FACULDADE DOCTUM DE CARATINGA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATEUS FREITAS DA COSTA
PAULO RICARDO

UMA ANÁLISE COMPARATIVA DE BIBLIOTECAS EM LINGUAGEM JAVASCRIPT PARA A RENDERIZAÇÂO DE MODELOS 3D EM TEMPO REAL EM AMBIENTES WEB

CARATINGA
2023

FACULDADE DOCTUM DE CARATINGA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATEUS FREITAS DA COSTA
PAULO RICARDO

UMA ANÁLISE COMPARATIVA DE BIBLIOTECAS EM LINGUAGEM JAVASCRIPT PARA A RENDERIZAÇÂO DE MODELOS 3D EM TEMPO REAL EM AMBIENTES WEB

Trabalho de Conclusão de Curso apresentado à Faculdade de Ciência da Computação da Faculdade Doctum de Caratinga, como requisito para a aprovação na disciplina de TCC I.
Orientador: XXXX-XXXX-XXXX

CARATINGA
2023

SUMÁRIO

GLOSSÁRIO **4**
1. APRESENTAÇÃO **5**
2. OBJETO DE ESTUDO **6**
3. HIPÓTESES **6**
4. OBJETIVOS **6**
4.1 Objetivos Gerais **6**
4.1 Objetivos Específicos **7**
5. JUSTIFICATIVA **7**
6. REFERENCIAL TEÓRICO **8**
6.1. Computação Gráfica **8**
6.2. OpenGL **10**
6.3. Gráficos 3D **11**
6.4. HTML **13**
6.5. Canvas **14**
6.6. Javascript **15**
6.7. WebGL **15**
7. METODOLOGIA **17**
7.1. Escolha de Bibliotecas **17**
7.1.1. Bibliotecas Disponíveis 17
7.1.2. Critérios de Seleção 19
7.1.2.1. Código Aberto *19*
7.1.2.2. Documentação Disponível *19*
7.1.2.3. Disponibilidade de Tutoriais e Exemplos *19*
7.1.2.4. Formatos de Modelos Suportados *20*
7.1.2.5. Linguagem Requerida *20*
7.1.2.6. Status de Desenvolvimento *20*
7.1.3. Bibliotecas Selecionadas 20
7.1.3.1. Three.js *20*

7.1.3.2. Babylon.js.....21
7.1.3.3. Xeokit SDK.....22
7.1.3.4. Playcanvas.....22
7.2. Métricas de Software.....22
7.2.1 Métricas Qualitativas.....22
7.2.1.1. Qualidade da Documentação.....22
7.2.2 Métricas Quantitativas.....23
7.2.2.1. Desenvolvimento do Projeto.....23
7.2.2.2. Formato de Arquivos Suportados.....23
7.2.2.3. Funcionalidades Presentes.....23
7.2.2.4. Performance.....24
7.2.2.5. Tamanho da Biblioteca.....25
7.3. Ferramentas utilizadas.....25
7.3.1. Navegadores e Mensuradores de Performance.....25
7.3.2. Configuração dos Computadores.....25
CRONOGRAMA.....27
SUMÁRIO HIPOTÉTICO.....28
REFERÊNCIAS.....30

GLOSSÁRIO

HTML	HyperText Markup Language
HTTP	HyperText Trasnfer Protocol
CSS	Cascading Style Sheets
WEB	World Wide Web
API	Application Programming Interface
CG	Computação Gráfica
CAD	Computer Aided Design
RGB	Red Green Blue
GPU	Graphics Processing Unit
W3C	World Wide Web Consortium
GLTF	GL Transmission Format
CPU	Central Processing Unit
SDK	Software Development Kit
OpenGL	Open Graphics Library

1. APRESENTAÇÃO

Uma das áreas mais proeminentes da computação dos dias de hoje refere-se a *World Wide Web*. Desenvolvida por Tim Berners-Lee em 1990 (WEBFOUNDATION, c2008-2022), as tecnologias fundamentais para o seu funcionamento consistem noa linguagem de formatação *HTML* e no protocolo *HTTP,* que permite o compartilhamento de recursos através da rede.

Além dos tradicionais arquivos de texto, o protocolo *HTTP* permite atualmente o compartilhamento de diversos outros tipos de mídia, como arquivos de estilo (*CSS), scripts (Javascript),* imagens, áudios, modelos 3D, entre outros (MDN, c1998-2023), permitindo assim o desenvolvimento de aplicações muito mais complexas e enriquecendo a experiência dos usuários conectados a *internet.*

Dada a evolução dos equipamentos de *hardware* nos últimos anos, se tornou cada vez mais comum a utilização de aplicações que usam de modelagem 3D para suas necessidades. A constante evolução da computação gráfica para o desenvolvimento de jogos, fotografia digital, design gráfico, cartografia, visualização de dados, entre muitos outros, fez crescer a demanda para que estas tecnologias se tornassem disponíveis também nos navegadores *Web*. Assim, foi desenvolvida o padrão *WebGL*, versão da *API* gráfica *OpenGL,* funcionando nativamente no navegador, permitindo assim o desenvolvimento de gráficos 3D com aceleração de *hardware* em *websites*. (GDAD-S-RIVERzz, 2017)

Na engenharia de *software*, Parnas (1972, p1053-1058, apud OLIVEIRA, 2017, p15) define a modularidade como a capacidade de dividir um sistema em submódulos, que podem ser modificados individualmente sem informações adicionais das outras. Assim, é possível que o desenvolvedor de uma aplicação utilize de bibliotecas de códigos fonte fornecidas por terceiros, que ao abstrair tarefas complexas de baixo nível, podem facilitar o desenvolvimento de um programa.

Considerando todos esses fatos, o seguinte trabalho busca fazer um estudo comparativo de diversas bibliotecas feitas em *Javascript* construídas em cima das capacidades dos navegadores em renderizar objetos 3D, fornecendo assim resultados qualitativos e quantitativos acerca de suas capacidades funcionais, com as vantagens e

desvantagens que um desenvolvedor terá ao optar por utilizar qualquer uma delas.

2. OBJETO DE ESTUDO

O objeto de estudo deste projeto trata da utilização de bibliotecas, ferramentas e frameworks javascript selecionados que sejam capazes de fornecer uma camada de abstração do *WebGL*, e dessa forma, possam ser utilizadas em projetos *Web* que buscam renderizar objetos de três dimensões em tempo real.

Dessa forma, busca-se catalogar e analisar as diversas opções que existem no mercado de software a fim de procurar as melhores alternativas que um desenvolvedor *Web* possa utilizar no seu projeto, a depender dos seus requerimentos.

3. HIPÓTESES

Tendo feito as comparações entre as diferentes ferramentas utilizadas, é esperado obter grandes e pequenas diferenças entre cada uma de suas métricas, e dessa forma, será possível obter um panorama geral das vantagens e desvantagens que um desenvolvedor terá ao utilizar em cada uma delas.

É esperado que algumas bibliotecas sejam melhores para uso geral, enquanto outras sejam superiores em propósitos específicos. Algumas bibliotecas podem performar melhor quando se trata de uma aplicação menor, com baixos requisitos de funcionalidades, além de ter um usabilidade mais simples, enquanto outras serão mais adaptadas para maiores projetos, de acordo com seu nível de complexidade.

4. OBJETIVOS

4.1 Objetivos Gerais

Investigar diversas bibliotecas para a linguagem de programação *Javascript* que possam ser utilizadas para o desenvolvimento de aplicações 3D em aplicações Web, e assim realizar uma análise comparativa entre elas utilizando diversas métricas de *software*, ressaltando os pontos positivos e negativos de cada uma.

4.1 Objetivos Específicos

A fim de realizar os propósitos definidos pelo objetivo geral, o seguinte trabalho apresenta as seguintes metas:

- Selecionar as bibliotecas *Javascript* que deverão ser utilizadas para a comparação, utilizando de critérios como: popularidade, disponibilidade da documentação e atualização do projeto.
- Fazer uma apresentação do uso de cada biblioteca escolhida, expondo sua origem, seu funcionamento em código fonte e suas funcionalidades.
- Buscar e entender as diversas métricas de software que deverão ser utilizadas para a análise de cada biblioteca analisada.
- Para cada biblioteca avaliada, fazer uma comparação de seus aspectos qualitativos, como a qualidade da documentação.
- Analisar, para cada biblioteca, os seus aspectos quantitativos básicos, tais como: funcionalidades presentes, formatos de arquivo suportados, e atualização do projeto pelos desenvolvedores. Além destes, aspectos quantitativos como performance *e* tamanho da biblioteca, serão avaliados pela execução de uma série de testes práticos através de programas que utilizarão as diversas funcionalidades presentes de renderização disponíveis em cada uma das bibliotecas.
- Fazer uma conclusão geral, resumindo os dados levantados ao longo do projeto, e ressaltando os pontos positivos e negativos descobertos na pesquisa e no uso das bibliotecas analisadas.

5. JUSTIFICATIVA

Em todas as áreas da computação, a evolução dos paradigmas é um fenômeno inexorável. A medida que a tecnologia se desenvolve, novas possibilidades antes não exploradas se tornam cada vez mais comuns no dia a dia dos desenvolvedores e usuários. O desenvolvimento dos navegadores e dos computadores *desktop* permitiu a criação *websites* cada vez mais complexos, com efeitos visuais detalhados, tais como: sombras, luzes, animações complexas, entre muitos outros.

Dessa forma, o uso do 3D, antes conhecido sobretudo nos vídeo games de computador ou consoles, nas programas de edição e modelagem, se tornaram possíveis também de serem executados no navegador, abrindo um novo leque de possibilidades, permitindo designs mais realistas, interações de usuário mais elaboradas, e proporcionando uma experiência mais rica ao usuário.

No entanto, no surgimento de muitas tecnologias, uma dificuldade no seu uso se manifesta na medida em que os desenvolvedores têm de adaptar-se a elas antes de poderem utilizá-la, e assim incoporarem-na no seu conjunto de habilidades. Dado que a computação gráfica é uma área notoriamente complexa, requerindo conhecimentos de matemática e algebra linear, disciplinas comumente desnecessárias no desenvolvimento da maioria das aplicações Web, a disponibilidade de ferramentas de mais alto nível que auxiliem estes desenvolvedores a produzir gráficos 3D sem exigir deles essas competências técnicas se faz necessária.

Assim, a fim de facilitar aos desenvolvedores a pesquisa e o estudo por estas tecnologias, o seguinte trabalho busca analisá-las e fazer uma comparação detalhada entre elas, permitindo que os programadores que queiram utilizá-las adquiram um conhecimento prévio que os permitam fazer uma melhor decisão na escolha de sua ferramenta.

6. REFERENCIAL TEÓRICO

Este referencial teórico é dividido em sete subcapítulos. O primeiro, descreve o conceito de computação gráfica, o segundo, a *API OpenGL*, o terceiro, o funcionamento de gráficos 3D. Já os capítulos quatro, cinco e seis tratam das tecnologias *WEB,* que são o *HTML,* a tag *canvas* e o *Javascript.* Por fim, o sétimo trata da *API WebGL.*

6.1. Computação Gráfica

A computação gráfica pode ser definida como:

> A Computação Gráfica reúne um **conjunto de técnicas que permitem a geração de imagens** a partir de modelos computacionais de objetos reais, objetos imaginários ou de dados quaisquer coletados por equipamentos na natureza. (SILVEIRA, 2018)

Dessa forma, entende-se que a computação gráfica é uma forma de representar objetos, que podem ser derivados do mundo real ou imaginários, num computador, assim gerando imagens que serão visualizadas por usuários, os quais podem ter objetivos educacionais, corporativos, científicos, lúdicos, entre outros.

A história da computação gráfica tem origem na década de 1950, época no qual pesquisadores do MIT foram capazes de criar um computador que podia processar imagens em três dimensões. Mais tarde, nos anos 1970, Ivan Sutherland e David Evans foram capazes de desenvolver um software que gerava e modelava gráficos. Posteriormente, foram criados os modelos 3D, utilizados amplamente na indústria cinematográfica, dando vida aos filmes da Disney e da Pixar. Outro grande marco a ser mencionado é o lançamento do computador da Apple, Macintosh, em 1980. (COUTINHO, 2021)

Devido ao alto custo que essa tecnologia demandava para ser utilizada, inicialmente a CG estava limitada às estações gráficas que possuíam recursos computacionais mais potentes, o que era caro. Foi nos anos 1990 que a evolução do hardware e dos dispositivos gráficos permitiu o barateamento desses recursos. Assim, acelerou-se o surgimento de ambientes que utilizavam da interface gráfica como o sistema operacional Windows. (SIQUEIRA, [s.d])

A computação gráfica se divide, principalmente, em três subcategorias (GONÇALVES, c2020-2021):

- Síntese de imagens, que se refere a produção de imagens sintéticas, em duas ou três dimensões;
- Análise de imagens, que busca obter dados provenientes de uma imagem, e traduzi-los em informações úteis para uma determinada função, como, por exemplo, reconhecimento facial.
- Processamento de imagens, o qual busca manipular uma determinada imagem com ajustes de cor, brilho, aplicações de filtros, etc. Exemplos incluem o *Photoshop* e os programas de edição de vídeo.

Atualmente, a computação gráfica é parte fundamental de diversos sistemas computacionais, e se tornou uma ferramenta essencial na vida e cotidiano de muitas

pessoas. Entre os seus usos, incluem-se: interface de usuário de programas e sistemas operacionais, visualização de gráficos, editoração eletrônica, *CADs* (mecânico, civil, elétrico), simulações e animações, arte e comércio, cartografia, e muitos outros. (PINHO, [s.d])

6.2. OpenGL

Um dos grandes instrumentos utilizados na computação gráfica é o *OpenGL*, que segundo o grupo Khronos (c2023) é a *API* gráfica 2D e 3D mais utilizada na indústria, trazendo milhares de aplicações em uma variedade de plataformas, sendo independente do sistema de janela e do sistema operacional. O *OpenGL* permite que desenvolvedores criem diversas aplicações gráficas visualmente atraentes de alta performance, como *CADs*, sistemas operacionais, jogos e realidade virtual.

A história do *OpenGL* remonta a década de 1980, época no qual não havia uma padronização comum das interfaces gráficas, fazendo com que os desenvolvedores do período tivessem de produzir seus softwares para cada hardware diferente. Dessa forma, uma empresa de nome *Silicon Graphics Inc. (SGI)* criou o *IRIS GL*, um padrão gráfico que chamou a atenção da indústria. Mais tarde, devido ao interesse da empresa em manter parte do código fonte como proprietário, a ramificação de código totalmente aberto, o *OpenGl*, surgiu, estando em contínuo desenvolvimento até hoje, sendo suportado por diversas empresas, como a *AMD, Dell, HP, Intel, Nvídia, Sun (Oracle)* e outras. (SILVA, 2012)

O *OpenGL* é independente da implementação de uma placa de vídeo específica, assim, o programador apenas deve dizer apenas o que o computador deve fazer, e não como. Possui 250 funções, utilizadas para controlar os desenhos que serão realizados na tela do computador. Tem suporte a diversas primitivas, como pontos, linhas e triângulos. *Rendering pipeline* é o termo que se utiliza para descrever o processo que acontece quando os comandos vão sendo passados para a *API* no *buffer* de comandos, nos quais são adicionados coordenadas, cores, texturas, iluminação e outros efeitos, até chegarem no *framebuffer*, do qual é enviado para a tela do computador. (MANSSOUR, 2003)

6.3. Gráficos 3D

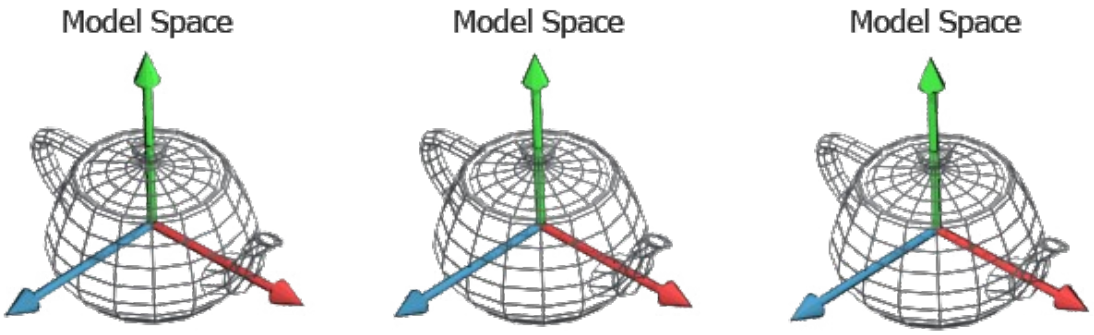
Uma tela de computador é composta por milhares de pixeis, pequenos quadrados constituídos de três feixes de luz, um de cada cor, vermelho, verde e azul, um padrão conhecido como *RBG*. Assim, num processo complexo de transformações, é possível determinar a luminosidade de cada feixe de luz para cada pixel, o qual misturando as cores aditivamente, pode formar milhões de cores diferentes, permitindo a criação de imagens complexas e realistas. (MEIRELLES, 2002)

Para determinar o local nos quais os pixeis serão desenhados, um sistema de coordenadas é necessário. No caso de gráficos 3D, o *OpenGL* um sistema conhecido como destro, no qual as coordenadas começam no quanto inferior esquerdo, apontando para cima e para a direita, e a terceira dimensão, o Z, aponta para a frente da tela. (VRIES, 2020)

Os objetos tridimensionais são formados por polígonos – usualmente triângulos -, que, por sua vez, compõem os dados de vértices, os quais além de guardar coordenadas no espaço tridimensional, também podem conter texturas, cores e o modo de reflexão de luz. Esses vértices são então enviados a *GPU* para serem processados por programas chamados *shaders*, responsáveis por transformar a representação 3D de um objeto numa imagem que a tela seja capaz de exibir. (GREGORY, c2015).

O *vertex shader* é o programa da *GPU* responsável por realizar estas transformações. Primeiro, o objeto, chamado de *model*, é transformado no seu espaço local, no qual são aplicados a rotação e escalonamento.

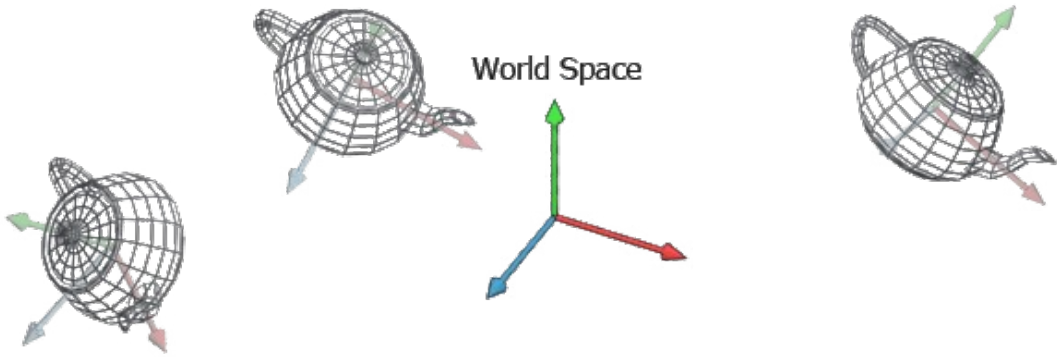
Figura 1: Um modelo de um objeto 3D: visão local



Fonte: Codinglabs[1].

Após o objeto ser transformado localmente, o *view space* determina a posição do objeto em relação a cena e ao restante dos objetos.

Figura 2: Um modelo de um objeto 3D: visão na cena



Fonte: Codinglabs[2].

Por fim, a matriz de projeção da câmera é responsável por fornecer uma visão do mundo a partir de uma localização e direção, criando a imagem 2D que se vê na tela do computador. (MÖLLER et al, 2018)

Tendo o *vertex shader* lidado com as posições dos polígonos, inicia-se a montagem da forma e da geometria, e depois, o processo de rasterização, no qual os

1 Disponível em: http://www.codinglabs.net/article_world_view_projection_matrix.aspx. Acesso em 24 abr. 2023

2 Disponível em: http://www.codinglabs.net/article_world_view_projection_matrix.aspx. Acesso em 24 abr. 2023

espaços entre os triângulos são determinados para serem coloridos, descartando os pixeis que não estiverem entre os limites da tela. Ainda, o *fragment shader* é responsável por aplicar a cor ao objeto, utilizando-se de cores, texturas e fontes de luz. Por fim, são realizados os testes de profundidade, e a partir deles pode-se destacar objetos obstruídos no plano de fundo, além de aplicar efeitos de transparência e mistura de cores. (VRIES, 2020)

6.4. HTML

HTML, segundo a (HTML Standard 2023) é uma linguagem de marcação utilizada para estruturar e exibir conteúdo na web. Ele é a base para a criação de páginas web e é interpretado pelos navegadores para renderizar o conteúdo visual que é apresentado aos usuários.

O *HTML* é uma linguagem importante para qualquer desenvolvedor web, pois é a base para a criação de páginas web. É importante entender os conceitos básicos da linguagem, como tags, elementos e atributos, bem como a estrutura da página e as especificações do *HTML5*. Com esse conhecimento, é possível criar páginas web eficazes e acessíveis, com um código limpo e organizado. (HTML Standard, 2023)

O *HTML* é baseado em *tags*, que são elementos com uma sintaxe específica usados para definir diferentes partes do conteúdo da página. As *tags* são escritas dentro de chevrons (<>) e podem incluir atributos que fornecem informações adicionais sobre a tag. Por exemplo, a *tag <p>* é usada para definir parágrafos e pode incluir o atributo *class* para indicar uma classe *CSS* que define o estilo do parágrafo. (HTML Standard, 2023)

Sua organização é feita em elementos que compõem a estrutura da página. O elemento *<html>* é o elemento raiz da página e contém todos os outros elementos. O elemento *<head>* contém informações sobre a página, como o título e metadados, enquanto o elemento *<body>* contém o conteúdo principal da página, como textos, imagens, vídeos, formulários e outros elementos. (HTML Standard, 2023)

A estrutura do *HTML* é definida por um conjunto de especificações, que são desenvolvidas pelo *W3C*, uma organização internacional que define padrões para a

web. A última versão do *HTML* é o *HTML5,* que traz diversas melhorias em relação às versões anteriores, como suporte para mídia integrada, semântica melhorada, recursos de acessibilidade e mais. (HTML Standard, 2023)

A linguagem também suporta a utilização de *CSS* e *JavaScript,* que permitem definir o estilo e comportamento da página, respectivamente. O *CSS* é utilizado para definir o estilo visual da página, como cores, fontes, layout e animações, enquanto o *JavaScript* é utilizado para adicionar interatividade à página, como validação de formulários, animações e manipulação de eventos. (HTML Standard, 2023)

6.5. Canvas

A *tag canvas* na (HTML Standard 2023) é um elemento *HTML5* que permite criar gráficos e animações diretamente no navegador, utilizando principalmente *JavaScript.* Com o *canvas,* é possível desenhar formas, linhas, curvas, imagens e textos em tempo real, tornando possível criar animações e gráficos interativos.

O *canvas* foi introduzido no *HTML5* como uma alternativa aos gráficos baseados em imagens, que muitas vezes são lentos e difíceis de manipular. O *canvas* permite que os desenvolvedores criem gráficos e animações complexas diretamente no navegador, sem precisar de *plugins* ou software adicional. (HTML Standard 2023)

Para utilizar o canvas, é necessário incluir a *tag* <*canvas*> no código *HTML*, que define a área em que os gráficos serão desenhados. A partir daí, é possível utilizar a *API* de desenho do canvas em *JavaScript* para desenhar e manipular os gráficos. Alguns dos recursos mais avançados do canvas incluem a capacidade de criar efeitos visuais complexos, como sombras, gradientes e transparências, e a capacidade de interagir com o usuário por meio de eventos do mouse e do teclado. (HTML Standard 2023)

O canvas é amplamente utilizado para criar jogos, visualização de dados, gráficos interativos e outras aplicações web que exigem gráficos e animações complexas. A sua utilização requer conhecimento em programação e em manipulação de gráficos em *JavaScript,* mas existem muitos recursos e tutoriais disponíveis para ajudar os desenvolvedores a aprender a utilizar o *canvas* de forma eficiente. (HTML

Standard 2023)

6.6. Javascript

JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos, que é amplamente utilizada na criação de aplicações web (JavaScript 2023). Desenvolvida originalmente pela *Netscape* em 1995, a linguagem foi padronizada pelo Ecma International na especificação *ECMAScript,* que é a base para a maioria das implementações modernas de *JavaScript.*

JavaScript é uma linguagem interpretada, ou seja, não precisa ser compilada antes de ser executada. Ela é executada diretamente no navegador do usuário, permitindo que as páginas web sejam dinâmicas e interativas. Além disso, ela é uma linguagem de *script,* o que significa que pode ser incluída diretamente em páginas web, sem a necessidade de arquivos externos. (JavaScript 2023)

Outra característica importante do *JavaScrip*t é sua capacidade de manipular o *DOM*, que é a estrutura de objetos que representa o conteúdo de uma página web. Com o *JavaScript*, é possível adicionar, remover ou modificar elementos *HTML* e *CSS* em tempo real, permitindo que as páginas web sejam dinâmicas e reativas às ações do usuário. (JavaScript 2023)

JavaScript é amplamente utilizado em todo o mundo para desenvolvimento de aplicações web, desde pequenos *scripts* até aplicativos web complexos e sofisticados. Além disso, o *JavaScript* é uma das principais linguagens de programação usadas em frameworks de desenvolvimento web, como *AngularJS, ReactJS e VueJS.* (JavaScript 2023)

6.7. WebGL

WebGl é uma *API* para *javascript* que tem o propósito de criar gráficos de duas ou três dimensões em alta performance em qualquer *browser* compatível sem a necessidade de utilizar *plugins* de terceiros. O *WebGL* realiza este feito ao implementar uma *API* quase totalmente compatível com o *OpenGl ES*, uma versão criada especificamente para rodar em dispositivos embarcados e *mobile.* (PETERBE et al,

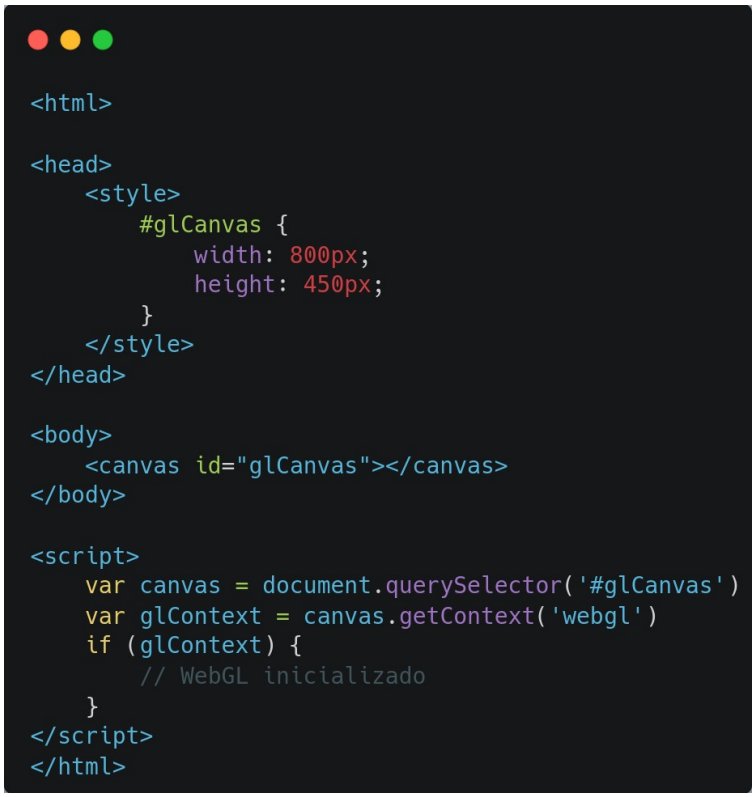
2023)

Quando os vídeogames já utilizavam de gráficos 3D, a *Web* ainda não possui um recurso nativo para a implementação deles. Assim, as fundações *Mozilla* e *Opera* mostraram alguns experimentos iniciais no qual renderizavam uma cena 3D no *HTML* utilizando a tag *canvas*. Mais tarde, vários navegadores colaboraram para o desenvolvimento do padrão *WebGL*, fornecendo uma camada independente que funcionaria em todas as plataformas, permitindo assim diversos jogos de alta qualidade serem desenvolvidos para a *WEB,* em como o surgimento de bibliotecas, como o *Three.js.* Mais tarde, o *WebGL* foi atualizado para sua versão 2. (GDAD-S-RIVER, 2017)

Teve origem no ano de 2009, quando uma organização nomeada *Khronous group* iniciou uma equipe para o projeto, no qual estavam presentes representantes da *Apple, Google, Mozilla* e *Opera*. O seu propósito era suportar gráficos 3D na *Web*. Criada por Vladmir Vukićević, escolheu o *OpenGL* como *API* pela quantidade de desenvolvedores familiares com a tecnologia. Apesar do nome, a *WebGL* nem sempre utiliza o *OpenGL* como motor subjacente, podendo utilizar o *DirectX* no sistema operacional *Windows*. (MANOR, 2021)

A fim de desenvolver uma aplicação utilizando o *WebGL*, o desenvolvedor deverá utilizar a tag <canvas> no código *HTML*, e então, utilizar o *javascript* e criar um contexto *WebGL* neste elemento, habilitando assim o uso das funções comuns do *OpenGL* que serão renderizadas no espaço delimitado pelo canvas. Isso pode ser feito manualmente, chamando as funções nativas do *WebGL*, ou utilizando uma biblioteca responsável por abstrair estes comandos, provendo funcionalidades de alto nível. Abaixo segue um exemplo de inicialização do *WebGL* num navegador.

Figura 3: Código HTML: inicializando WebGL no elemento canvas



```html
<html>

<head>
    <style>
        #glCanvas {
            width: 800px;
            height: 450px;
        }
    </style>
</head>

<body>
    <canvas id="glCanvas"></canvas>
</body>

<script>
    var canvas = document.querySelector('#glCanvas')
    var glContext = canvas.getContext('webgl')
    if (glContext) {
        // WebGL inicializado
    }
</script>
</html>
```

Fonte: Os autores, 2023.

7. METODOLOGIA

7.1. Escolha de Bibliotecas

Para a comparação a ser realizada neste projeto, deverão ser utilizadas diversas bibliotecas programadas em linguagem *Javascript* que sejam capazes de renderizar os gráficos 3D. Dessa forma, o presente capítulo ocupa-se de apresentar uma lista de diversas bibliotecas disponíveis, explicar os critérios de seleção para filtragem, e por fim, apresentar as bibliotecas escolhidas para análise.

7.1.1. Bibliotecas Disponíveis

A seguir estão listadas as diversas bibliotecas que foram encontradas na *Web* para serem selecionadas. A tabela apresenta o nome da biblioteca, sua licensa, site no qual é disponibilizada, se foi feita em linguagem *Javascript* ou o seu derivado, o *Typescript*. Também apresenta a data da última atualização feita pelos seus

desenvolvedores, indicando caso ainda esteja ativo com a atributo “Ativo” para as bibliotecas que sofreram mais de 10 atualizações no ano de 2023.

Nome	Licensa	Disponibilidade	Linguagem	Última Atualização
Three.js	MIT	Github / Site	Javascript	Ativo
Babylon.js	Apache License 2.0	Github / Site	Javascript	Ativo
RedGL	MIT	Github	Javascript	17/08/2022
Webgl-operate	MIT	Github / Site	Typescript	12/12/2022
Zogra Render	MIT	Github	Typescript	16/02/2023
ClayGL	BSD-2-Clause license	Github / Site	Javascript	31/10/2021
Xeogl	MIT	Github / Site	Javascript	14/05/2020
Litescene	MIT	Github / Site	Javascript	02/08/2020
Playcanvas	MIT	Github / Site	Javascript	Ativo
CopperLicht	CopperLicht License	Site	Javascript	Não Disponível
Xeokit SDK	AGPL V3	Github / Site	Javascript	Ativo

OpenJSCAD	MIT	Github / Site	Javascript	Ativo

7.1.2. Critérios de Seleção

Esta seção tem o objetivo de detalhar os critérios a serem utilizados para a filtragem das bibliotecas expostas na tabela do subcapítulo anterior.

7.1.2.1. Código Aberto

Para que o desenvolvedor possa ler, analisar e potencialmente contribuir e realizar modificações nas funcionalidades da biblioteca em questão, é fundamental que seja de código aberto. Dessa forma, todas as bibliotecas analisadas para este trabalho deverão cumprir este requisito.

7.1.2.2. Documentação Disponível

A documentação de uma biblioteca pública é necessária para os desenvolvedores possam utilizá-la corretamente. Módulos, classes, funções e globais e toda a *API* de cada biblioteca deve estar disponível a fim de que suas funcionalidades e uso sejam claros para o desenvolvedor.

7.1.2.3. Disponibilidade de Tutoriais e Exemplos

Além de um documentação completa da *API,* é de grande conveniência que o desenvolvedor possa encontrar explicações gerais, lista de funcionalidades, exemplos variados de códigos fontes ou aplicações desenvolvidas com a biblioteca a fim de viabilizar o seu aprendizado. Portando, as bibliotecas selecionadas devem possuir tutoriais em forma de texto disponível, ou comentários presentes da *API Docs*, com trechos de código fonte que exemplifiquem seus usos.

7.1.2.4. Formatos de Modelos Suportados

A fim de realizar os testes práticos de avaliação de performance, será necessário o uso extensivo de modelos 3D que serão carregados e renderizados pela biblioteca. Dessa forma, é necessário utilizar um formato de arquivo único a fim de padronizar os testes. O formato *GLFT* foi escolhido por sua popularidade, e assim, todas as bibliotecas possuir compatibilidade com modelos codificados nesse tipo de arquivo.

7.1.2.5. Linguagem Requerida

A linguagem de programação Javascript *é* o padrão implementado nos navegadores para aplicações *Web*. Outras linguagens, como o *Typescript,* derivado e compilado para o *Javascript*, também é utilizado frequentemente em diversos sistemas e bibliotecas. Os autores deste trabalho optaram por utilizar apenas bibliotecas que estão escritas diretamente na linguagem *Javascript*, descartando seus derivados.

7.1.2.6. Status de Desenvolvimento

É importante ao desenvolvedor cliente de uma biblioteca que a mesma esteja em status de desenvolvimento, ou seja, que o seu código fonte esteja sendo atualizado constantemente para a adição de funcionalidades, correção de bugs, aumento de compatibilidade e acompanhamento dos padrões do *Web.* Por causa disso, foram descartadas todas as bibliotecas que não tiveram atualizações no ano de 2023.

7.1.3. Bibliotecas Selecionadas

A partir dos critérios definidos acima, foram selecionados quatro exemplares que cumprem todos os requisitos: *Three.js, Babylon.js, Xeokit SDK e Playcanvas.*

7.1.3.1. Three.js

O *Three.js* é uma biblioteca open-source para a linguagem *JavaScript* que permite aos desenvolvedores criar cenas 3D de forma organizada e renderizá-las diretamente do navegador web. Ela tem se tornado uma das bibliotecas de desenvolvimento de gráficos 3D mais populares da atualidade, sendo principalmente

utilizada para o desenvolvimento de jogos online, demonstrações e modelos. (BOSNJAK, 2018)

O *Three.js* foi lançado por Ricardo Cabello em 2010 na plataforma *GitHub.* Originalmente estava sendo desenvolvido em *ActionScript* mas em 2009 foi portado para *JavaScript*, removendo a necessidade de ser previamente compilado pelo desenvolvedor. Desde os treze anos de atividade desde seu lançamento, o *Three.js* recebeu inúmeras contribuições e atualizações de diversos desenvolvedores, chegando a exceder 25.000 *commits.*(Lilly021, 2022)

É uma ótima escolha tanto para desenvolvedores experientes quanto para desenvolvedores novos no desenvolvimento 3D, sendo significativamente mais fácil de ser utilizado que o *WebGL* puro. Sendo especialmente feito para desenvolver cenas e animações 3D sem se preocupar com sua interação com o hardware. Para utilizar o *Three.js* é necessário que o desenvolvedor possua pelo menos conhecimento básico em programação em *JavaScript e HTML,* além de saber *CSS* e seus seletores. (Three.js, 2023).

Outro fator que pode beneficiar os desenvolvedores ao utilizar esta biblioteca é a grande disponibilidade de exemplos e tutoriais na qual os desenvolvedores podem achar facilmente que procuram para desenvolver seus projetos. Porém sua documentação não está escrita muito bem, e possui uma falta significativa de descrições e informações oque pode forçar os desenvolvedores a gastar mais tempo para encontrar a informação que precisa. (Web Development, 2019)

7.1.3.2. Babylon.js

Babylon.js é uma 3D engine open-source baseada em *WebGL* e *JavaScript* usada para desenvolver elementos 3D complexos e interativos que podem ser executados por navegadores web. Ela é popular entre desenvolvedores que procuram desenvolver jogos *Web*, por possuir inúmeras funcionalidades embutidas e possuir uma documentação organizada e bem explicada. (WEBER, 2015)

Originalmente foi criada como um projeto *open-source* pelo engenheiro da *Microsoft* David Catuhe e posteriormente se tornando seu trabalho em tempo integral.

Conduzindo seu time com o objetivo de desenvolver a mais simples e poderosa ferramenta de renderização web, em 2015 o *Babylon* já demonstrava suas grandes capacidades, conquistando a atenção de seus concorrentes. (IRWIN, 2021)

Por ser focada principalmente em desenvolvimento de jogos, *Babylon* possui funções específicas que outras bibliotecas não possuem. Algumas destas funções são detecção de colisão, gravidade de cenário, câmeras orientadas, que facilitam o desenvolvimento destas aplicações, podendo ser desenvolvido usando apenas o código em *JavaScript*. (WEBER, 2015)

O *Babylon* possui também o *Babylon Playground,* que é um site simples em que desenvolvedores podem aprender a utilizar o *Babylon* com mais facilidade. Ele permite que o desenvolvedor escreva o código no espaço à esquerda e veja a direita atualização renderizada instantaneamente.(Babylon.js, 2023)

7.1.3.3. Xeokit SDK

Xeokit é um SDK de gráficos 3D para JavaScript desenvolvida pela Xeolabs para visualização de modelos BIM (Building information modeling) e engenharia 3D de alta precisão e detalhes. Capaz de renderizar modelos grandes na maioria dos navegadores sem a necessidade de plugins, inclusive em mobile.(Xeokit SDK, 2023)

Com o xeokit é possível manter os arquivos, visualizador e ferramentas de conversão em seu próprio servidor. Além de possuir uma velocidade de carregamento alta, utilizando principalmente o formato .XKT um dos formatos geométricos mais compactos existentes. O xeokit é construído especialmente para renderizar modelos grandes e detalhados com precisão direto do navegador sem perder qualidade. (CREOOX, 2023)

Xeokit possui o código aberto batendo ser modificado de acordo com a necessidade do desenvolvedor com poucas restrições. Além de possuir uma ampla documentação, exemplos e tutoriais disponíveis gratuitamente para ajudar desenvolvedores iniciantes a utilizar a biblioteca.(Xeokit SDK, 2023)

7.1.3.4. Playcanvas

Playcanvas é uma engine de aplicação interativa 3D com base em HTML5 e JavaScript focada no desenvolvimento de jogos. Ela é uma plataforma hospedada na web e não possui a necessidade de fazer nenhuma instalação, e pode ser acessada de qualquer dispositivo e qualquer navegador suportado. (Playcanvas Manual, 2023)

Playcanvas é uma das lideres de mercado em relação a game engine em WebGL, sendo capaz de criar aplicações em AR (Augmented Reality) e VR (Virtual Reality). É utilizada por desenvolvedores independentes e também por empresas conhecidas no mercado, sendo algumas delas por exemplo King, Disney e Nickelodeon. (Playcanvas, 2023)

Também é utilizada para criar aplicações de configurações e visualização arquitetural. Utilizando suas ferramentas de edição visual e manejamento de assets os desenvolvedores podem criar incríveis elementos gráficos interativos que podem ser suportados em diferentes dispositivos com diferença mínima de performance. (Playcanvas, 2023)

O Playcanvas possui uma facilidade de utilização significativa para desenvolvedores iniciantes, por seus extensos recursos para desenvolvedores que facilitam o aprendizado. Sendo eles o manual do usuário, a documentação, o fórum oficial, tutoriais e inúmeros exemplos disponíveis pela internet. (Playcanvas Manual, 2023)

7.2. Métricas de Software

O capítulo apresentado a seguir tem como função apresntar as diversas métricas de *software* que deverão ser utilizadas para a análise e comparação de cada biblioteca a ser avaliada no projeto. Segue-se um subcapítulo para as métricas qualitativas, e outro para as quantitativas.

7.2.1 Métricas Qualitativas

7.2.1.1. Qualidade da Documentação

A qualidade de uma documentação é um aspecto crucial a ser avaliado pelo

desenvolvedor que pretende consumir uma biblioteca como *API* em seu projeto. A documentação permite que desenvolvedores conheçam as funcionalidades presentes da biblioteca, e entendam *modus operanti* de sua utilização em código. Assim sendo, serão analisados a qualidade da documentação disponível para cada uma das bibliotecas avaliadas. Aspectos a serem considerados incluem: comentários e explicações da *API Docs*, guia e manual do usuário, desmontrações em código fonte das funcionalidades, presença de projetos completos desenvolvidos e comentários presentes no código fonte.

7.2.2 Métricas Quantitativas

7.2.2.1. Desenvolvimento do Projeto

Esta métrica se refere ao engajamento da comunidade responsável pelo desenvolvimento do projeto. Todas as bibliotecas selecionadas para avaliação possui pelo menos uma atualização no ano em que este trabalho foi escrito. No entanto, as bibliotecas não possuem todas a mesma velocidade de desenvolvimento, nem a mesma quantidade de código sendo alterada ou adicionada constantemente. Assim, pode-se avaliar, a partir de dados disponíveis no *Github:* número total e recorrente de *commits*, número de *pull requests*, e número de contribuidores do projeto.

7.2.2.2. Formato de Arquivos Suportados

As bibliotecas de renderização 3D são capazes de desenhar modelos na tela do computador. Além de formas primitivas, como cubos, cilindros e outras formas geométricas, as bibliotecas também são capazes de carregar modelos customizados. Estes modelos podem ser encontrados na internet, ou serem desenvolvidos por através de um *software* de modelagem 3D*,* como o *Blender.* Estando disponíveis em diversos formatos de arquivos, a depender de sua origem, esta métrica busca listar todos os formatos suportados para o carregamento e renderização de cada biblioteca.

7.2.2.3. Funcionalidades Presentes

Computação gráfica envolve muito mais do que apenas carregar modelos e

dispô-los da tela do dispositivo. Uma biblioteca responsável por abstrair as funções do *OpenGL* pode oferecer muito mais do que uma simples renderização 3D, e pode conter com recursos muito mais avançados, tais como:

- Sistema de partículas
- Animação esqueleta
- Sistema de câmeras
- Compressão de modelos
- Sistema de física e colisão
- Materiais e geometrias disponíveis
- Sistema de iluminação e sombra
- Geração de *lightmaps*
- Efeitos de alteração de cores
- *Anti-aliasing*
- *Ambient occlusion*
- *Fog*
- *Motion Blur*

Dessa forma, bibliotecas que tiveram mais funcionalidades úteis ao desenvolvedor obterão vantagem nessa métrica. No entanto, há de ser considerado também os diversos usos de gráficos 3D, nos quais algumas funcionalidades podem ser úteis, enquanto outras, por aumentarem a complexidade do aprendizado a biblioteca, podem se apresentar como um estorvo ao cliente que deseja desenvolver certas aplicações que não necessitam desses recursos.

7.2.2.4. Performance

Aplicações como jogos, *CADs*, simulações, programas de visualizações e outros demandam do computador uma quantidade generosa de recursos para a renderização das complexias geometrias responsáveis por dar vida a suas funcionaliades. Segue-se então, que na escolha de uma biblioteca a ser utilizada para esse propósito, a sua capacidade de aproveitar eficientemente os recursos do computador, provendo uma alta performance ao usuário final é um aspecto essencial a ser avaliado.

Dessa forma, na análise da seguinte métrica propõe-se a coletar estatísticas que mensurem a performance da biblioteca avaliada, como o uso de *CPU,* memória alocada, tempo de inicialização e quadros por segundo.

Para que isso seja feito, serão desenvolvidos testes envolvendo cada uma das bibliotecas, no qual serão utilizados modelos 3D em formato *GLTF* obtidos na *Web*, e assim, carregados para a renderização de uma cena. Serão feitos diversos testes de renderização, avaliando diferentes funcionalidades e cenários, na medida em que as bibliotecas consigam suportar:

- Cena simples, com poucos objetos e contagem de polígonos.
- Cena complexa, com grande quantidade de objetos e polígonos.
- Com fontes de luzes.
- Cena animada.
- Sistema de partículas.
- Carregamento dos modelos.
- Pós-processamento de efeitos e cores.

7.2.2.5. Tamanho da Biblioteca

O tamanho da biblioteca se refere a quantidade de memória que o código fonte da mesma ocupa ao ser importada pelo desenvolvedor cliente para a sua utilização. Uma biblioteca maior pode desacelerar o tempo de carregamento da página, e por isso, este aspecto será levado em consideração no comparativo.

7.3. Ferramentas utilizadas

7.3.1. Navegadores e Mensuradores de Performance

A fim de padronizar os resultados e obter um conjunto variado de dados, serão feitos testes nos navegador *Google Chrome e* M*icrosoft Edge*, nos quais é possível utilizar a ferramenta de mensuração de performance *DevTools*.

Além das ferramentas contidas no navegador, as funções nativas da linguagem *Javascript* também serão utilizadas na coleta de dados de performance.

7.3.2. Configuração dos Computadores

Neste trabalho, a fim de obter uma variedade maior de resultados, as coletas de dados serão feitas de forma múltipla, através de testes realizados em dois computadores diferentes:

- Computador com processador AMD Ryzen 5 5500 (12 CPUs) ~3.6GHz, de placa de vídeo NVIDIA GeForce GTX 1660 SUPER, de memória RAM 16384MB DDR4, e sistema operacional Windows 11 Home 64 bits.
- Computador com processador Intel Core i3-4170 CPU (4 CPUs) ~3.7GHz, de placa de vídeo Mesa Intel HD Graphics 4400 (1.5 Gib), de memória RAM 3.7 GiB DDR3, e sistema operacional Arch Linux Kernel 6.1.26-1-lts.

CRONOGRAMA

ATIVIDADE	1° mês	2° mês	3° mês	4° mês	5° mês	6° mês
Revisão Bibliográfica						
Estudo das ferramentas						
Desenvolvimento dos Testes						
Coleta de						

Dados						
Análise dos Dados						
Escrita dos Dados						
Formatação						
Revisão Final						
Protocolamento						
Defesa Pública						

SUMÁRIO HIPOTÉTICO

GLOSSÁRIO

1. INTRODUÇÃO

2. REFERENCIAL TEÓRICO

2.1. Computação Gráfica

2.2. OpenGL

2.3. Gráficos 3D

2.4. HTML

2.5. Canvas

2.6. Javascript

2.7. WebGL

3. METODOLOGIA

3.1. Escolha de Bibliotecas

3.1.1. Bibliotecas Disponíveis

3.1.2. Critérios de Seleção

3.1.2.1. Código Aberto

3.1.2.2. Documentação Disponível

3.1.2.3. Disponibilidade de Tutoriais e Exemplos

3.1.2.4. Formatos de Modelos Suportados

3.1.2.5. Linguagem Requerida

3.1.2.6. Status de Desenvolvimento

3.1.3. Bibliotecas Selecionadas

3.1.3.1. Three.js

3.1.3.2. Babylon.js

3.1.3.3. Xeokit SDK

3.1.3.4. Playcanvas

3.2. Métricas de Software

3.2.1 Métricas Qualitativas

3.2.1.1. Qualidade da Documentação

3.2.2 Métricas Quantitativas

3.2.2.1. Desenvolvimento do Projeto

3.2.2.2. Formato de Arquivos Suportados

3.2.2.3. Funcionalidades Presentes

3.2.2.4. Performance

3.2.2.5. Tamanho da Biblioteca

3.3. Ferramentas utilizadas

3.3.1. Navegadores e Mensuradores de Performance

3.3.2. Configuração dos Computadores

4. PESQUISA

4.1. Análise das Métricas Qualitativas

4.1.1. Qualidade da Documentação

4.2. Análise das Métricas Qualitativas

4.2.1. Desenvolvimento do Projeto

4.2.2. Formato de Arquivos Suportados

4.2.3. Funcionalidades Presentes

4.2.4. Performance

4.2.5. Tamanho da Biblioteca

5. CONCLUSÃO

REFERÊNCIAS

ANEXO

REFERÊNCIAS

Web Foundation. **History of the Web**. Web Foundation, c2008-2022. Disponível em: https://webfoundation.org/about/vision/history-of-the-web/. Acesso em: 16 mar. 2023.

Mfuji09 etc al. **MIME Types (IANA media types)**. MDN Web Docs, c1998-2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Acesso em: 16 mar. 2023.

Gdad-s-River. A **Brief History of Web Graphics**. Fossbytes, 2017, Disponível em: https://fossbytes.com/history-web-graphics/. Acesso em: 16 mar. 2023.

PARNAS, D. L. **On the criteria to be used in decomposing systems into modules**. Communications of the ACM, ACM, v. 15, 1972.

SILVEIRA, André. **O que é Computação Gráfica**. Ambiente DESIGN, 2018. Disponível em: http://www.um.pro.br/index.php?c=/computacao/definicao. Acesso em: 25 abr. 2023.

COUTINHO, Thiago. **O que é Computação Gráfica? Descubra as melhores oportunidades na área!.** Voitto, 2021. Disponível em: https://www.voitto.com.br/blog/artigo/o-que-e-computacao-grafica. Acesso em: 25 abr. 2023.

SIQUEIRA, Fernando. **Conceitos de Computação Gráfica.** Sites Google, [s.d]. Disponível em: https://sites.google.com/site/profferdesiqueiracompgrafica/aulas/aula-1---conceitos-decomputacao-grafica. Acesso em: 25 abr. 2023.

GONÇALVES, Júnior. I**ntrodução a computação gráfica.** HyperBytes, c2020-2021. Disponível em: https://www.hiperbytes.com.br/introducao-a-computacao-grafica. Acesso em: 25 abr. 2023.

PINHO, Márcio. **Origens da Computação Gráfica**. Pucrs, Escola Politécnica, [s.d]. Disponível em: https://www.inf.pucrs.br/~pinho/CG/Aulas/Intro/introOLD.htm. Acesso em: 25 abr. 2023.

KHRONOS GROUP. **OpenGL Overview**. Khronos, c2023. Disponível em: https://www.khronos.org/opengl/. Acesso em 30 abr. 2023.

SILVA, Edson. **Opengl completa três décadas de constante evolução.** Diolinux, 2012. Disponível em: https://diolinux.com.br/noticias/opengl-completa-tres-decadas.html. Acesso em: 25 abr. 2023.

MANSSOUR, Isabel. **Introdução à OpenGL.** Pucrs, Escola Politécnica, 2003. Disponível em: https://www.inf.pucrs.br/~manssour/OpenGL/Introducao.html. Acesso em: 25 abr. 2023.

MEIRELLES, Adriano. **Como funciona o LCD.** Hardware, 2002. Disponível em: https://www.hardware.com.br/livros/hardware-manual/como-funciona-lcd.html. Acesso em: 26 abr. 2023.

VRIES, Joey. **Learn OPENGL – Graphics Programming**, 2020. Disponível em: https://learnopengl.com/book/book_pdf.pdf. Acesso em: 26 abr. 2023.

GREGORY, Jason. **Game Engine Architecture.** 2. ed. Boca Raton: Taylor & Francis Group, c2015. Disponível em: http://ce.eng.usc.ac.ir/files/1511334027376.pdf. Acesso em: 26 abr. 2023

MÖLLER et al. **Real Time Rendering.** 4. ed. Boca Raton: Taylor & Francis Group, 2018. Disponível em: http://cinslab.com/wp-content/uploads/2021/03/chenjiahao-Real-Time-Rendering-Fourth-Edition-2018-CRC-Press.pdf. Acesso em: 26 abr. 2023.

PETERBE et al. **WebGL: 2D and 3D graphics for the web,** 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Acesso em 26 abr. 2023.

Ian Hickson**. HTML Standart,** 2023. Disponível em: https://html.spec.whatwg.org. Acesso em 27 abr. 2023.

Arpitgoyalgg et al. **JAVASCRIPT**, 2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Acesso em 27 abr. 2023.

BOSNJAK, Dusan. **What is three.js?.** Medium, 2018. Disponível em: https://medium.com/@pailhead011/what-is-three-js-7a03d84d9489. Acesso em 18 mai. 2023

3D computer graphics with Three.js. Lilly021, 2022. Disponível em: https://lilly021.com/3d-computer-graphics-with-three-js. Acesso em 18 mai. 2023

Three.js, 2023. Disponível em: https://threejs.org. Acesso em 18 mai. 2023

Web Development. **Three.js,** c2019. Disponível em: https://2019-spring-web-dev.readthedocs.io/en/latest/final/roen/index.html. Acesso em 18 mai. 2023

IRWIN, Emma. M**icrosoft Open Source success story—Babylon.** Microsoft, 2021. Disponível em: https://cloudblogs.microsoft.com/opensource/2021/02/22/microsoft-open-source-success-story-babylon/. Acesso em 25 maio 2023.

WEBER, Raanan. **Game Development - Babylon.js: Building a Basic Game for the Web.** Microsoft. 2015. Disponível em: https://learn.microsoft.com/en-us/archive/msdn-magazine/2015/december/game-development-babylon-js-building-a-basic-game-for-the-web. Acesso em 25 maio 2023.

Babylon.js Documentation. Babylon.js, c2023. Disponível em: https://doc.babylonjs.com. Acesso em 25 maio 2023.

CREOOX. **Xeokit: online Viewer for 3D and IFC objects**. c2023. Disponível em: https://creoox.com/en/3d-viewer/. Acesso em: 25 maio 2023.

Xeokit: Web Programming Toolkit for AEC Graphics. Xeokit SDK, c2023. Disponível em: https://xeokit.io. Acesso em: 25 maio 2023.

Playcanvas: THE WEB-FIRST GAME ENGINE. Playcanvas, 2023. Disponível em: https://playcanvas.com. Acesso em: 25 maio 2023..

PlayCanvas Developer Resources. Playcanvas Manual, 2023. Disponível em: https://developer.playcanvas.com/en/. Acesso em: 25 maio 2023.