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

Mateus Freitas da Costa*
Paulo Ricardo Sampaio Martins**
Jefte de Lima Ferreira***

RESUMO

Com a evolução das tecnologias de hardware, os *websites têm se tornado* cada vez mais complexos e interativos, e hoje, além de textos e imagens, já é possível integrar gráficos de três dimensões nos navegadores. O seguinte trabalho teve o objetivo de realizar uma comparação entre diversas bibliotecas de linguagem *Javascript* feitas para auxiliarem o desenvolvimento de aplicações 3D no ambiente WEB. Os critérios que foram analisados incluem a qualidade da documentação, funcionalidades presentes e desempenho computacional. A fim de obter dados sobre o desempenho de cada biblioteca, diversos testes de *benchmark* foram realizados no navegador utilizando as funcionalidades presentes em cada uma delas. Ao obter os dados, foi possível concluir quais bibliotecas são mais eficientes para o desenvolvimento de gráficos em aplicativos WEB.

Palavras-Chave: Computação Gráfica; Javascript; WebGL.

ABSTRACT

With the evolution of hardware technologies, websites have become increasingly complex and interactive, and today, in addition to text and images, it is possible to integrate three-dimensional graphics into browsers. The aim of this project was to compare various Javascript language libraries designed to help develop 3D applications in the web environment. The Criteria to be analyzed include the quality of the documentation, present functionalities and computational performance. In order to obtain data on the performance of each library, various benchmark tests were carried out in the browser using the functionalities available in each of them. By obtaining the data, it was possible to conclude which libraries are most efficient for developing graphics in WEB applications.

Keywords: Computer Graphics; Javascript; WebGL.

* Rede de Ensino Doctum – Unidade Caratinga – mateusfcosta2002@gmail.com – Graduando em Ciência da Computação
** Rede de Ensino Doctum – Unidade Caratinga – aluno.paulo.martins@doctum.edu.br – Graduando em Ciência da Computação
*** Rede de Ensino Doctum – Unidade Caratinga – prof.jefte.ferreira@doctum.edu.br – Especialista em Ciência da Computação – Jefte de Lima Ferreira

SUMÁRIO

1. INTRODUÇÃO

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 na linguagem de formatação HTML – *HyperText Markup Language* - e no protocolo HTTP – *HyperText Transfer Protocol -,* 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 desenvolvido o padrão *WebGL – Web Graphics Library* -, versão da *API* gráfica *OpenGL,* funcionando nativamente no navegador, permitindo assim o desenvolvimento de gráficos 3D com aceleração de *hardware* em *websites*. (GDAD-S-RIVER, 2017)

No entanto, no surgimento de muitas tecnologias, uma dificuldade no seu uso se manifesta enquanto os desenvolvedores têm de adaptar-se a elas antes de poderem utilizá-la, e assim incorporarem-na no seu conjunto de habilidades. Dado que a computação gráfica é uma área notoriamente complexa, requerendo conhecimentos de matemática e álgebra 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

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.

Assim, um desenvolvedor WEB que deseja criar uma aplicação 3D tem a opção de usar uma biblioteca desenvolvida por terceiros que seja capaz de oferecer diversas funcionalidades e ferramentas mais sofisticadas em cima da especificação mais complexa *WebGL*, que seja de uso mais fácil e conveniente ao desenvolvedor, permitindo que o mesmo desenvolva seu programa de forma mais simples, rápida, e livre de erros.

Considerando todos estes fatos, o seguinte trabalho busca selecionar, a partir de uma amostragem mais ampla, bibliotecas *Javascript* que foram construídas em cima do *WebGL* a fim de facilitar o desenvolvimento de aplicações gráficas no navegador, apresentando sua origem, seu funcionamento em código fonte e funcionalidades. Depois, para cada biblioteca selecionada, fazer uma análise comparativa de seus aspectos qualitativos e quantitativos, como a qualidade da documentação,

funcionalidades presentes, desenvolvimento do projeto, tamanho em memória das bibliotecas, desempenho e uso de recursos computacionais.

A fim de obter os resultados práticos de desempenho computacional, serão desenvolvidos diversos testes na linguagem *Javascript* utilizando as funcionalidades de cada biblioteca que serão executados no navegador, e que, com o auxílio de ferramentas de *benchmark,* fornecerão dados de uso de processador, memória RAM, e tempo de renderização e carregamento.

Por fim, ao obter os dados e analisar as diferentes características de cada biblioteca, ressaltando seus pontos positivos e negativos, será possível tirar uma conclusão de suas diferentes especialidades e finalidades, que pode servir de ajuda aos desenvolvedores que desejam utilizar uma biblioteca gráfica no seus *websites.*

2. 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:* o *HTML,* a tag *canvas* e o *Javascript.* Por fim, o último subcapítulo sétimo trata de explicar a *API WebGL.*

2.1. 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])

2.2. 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 desempenho, 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, Nvidia, 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)

2.3. 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

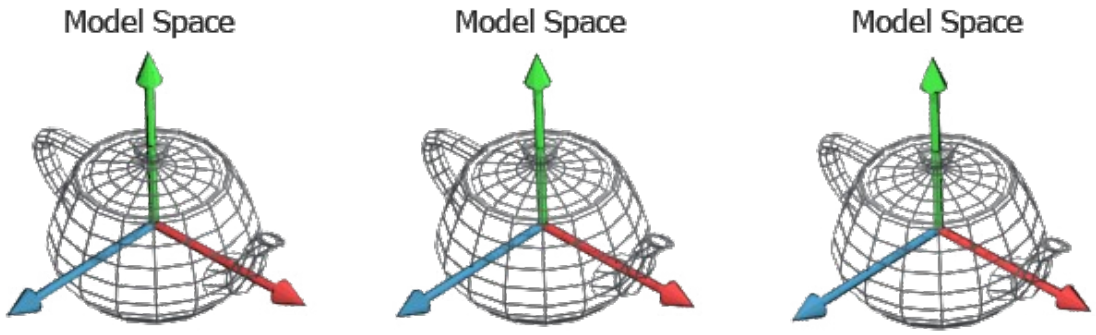
possível determinar a luminosidade de cada feixe de luz para cada píxel, o qual misturando as cores aditivamente, pode formar milhões de cores diferentes, permitindo a criação de imagens complexas e realistas. (MEIRELLES, 2002)

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



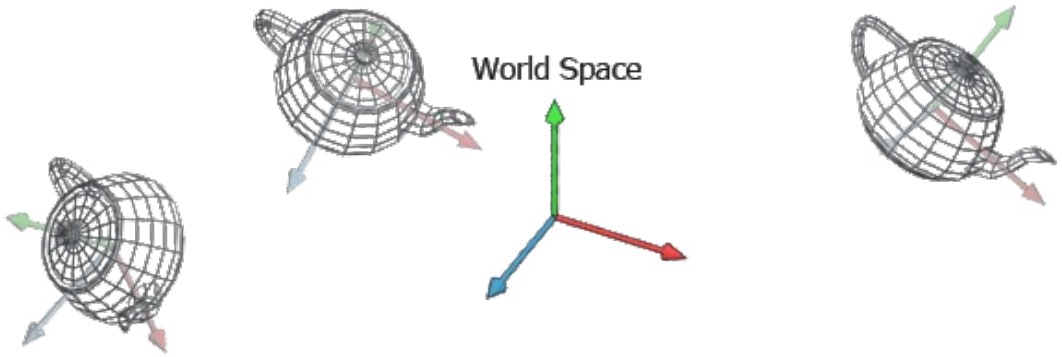
Fonte: Codinglabs, 2023[1].

Após o objeto ser transformado localmente, o *view space* determina a posição do

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

objeto em relação à cena e ao restante dos objetos.

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



Fonte: Codinglabs, 2023[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 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)

2.4. 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 apresentado aos usuários.

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

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*, 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 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)

2.5. 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, permitindo 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)

2.6. 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*, 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)

2.7. WebGL

WebGl é uma *API* para J*avascript* que tem o propósito de criar gráficos de duas ou três dimensões em alta desempenho 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 possuia 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 que diversos jogos de alta qualidade fossem desenvolvidos para a *Web,* bem como o surgimento de bibliotecas, como o *Three.js.* Posteriormente, 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.

3. METODOLOGIA

3.1. Escolha de Bibliotecas

Este projeto trata-se da comparação das diversas bibliotecas feitas em linguagem de programação *Javascript* que possuam recursos de renderização de gráficos 3D para a *Web*. Para que esta comparação seja feita, é necessário que seja construído uma amostragem das diferentes bibliotecas desenvolvidas para este propósito. Este subcapítulo trata de apresentar diversas bibliotecas disponíveis na *Web*, os critérios que serão utilizados para a filtragem e selação das mesmas, e por fim, fazer uma apresentação das que forem escolhidas a partir dos critérios definidos.

3.1.1. Bibliotecas Disponíveis

A tabela abaixo trata de listar e descrever as bibliotecas de linguagem *Javascript* encontradas na Web feitas para a renderização de gráficos 3D no navegador. As colunas indicam o nome da biblioteca, sua licensa, linguagem em que foi desenvolvida, e sua última atualização no site de repositórios *Github,* cujo valor “Ativo” indica uma biblioteca que está em constante desenvolvimento:

Tabela 1: Bibliotecas de Gráficos 3D disponíveis na WEB.

Nome	Licensa	Localização	Linguagem	Última Atualização	Estrelas Github
Three.js	MIT	Github / Site	Javascript	Ativo	94.718
Babylon.js	Apache License 2.0	Github / Site	Javascript	Ativo	21.442
RedGL	MIT	Github	Javascript	17/08/2022	152
Webgl-operate	MIT	Github / Site	Typescript	12/12/2022	163
Zogra Render	MIT	Github	Typescript	16/02/2023	21
ClayGL	BSD-2-Clause license	Github / Site	Javascript	31/10/2021	2.679
Xeogl	MIT	Github / Site	Javascript	14/05/2020	1.100
Litescene	MIT	Github / Site	Javascript	02/08/2020	340
Playcanvas	MIT	Github / Site	Javascript	Ativo	8.664
CopperLicht	CopperLicht License	Site	Javascript	-	-
Xeokit SDK	AGPL V3	Github / Site	Javascript	Ativo	601
OpenJSCAD	MIT	Github / Site	Javascript	Ativo	2.330

3.1.2. Critérios de Seleção

Previamente a comparação que será efetuada, é possível realizar uma filtragem das bibliotecas encontradas e assim diminuir o tamanho do grupo final que será analisado. A partir de diversos critérios de seleção, é possível reduzir o número de bibliotecas comparáveis a uma amostragem viável, descartando as bibliotecas que apresentem indícios de menor qualidade desde o ínicio.

Assim, as seções abaixo tratam de explicar os critérios iniciais que cada biblioteca precisa possuir a fim de passar no processo de seleção.

3.1.2.1. Código-fonte Aberto

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

3.1.2.2. Documentação Disponível

A documentação de uma biblioteca pública é necessária para que 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.

3.1.2.3. Disponibilidade de Tutoriais e Exemplos

Além de uma 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 algum tipo de tutorial em forma de texto disponível, ou comentários presentes da *API Docs*, com trechos de código-fonte que exemplifiquem sua usabilidade.

3.1.2.4. Formatos de Modelos Suportados

A fim de realizar os testes práticos de avaliação de desempenho, 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 GLTF (*GL Transmission Format)* é um tipo de arquivo binário, de código-aberto, altamente popular e com amplo suporte, eficiente, capaz de armazenar

ampla informação de modelos 3D[3], será utilizado para a realização dos testes de desempenho. Dessa forma, todas as bibliotecas precisam ter compatibilidade com modelos codificados nesse tipo de arquivo.

3.1.2.6. Funcionalidades Presentes

Na renderização de gráficos 3D, existem diversos domínios com propósitos diferentes. Uma vez que o seguinte trabalho trata de selecionar bibliotecas feitas para a renderização de gráficos realísticos em tempo real, é necessário utilizar as bibliotecas que foram desenvolvidas para este propósito. As funcionalidades necessárias para cumpri-lo são: sistema de renderização baseado em física, sistema de luzes com suporte à sombras, e sistema de animação esqueletal.

3.1.2.7.1 Popularidade

A popularidade de uma biblioteca costuma significar maior quantidade de tutoriais disponíveis online, menor possibilidade de bugs e problemas, e é um bom indicador geral de qualidade. Assim, das bibliotecas que cumprem todos os requisitos anteriores, apenas as quatro com maior quantidade de estrelas dadas no seu repositório Github serão utilizadas para as comparações feitas neste trabalho.

3.1.3. Bibliotecas Selecionadas

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

3.1.3.1. 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.

3 SmartPixels, [s.d]. **GLTF vs FBX: their 5 key features.** Disponível em: https://www.smartpixels.fr/gltf-vs-fbx-5-key-features-to-the-formats/. Acesso em 30 set. 2023.

(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).

Além da biblioteca gráfica, o Three.js também contêm ferramentas extras para facilitarem o desenvolvimento, como o editor, disponível online, que permite carregar modelos, visualizá-los e construir uma cena posicionando objetos e exportá-la em um formato pronto para ser carregado. Também possui um *playground* que permite a construção de uma cena visualmente por meio de nós que se conectam representando geometrias, materiais, *scripts* e efeitos de cores.

O código-fonte abaixo demonstra a criação de uma cena em Three.js, no qual é inicializado uma câmera e uma geometria de um cubo com um material de cor azul.

Figura 3: Three.Js: Inicialização de Uma Cena

```
const colorBlue = 0x0000ff
const scene = new Three.Scene();
const camera = new Three.PerspectiveCamera(fov, width / height, minZ, maxZ);
const renderer = new Three.WebGLRenderer();
const geometry = new Three.BoxGeometry(1, 1, 1);
const material = new Three.MeshBasicMaterial({ color: colorBlue });
const cube = new Three.Mesh(geometry, material);

renderer.setSize(width, height);
document.body.appendChild(renderer.domElement);
scene.add(cube);
```

Fonte: Three.js, [s.d][4].

3.1.3.2. 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 para desenvolver a mais simples e poderosa ferramenta de renderização *Web*, em 2015 o *Babylon.js* já demonstrava suas grandes capacidades, conquistando a atenção de seus concorrentes. (IRWIN, 2021)

Por ser focada em desenvolvimento de jogos, *Babylon.js* 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.js,* além da engine, possui diversas ferramentas auxiliares para ajudarem no desenvolvimento, como o *Playground*, que permite que se escreva código e o visualize em tempo real ao lado, um avançado editor *offline*, um criador de interface gráfica, um editor de nós, uma biblioteca de *assets* contendo diversas geometrias e materiais, e por fim, ainda possuí o *inspector,* um painel de *debug* que pode ser utilizado para visualizar todos os objetos de uma cena e suas propriedades.

O codigo-fonte abaixo exemplifica a criação de uma cena, uma câmera, uma luz direcional, uma esfera e um chão utilizando a biblioteca.

4 Disponível em: https://threejs.org/docs/#manual/en/introduction/Creating-a-scene. Acesso em 18 set. 2023.

Figura 4: Babylon.Js: Inicialização de Uma Cena

```
const scene = new BABYLON.Scene(engine);
const camera = new BABYLON.FreeCamera("camera1", new BABYLON.Vector3(0, 5,-10), scene);
camera.setTarget(BABYLON.Vector3.Zero());
camera.attachControl(canvas, true)
const light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0, 1, 0), scene);
light.intensity = 0.7;
const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 2, segments: 32},
scene);
const ground = BABYLON.MeshBuilder.CreateGround("ground", {width: 6, height: 6}, scene);
```

Fonte: Babylon.js, [s.d][5].

3.1.3.3. ClayGL

ClayGL é uma biblioteca baseada em *WebGL* para a criação de aplicações Web3D escaláveis. É fácil de utilizar, e configuravel para a criação de gráficos de alta qualidade. É beneficiada pela modularidade e por ter a capacidade de ter seu código-fonte não utilizado removido, fazendo com que seja uma biblioteca extremamente leve de ser incluída no seu projeto. (Contribuídores ClayGL, 2018)

Tendo seu início em 2013, é um projeto menor e possui poucos contribuidores, sendo atualizada pela última vez no ano de 2021. Apesar de possuir suporte avançado a gráficos 3D, não conta com ferramentas extras como um editor em interface gráfica.

Possui apenas uma documentação em API com breves comentários e uma página de exemplos, nos quais algumas das funcionalidades mais avançadas da biblioteca é exibida. Apesar da brevidade, a API possui o código limpo e é fácil de ser utilizada.

A biblioteca pode ser utilizada conforme a figura abaixo, no qual uma cena com uma câmera, um cubo e uma luz direcional são criadas. O código-fonte também contêm a função *loop*, que rotaciona o cubo no eixo Y a cada quadro.

5 Disponível em: https://doc.babylonjs.com/features/starterSceneCode. Acesso em 18 set. 2023.

Figura 5: ClayGL: Inicialização de Uma Cena

```
clay.application.create('#main') {
      width: window.innerWidth,
      height: window.innerHeight,
      init(app) {
        this._camera = app.createCamera([0, 2, 5], [0, 0, 0]);
        this._cube = app.createCube( {color: '#f00' } );
        this._mainLight = app.createDirectionalLight([-1, -1, -1]);
      },
      loop(app) {
        this._cube.rotation.rotateY(app.frameTime / 1000);
      }
    });
}
```

Fonte: ClayGL Github, 2018[6].

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 líderes de mercado em relação a game engine em *WebGL*, sendo capaz de criar aplicações em AR e VR. É utilizada por desenvolvedores independentes e também por empresas conhecidas no mercado, como a 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 desempenho. (Playcanvas, 2023)

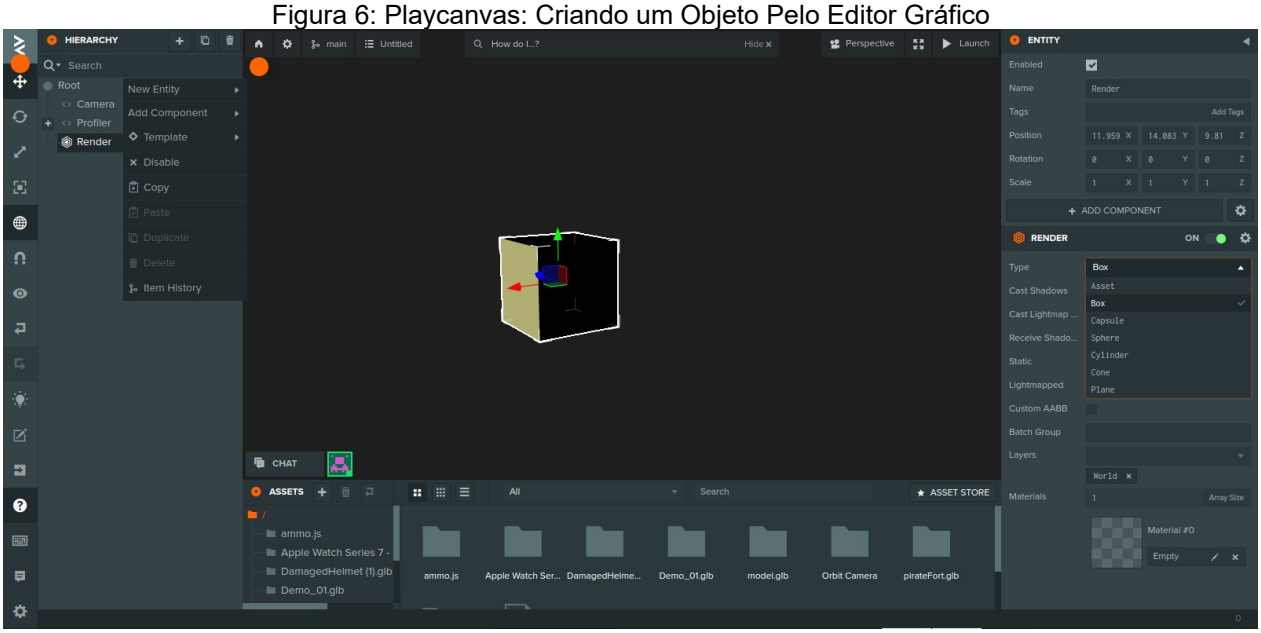
O Playcanvas possui uma facilidade de utilização significativa para

6 Disponível em: https://github.com/pissang/claygl. Acesso em 18 set. 2023.

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).

O playcanvas, ao contrário das outras ferramentas, tem seu modo principal utilizando o seu editor gráfico, no qual é possível criar as cenas e adicionar os objetos, utilizando os arquivos de *script* somente para dar lógica a aplicação. A figura abaixo demonstra a criação de um cubo utilizando o editor da biblioteca disponível online.

Figura 6: Playcanvas: Criando um Objeto Pelo Editor Gráfico



Fonte: Os autores, 2023.

3.2. Métricas de Software

O capítulo apresentado a seguir tem como função apresentar as diversas métricas de *software* que deverão ser utilizadas para a análise e comparação de cada biblioteca a ser avaliada no projeto. Semelhante escolha de métricas foi feita no trabalho de conclusão de curso do aluno Oggo Petersen, da Universidade Federal do Rio Grande do Sul, que analisava bibliotecas geradoras de gráficos de dados[7]

A seção 7.1 trata das métricas qualitativas utilizadas para a a comparação,

7 PETERSEN, Oggo. **Análise de bibliotecas para geração de gráficos na WEB**, 2013. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/86642/000910051.pdf?sequence=1&isAllowed=y. Acesso em 06 jun. 2023.

seguida das métricas quantitativas, na seção 7.2.

3.2.1 Métricas Qualitativas

3.2.1.1. 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, demonstrações em código-fonte das funcionalidades, presença de projetos completos desenvolvidos, e por fim as redes sociais ou fórums que a biblioteca possa possuir.

3.2.2 Métricas Quantitativas

3.2.2.1. Desenvolvimento do Projeto

Um fator de qualidade de uma biblioteca é o engajamento de comunidade responsável por seu desenvolvimento. O quanto uma biblioteca possui seu código-fonte alterado para a adição de funcionalidades, correção de bugs, e atualização para os novos padrões da *Web* é um fator a ser considerado antes da inclusão da mesma em um projeto. Assim, pode-se avaliar, a partir de dados disponíveis no *Github:* data da última atualização, quantidade total e recente de *commits (*taxa de desenvolvimento pelos programadores*)*, quantidade total e recente de *pull requests (*atualizações sendo adicionadas na versão de produção*)*, quantidade total e recente de *issues (*mudanças requisitadas pela comunidade*),* quantidade de projetos que utilizam a biblioteca e quantidade de contribuidores do projeto.

3.2.2.2. 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 outras funções além de uma simples renderização 3D, e pode possuir mais funcionalidades capazes de gerar gráficos mais realistas, ou também oferecer funções extras e customizações ao programador, permitindo-lhe que tenha melhor controle sobre sua aplicação e que desenvolva diversos tipos de aplicativos mais facilmente. Algumas das funcionalidades possíveis de serem avaliadas são:

- Configuração e propriedades dos materiais baseados em física.
- Formatos de arquivo suportados para importação.
- Utilidades extras como: Sistema de física, física e colisão, partículas e fluídos e exportação de cenas.
- Ferramentas extras além da biblioteca, como editores gráficos e suas capabilidades.
- Versão do *WebGL* suportada.
- Biblioteca de recursos pré-disponíveis, como materiais, geometrias e texturas prontas para uso.
- Sistema de iluminação como tipos de luzes suportadas, sombras, e capacidade de gerar *lightmaps*.
- Otimizações disponíveis.
- Efeitos de pós-processamento suportados, como *fog, bloom, anti-aliasing* etc.

Dessa forma, bibliotecas que oferecem mais funcionalidades úteis ao desenvolvedor obterão vantagem nessa métrica. No entanto, há de ser considerado também as diferentes aplicações e usos da computação gráfica, os quais algumas podem se beneficiar de todas as funcionalidades presentes, enquanto outras, em razão de sua simplicidade, tratariam complexidades adicionais como um incoveniente por não necessitarem desses recursos.

3.2.2.3. Desempenho

Aplicações como jogos, *CADs*, simulações, programas de visualizações e outros demandam do computador uma quantidade generosa de recursos para a renderização das complexas geometrias responsáveis por dar vida a suas funcionalidades. 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 desempenho 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 desempenho da biblioteca avaliada, como o uso de *CPU,* memória RAM alocada, tempo de inicialização, tempo de carregamento de modelos, e tempo de renderização.

Para que isso seja feito, serão desenvolvidos testes envolvendo cada uma das bibliotecas, nos quais modelos modelos 3D em formato *GLTF* serão carregados para a renderização de uma cena. Serão feitos diversos testes a fim de obter um resultado mais amplo, no qual se avaliarão os diferentes aspectos da renderização 3D ao renderizar cenas contendo:

- Poucos objetos.
- Muitos objetos.
- Objeto com alta contagem de polígonos.
- Alta contagem de fontes de luz.
- Animação.
- Pós-processamento de efeitos e cores.

3.2.2.4. Tamanho da Biblioteca

Um código-fonte que ocupa muita memória em disco aumenta a carga dos servidores *WEB* e aumenta os tempos de requisição e carregamento da página, o que pode afetar a experiência final dos usuários. Assim, está métrica busca catalogar os seguintes dados de tamanho da biblioteca:

- Memória ocupada total dos arquivos do repositório público da biblioteca baixado por um gerenciador de pacotes, como o NPM*,* tratado na seção 3.3.2.1. Aqui também serão também avaliados repositórios extras, que estão disponíveis em algumas bibliotecas como extensão.
- Memória ocupada do código-fonte da biblioteca após juntado num único arquivo por um programa *bundler*, como o *Esbuild,* explicado na seção 3.3.2.2.
- Memória ocupada do mesmo arquivo unificado após passar por algoritmos de

minificação e compressão, que reduzem consideravelmente o seu espaço em disco, sem que se tornem inutilizáveis nos servidores *WEB*.

3.3. Ferramentas utilizadas

Este subcapítulo trata das ferramentas utilizadas para a realização dos testes e comparações a serem feitas para cada biblioteca selecionada. A seção 3.3.1 trata das métricas que podem ser avaliadas a partir de uma pesquisa nos sites oficiais dos desenvolvedores da biblioteca. Na seção 3.2.2 explica-se o modo de medir o tamanho ocupado por uma biblioteca utilizando o gerenciador de pacotes *NPM*, o *bundler Esbuild* e os algoritmos de compressão. A seção 3.3.3 trata de explicar como os navegadores e as funções nativas da linguagem *Javascript* podem ser utilizados para a coleta e análise de dados de desempenho. Por fim, o ambiente de desenvolvimento tem suas informações detalhadas na seção 3.3.4.

3.3.1. Documentação e Repositórios

Dos critérios de comparação, alguns podem ser avaliados facilmente através de uma pesquisa em seus sítios oficiais. O desenvolvimento do projeto possui estatísticas diretamente disponíveis seu repositório de código *Github*, enquanto as funcionalidades presentes, os formatos de arquivo suportados e qualidade da documentação podem ser avaliados a partir das informações disponíveis site oficial da biblioteca e/ou no seu repositório.

3.3.2. Ferramentas de Medição de Tamanho

Esta seção trata da explicar as ferramentas citadas que serão utilizadas para a medição do tamanho de uma biblioteca. A primeira subseção trata de explicar o popular gerenciador de pacotes *NPM*, o segundo explica o funcionamento do *bundler Esbuild*. Por fim, o terceiro capítulo explica a compressão de arquivos utilizando o algoritmo *Deflate* em formato Gzip.

3.3.2.1. NPM

3.3.2.2. Esbuild

3.3.2.3. Gzip/Deflate

3.3.3. Mensuradores de Desempenho

Nos testes de desempenho, nos quais serão analisados estatísticas de uso de CPU*,* memória RAM, tempo de inicialização e tempo de renderização, três ferramentas serão utilizadas, os navegadores, o painel *Devtools* e as funções nativas da linguagem *Javascript*.

3.3.3.1. Navegadores

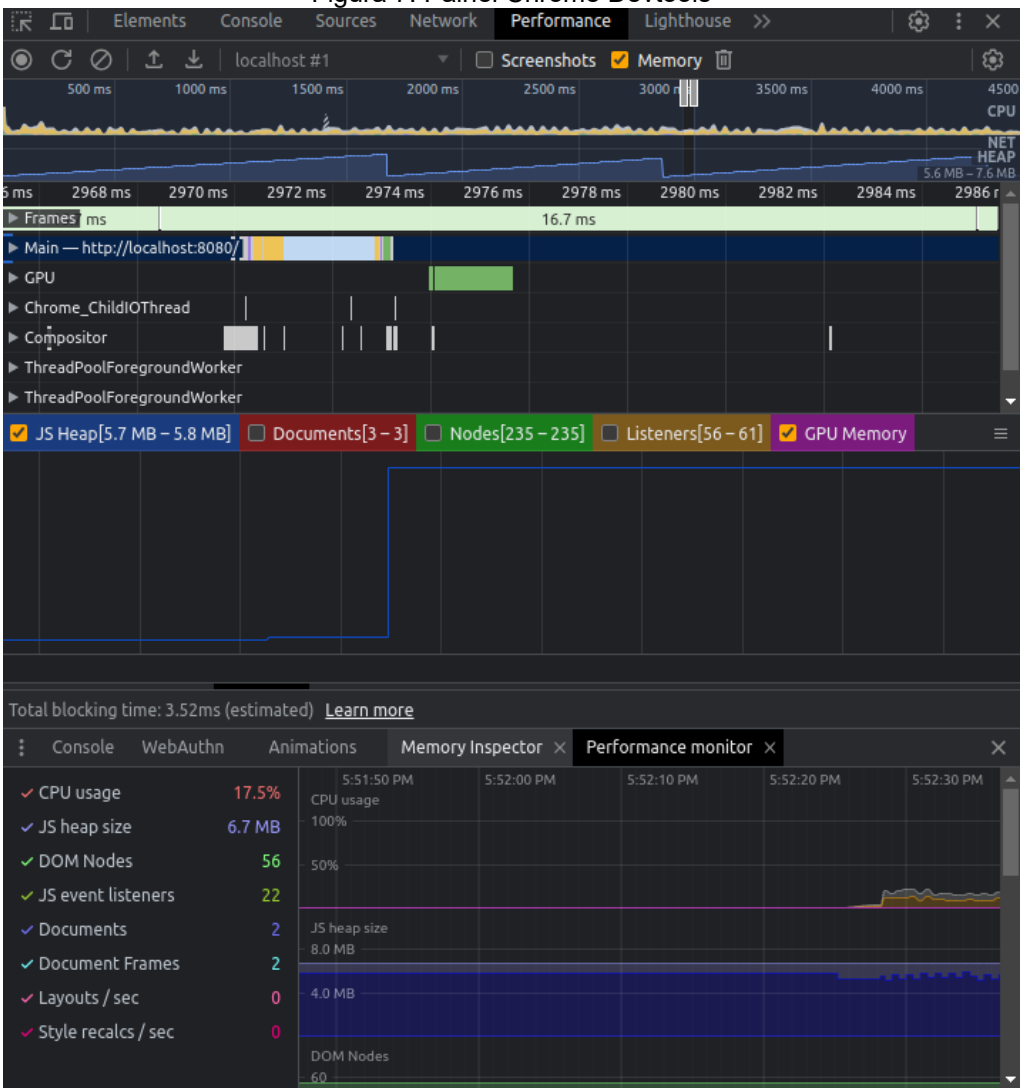
As bibliotecas deste projeto foram desenvolvidas para serem utilizadas no contexto de uma página *Web*, e assim, requerem um navegador para serem executados no computador. Para este trabalho, o navegador *Google Chrome* foi selecionado, que além de ser o mais utilizado pelos usuários em todo o mundo[8], possuí, integrado ao seu painel de desenvolvedor *Devtools*, funções importantes na análise de mensuração de desempenho, que será tratado na seção seguinte.

3.3.3.2. Devtools

O painel *Devtools* é uma ferramenta embutida em diversos navegadores que permite aos desenvolvedores analisar, alterar e diagnosticar problemas em páginas *Web*. Neste trabalho, o painel *Devtools* do Google Chrome será utilizado para obter dados de memória RAM e uso de CPU enquanto a aplicação gráfica estiver rodando.

8 **Top Browser Market Share,** 2023. Disponível em: https://www.similarweb.com/browsers/. Acesso em 06 jun. 2023.

Figura 7: Painel Chrome Devtools



Fonte: Os Autores, 2023.

3.3.3.3. Funções Javascript

Para o tempo de carregamento das cenas 3D, é possível utilizar as funções nativas disponíveis na linguagem *Javascript*. Basta que se tire quantos milisegundos passaram-se quando a biblioteca inicializou utilizando a função *performance.now()* e calcular a diferença do tempo que a página começou a carregar. Na análise do tempo de renderização, faz-se o mesmo, obtendo o tempo de início e final de cada quadro. Ao fazer a divisão de 1000 por este número, é possível obter o FPS (quadros por segundo), e a partir deste dado, também é possível obter dados derivados úteis na

análise de desempenho que são comumente utilizados em jogos, tais como: FPS médio, FPS mínimo e máximo (média dos dados coletados no tempo de um segundo), média truncada de FPS (os valores em 80% do tempo, descartando os extremos) e FPS 1% low (piores valores em 1% do tempo, importante para conhecer os picos de atraso)[9].

3.3.4. Ambiente de Desenvolvimento

Neste trabalho, a fim de obter uma variedade maior de resultados e enriquecer os dados da pesquisa, os testes de desempenho terão seus dados coletados em diferentes ambientes, mediante execução e coleta em dois computadores de configurações 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.

4. RESULTADOS

4.1. Análise das Métricas Qualitativas

4.1.1. Qualidade da Documentação

4.1.1.1. Three.js

A Biblioteca Three.js além de sua API Docs comentada possui um amplo manual de usuário, demonstrando suas funcionalidades principais com exemplos e explicações detalhadas. Também Possui uma página no seu site com amplos projetos desenvolvidos pela comunidade. Ainda, por ser uma biblioteca popular, conta com diversos tutoriais disponíveis em outras plataformas desenvolvidos por terceiros.

9 BUTLER, Sidney. **Why 1% Lows Matter in Video Games (And What are They?)**, 2023. Disponível em: https://www.howtogeek.com/892766/why-1-lows-matter-in-video-games-and-what-are-they/. Acesso em 12 set. 2023.

Suporte: Fórum e Discord.

Outras Redes: Twitter.

4.1.1.2. Babylon.js

A Biblioteca Babylon.Js possui sua API Docs plenamente comentada. Possui também um guia de usuário amplo demonstrando praticamente todas as funcionalidades e configurações de Biblioteca, além de muitos projetos completos desenvolvidos pelos desenvolvedores e comunidades.

Suporte: Fórum e Discord.

Outras Redes: Blog no Medium, Youtube e Twitter.

4.1.1.3. Playcanvas

A biblioteca Playcanvas possui sua API Docs plenamente comentada, e seu guia de usuário é bastante abrangente, demonstrando quase todas as funcionalidades do seu editor e alguns exemplos com código-fonte. No entanto sua documentação para os usuários da *engine* que não utilizam o editor é limitada e possui poucos exemplos. Também possui amplos projetos desenvolvidos pela comunidade.

Suporte: Forum e Discord.

Outras Redes: Youtube.

4.1.1.4. ClayGL

A biblioteca ClayGL possui sua API Docs plenamente comentada, no entanto não possui guia de usuário, mas apenas alguns trechos de código-fonte demonstrando algumas de suas funcionalidades. Há apenas quatro projetos completos desenvolvidos disponíveis de exemplo no seu repositório Github, e a biblioteca não oferece mais suporte nem é mantida pelos desenvolvedores.

4.2. Análise das Métricas Quantitativas

4.2.1. Desenvolvimento do Projeto

Tabela 2: Desenvolvimento do projeto - Dados do Github

Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**
***Pull Requests* (Ativos)**	145	8	11	21
***Pull Requests* (Fechados)**	14627	11371	11386	3692
***Issues* (Abertos)**	386	93	37	525
***Issues* (FechadosI**	11550	2836	48	1422
Contribuidores	1736	485	11	130
Commits	42170	41403	1191	10902
Estrelas	94810	21464	2671	8671
Usado Por	228.190	4648	0	1

Dados obtidos em 5 de Outubro de 2023.

Tabela 3: Desenvolvimento do projeto das bibliotecas num período de um mês – Dados do Github.

Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**
Pull Requests (Criados)	10	5		4
Pull Requests (Merged)	120	118		66
***Issues* (Criados)**	15	6		22
***Issues* (Concluídos)**	46	19		20
Autores	40	20		14
Commits	134	221		68

Dados referentes ao período 5 de Setembro de 2023 a 5 de Outubro de 2023.

4.2.2. Funcionalidades Presentes

Tabela 4: Funcionalidades - Configurações disponíveis dos materiais PBR.

Biblioteca	Three.js	Babylon.js	ClayGL	Playcanvas
Albedo	Não Suporta	Total	Não Suporta	
Diffuse	Total	Total	Total	Total
Alpha	-	-	-	
Metalic	Total	Total	Total	Total
Rough/Gloss	Total	Total	Total	Total
Refraction	Parcial	Total	Não Suporta	Total
Reflectivity	Parcial	Total	Não Suporta	Parcial
Environment/Reflection	Total	Total	Não Suporta	Total
Bump	Total	Total	Não Suporta	Parcial
Emissive	Total	Total	Total	Total
Opacity	-	-	-	-
LightMap	Total	Total	Não Suporta	Total
Detail	Não Suporta	Total	Não Suporta	Total
Iridescence	Total	Total	Não Suporta	Total
Anisotropic	Não Suporta	Total	Não Suporta	Parcial
Sheen	Suporta	Total	Não Suporta	Total
Specular	Suporta	Parcial	Não Suporta	Total
Transmission	Total	Não Suporta	Não Suporta	Não Suporta
Normal	Total	Total	Total	Total
ClearCoat	Total	Total	Não Suporta	Total
Paralax	Não Suporta	Total	Não Suporta	Não Suporta
Decal	Não Suporta	Total	Não Suporta	Não Suporta
Ambient Occlusion	Total	Total	Não Suporta	Total
Displacement	Total	Não Suporta	Não Suporta	Não Suporta
ThickNess	Parcial	Total	Não Suporta	Total

Suporte Total - Texturas configuráveis.
Suporte Parcial – Apenas Intensidade ou cor configurável.

4.2.3. Testes de Desempenho

Tabela 5: Dados de Renderização da Cena "Pirate Ford" - Cena Simples com poucos objetos

Configuração	**Nome**	**Triângulos**	**Texturas**	**Luzes**	**Sombras**	**Objetos**
Descrição	Pirate Ford	90.334	4 1024x 1 512x	2 Omni 1 Direcional	Desativado	7

Distância da Câmera			Perto	
Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**
Média FPS	169.00	102.35	116.32	75.79
Média Truncada FPS	163.88	105.23	116.56	72.76
Média FPS Máximo	275.14	272.31	290.51	251.39
Média FPS Mínimo	110.29	26.33	34.95	13.46
Média FPS 1% Low	5.44	24.80	8.39	8.10

Distância da Câmera			Longe	
Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**
Média FPS	215.52	183.12	248.07	147.44
Média Truncada FPS	217.22	175.19	250.95	148.24
Média FPS Máximo	362.99	327.27	375.31	215.67
Média FPS Mínimo	112.14	81.54	125.06	107.53
Média FPS 1% Low	5.34	6.27	5.22	7.15

Tabela 6: Dados de Renderização da Cena "Floresta Low Poly" - Cena complexa com muitos objetos

Configuração	**Nome**	**Triângulos**	**Texturas**	**Luzes**	**Sombras**	**Objetos**
Descrição	Floresta Low Poly	2076694	1 256x	1 Direcional	Desativado	1228
Distância da Câmera				Perto		
Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**		
Média FPS	81.01	39.20	44.19	39.95		
Média Truncada FPS	80.62	38.10	43.81	40.81		
Média FPS Máximo	137.48	75.80	97.44	88.25		
Média FPS Mínimo	24.07	8.93	8.27	7.16		
Média FPS 1% Low	11.98	35.67	31.40	21.72		
Distância da Câmera				Longe		
Biblioteca	**Three.js**	**Babylon.js**	**ClayGL**	**Playcanvas**		
Média FPS	86.38	58.51	70.51	65.12		
Média Truncada FPS	85.48	60.84	70.11	64.31		
Média FPS Máximo	132.86	118.56	150.67	143.31		
Média FPS Mínimo	61.79	12.71	14.45	14.90		
Média FPS 1% Low	10.72	28.56	16.22	15.48		

4.2.4. Tamanho da Biblioteca

Utilizando das ferramentas tratadas na seção 3.3.2, foi possível obter os seguintes resultados em relação ao tamanho das bibliotecas. A biblioteca *Playcanvas,* por estar disponível em duas formas, editor e código, foi dividida em 2 categorias, com os campos em escuro representando não disponibilidade da informação:

Tabela 7: Tamanho das bibliotecas, espaço ocupado por pacote NPM e código-fonte

Biblioteca	Three.js	Babylon.js	ClayGL	PlayCanvas (editor)	PlayCanvas (engine)
Pacote NPM (Principal)	25M	70M	12M		52M
Pacote NPM (Principal + Extras/Addons)	27M	194M	13M		
Código-Fonte (Bundled)	1.2M	6.7M	644K		2.2M
Código-Fonte (Minificado)	644K	4.8M	304K	1.5M	1.3M
Código-Fonte (Minificado e GZIP)	176K	1.1M	88K	780K	328K

Unidades de Medida - M - MegaBytes, K - KiloBytes.

REFERÊNCIAS

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

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

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

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.

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

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

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

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

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.

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.

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

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.

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.

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

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

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.

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.

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

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

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.

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.

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