

ANÁLISE COMPARATIVA DE BIBLIOTECAS JAVASCRIPT PARA A RENDERIZAÇÃO DE GRÁFICOS 3D EM TEMPO REAL EM APLICAÇÕES 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; WebSites.

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; Websites.

* 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

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

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])

2.3 - Gráficos 3D

Uma tela de computador é composta por milhares de pixels, 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)

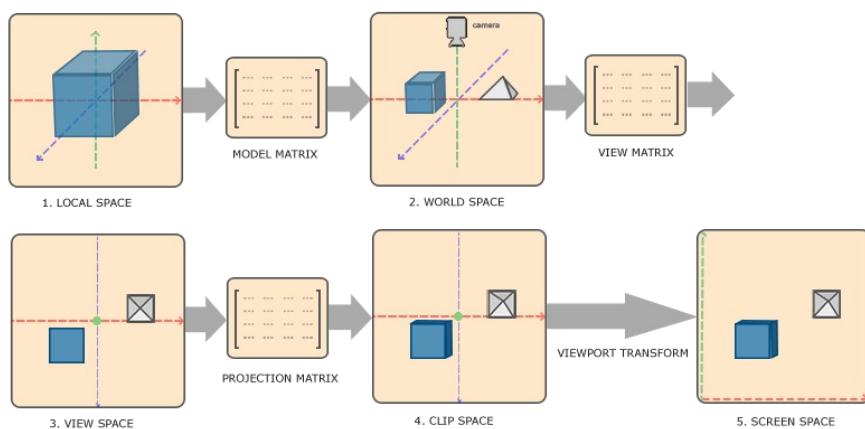
Para determinar o local nos quais os pixels serão desenhados, um sistema de coordenadas é necessário. No caso de gráficos 3D, o *OpenGL* um sistema conhecido como destro, no qual as coordenadas começam no quanto inferior esquerdo, apontando para cima e para a direita, e a terceira dimensão, o Z, aponta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

Para realizar as transformações de 3D para 2D, uma câmera é utilizada, que consiste num objeto invisível que permite a renderização e visualização da cena. As

câmeras podem ser divididas em dois principais tipos, perspectiva e ortográfica. A câmera perspectiva reflete a visão do mundo real, no qual os objetos mais distantes tendem ao centro do campo de visão, já a câmera ortográfica permite visualizar os objetos sem distorção e no mesmo tamanho independente da distância, permitindo um melhor alinhamento de objetos e visualização para comparação de tamanhos, sendo mais utilizada na modelagem. (BLENDER, c2023)

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. Depois, tem sua posição determinada na cena, seguido da sua posição na câmera e na tela do computador, no qual são descartadas (*Clip Space*) tudo aquilo que estiver além dos limites do ecrã do dispositivo. (VRIES, 2020).



Fonte: LearnOpenGL, 2020.¹

Tendo o *vertex shader* lido 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 pixéis 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. O processo de passar um objeto através dessas diversas etapas de processamento é conhecido como *rendering pipeline*. (VRIES, 2020)

2.4 - Geometrias e Materiais

Geometria é a área da matemática que desempenha o papel de representar formas em um plano espacial. Enquanto a geometria euclidiana tradicional concentra-se na representação de objetos bidimensionais, a geometria 3D ou Espacial refere-se a análise de figuras e objetos que ocupam o espaço tridimensional, ou seja, aqueles que têm comprimento, largura e altura, utilizando coordenadas tridimensionais (x, y, z) para descrever pontos no espaço. (SILVA)

O entendimento da geometria é essencial para a criação de malhas poligonais, que são uma das formas de modelagem gráfica. As malhas poligonais podem ser descritas como uma coleção de pontos, arestas e faces conectados em polígonos como triângulos e quadriláteros, que servem para simplificar o processo de renderização. Estes polígonos constituem a superfície da malha poligonal que é utilizada para modelar e representar objetos tridimensionais (TIIGIMÄGI, 2021). Qualquer tipo de objeto, como pontos, linhas, cubos, esferas, edifícios, terrenos e animais pode ser construído e renderizado computacionalmente através das malhas poligonais.

¹ Disponível em: https://learnopengl.com/book/book_pdf.pdf. Acesso em: 20 out. 2023.

Além da malha poligonal, um objeto 3D também guarda informações sobre o seu material, constituído de atributos que são aplicados ao modelo para modificar como ele vai ser visto no seu estado final, deixando-o com uma aparência mais característica e dinâmica. Estes materiais são classificados em diferentes tipos, tais como cor, brilho, reflexo, transparência e refração. Cada um destes tipos de materiais adicionam ou modificam atributos básicos dos modelos como e por exemplo a transparência que permite que seja possível enxergar através dele, sendo útil para simular objetos como vidro, e também o reflexo que adiciona a capacidade de refletir a luz, útil para simular espelhos e objetos de metal. (SANTOS, 2022)

Durante a modelagem, os objetos 3d normalmente são criados com uma cor cinza padrão, e para que o modelo possua uma aparência mais detalhada é necessário a aplicação de uma ou mais textura ao modelo. Textura são normalmente imagens 2d aplicadas ao redor de modelos 3D para criar uma aparência mais realista e detalhada. Além de definir a cor primária de um objeto, as texturas podem definir suas outras propriedades de seu material, como seu brilho ou transparência. (SHAHBAZI, 2023)

2.5 - Iluminação e Sombra

A iluminação é a parte da modelagem 3D que lida com a configuração de simulação de fontes de luz em cenas tridimensionais permitindo a visão mais clara de seus componentes. Essa é uma parte importante, pois tem grande impacto na renderização final da cena, e uma iluminação mal feita pode fazer com que os modelos não apresentem muito bem todos os seus detalhes. (NAGHDI, 2020)

Existem diversas formas de configurar formas de luz em uma cena de acordo com o objetivo que deseja alcançar com ela. Algumas destas formas de iluminação são a direcional, que gera uma fonte de luz que segue uma direção fixa e ilumina igualmente todos os objetos em seu caminho; a de ponto, que diferente da direcional gera uma fonte de luz em um ou mais pontos específicos que se expande em todas as direções e perdendo intensidade à medida que se distancia; a holofote, que gera uma fonte de luz em um ponto que se expande de forma cônica em uma direção com

intensidade focado no centro; e a ambiente, que gera luz em uma área determinada que expande em todas as direções muito utilizada como luz padrão em cenas. (Tina Lee, 2023)

O funcionamento de luz no mundo real é extremamente complicado e depende de muitos fatores. Portanto, para ser capaz de simular um ambiente iluminado, um computador deve utilizar um modelo simplificado que possa ser renderizado em tempo real, ao mesmo tempo que oferece resultados semelhantes aos encontrados no mundo real. Um destes modelos é o *Phong Lightning Model*, que baseia-se em certas propriedades dos materiais que determinam como a luz será refletida no ambiente: *Ambient*, cor base do objeto na ausência de luz; *Diffuse*, cor do objeto ao refletir uma luz e *Specular*, pequeno ponto brilhante de uma luz que aparece no objeto. (VRIES, 2020).

Outra forma de simular a luz é pelo método PBR (*Physically Based Rendering*), ou renderização baseada em física, uma técnica de simular iluminação similar ao da vida real. O PBR foca em calcular a interação da luz nas superfícies e textura dos modelos de forma fisicamente realista, por exemplo como a luz é refletida, absorvida e refratada pelo material. É uma técnica amplamente utilizada por ser mais prática e eficiente do que seus antecessores, além de ser compatível com diferentes sistemas de iluminação e prover um resultado final mais realista. (RUSSELL, 2023)

Pelo PBR, os materiais se tornam muito mais configuráveis, fornecendo atributos – em forma de texturas ou escala - como *metalness*, reflexão do ambiente; *roughness*, aspereza; *refraction*, refração; *environment*, reflexão do plano de fundo; *normal*, elevação e profundidade que afetar a geometria; *displacement*, elevação maior, afetando a geometria; *opacity*, transparência; *sheen*, para tecidos ou roupas; *transmission*, transparência com reflexão; *clearcoat*, reflexão limpa em material aspero; *parallax*, acentuação de relevo; *decal*, detalhes independentes adicionais; *ambient occlusion*, sombras suaves nos detalhes da superfície; *thickness*, espessura utilizada na refração; *iridescence*, mudança na faixa de cores pelo ângulo do observador; *attenuation*, mudança de cor pela distância; *anisotropic*, afeta a reflexão a depender do

ângulo do observador. (A23D, c2023; BABYLON.JS, c2023; THREE.JS, 2023; PLURALSIGHT, 2022).

Figura 1: Renderização Phong (esquerda) versus renderização baseada em física (direita)



Fonte: Playcanvas Blog, 2014²

Além da iluminação, uma cena precisa de sombras para dar maior realismo. As sombras são resultados da ausência de luz devido à oclusão de algum objeto, e auxiliam o observador a visualizar a relação espacial entre os objetos, dando um melhor senso de profundidade a uma cena. Geralmente o processo de cálculo de sombras é feito em tempo real através de diversas técnicas, como o uso de *shadow maps*, que renderiza a cena várias vezes a partir do ponto de vista das fontes de luzes presentes (VRIES, 2020). Também é possível pré-computar os *lightmaps*, texturas que definem as partes do objetos iluminadas pelas fontes de luzes da cena, para os objetos e fontes de luzes imóveis e assim não precisar recomputar seus valores a cada quadro, obtendo grande ganho de desempenho. (PLAYCANVAS, 2023)

No entanto, no processo de geramento de sombras é muito comum que estas se tornem pixeladas ou tenham um grande serrilhado. Para resolver este problema,

² Disponível em: <https://blog.playcanvas.com/physically-based-rendering-comes-to-webgl/>. Acesso em: 16 out. 2023.

diversos algoritmos são utilizados a fim de dar às sombras uma penumbra suavizada e remover o serrilhado, tais como: PCF (*Percentage Close Filtering*); PCSS (*Percentage Close Soft-Shadows*), que gera resultados mais realistas a custa de processamento; CSM (*Cascaded Shadow Mapping*), usado junto aos outros, é uma técnica de otimização; VSM (*Variance Shadow Mapping*), uma alternativa ao PCF que gera resultados semelhantes ao PCF com considerável ganho de desempenho. (BABYLON, c2023; MYERS, 2007)

Outro efeito que pode adicionar realismo a cenas é o uso de iluminação global, no qual é utilizado a informação de vários objetos espalhados pela cena para calcular as reflexões, transparência e sombras. Um destes algoritmos é o *Ray Tracing*, que calcula as intensidades de cores de um objeto ao seguir sua reflexão através dos diferentes objetos dispersos na cena. Este método produz resultados muitos mais realistas que a rasterização direta, que renderiza a cena por apenas um ponto de vista. O *Ray Tracing* consegue produzir apenas um número limitado de efeitos, como reflexões agudas e sombras. Uma técnica ainda mais avançada é o *Path Tracing*, que traça um caminho de luz ao lançar vários raios de luz dos objetos visualizados em diferentes direções aleatórias e recursivamente a fim de captar os objetos e fontes de luzes dispersos pela cena, sendo capaz de prover resultados ainda mais realistas e fisicamente acurados. (MARTINDALE, 2022; MÖLLER et al, 2018)

Figura 2: Ray Tracing (esquerda) versus Path Tracing (direita). O método Path Tracing consegue captar melhor as cores do ambiente nos objetos refletidos



Fonte: Hardware Times, 2023³

3 Disponível em: <https://www.hardwaretimes.com/path-tracing-vs-ray-tracing-what-is-the-difference/>.

2.6 - Pós-Processamento, Sistema de Animação e Particulas

Ainda, no final da renderização, é possível adicionar efeitos de pós-processamentos e assim melhorar significamente a aparência da imagem gerada na tela. Alguns dos efeitos mais comuns são: *fog* (neblina ou névoa no plano de fundo); *anti-aliasing* (remover os serrilhados das bordas); *color curves* (controle de saturação, *hue*, sépia, contraste, luminosidade, exposição etc); *depth of field* (desfoque da lente, criando um efeito cinematográfico); *chromatic aberration* (dispersão de cores); *bloom* (brilho das áreas mais luminosas); *motion blur* (borramento em objetos em movimento); *tonemapping* (alteração de cores para HDR); *vignette* (sombreamento nas bordas da tela); SSAO (*Screen Space Ambient Occlusion*, uma forma de calcular o *ambient occlusion* somente com dados da tela, salvando desempenho); SSR (*Screen Space Reflection*, reflexões mais sutis que podem ser usados em poças de água); lens distortion (distorção da imagem pelo formato da lente). (BABYLON.JS, 2023; UNITY, 2022).

Além destas técnicas, também é preciso um meio de trazer os objetos a vida. Um método comum de simular os movimentos de um objeto é pelo sistema de animação esqueletal, que adiciona ao modelo 3D do objeto uma estrutura esquelética constituída de múltiplas articulações, sendo um deles o nó raiz, no qual é organizado a hierarquia constituinte da estrutura esquelética. Cada articulação guarda uma lista de polígonos conectados diretamente a ele, movendo-os ao ter sua posição ou rotação alterada pela animação, além de propagar o movimento para as articulações no nível menor de sua hierarquia. Uma animação é construída através *keyframes*, quadros chaves definidos pelo modelador, que são interpolados pelo programa a fim de criar uma animação fluída (VRIES, 2020)

Uma cena gráfica também pode conter um plano de fundo conhecido como Skybox, uma técnica de criação de cenas 3D com o intuito de melhorar sua ambientação e diminuir o custo de renderização. Ela consiste em criar uma caixa

texturizada ao redor da cena para criar um plano de fundo do ambiente à cena sem ter que modelar e renderizar grandes áreas, melhorando o desempenho e velocidade de renderização da cena. (Yana Krasnolutska, 2023)

Além de objetos, iluminação, e planos de fundo, é comum em aplicações gráficas utilizar um sistema de partículas, uma técnica usada para simular uma quantidade muito grande de objetos em movimentação que se comportam de uma maneira a simular um fenômeno do mundo real, como chuva, fogo, fumaça, fluídos, neve etc. (PLAYCANVAS, 2023)

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 seleção das mesmas, e por fim, fazer uma apresentação das que forem escolhidas a partir dos critérios definidos.

3.1.1 - Bibliotecas Disponíveis

A tabela do apêndice A 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 licença, 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. São elas: Three.js, Babylon.js, RedGL, Webgl-operate, Zogra Renderer, ClayGL, Xeogl, Litescene, Playcanvas, CopperLicht, Xeokit SDK e OpenJSCAD.

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. A *API (Application Programming Interface)* listando todos os módulos, classes e funções de cada biblioteca deve estar disponível com comentários a fim de que suas funcionalidades e uso sejam claros para o desenvolvedor. Também deve haver trechos de código-fonte explicando o funcionamento geral da biblioteca.

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. Portanto, 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 (SMARTPIXELS, [s.d.]), 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. (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 contém 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

Figura 3: Inicialização de uma cena pelo Three.js

```
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

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

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 durante a execução da aplicação. (BABYLON, c2023).

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

Figura 4: Inicialização de uma cena pelo Babylon.js

```
const scene = new BABYLON.Scene(engine);
const camera = new BABYLON.FreeCamera("cameral", 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 configurável 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. (PISSANG et al, 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.

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

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 contém a função *loop*, que rotaciona o cubo no eixo Y a cada quadro.

Figura 5: Inicialização de uma cena pelo ClayGL

```
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, 2018⁶

3.1.3.4 - 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, 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

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

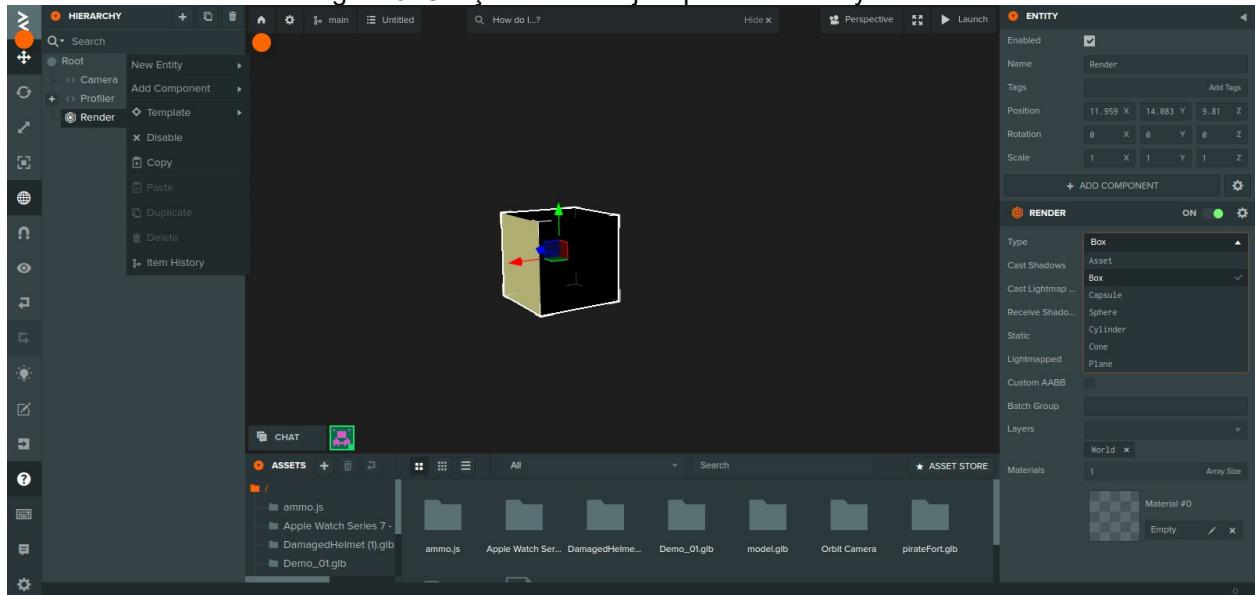
desenvolvedores independentes e também por empresas conhecidas no mercado, como a King, Disney e Nickelodeon. (PLAYCANVAS, 2023)

Também é utilizada para criar aplicações de configurações e visualização arquitetural. Utilizando suas ferramentas de edição visual e manejo 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 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, 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: Criação de um objeto pelo editor Playcanvas



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.

A seção 7.1 trata das métricas qualitativas utilizadas para a comparação, 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 fóruns 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 e correção de bugs é um fator a ser considerado antes da inclusão da mesma em um projeto. Assim, pode-se avaliar, a partir de dados disponíveis nos seus repositórios no Github: data da última atualização, quantidade total e recente de *commits* (atualizações e mudanças feitas em arquivos),

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 PBR disponíveis.
- Efeitos de pós-processamento e de filtros de sombras suportados.
- Utilidades de renderização e otimização, tais como: *Instanced Rendering*, a capacidade de mandar vértices de vários objetos do mesmo para a GPU e renderizá-los numa única chamada; *Mesh Merging*, que consiste em combinar objetos num único; *Level of Details*, que permite reduzir a qualidade dos modelos quando distantes da câmera; *Basis Texture Compression*, o suporte a um formato de textura otimizado para a GPU, que reduz consideravelmente o uso de memória gráfica.
- Funcionalidades e ferramentas extras, como: Sistema de física e colisão; áudio, suporte a realidade virtual, exportação de cenas; editor gráfico de cenas; depurador gráfico de cenas, no qual é possível visualizar os atributos de todos os objetos durante a execução da aplicação; sistema de criação de GUI (interface gráfica) 3D; suporte a renderização com *path tracing*; sistema de *NavMesh*, um sistema que define o caminho pelos quais os NPCs podem caminhar.

- Formatos de arquivos suportados de modelos 3D para importação.
- Suporte a versão 2 do WebGL.

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, que aumentam o peso da biblioteca, 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 que contêm: poucos e muitos objetos; objetos simples e complexos; muitas fontes de luzes; sombras ativadas; pós-processamento *Anti-Aliasing FXAA*, *SSAO* e *color curves*.

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, esta métrica busca catalogar os seguintes dados de tamanho da biblioteca:

- Memória ocupada total dos arquivos do repositório público da biblioteca e de suas dependências ao serem baixados por um gerenciador de pacotes. 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.
- 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*. Medir os tamanhos neste formato será útil para obter a diferença real entre o peso de cada biblioteca em ambientes de produção.

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

3.3.2.1 - Gerenciador de pacotes NPM

O gerenciador de pacotes NPM pode ser definido como:

NPM ou *Node Package Manager* é uma biblioteca e repositório de código aberto para pacotes de JavaScript softwares. Ele é uma ferramenta muito útil do Node.js que com funções simples permite aos desenvolvedores instalar, desinstalar, atualizar e gerenciar dependências em uma aplicação de forma prática e fácil. (ABRAMOWSKI, 2022).

Quando uma biblioteca é baixada pelo NPM num diretório de um projeto, é gerado um subdiretório para essa biblioteca e mais para cada dependência dessa biblioteca. Assim, basta verificar o tamanho dos subdiretórios criados pelo gerenciador de arquivos para obter o tamanho total de *download* da biblioteca em questão.

3.3.2.2 - Javascript Bundler Esbuild

É comum que as bibliotecas desenvolvidas sejam divididas em múltiplos arquivos de código-fonte para melhorar a organização do projeto, e como consequência, torna-se mais difícil de analisar qual o tamanho dos arquivos que realmente serão requisitados pelo navegador ao desenvolver uma aplicação que utiliza esta biblioteca. Um *bundler*, como o Esbuild, é capaz de pegar todos os arquivos de código-fonte utilizadas por uma aplicação e juntá-lo num único arquivo, tornando mais fácil a medição do tamanho real em memória que a biblioteca irá utilizar. Também é possível aplicar o algoritmo de minificação, que reduz consideravelmente o tamanho

final do arquivo gerado. (ESBUILD, [s.d])

3.3.2.3 - Compressão DEFLATE

Além da minificação, é possível reduzir ainda mais o tamanho final do arquivo gerado ao aplicar o algoritmo DEFLATE, que pode ser feito através da ferramenta de linha de comando Gzip, disponível em sistemas operacionais Linux (FREE SOFTWARE FOUNDATION, c2009-2023). Este algoritmo foi escolhido porque é suportado por diversos servidores WEB, como o Nginx e Apache (KIRUBAI, 2023), e pelos navegadores, que consegue descomprimir os arquivos nesta codificação de volta ao seu formato funcional.

3.3.3 - Mensuradores de Desempenho

Esta seção trata das ferramentas utilizadas nos testes de desempenho, no qual serão avaliados o uso de CPU, memória RAM, tempo de inicialização e renderização de cada biblioteca. Serão utilizadas para este propósito três ferramentas, um navegador, o painel *Devtools*, e as funções na linguagem de programação *Javascript*.

3.3.3.1 - Navegadores

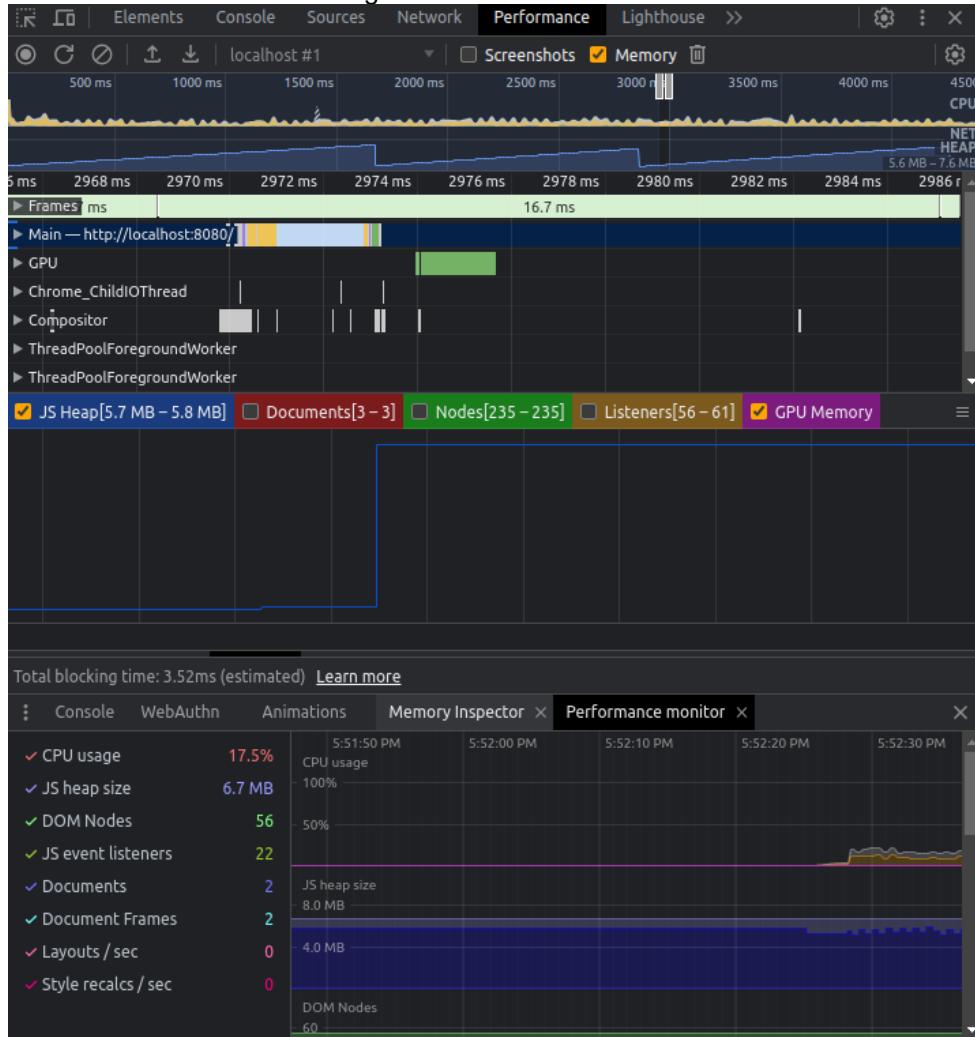
As bibliotecas deste projeto foram desenvolvidas para serem utilizadas no contexto de uma página *Web*, e assim, requerem um navegador para serem executados no computador. Inicialmente, o navegador *Google Chrome* foi selecionado por ser o mais utilizado pelos usuários (SIMILARWEB, 2023) e por ter um mensurador de CPU e RAM no seu painel de desenvolvedor *Devtools*. Os dados relacionados ao tempo de renderização também foram coletados nos navegadores *Microsoft Edge* e *Mozilla Firefox*.

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.

Figura 7: Painel Devtools.



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) e FPS 1% low (piores valores em 1% do tempo, importante para conhecer os picos de atraso) (BUTLER, 2023).

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.

- PC-1: 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.
- PC-2: 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.

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. Redes sociais: Fórum, Discord e 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. Redes sociais: Fórum, Discord, 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 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. Rede sociais: Fórum, Discord e Twitter.

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 suporte nem é mantida pelos desenvolvedores.

4.2 - Análise das Métricas Quantitativas

4.2.1 - Desenvolvimento do Projeto

A tabela abaixo mostra os dados referentes ao desenvolvimento do projeto no período de um mês, no qual se observa que a biblioteca Three.js e Babylon.Js possuem o desenvolvimento mais ativo com grande número de *commits*, sendo que a primeira tem o dobro de contribuídores. A Playcanvas fica em terceiro lugar, enquanto a

biblioteca ClayGL já teve seu desenvolvimento descontinuado em 2021, não recebendo mais nenhuma adição de funcionalidade ou correção de *bugs* desde então.

Tabela 1: Desenvolvimento de cada biblioteca ao longo de um mês

Métricas Recentes	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
Pull Requests (Criados)	10	5	0	4
Pull Requests (Merged)	120	118	0	66
Issues (Criados)	15	6	0	22
Issues (Concluídos)	46	19	0	20
Autores	40	20	0	14
Commits	134	221	0	68

Fonte: Dados obtidos no Github referente aos dias entre 5 de Setembro de 2023 e 5 de Outubro de 2023

Enquanto isso, na tabela abaixo observa-se a popularidade da Three.js pelo número de estrelas e contribuídores, além do número de projetos hospedados no Github que a utilizam. Em segundo lugar fica a Babylon.js, seguida da Playcanvas, e por fim a ClayGL, com popularidade muito menor.

Tabela 2: Desenvolvimento de cada biblioteca total

Métricas Totais	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
Pull Requests (Ativos)	145	8	11	21
Pull Requests (Fechados)	14.627	11.371	48	3.692
Issues (Abertos)	386	93	37	525
Issues (Fechados)	1.1550	2.836	48	1.422
Contribuídores	1.736	485	11	130
Commits	42.170	41.403	1.191	10.902
Estrelas	94.810	21.464	2.671	8.671
Usado Por	228.190	4.648	0	1

Fonte: Dados obtidos no Github no dia 5 de Outubro de 2023

4.2.2 - Funcionalidades Presentes

4.2.2.1 - Propriedades dos Materiais PBR Disponíveis

A tabela 4 do apêndice demonstra as propriedades dos materiais PBR disponíveis em cada biblioteca. Um campo marcado com “Mapa” indica que a propriedade é configurável através de uma textura, ou seja, por pixel, caso seja marcado com “Escalar” significa que a propriedade só pode ser configurada em forma de intensidade (um número de 0.0 a 1.0) ou em forma de coloração geral (um valor RGB), enquanto o campo vazio implica a ausência total da propriedade.

É possível perceber que a Babylon.js possui todas as 23 propriedades listadas, seguida da Playcanvas, com 20, e da Three.js, com 19, e por último a ClayGL, que apresenta um suporte apenas as 6 propriedades mais básicas.

Das propriedades que faltam ao Playcanvas, destacam-se a *transmission* – reflexão de materiais transparentes, utilizadas em vidros de carros, por exemplo – e os *decals* – texturas extras que podem ser utilizadas para dar mais detalhes dinâmicos a um objeto, como um vidro com furos de tiros ou janelas janelas quebradas.

Já em relação ao Three.js, nota-se a falta dos atributos *anisotropic* e dos *details*. Também não possui suporte a *decals* pelas propriedades dos materiais, no entanto permite a criação de geometrias que funcionam como os tais, ou seja, implementa a funcionalidade através de outro sistema.

A propriedade *Parallax* está disponível apenas na biblioteca Babylon.js, e é responsável por criar um efeito de profundidade e de sobressaliente em superfícies irregulares.



Fonte: Post de ddd no fórum Blender, 2019⁷

4.2.2.2 - Efeitos de Pós-Processamento

A tabela 5 do apêndice contém os dados em relação a alguns dos efeitos de pós-processamento mais populares, enquanto a tabela 6 contém a listagem dos filtros de sombras modernos que são suportados por cada uma das bibliotecas. Um campo marcado com “X” significa o filtro está disponível nos repositórios oficiais da biblioteca, enquanto os campos marcados com “Plugin” significam que a funcionalidade está disponível pela extensão de um terceiro.

Observa-se que a biblioteca Three.js contém a maior biblioteca de efeitos disponíveis com o auxílio de *plugins*, estando a Babylon.js em segundo lugar, na qual falta-se notoriamente alguns efeitos de *anti-aliasing*, os SMAA / TAA – efeitos mais modernos e balanceados -, e em contraste possuindo o FXAA, o efeito mais rápido e menos eficiente, e o MSAA, que embora possa produzir resultados superiores, demanda muito mais do *hardware* (AREEJ, 2023).

Nota-se também que a biblioteca ClayGL não possui o efeito de *fog*, que é muito utilizado para gerar névoas em longas distâncias em diversos jogos, além de outros efeitos acessórios, como *vignette*, útil para sombrar as bordas da tela, e efeitos de *blur*.

Em último lugar se apresenta a biblioteca Playcanvas, que possui a menor quantidade de efeitos de pós-processamento em relação as outras, que, além da falta dos efeitos de *anti-aliasing* mais avançados e o efeito de *blur*, destaca-se a ausência dos efeitos de SSR e Depth of Field, que são capazes de mudar significativamente a ambientação de uma cena. Para agravar a situação, todos os efeitos que a PlayCanvas afirma suportar não funcionaram nos testes de desempenho (seção x.y).

4.2.2.3 - Efeitos de Sombras Suportados

A tabela abaixo contém a lista dos diferentes filtros de sombras suportados em cada biblioteca. Os filtros, com exceção do CSM, que é uma técnica de otimização, são

⁷ Disponível em: <https://blender.community/c/rightclickselect/t9bbbc/?sorting=hot>. Acesso em: 22 out. 2023.

excludentes entre si, e podem ser escolhidos a depender da preferência do usuário por maior desempenho ou qualidade gráfica.

Técnica de Sombreamento	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
PCF	X	X	X	X
PCSS		X		X
VSM	X		X	X
CSM	X	X	X	X

Uma importante observação é que nos testes de desempenho no qual se foram avaliados a capacidade de renderização de sombras utilizando o filtro PCF, a biblioteca ClayGL não produziu o efeitos esperado (seção x.y).

4.2.2.3 - Utilidades de Otimização

A tabela abaixo trata de listar algumas das técnicas de otimização de eficiência utilizadas por aplicações de computação gráfica, que podem ser utilizadas em situações específicas a fim de economizar recursos de *hardware* ou melhorar o tempo de renderização.

Técnica de Otimização	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
Instanced Rendering	X	X	X	X
Combinação de Malhas	X	X		X
Compressão de Texturas BASIS	X	X		X
Geração de Lightmaps				X
LOD (Nível de Detalhes)	X	X		

É importante notar que é possível efetuar a geração de Lightmaps e combinação de malhas em um programa de modelagem 3D , exportando a cena para ser utilizada pelas bibliotecas que não possuem a funcionalidade.

4.2.2.4 - Formatos de Arquivos 3D para Importação Suportados

A tabela abaixo trata dos diferentes formatos de arquivo que cada biblioteca suporta para a importação de objetos 3D. Os formatos GLTF, COLLADA, OBJ e FBX são formatos de propósito geral, enquanto o MMD e 3DS são vinculados a programas de modelagem proprietários, e os outros formatos são utilizados em certos domínios de aplicações, como CADs, modelagem de avatares, dados científicos e Lego. O último campo se refere a capacidade em carregar arquivos comprimidos pela biblioteca DRACO.

Formato	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
GLTF / GLB (Open Source)	X	X	Parcial	X
COLLADA / DAE (Open Source)	X			X
OBJ (Open Source)	X	X		X
FBX (Proprietário)	X			X
MMD (Proprietário)	X			
3DS (Proprietário)	X			
3dm (CAD)	X			
STL (CAD)	X	X		
VRM (Avatares)	X			
PCD (Point Cloud Data)	X			
PDB (Protein Data Blank)	X			
Ldraw (LEGO)	X			
Compressão de Malha DRACO	X	X		X

Nota-se que a ClayGL possui somente suporte parcial ao formato GLTF, dado explicado na seção xyz.

4.2.2.5 - Funcionalidades e Ferramentas Extras Presentes

A tabela 9 trata de listar diversas funcionalidades extras que cada biblioteca implementa e que possuem diferentes utilidades a depender do domínio da aplicação desenvolvida. A biblioteca ClayGL aparece com mínimas utilidades, tendo apenas um sistema de partículas e animação.

Nota-se que funcionalidades úteis no desenvolvimento de jogos, como sistema de partículas, áudio, física (através da integração com um plugin externo) e *NavMesh* estão presentes nas bibliotecas Playcanvas e Babylon.js. Já a biblioteca Three.js requer o uso de *plugins* para fornecer tais funcionalidades – com exceção do áudio, que funciona nativamente -, não obstante, não possui integração com um sistema de física, cabendo ao desenvolvedor da aplicação implementar suas próprias integrações para conseguir obter tal utilidade. A necessidade de muitos *plugins* também implica na limitação do editor gráfico da Three.js em relação aos outros, uma vez que não é capaz de integrar suas funcionalidades como se fossem nativas.

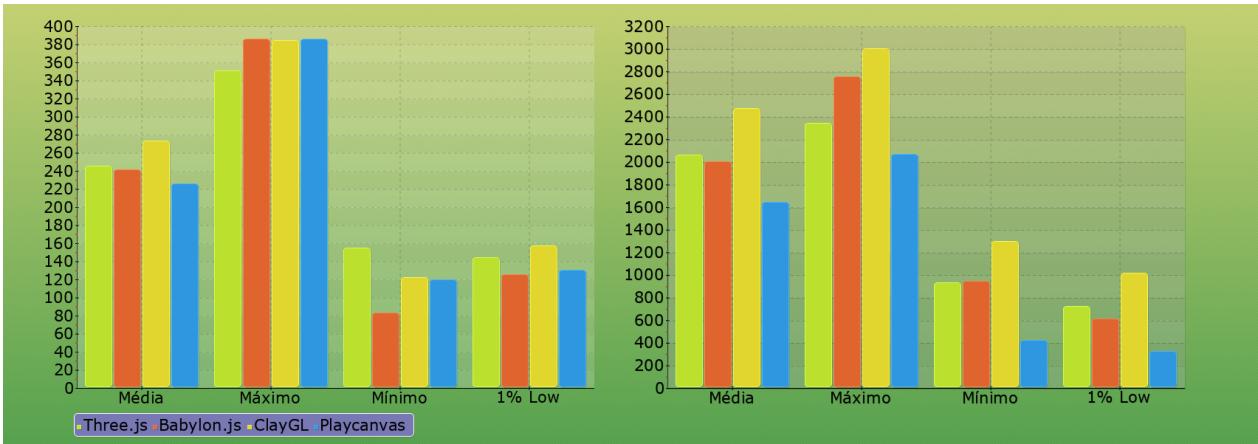
Também é possível observar que as bibliotecas Three.js e Babylon.js são as únicas que contêm *plugins* para renderizar cenas altamente realistas pelo *Path Tracing*, além de possuírem um sistema de depurador de cenas em tempo real, uma ferramenta que pode auxiliar o desenvolvedor a diagnosticar falhas e *bugs* em suas aplicações. A Three.js também falta em prover um sistema de criação de GUI que com se mescle com os objetos 3D, sendo apenas possível criá-las pelo navegador utilizando HTML e CSS.

4.2.3 - Testes de Desempenho

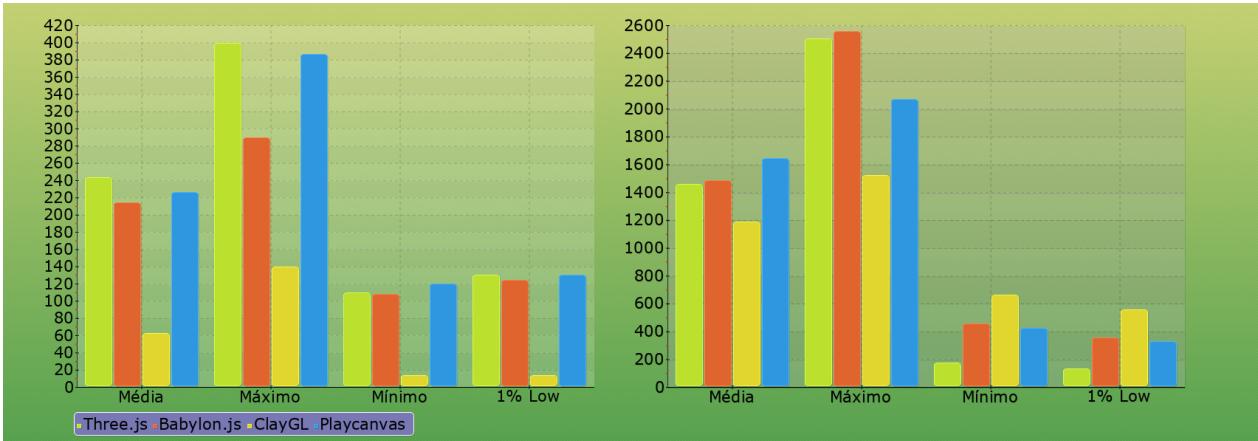
Este subcapítulo é dedicado a demonstrar todas as estatísticas relacionadas aos testes de desempenho, nos quais foram renderizados diferentes tipos de cenas em dois diferentes computadores e para cada biblioteca no navegador Chrome. A tabela X e Y do apêndice contêm as informações das cenas e dos testes. Já as imagens do apêndice Z demonstra as imagens geradas.

4.3.3.1 – Renderização e FPS

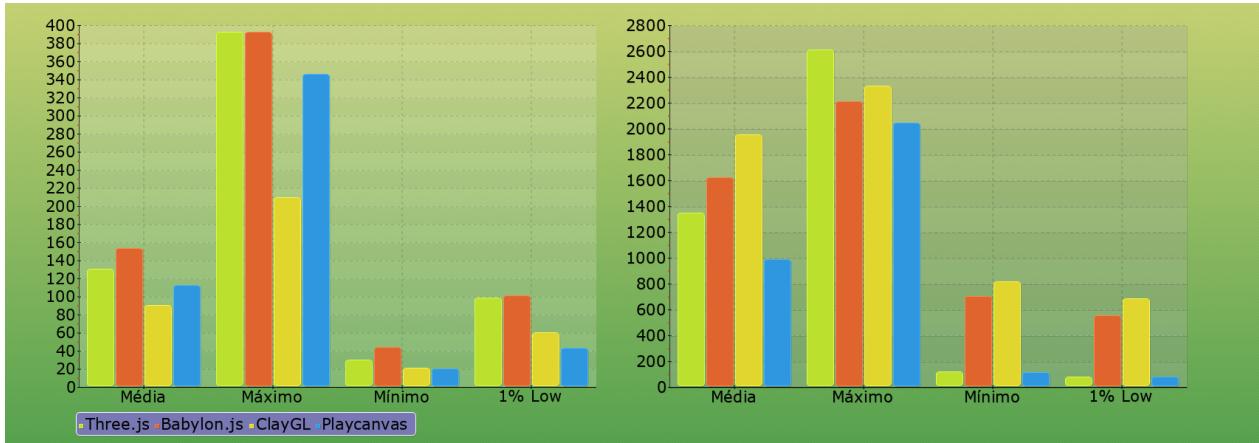
O gráfico abaixo contêm os dados de FPS referentes ao primeiro teste de desempenho, que renderiza a cena “Forte Pirata”, cena simples com baixo contingente de objetos (7), triângulos (90.334) e luzes (4 omni), e câmera distante. A esquerda, os dados coletados no PC-1, e na direita, no PC-2.



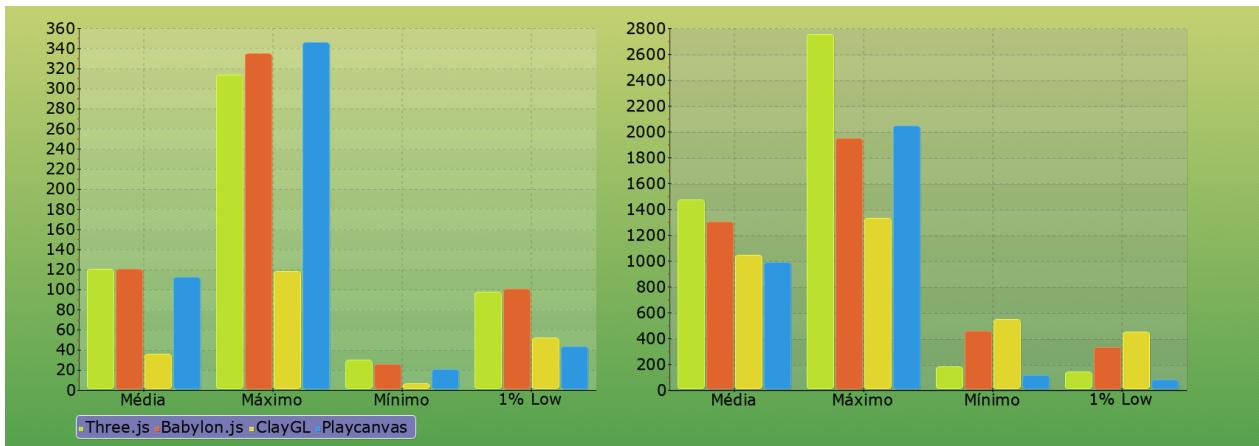
O gráfico abaixo se refere à mesma cena, mas com o pós-processamento de *color curves* habilitado para as três bibliotecas que não a Playcanvans, sendo que esta já ativa o efeito por padrão, tendo seus dados repetidos do gráfico anterior.



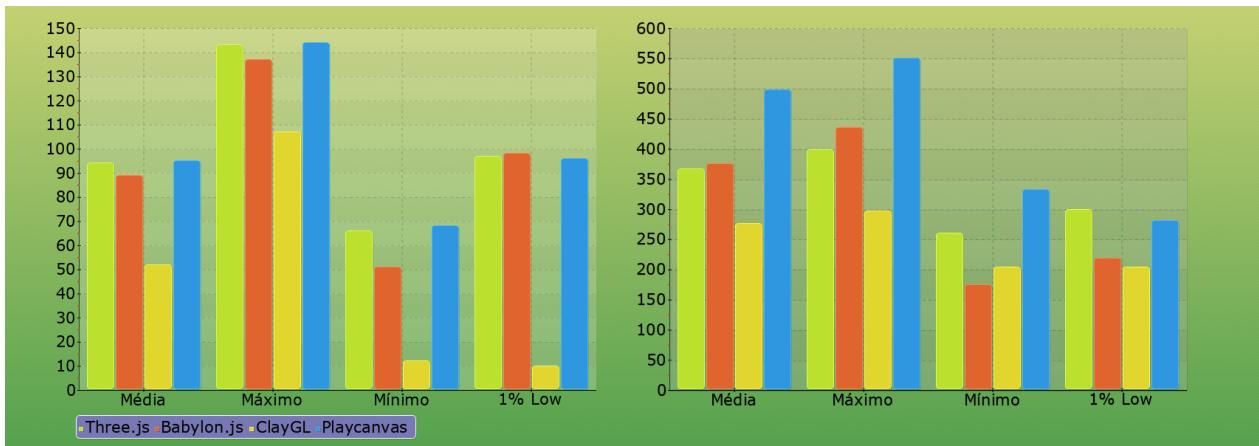
A seguir são apresentados aos dados referentes a renderização do “Forte Pirata”, mas com a câmera próxima dos objetos.



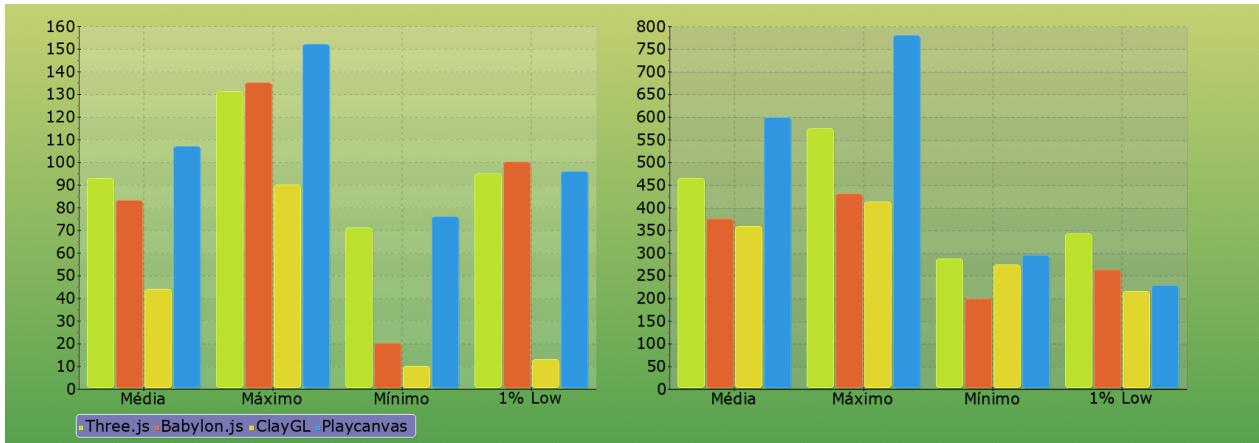
O gráfico abaixo mostra os dados do mesmo teste com a câmera perto e com os mesmos efeitos de pós-processamento ativados para as três bibliotecas que não o ativam por padrão.



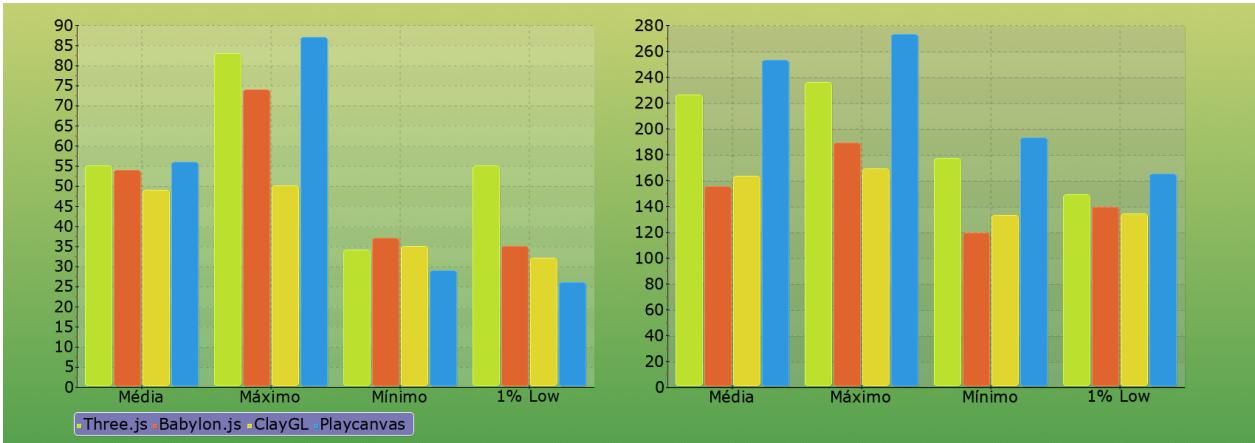
A seguir apresenta-se os dados referentes a renderização da cena da “Floresta”, que contém uma grande quantidade de objetos (1228) e triângulos (2.076.694), e uma luz direcional, com a câmera distante.



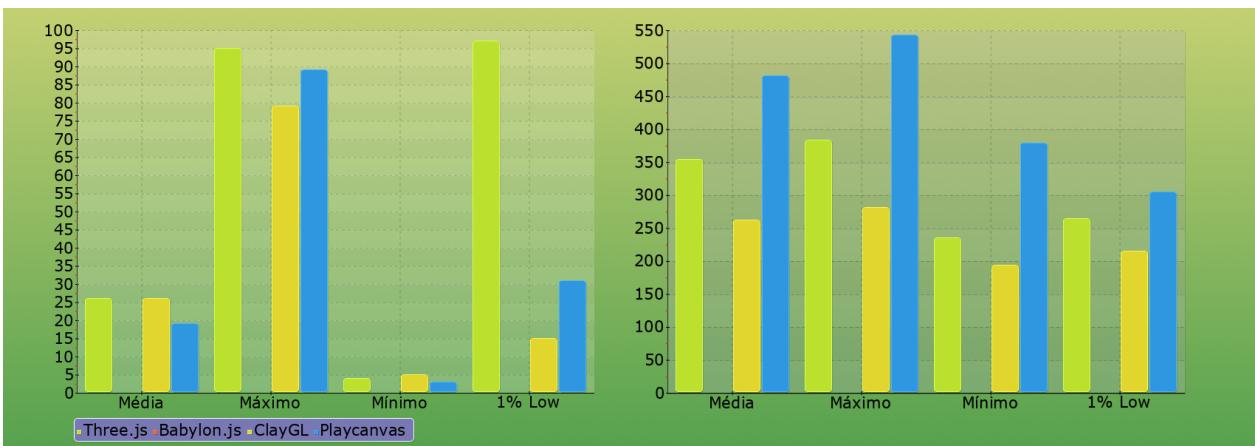
A seguir os dados com a câmera próxima.



Abaixo são apresentados os dados referentes a renderização da cena da “Floresta”, que avaliou a capacidade das cenas renderizarem sombras (utilizando filtro PCF).

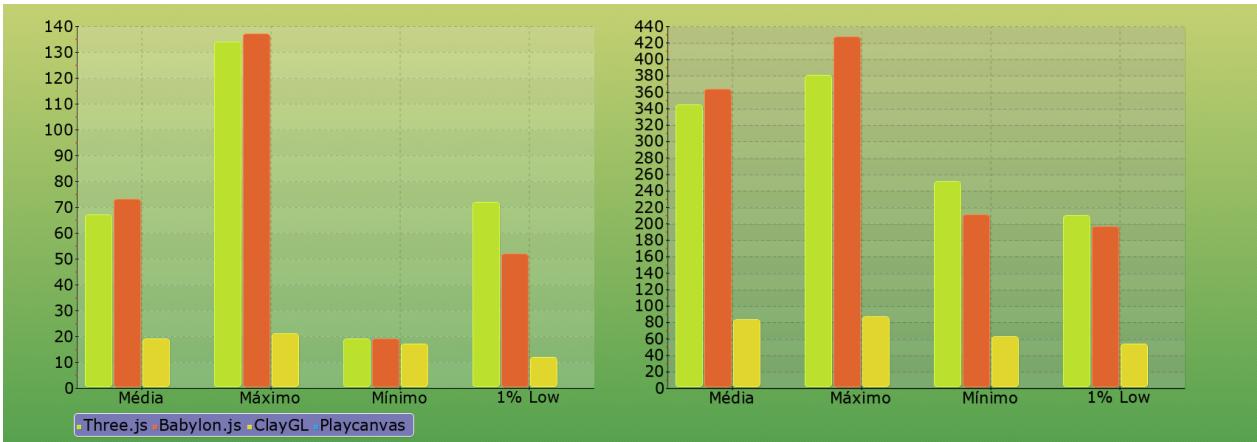


A seguir são apresentados os dados referentes a renderização da mesma cena, mas com sombras desligadas e 28 fontes de luzes do tipo omni espalhadas pela cena. A biblioteca Babylon.js não suporta mais do que 4 luzes simultâneas, estando excluída do teste.

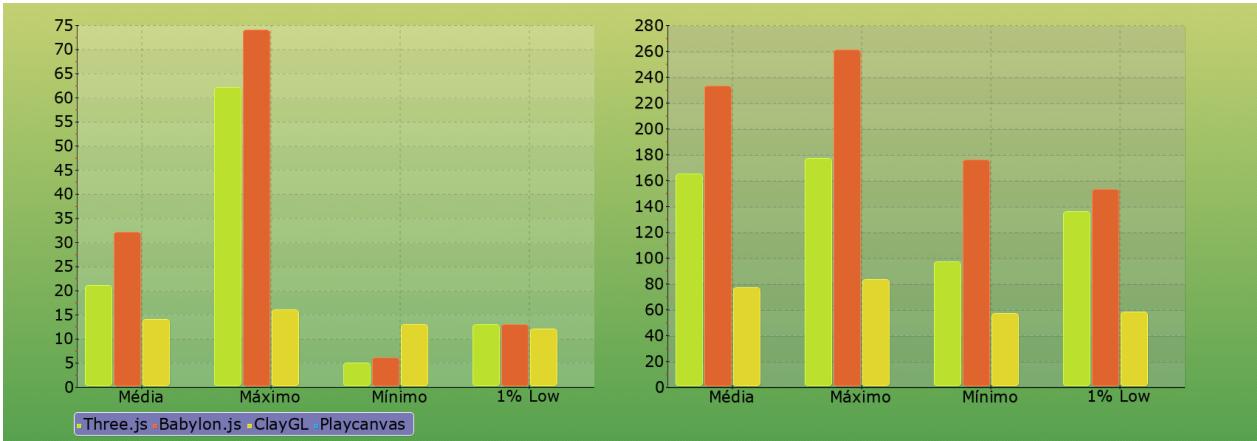


A seguir seguem-se os dados referentes ao teste no qual se testou o pós-processamento das bibliotecas ao renderizar a “Floresta” com o efeito *Anti-Aliasing FXAA* ligado. A biblioteca Playcanvas não está presente devido ao script relacionado a

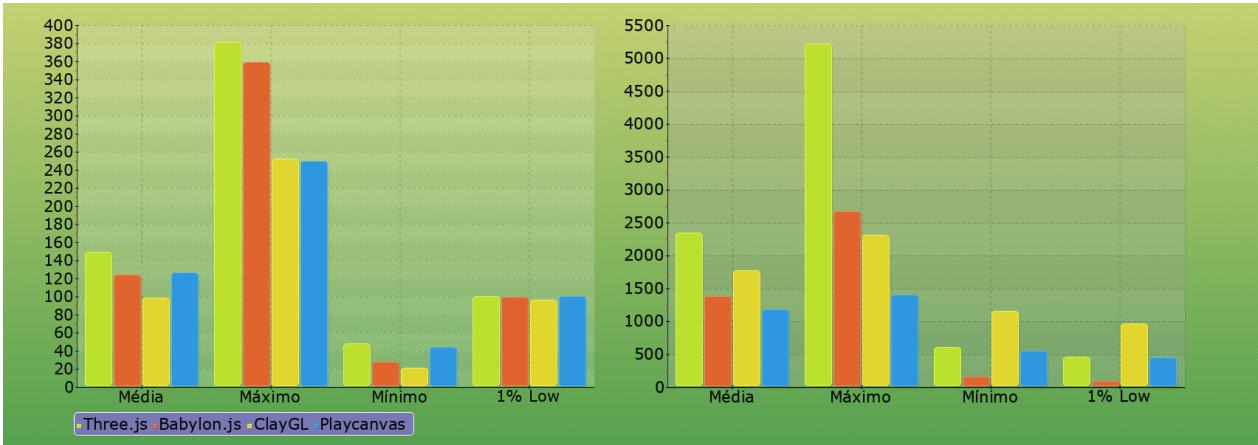
tal funcionalidade não ter funcionado (seção x.y)



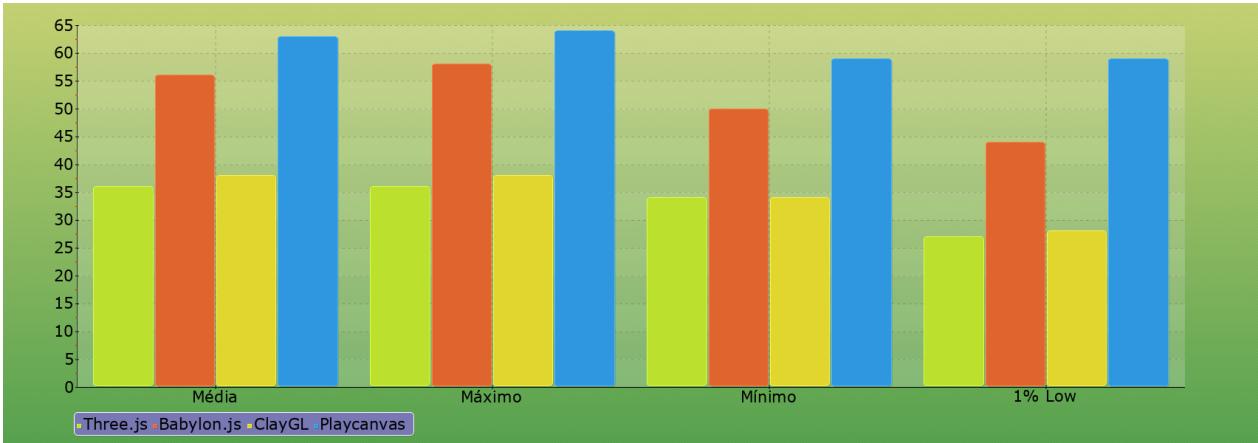
Os gráficos abaixo tratam do teste que analisa o uso de efeitos de pós-processamento simultâneos pelas biblioteca, utilizando o *Anti-Aliasing FXAA*, *color curves* e *SSAO*, também realizado na cena da “Floresta”,



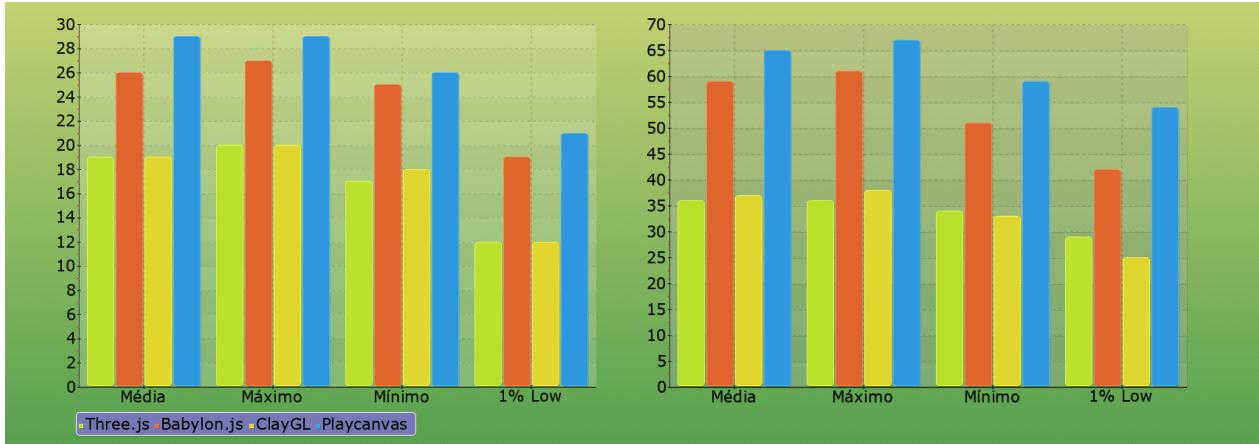
Abaixo segue-se os dados referentes ao teste que analisa a capacidade das bibliotecas de renderizar um modelo muito complexo, assim foi utilizado a cena da “Floresta (Combinado)”, que contém a mesma cena da “Floresta”, mas com todos os objetos condensados numa única malha poligonal.



O gráfico abaixo trata dos dados do teste no qual se renderiza a cena “Deserto”, uma cena extremamente complexa, realizado apenas no PC-2. Apesar da biblioteca ClayGL ter ficado em primeiro lugar, ela não conseguiu renderizar a cena apropriadamente (seção x.y, figura z).

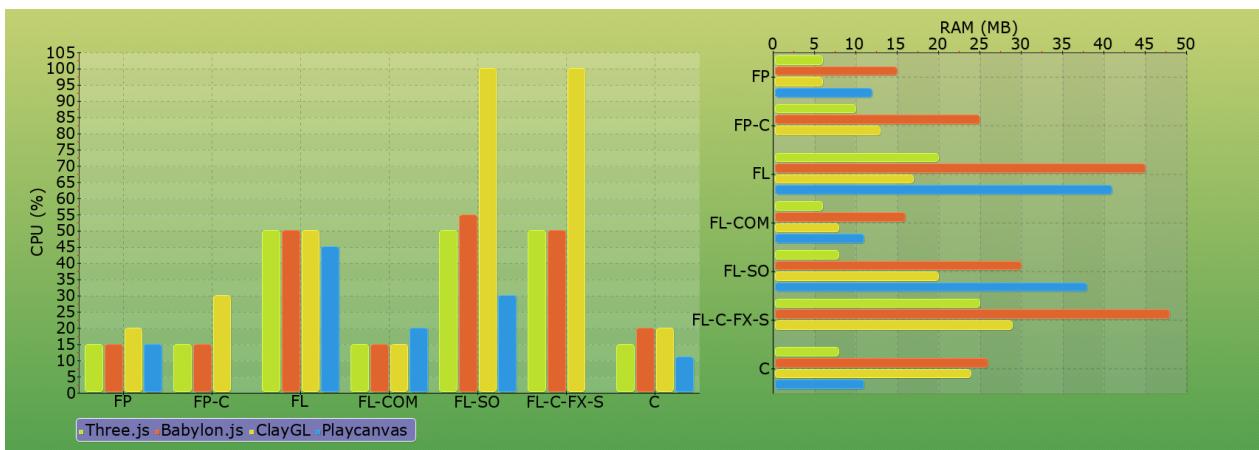


A seguir os gráficos referentes a renderização da cena do “Deserto”, mas com efeitos de sombra ativados (à esquerda), e com maior contagem de luzes, mas sem sombras (à direita), também realizados no PC-2.

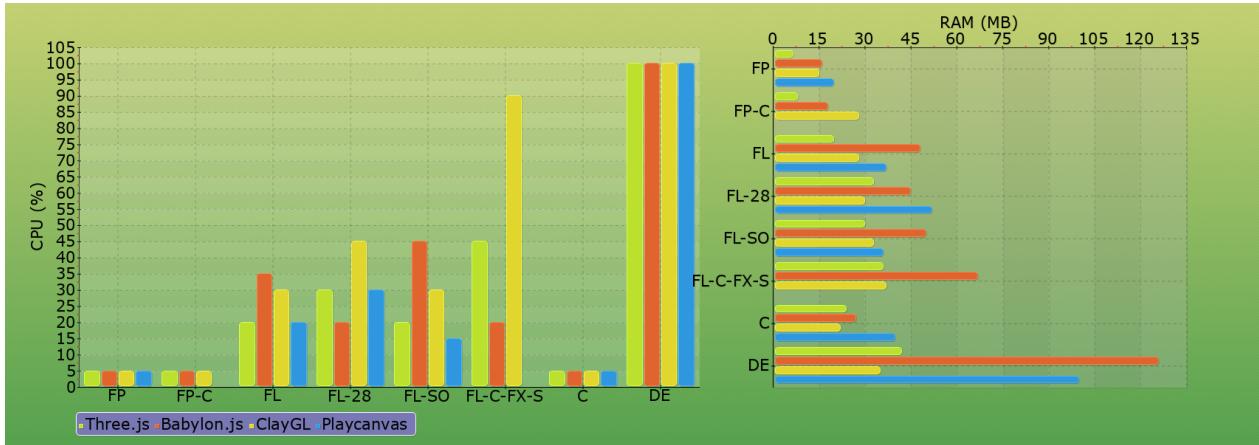


4.3.3.2 – Uso de CPU e memória RAM

A seguir são apresentados os dados do uso de CPU e de gasto de memória RAM dos programas pelas bibliotecas em cada uma das cenas. O significado das siglas corresponde a tabela X do apêndice y. Abaixo os gráficos referentes aos testes realizados no PC-1.

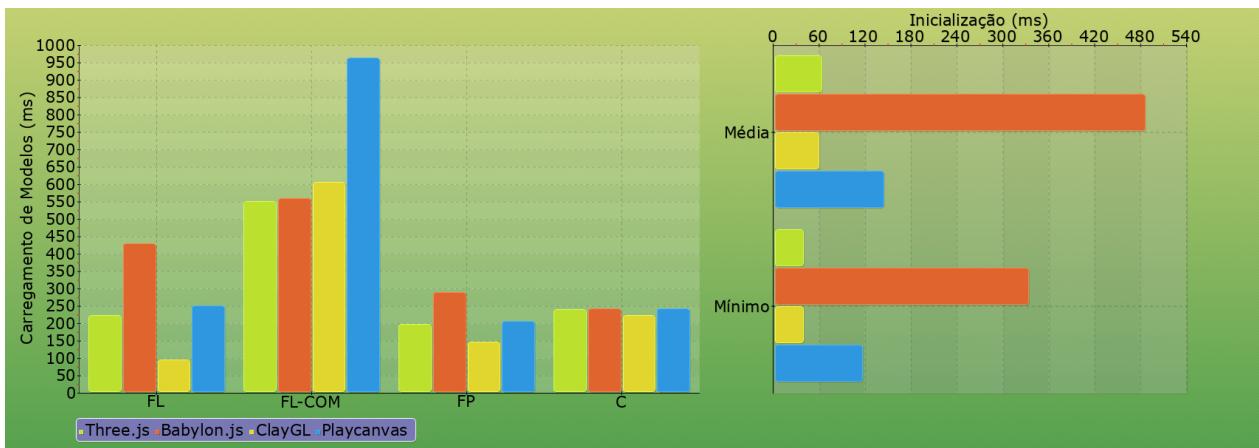


Abaixo os dados coletados no PC-2.

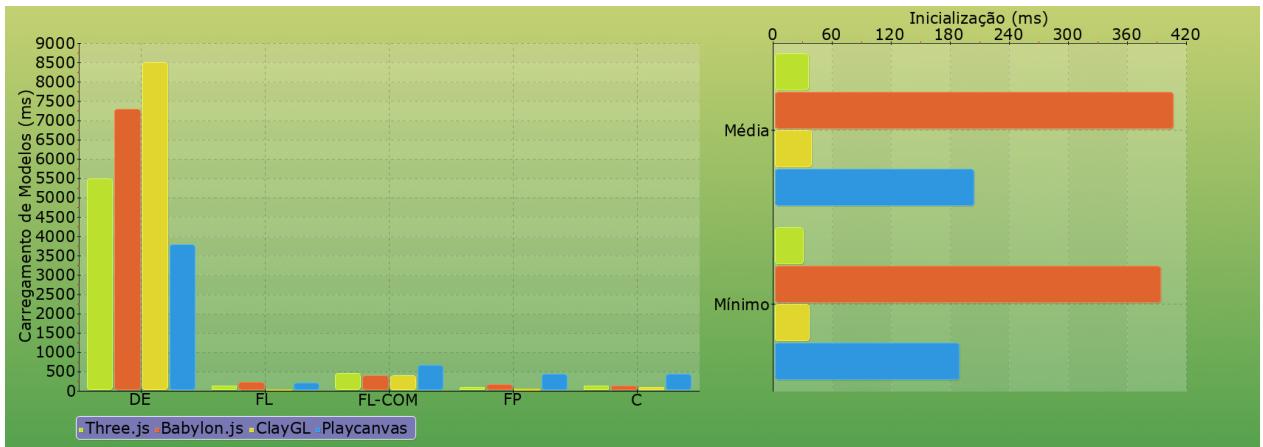


4.3.3.4 – Tempo de Inicialização e Carregamento de Modelos

A seguir são apresentados os dados do tempo de carregamento de modelos (gráficos à esquerda) e do tempo de inicialização das bibliotecas (gráficos à direita), isto é, o tempo que a biblioteca demorá para carregar no navegador. A informação referente a inicialização está disponível tanto no tempo mínimo verificado, como na média dos valores coletados. Segue abaixo o gráfico coletado no PC-1.



A seguir tem-se os dados coletados no PC-2.



4.3.3.3 – Observações

4.3.3.4 – Problemas Encontrados

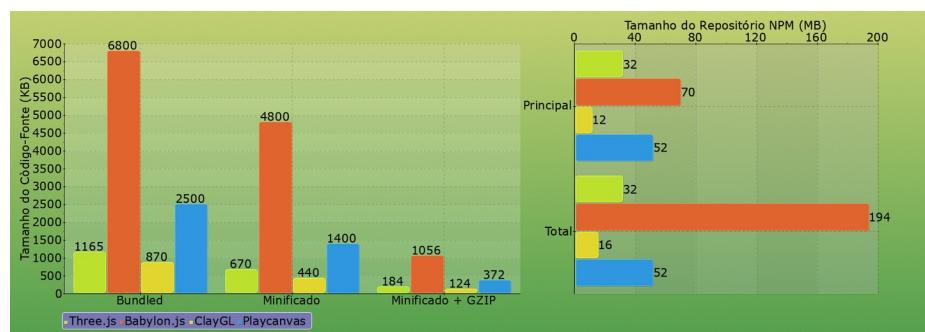
Ao realizar os testes de renderização, alguns defeitos e falhas na renderização foram encontrados:

- Na renderização de cenas com sombras ativadas pela biblioteca Three.js, no navegador Google Chrome e no PC-1, a cena ficou sem cores e repleta de artefatos visuais, conforme imagem X do apêndice Y.
- Na renderização da cena do deserto, realizado somente no PC-2, a biblioteca

ClayGL gerou cores severamente incorretas. As sombras também não renderizaram corretamente. Os efeitos foram observados nos dois navegadores (Chrome e Edge).

4.2.4 - Tamanho da Biblioteca

O gráfico abaixo trata de mostrar o tamanho em memória de cada biblioteca. O gráfico a esquerda trata do tamanho do código-fonte, antes e depois das transformações de minificação e compressão, e o da direita trata do tamanho de *download* dos repositórios públicos das bibliotecas.



Ao analisar o tamanho do código-fonte, percebe-se que a biblioteca Babylon.js é significativamente mais pesada que as outras, o significa maior carga no servidor e maior demora para o carregamento das páginas. Mas ao aplicar os algoritmos de compressão e minificação, consegue-se reduzir consideravelmente as diferenças entre elas, e assim a Babylon.js passa a possuir apenas 1.1MB, tamanho semelhante a uma imagem em resolução FULL HD, reduzindo drasticamente tais possíveis problemas.

5 – Conclusão

Referências

3D computer graphics with Three.js. LILLY021 Blog, 2022. Disponível em: <https://lilly021.com/3d-computer-graphics-with-three-js>. Acesso em: 18 mai. 2023.

Abramowski, Nicole. **What is NPM? A Beginner's Guide.** CareerFoundry, 2022; Disponível em: <https://careerfoundry.com/en/blog/web-development/what-is-npm/>. Acesso em: 05 out. 2023.

AREEJ. **TAA vs SMAA vs FXAA Graphics Settings Compared: Which One Should You Choose?**. Hardware Times, 2023. Disponível em: <https://www.hardwaretimes.com/taa-vs-smaa-vs-fxaa-graphics-settings-compared-which-one-should-you-choose/>. Acesso em: 23 out. 2023.

BABYLON.JS, Documentation. Babylon.js, 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.

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

CAMERAS. Blender, 2023. Disponível em: <https://docs.blender.org/manual/en/3.6/render/cameras.html>. Acesso em: 10 out. 2023.

CAULFIELD, Brian. **O que é Path Tracing?**. Blog NVIDIA, 2022. Disponível em: <https://blog.nvidia.com.br/2022/05/10/o-que-e-path-tracing/>. Acesso em: 16 out. 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.

DREWOREN. **Three.Js**, 2019. Disponível em: <https://2019-spring-web-dev.readthedocs.io/en/latest/final/roen/index.html>. Acesso em: 18 mai. 2023.

ESBUILD: An extremely fast bundler for the web. Esvuild, [s.d]. Disponível em: <https://esbuild.github.io/>. Acesso em: 21 out. 2023.

Free Software Foundation. **GNU Gzip: General File (de)compression.** GNU, c2009-2023. Disponível em: <https://www.gnu.org/software/gzip/manual/gzip.html>.

Acesso em: 21 out. 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.

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

GONÇALVES, Júnior. **Introdução 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.

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

IRWIN, Emma. **Microsoft 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 mai. 2023.

KEIZER, Gregg. **The Brave browser basics: what it does, how it differs from rivals**. Computer World, 2021. Disponível em: <https://www.computerworld.com/article/3292619/the-brave-browser-basics-what-it-does-how-it-differs-from-rivals.html>. Acesso em: 21 out. 2023.

KIRUBAI, Lydia. **Gzip Compression for Faster Web Pages (Apache, Nginx, WordPress)**. Atatus, 2022. Disponível em: <https://www.atatus.com/blog/gzip-compression-for-faster-web-pages/>. Acesso em: 21 out. 2023.

Krasnolutska, Yana. **Unlocking the Skies: Exploring the Immersive Potential of Unity 3D Skyboxes**. MARKETSPLASH, c2023. Disponível em: <https://marketsplash.com/tutorials/unity-3d/unity-3d-skyboxes/>. Acesso em: 07 out. 2023.

LEE, Tina. **Lights and Shadows: CG Lighting Types for 3D Animation**. Academy Of Animated Art, c2023. Disponível em: <https://academyofanimatedart.com/lights-and-shadows-cg-lighting-types-for-3d-animation/>. Acesso em: 27 set. 2023.

MARTINDALE, Jon. **Ray tracing vs. path tracing — which is the best dynamic lighting technique?**. DigitalTrends, 2022. Disponível em: <https://www.digitaltrends.com/computing/ray-tracing-vs-path-tracing>. Acesso em: 16 out. 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.

MYERS, Kevin. **Variance Shadow Mapping.** Developer Nvidia, 2007. Disponível em: <https://developer.download.nvidia.com/SDK/10/direct3d/Source/VarianceShadowMapping/Doc/VarianceShadowMapping.pdf>. Acesso em: 21 out. 2023.

MÖLLER, Tomas 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.

NAGHDI, Arash. “The ultimate guide to lighting fundamentals for 3D”. Dream Farm Studios, c2020. Disponível em: <https://dreamfarmstudios.com/blog/the-ultimate-guide-to-lighting-fundamentals-for-3d/> Acesso em: 27 set. 2023.

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

PETERSEN, Osgo. 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.

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 User Manual. Playcanvas, 2023. Disponível em: <https://developer.playcanvas.com/en/>. Acesso em: 25 mai. 2023.

PLAYCANVAS: THE WEB-FIRST GAME ENGINE. Playcanvas, 2023. Disponível em: <https://playcanvas.com>. Acesso em: 25 mai. 2023.

POST-PROCESSING and full-screen effects. Unity User Manual, 2022. Disponível em: <https://docs.unity3d.com/Manual/PostProcessingOverview.html>. Acesso em: 16 out. 2023.

RIGGING and skeletal animation: what it is and how it works. Adobe, c2023. Disponível em: <https://www.adobe.com/uk/creativecloud/animation/discover/rigging.html>. Acesso em: 07 out. 2023.

RUSSELL, Jeff. **Basic Theory of Physically-Based Rendering**. Marmoset LLC, 2023 Disponível em: <https://marmoset.co/posts/basic-theory-of-physically-based-rendering/>. Acesso em: 02 out. 2023.

SANTOS, Cleyder, **O que são materiais e texturas em um software 3D?** Alura, c2022. Disponível em: <https://www.alura.com.br/artigos/o-que-sao-materiais-texturas-software-3d>. Acesso em: 26 set. 2023.

SHAHBAZI, Nazanin. **Exploring the World of 3D Textures: A Comprehensive Guide**. Pixtune, c2023. Disponível em: <https://pixtune.com/blog/3d-texturing/>. Acesso em: 26 set. 2023.

SHEN, Yi et al. **ClayGL** Github, 2018. Disponível em: <https://github.com/pissang/claygl/blob/master/README.md>. Acesso em: 16 out. 2023.

SILVA, Luiz . **O que é geometria?**. Brasil Escola, [s.d]. Disponível em: <https://brasilescola.uol.com.br/o-que-e/matematica/o-que-e-geometria.htm>. Acesso em: 24 set. 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.

SINGH, Balpreet. **How to make photorealistic 3D graphics with different texture maps?**. Webdew, c2022. Disponível em: <https://www.webdew.com/blog/how-to-make-photorealistic-3d-graphics>. Acesso em: 27 set. 2023.

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

THREE.JS Docs. Three.js, 2023. Disponível em: <https://threejs.org/docs/>. Acesso em: 18 mai. 2023.

TIIGIMÄGI, Siim. **O que é uma malha Polygon e como editá-la?**. 3D Studio, c2014-2023. Disponível em: <https://3dstudio.co.pt/polygon-mesh>. Acesso em: 26 set. 2023.

TOP Browser Market Share. SimilarWeb, 2023. Disponível em: <https://www.similarweb.com/browsers/>. Acesso em: 06 jun. 2023.

UNDERSTANDING Ambient Occlusion. PluralSight, 2022. Disponível em: <https://www.pluralsight.com/blog/film-games/understanding-ambient-occlusion>. Acesso em: 15 out. 2023.

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

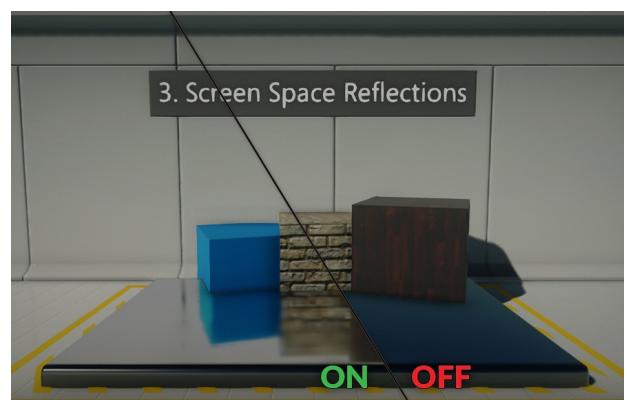
WEBGL 2.0. Can I use?, 2023. Disponível em: <https://caniuse.com/webgl2>. Acesso em: 24 out. 2023.

WHAT are Decals?. A23D, c2023. Disponível em:
<https://www.a23d.co/blog/what-are-decals/>. Acesso em: 15 out. 2023.

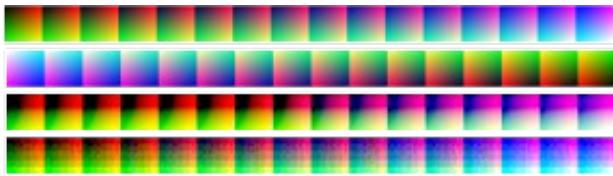
Anexos

Anexo A – Visualização dos Efeitos de Pós-Processamento









Anexo B – Visualização das Propriedades PBR

Apêndice

Apêndice A – Bibliotecas de Gráficos 3D Disponíveis

Nome	Licença	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

Apêndice B – Dados das Propriedades PBR por Biblioteca

Propriedade PBR	Descrição	Biblioteca			
		Three.js	Babylon.js	ClayGL	Playcanvas
Albedo		Mapa	Mapa	Mapa	Mapa
Metalness		Mapa	Mapa	Mapa	Mapa
Rough / Gloss		Mapa	Mapa	Mapa	Mapa
Refraction		Escalar	Mapa		Mapa
Environment/Reflection		Mapa	Mapa		Mapa
Normal		Mapa	Mapa		Escalar
Emissive		Mapa	Mapa	Mapa	Mapa
Opacity		Mapa	Mapa	Escalar	Mapa
LightMap		Mapa	Mapa		Mapa
Detail			Mapa		Mapa
Iridescence		Mapa	Mapa		Mapa
Anisotropic			Mapa		Escalar
Sheen		Mapa	Mapa		Mapa
Specular		Mapa	Mapa		Mapa
Transmission		Mapa	Escalar		

<i>ClearCoat</i>		Mapa	Mapa		Mapa
<i>Parallax</i>			Mapa		
<i>Decal</i>			Mapa		
<i>Ambient Occlusion</i>		Mapa	Mapa		Mapa
<i>Displacement</i>		Mapa	Mapa		Mapa
<i>Attenuation</i>		Escalar			Escalar
<i>Thickness</i>		Parcial	Mapa		Mapa

Apêndice C – Efeitos de Pós-Processamento por Biblioteca

Efeitos	Descrição	Biblioteca			
		Three.js	Babylon.js	ClayGL	Playcanvas
Fog	Efeito de neblina nas áreas mais distantes de cena	X	X		X
Anti-Aliasing (FXAA)	Redução de serrilhado aproximado e rápido	X	X	X	X
Anti-Aliasing (SMAA)	Redução de serrilhado mais eficaz que FXAA	X			
Anti-Aliasing (TAA)	Redução de serrilhado com estabilidade (eficiente em movimentação)	X		X	
Anti-Aliasing (MSAA)	Redução de serrilhado super eficaz.	Plugin	X		
Color Curves	Ajuste de cores como matiz, saturação e brilho	X	X	X	X
Bloom	Brilho em partes brilhantes de imagem	X	X	X	X
Vignette	Sombreamento das bordas da imagem	X	X		X
Depth of Field	Obfusca o fundo da imagem e foca os objetos da frente	Plugin	X	X	
SSAO	Uma forma eficiente de calcular o <i>ambient occlusion</i> , as sombras em detalhes da superfície de objetos)	X	X	X	X
SSR	Sutis reflexões em objetos como poças d'água	X	X	X	
Tone Mapping	Remapeia cores para valores HDR	X	X	X	X
Blur	Desfoca certos objetos	Plugin	X		

Motion Blur	Desfoca objetos em movimento		X		
Chromatic Aberration	Dispersão de cores ao longo das bordas dos objetos (falha na câmera)	Plugin	X		
Lens Distortion	Distorção da imagem pelo formato da lente	Plugin	X		
Lookup Tables	Mapear cores para outras a partir de valores pré-definidos	Plugin	X	X	

Fonte: Elaborados pelo autores a partir de AREEJ, 2023; Babylon.js; 2023; Unity, 2023.

Apêndice D – Outras Funcionalidades e Ferramentas Extras por Biblioteca

Funcionalidade	Biblioteca			
	Three.js	Babylon.js	ClayGL	Playcanvas
Sistema de Animação	X	X	X	X
Sistema de Partículas	Plugin	X	X	X
Sistema de Física		Plugin		Plugin
Sistema de Áudio	X	X		X
Gerador de Lightmaps				X
NavMesh	Plugin	X		Plugin
Sistema de GUI		X		X
Editor de Cena Gráfico	X	X		X
Visualizador de Cena / Depurador	Plugin	X		
Path Tracing	Plugin	Plugin		
Exportador de Cenas	X	X		X
Supporte a WebGL2	X	X		X
Supporte a Realidade Virtual	X	X		X

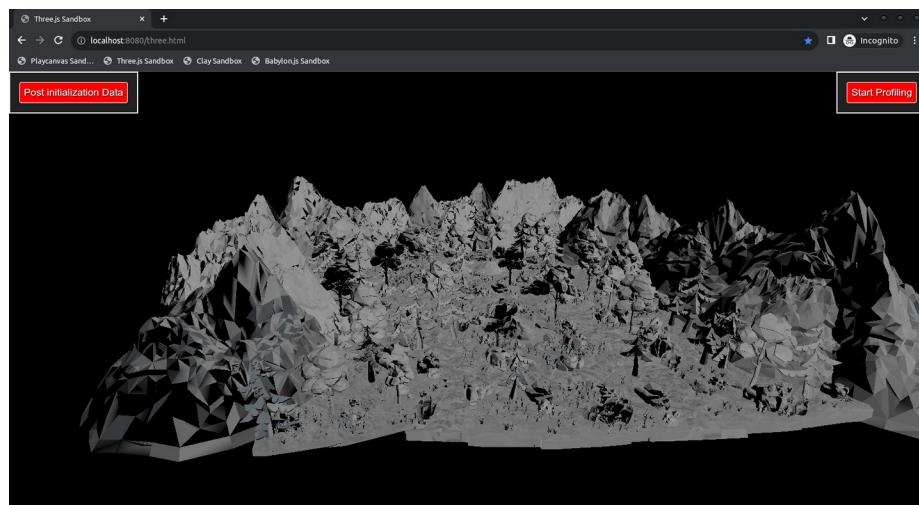
Apêndice E – Descrição das Cenas e dos Testes de Desempenho

Nome da Cena	Contagem de Triângulos	Contagem de Objetos
Forte Pirata	90.334	7
Floresta	2.076.694	1228
Floresta (Combinado)	2.076.694	1
Deserto	42.074.389	10.662

Sigla	Cena	Informações
PF	Forte Pirata	Câmera próxima e distante - 4 Luzes omni - Sombras desligadas - PC 1 e 2 - Sem efeitos
PF-C	Forte Pirata	Pós-processamento de Color Curves ligado - Câmera próxima e distante – 4 Luzes omni – Sombras desligadas – PC 1 e 2
FL	Floresta	Câmera próxima e distante – 1 Luz direcional – Sombras desligadas – PC 1 e 2 – Sem efeitos
FL-COM	Floresta (Combinado)	Câmera distante – 1 Luz direcional – Sombras desligadas – PC 1 e 2 – Sem efeitos
FL-SO	Floresta	Câmera distante – 1 Luz direcional – Sombras ligadas – PC 1 e 2 – Sem efeitos
FL-28	Floresta	Câmera distante – 28 Luzes omni – Sombras desligadas – PC 1 e 2 – Sem efeitos
FL-C-FX-S	Floresta	Pós-processamento de Colors Curves, Anti-Aliasing FXAA e Space Screen Ambient Occlusion - Câmera distante – 1 Luz direcional – Sombras desligadas – PC 1 e 2
DE	Deserto	Câmera distante – 1 Luz direcional – Sombras desligadas – PC 2 – Sem efeitos
DE-SO	Deserto	Câmera distante – 1 Luz direcional – Sombras ligadas – PC 2 – Sem efeitos
DE-4	Deserto	Câmera distante – 4 Luzes omni – Sombras desligadas – PC 2 – Sem efeitos

Apêndice F – Visualização dos Testes de Desempenho

Apêndice G – Bugs Visuais dos Teste de Desempenho



Apêndice G – Dados dos Testes de Desempenho Realizados no Mozilla Firefox

Apêndice H – Dados dos Testes de Desempenho Realizados no Microsoft Edge