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. Para análise, foi coletada uma amostra de três bibliotecas através de uma criteriosa seleção prévia, sendo elas Three.js, Babylon.js e Playcanvas. Para a comparação, foram analisados a qualidade da documentação, funcionalidades presentes, tamanho em memória, desenvolvimento do projeto e desempenho computacional. A fim de obter dados sobre o desempenho de cada biblioteca, diversos testes de *benchmark* foram realizados ao renderizar cenas 3D em vários cenários diferentes. Ao obter os dados, foi possível derivar os melhores usos para cada biblioteca.

Palavras-Chave: Computação Gráfica. WebGL. *Benchmark*. Renderização Baseada em Física. Desenvolvimento WEB.

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. For the analysis, a sample of three libraries was taken through a careful selection process, namely Three.js, Babylon.js and Playcanvas. For comparison, the quality of the documentation, functionalities present, memory size, project development and computational performance were analyzed. In order to obtain data on the performance of each library, several benchmark tests were carried out by rendering 3D scenes in various different scenarios. By obtaining the data, it was possible to derive the best uses for each library.

Keywords: Computer Graphics. WebGL. Benchmark. Physically Based Rendering. WEB Development.

* 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 atualmente 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 WEB da API gráfica multiplataforma OpenGL*,* que, funcionando nativamente no navegador, permite 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 não utilizadas 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.

Logo, 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 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, para então fazer uma análise comparativa de seus aspectos qualitativos e quantitativos, como a qualidade da documentação, funcionalidades presentes, desenvolvimento do projeto, tamanho em memória, 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, os quais serão executados no navegador, e que, com o auxílio de ferramentas de *benchmark,* fornecerão dados de uso de processador, memória RAM, 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 em seus websites.

2 – Referencial Teórico

Este referencial teórico busca explicar os principais conceitos relacionados à

computação gráfica. A seção 2.1 contém uma introdução ao tema, a seção 2.2 explica mais detalhadamente os gráficos 3D. Nas seções 2.3, 2.4 e 2.5 há a explicação de temas mais específicos relacionados às bibliotecas gráficas e aos testes que serão realizados.

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 computação gráfica 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 (criação de *designs*, como produtos, equipamentos e cidades), simulações e animações, arte e comércio, cartografia, e muitos outros. (PINHO, [s.d])

2.2 - 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 RGB. 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 (assim como o WebGL) utiliza um sistema conhecido como destro, no qual as coordenadas começam no canto 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 feito para a renderização e visualização de uma cena. As câmeras podem ser divididas em dois principais tipos, perspectiva e ortográfica. A câmera perspectiva reflete a visão do mundo real, na qual os objetos mais distantes tendem ao centro do campo de visão, enquanto a câmera ortográfica permite visualizar os objetos sem distorção e no mesmo tamanho de forma independente da distância, permitindo um melhor alinhamento dos objetos e visualização para a 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 (*Graphics Processing Unit)* 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 modelo, 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). A figura 01 demonstra este processo:

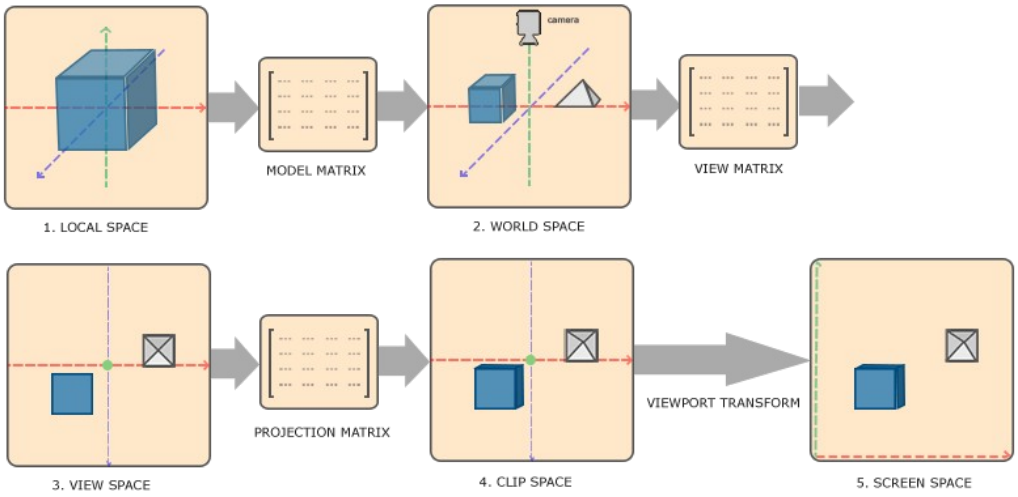

Figura 01 - Transformação de um objeto 3D nos diferentes espaços geométricos
Fonte: VRIES (2020)[1]

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 pixels que não estiverem entre os limites da tela. Outro programa, 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, pelos quais 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)

2.3 - 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, [s.d])

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

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 serve 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.

Além da malha poligonal, um objeto 3D também guarda informações sobre o seu material, constituído de atributos aplicados ao modelo para modificar como ele vai ser visto no seu estado final, deixando-o com uma aparência mais característica e dinâmica. Cada um desses atributos adicionam ou modificam aspectos básicos dos modelos, como, por exemplo: cor principal; transparência, que permite que seja possível enxergar através dele, sendo útil para simular objetos como vidro; 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 são normalmente 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, geralmente, 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)

2.3 - 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 a luz de uma cena conforme o objetivo que se deseja alcançar com ela. Algumas destas formas de iluminação são: direcional, que gera uma fonte de luz que segue uma direção fixa e ilumina igualmente todos os objetos em seu caminho; omni, que, diferente da direcional, gera uma fonte de luz em um ou mais pontos específicos que se expande em todas as direções e perde intensidade à medida que se distancia; holofote, que gera uma fonte de luz em um ponto que se expande de forma cônica em uma direção, e intensidade focada no centro; ambiente, que gera luz em uma área determinada que se expande em todas as direções, muito utilizada como luz padrão em cenas. (LEE, 2023)

O funcionamento da 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 Lighting Model*, que se baseia 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; *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 do mundo real. O PBR foca em calcular a interação da luz nas superfícies e textura dos modelos de forma fisicamente realista, como, 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 gerar um resultado mais realista. (RUSSELL, 2023)

Pelo PBR, os materiais se tornam muito mais configuráveis, fornecendo atributos além daqueles disponíveis no modelo *Phong,* na forma de um número escalar, cor ou textura, como *albedo,* equivalente ao *diffuse*, *metalness (*reflexão do ambiente), *roughness (*aspereza da superfície), *refraction (*que a controla como a luz atravessa o

objeto), *environment (*reflexão do plano de fundo), *normal (*elevação e profundidade sem afetar a geometria), *displacement (*elevação maior, afetando a geometria), *opacity (*transparência), *sheen (*simulação de tecidos ou roupas), *transmission (*transparência com reflexão), *clearcoat (*reflexão limpa em material áspero), *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). A figura 02 demonstra a diferença entre os dois métodos de renderização:

Figura 02 - Renderização Phong (esquerda) versus renderização baseada em física (direita)
Fonte: Playcanvas Blog (2014)[2]

Além da iluminação, uma cena precisa de sombras para dar maior realismo. As sombras são resultado 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 as *shadow maps*, uma técnica que renderiza a cena várias vezes a partir do ponto de vista das fontes de

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

luzes presentes (VRIES, 2020). Também é possível pré-computar os *lightmaps*, texturas que definem as partes dos objetos atingidas pelas luzes da cena, para os objetos e luzes imóveis e assim não precisar recomputar seus valores a cada quadro, obtendo grande ganho de desempenho. (PLAYCANVAS, 2023)

No entanto, no processo de geração de sombras, é muito comum que elas se tornem pixeladas ou tenham um grande serrilhado. Para resolver este problema, 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 com considerável ganho de desempenho. (BABYLON, c2023; MYERS, 2007)

Outro efeito que pode adicionar realismo a cenas é o uso da iluminação global, na 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 um resultado muito mais realista 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). A figura 03 mostra como o método de *Path Tracing* consegue capturar melhor as cores do ambiente em relação ao *Ray Tracing* comum:

Figura 03 - *Ray Tracing* (esquerda) versus *Path Tracing* (direita).
Fonte: AREEJ (2023)[3]

2.4 - Pós-Processamento, Sistema de Animação e Partículas

Também é possível, no final da renderização, adicionar efeitos de pós-processamento e assim melhorar significativamente a aparência da imagem gerada na tela. Alguns dos efeitos mais comuns são: *fog* (neblina ou névoa no plano de fundo); *anti-aliasing* (remoção dos serrilhados das bordas); *color curves* (controle de saturação, matiz, 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 à 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

3 Disponível em: https://www.hardwaretimes.com/path-tracing-vs-ray-tracing-what-is-the-difference/. Acesso em: 16 de outubro de 2023.

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 na sua hierarquia. Uma animação é construída por meio de *keyframes*, quadros-chave definidos pelo modelador, os quais serão interpolados pelo programa a fim de criar uma animação fluida. (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 movimento que se comportam de uma maneira a simular um fenômeno do mundo real, como chuva, fogo, fumaça, fluídos, neve, etc. (PLAYCANVAS, 2023)

3- Metodologia

O objetivo deste projeto é exploratório ao realizar uma comparação entre diferentes bibliotecas, utilizando de métricas qualiquantitativas. A seção 3.1 trata de explicar quais bibliotecas serão utilizadas para a comparação e a razão da escolha. A seção 3.2 explica as métricas que serão utilizadas na comparação e sua relevância na análise, enquanto a seção 3.3 demonstra as ferramentas e métodos utilizados para a coleta dos resultados.

3.1 - Escolha de Bibliotecas

A fim de realizar a comparação entre bibliotecas desenvolvidas para a geração

de gráficos 3D na WEB, primeiramente é necessário a aquisição de uma amostragem dos diferentes projetos criados para este propósito. Primeiro, será apresentada uma lista das diversas bibliotecas que podem ser encontradas em repositórios públicos, como o Github, NPM ou em um sítio próprio. Depois, serão explicados os critérios a serem utilizados para a filtragem e seleção das mesmas, e por fim, fazer uma apresentação mais detalhada das que forem escolhidas a partir dos critérios definidos.

3.1.1 - Bibliotecas Disponíveis

O quadro 01 do apêndice A contém a lista das bibliotecas de linguagem JavaScript (ou sua contraparte, o TypeScript), encontradas na WEB capazes de renderizar gráficos 3D no navegador. As colunas indicam o nome da biblioteca, sua licença, onde pode ser encontrada, linguagem em que foi desenvolvida, sua última atualização no site de repositórios Github*,* cujo valor “Ativo” indica uma biblioteca que está em constante desenvolvimento, e o número de estrelas recebidas no Github. 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 início.

Assim, as seções abaixo tratam de explicar os critérios iniciais que cada biblioteca deve 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*) com a lista de 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 algum tipo de manual ou tutorial, com trechos de código-fonte explicando o funcionamento geral da biblioteca.

3.1.2.3 - 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-fonte aberto, altamente popular, com amplo suporte, eficiente, capaz de armazenar extensa informação de modelos 3D (SMARTPIXELS, [s.d]), e será utilizado para a realização dos testes de desempenho. Dessa forma, todas as bibliotecas precisam ter compatibilidade com modelos codificados nesse tipo de arquivo.

3.1.2.4 - Funcionalidades Presentes

A renderização de gráficos 3D é realizada com diversos propósitos diferentes, e atinge diferentes domínios. 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 desenvolvidas para este propósito. As funcionalidades que as bibliotecas precisam ter para cumpri-lo são: renderização baseada em física; sistema de luzes com sombras; sistema de animação esqueletal.

3.1.2.5 – Atualização Recente

O desenvolvimento do projeto da biblioteca se refere a atualização do seu código-fonte pelos seus desenvolvedores ao longo do tempo. Uma biblioteca abandonada implica em bugs não resolvidos, ausência de suporte e ausência de novas funcionalidades sendo adicionadas. Assim, foram descartadas todas as bibliotecas que não sofreram ao menos uma atualização no ano de 2023.

3.1.3 - Bibliotecas Selecionadas

A partir dos critérios definidos acima, apenas três exemplares cumprem todos os requisitos: Three.js, Babylon.js e Playcanvas.

3.1.3.1 - Three.js

A Three.js é uma biblioteca de código-fonte aberto para a linguagem JavaScript que permite aos desenvolvedores criar cenas 3D de forma organizada e renderizá-las diretamente do navegador WEB. Sendo lançada em 2010 na plataforma Github, tem se tornado uma das bibliotecas de desenvolvimento de gráficos 3D mais populares da atualidade, sendo utilizada em jogos online, demonstrações e modelagem. Por ser muito mais fácil de utilizar que o WebGL, é uma ótima escolha tanto para desenvolvedores experientes como iniciantes. (BONSJAK, 2018; LILLY021, 2022)

Além da biblioteca gráfica, a Three.js também conta com ferramentas extras que facilitam o desenvolvimento, como o editor gráfico, disponível online, que permite carregar modelos, visualizá-los, posicionar objetos, construir uma cena e exportá-la em um formato pronto para ser carregado pela biblioteca. 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.

3.1.3.2 - Babylon.js

Babylon.js é uma *engine* 3D 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, como detecção de colisão, gravidade de cenário, câmeras orientadas, além de possuir uma documentação organizada e bem explicada. (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 editor offline desenvolvido pela comunidade, um criador de interface gráfica, um editor de nós, uma biblioteca de *assets* contendo diversas geometrias e materiais, e por fim, ainda possui o *inspector,* um painel de depuração que pode ser utilizado para visualizar todos os objetos de uma cena e suas propriedades durante a execução da aplicação. (BABYLON, c2023).

3.1.3.3 - 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. É uma das líderes de mercado em relação a *game engines* para WebGL, sendo utilizada por desenvolvedores independentes e empresas conhecidas, como King, Disney e Nickelodeon (PLAYCANVAS, 2023)

O Playcanvas, ao contrário das outras ferramentas, tem as cenas construídas primariamente pelo seu poderoso editor gráfico, disponível online, no qual é possível criar cenas, subir arquivos de modelos, sons, construir e visualizar hierarquia de animações, e controlar praticamente todos os aspectos disponibilizados pela biblioteca. Através do editor, é possível fazer download do código-fonte do aplicativo ou utilizar o sistema gratuito de hospedagem no sítio da biblioteca.

3.2 - Métricas de Software

O capítulo 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 estatísticos.

A primeira seção trata de uma métrica qualitativa, a qualidade da documentação, e as quatro seções seguintes, das métricas quantitativas.

3.2.1 - 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 a sua utilização em código. Assim sendo, será analisada a qualidade da documentação disponível para cada uma das bibliotecas avaliadas. Aspectos a serem considerados incluem: comentários e explicações da documentação API, guia e manual do usuário, demonstrações em código-fonte das funcionalidades, além da presença de projetos completos desenvolvidos como exemplos.

3.2.2 - 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 ou *bugs* identificados)*,* quantidade de projetos que utilizam a biblioteca e quantidade de contribuidores do projeto.

3.2.3 - Funcionalidades Presentes

Computação gráfica envolve muito mais do que apenas carregar modelos e dispô-los da tela do dispositivo. Uma biblioteca responsável por abstrair as funções do WebGL pode oferecer outras funções além de uma simples renderização 3D, como funcionalidades capazes de gerar gráficos mais realistas, ou também disponibilizar funções extras e customizações ao programador, permitindo-lhe que tenha melhor controle sobre sua aplicação e que desenvolva diversos tipos de aplicativos mais facilmente. As funcionalidades a serem avaliadas estão descritas a seguir.

- Configuração e propriedades dos materiais PBR disponíveis, efeitos de pós-processamento e filtros de sombras suportados, que enriquecem a riqueza dos gráficos gerados.
- Utilidades de otimização, tais como: *Instanced Rendering,* ou renderização instanciada*,* a capacidade de mandar vértices de vários objetos do mesmo tipo para a GPU e renderizá-los numa única chamada*; Mesh Merging,* ou combinação de malhas, que consiste em combinar vários objetos num único, igualmente renderizando-o numa única chamada*; Level of Details,* ou nível de detalhes*,* 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) no espaço 3D; suporte a renderização com *Path Tracing;* sistema de *NavMesh,* um sistema que define o caminho pelos quais os NPCs (*Non-player character)* podem caminhar.
- Formatos de arquivos de modelos 3D suportados 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, já que 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 inconveniente por não necessitarem desses recursos.

3.2.4 - 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. Por conseguinte, faz-se necessário avaliar a capacidade da biblioteca em aproveitar eficientemente os recursos do computador e prover uma experiência fluida ao usuário final.

Dessa forma, na análise da seguinte métrica propõe-se a coletar estatísticas que mensuram o desempenho da biblioteca avaliada: uso de CPU; memória RAM alocada; tempo de inicialização; tempo de carregamento de modelos; tempo de renderização.

Para que isso seja feito, serão desenvolvidos testes envolvendo cada uma das bibliotecas, nos quais modelos 3D em formato GLTF serão carregados para a renderização de uma cena. A fim de obter um resultado diversificado, serão renderizadas diferentes cenas, com variação na quantidade e complexidade dos objetos, na distância da câmera e na ativação ou não do sistema de sombras.

3.2.5 - 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 total ocupada 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, no qual se pode ainda aplicar algoritmos de minificação e compressão, a fim de reduzir seu tamanho.

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 sítios oficiais dos desenvolvedores da biblioteca. Na seção 3.3.2 explica-se como medir com precisão o tamanho ocupado por uma biblioteca. A seção 3.3.3 mostra as ferramentas de *benchmark* utilizadas nos testes de desempenho, seguido da seção 3.3.4 que detalha a configuração dos computadores utilizados nos mesmos.

3.3.1 - Documentação e Repositórios

Dos critérios de comparação, alguns podem ser avaliados facilmente via uma pesquisa em seus sítios oficiais. O desenvolvimento do projeto possui estatísticas diretamente disponíveis em 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 no sítio oficial da biblioteca ou também no Github.

3.3.2 - Ferramentas de Medição de Tamanho

A seguir são apresentados como calcular o tamanho de uma biblioteca utilizando três ferramentas, o gerenciador de pacotes NPM, o JavaScript *bundler* Esbuild e o algoritmo de compressão DEFLATE.

3.3.2.1 - Gerenciador de pacotes NPM

O gerenciador de pacotes NPM (*Node Package Manager)* é uma biblioteca e repositório de código-fonte aberto que permitem a fácil instalação de bibliotecas JavaScript e controlar suas dependências. (ABRAMOWSKI, 2022)

Quando uma biblioteca é baixada pelo NPM num diretório de um projeto, é gerado um subdiretório para a biblioteca e para cada dependência da mesma. 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 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 utilizados por uma aplicação e juntá-los num único arquivo, facilitando a medição do tamanho real em memória que a biblioteca utiliza. 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 conseguem 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, os navegadores, 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. O navegador Google Chrome foi selecionado como o principal para os testes 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 de 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. Na figura 10 do apêndice J pode-se visualizar o mensurador de CPU e RAM no canto inferior do painel.

3.3.3.3 - Funções JavaScript

Na coleta dos dados de tempo de inicialização e tempo de carregamento de arquivos de modelos 3D, basta utilizar a função `performance.now` no início e final dos períodos de medição, e depois tirar sua diferença em milissegundos.

Para obter o tempo de renderização, é necessário utilizar a função

`requestIdleCallback` no final de cada quadro, que recebe uma função como argumento, e que será automaticamente chamada com o valor remanescente em relação ao tempo mínimo de renderização (inverso da frequência) suportado pelo monitor. Assim, subtrai-se o tempo mínimo pelo remanescente a fim de obter o tempo real de renderização. A razão deste método é que os laços infinitos em JavaScript são realizados através da função `requestAnimationFrame`, que limita a sua frequência de chamada a do monitor, de modo que se utilizasse a função `performance.now` para calcular a diferença entre os quadros não seria possível captar valores excedentes a tal limite. Depois, repete-se o processo milhares de vezes a fim de tirar uma média precisa do resultado, e no final, divide-se 1000 pelo resultado obtido a fim de obter o FPS (quadros por segundo).

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 serão executados e terão seus dados coletados em dois computadores de configurações diferentes, sendo que o primeiro é mais antigo, enquanto o segundo é mais apropriado para aplicações gráficas modernas.

- PC-1: processador Intel Core i3-4170 CPU (4 CPUs) ~3.7GHz, de placa de vídeo Mesa Intel HD Graphics 4400 (1.5 Gib), memória RAM 3.7 GiB DDR3, sistema operacional Arch Linux Kernel 6.1-lts, e monitor 1600x900 ~60HZ.
- PC-2: processador AMD Ryzen 5 5500 (12 CPUs) ~3.6GHz, de placa de vídeo NVIDIA GeForce GTX 1660 SUPER, memória RAM 16384 MB DDR4, sistema operacional Windows 11 Home 64 bits, e monitor 2560x1440 ~75HZ.

4 – Resultados

A seguir são apresentados os dados coletados conforme explicado no capítulo anterior: a qualidade da documentação (seção 4.1), o desenvolvimento do projeto (seção 4.2), funcionalidades presentes (seção 4.3), desempenho (seção 4.4) e

tamanho da biblioteca (seção 4.5).

4.1 - Qualidade da Documentação

Todas as bibliotecas Three.js, Babylon.js e Playcanvas possuem a API Docs completa e documentada. Também possuem um amplo manual de usuário com exemplos de códigos que demonstram sua usabilidade e funcionamento, além de projetos desenvolvidos pela comunidade disponíveis no seu sítio principal. Assim, todas possuem uma documentação de alta qualidade e que pode ser utilizada como meio útil de aprendizado.

4.2 - Desenvolvimento do Projeto

A tabela 01 do apêndice B 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 contribuidores, enquanto a Playcanvas possui apenas um terço dos contribuidores da Three.js e metade dos *commits*:

Enquanto isso, na tabela 02 do mesmo apêndice observa-se a popularidade da Three.js pelo número de estrelas, contribuídores e *issues*, além do número de projetos hospedados no Github que a utilizam. Em segundo lugar fica a Babylon.js, seguida da Playcanvas.

4.3 - Funcionalidades Presentes

As funcionalidades presentes tratam daquilo que a biblioteca disponibiliza para auxiliar o desenvolvimento, sendo detalhado neste capítulo as propriedades dos materiais PBR (4.3.1), os efeitos de sombras (4.3.2) e pós-processamento (4.3.3), as utilidades de otimização (4.3.4), os formatos de arquivos 3D suportados (4.3.5) e outras funcionalidades extras (4.3.6).

4.3.1 - Propriedades dos Materiais PBR Disponíveis

O quadro 02 do apêndice C 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. No mesmo apêndice encontra-se também uma lista de figuras que demonstram como cada propriedade afeta o objeto renderizado.

É possível perceber que a Babylon.js possui todas as 23 propriedades listadas, seguida da Playcanvas, com 20, e da Three.js, com 19.

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 quebradas.

Já em relação à 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 por meio de outro sistema.

A propriedade *parallax* está disponível apenas na biblioteca Babylon.js, e é responsável por criar um efeito de profundidade e de sobressalência em superfícies irregulares. O efeito pode ser visto na figura 05:

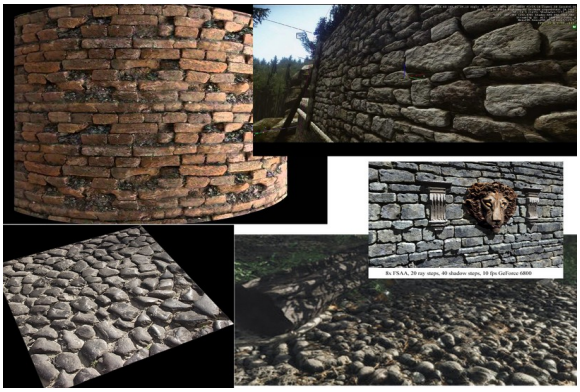

Figura 04 - Propriedade *parallax*
Fonte: Blender Forum (2019)[4]

4.3.2 - Efeitos de Pós-Processamento

O quadro 03 disponível no apêndice D contém os dados em relação a alguns dos efeitos de pós-processamento mais populares. Um campo marcado com “X” significa que 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. Campos vazios significam que a biblioteca não implementa a funcionalidade de forma alguma, mas não implicam que a funcionalidade é impossível de ser integrada através de outras bibliotecas ou implementação própria. No mesmo apêndice há também uma lista de figuras representando o funcionamento de cada um.

Observa-se que a biblioteca Three.js contêm a maior biblioteca de efeitos disponíveis com o auxílio de *plugins*, estando a Babylon.js em segundo lugar, na qual se faltam notoriamente alguns efeitos de *anti-aliasing,* os SMAA e 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*)*.

Em último lugar se apresenta a biblioteca Playcanvas, que possui a menor quantidade de efeitos de pós-processamento em relação às outras, que, além da falta dos efeitos de *anti-aliasing* mais avançados e o efeito de *blur,* destaca-se a ausência

4 Disponível em: https://blender.community/c/rightclickselect/t9bbbc/?sorting=hot. Acesso em: 22 de outubro de 2023.

dos efeitos de SSR e *depth of field*, que são capazes de mudar significativamente a ambientação de uma cena.

4.3.3 - Efeitos de Sombras Suportados

O quadro 04 do apêndice E 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 excludentes entre si, e podem ser escolhidos a depender da preferência do usuário por maior desempenho ou qualidade gráfica. Pode-se ver que a biblioteca Playcanvas possui o maior número de opções.

4.3.4 - Utilidades de Otimização

O quadro 05 (apêndice F) trata de listar algumas das técnicas de otimização de eficiência utilizadas por aplicações de computação gráfica, que podem ser empregadas em situações específicas a fim de economizar recursos de hardware ou melhorar o tempo de renderização. Nota-se que a Playcanvas não possui o LOD (nível de detalhes), mas é a única que possui a geração de *lightmaps*, o que pode acelerar bastante o desenvolvimento, já que ao utilizar as outras bibliotecas será preciso realizar esse passo num programa de modelagem 3D externo.

4.3.5 - Formatos de Arquivos 3D para Importação Suportados

O quadro 06 (apêndice G) 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 de pontos (utilizados em robótica, realidade aumentada, etc), dados de moléculas e LEGO. O último campo se refere à capacidade de carregar arquivos comprimidos pela biblioteca DRACO.

Em geral, para construção de aplicações WEB, o formato GLTF, suportado por

todas, vai ser suficiente por ser otimizado para ocupar pouca memória e ser carregado rapidamente, qualidades muito úteis para utilização em websites, mas aplicações específicas podem necessitar dos outros formatos disponíveis pela Three.js.

4.3.6 - Funcionalidades e Ferramentas Extras Presentes

O quadro 07 do apêndice H trata de listar diversas funcionalidades extras que cada biblioteca implementa e que possuem diferentes utilidades a depender do domínio da aplicação desenvolvida.

Nota-se que funcionalidades úteis no desenvolvimento de jogos, como sistema de partículas, áudio, física (através da integração com um *plugin*) 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 a 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 se mescle com os objetos 3D, sendo apenas possível criá-las pelo navegador utilizando HTML e CSS.

4.4 - Testes de Desempenho

Este subcapítulo é dedicado a demonstrar as estatísticas relacionadas aos testes de desempenho realizadas em cada biblioteca. O quadro 08 do apêndice I informa a quantidade de triângulos e objetos de cada modelo 3D, enquanto o quadro 09 explica os detalhes específicos dos testes realizados. A visualização das imagens

geradas pode ser encontrada no apêndice L.

Neste capítulo estão disponíveis as estatísticas coletadas no navegador Chrome no PC-1 e PC-2, enquanto os dados coletados no Mozilla Firefox foram descartados por apresentarem imprecisões. No apêndice N encontram-se os dados de FPS obtidos no PC-2 e no Microsoft Edge.

No teste de nome “Vale + sombra”, no qual se renderiza a cena do vale com sombras ativadas, a biblioteca Three.js não conseguiu renderizar a cena corretamente no PC-1 (figura 16 do apêndice M), razão pela qual o teste foi realizado no Brave, um navegador semelhante ao Chrome, que utiliza a mesma engine *Chromium* (KEIZER, 2021).

4.4.1 – Configuração das Bibliotecas

A fim de realizar uma comparação mais sólida, as bibliotecas serão previamente configuradas a fim de obter o máximo de desempenho possível, mas sem alterar a qualidade gráfica de uma em relação a outra. Primeiro, todas as bibliotecas tiveram o *auto clear de buffers* e o *frustum culling* (a checagem de objetos fora do campo de visão da câmera) desativados. O WebGL foi configurado com os parâmetros `{powerPreference: “high-performance”, antialias: false}`.

Também foram feitos testes utilizando a biblioteca Playcanvas tanto no pelo editor gráfico, como no modo “*engine only*”, que utiliza apenas código-fonte. Foi constatado que o primeiro modo é mais eficiente que o segundo, sendo utilizado nas comparações.

Para a Babylon.js, foi habilitado o modo de otimização agressiva, que retira algumas funcionalidades extras não utilizadas nas renderizações a fim de melhorar o desempenho geral. Também foi utilizado o otimizador de cena, cuja uma das funcionalidades é a realização automática do *Merge Meshing* das malhas poligonais que compartilham o mesmo material.

As versões das bibliotecas utilizadas foram Three.js 0.157.0, Babylon.js 6.16.2 e Playcanvas 1.66.3.

4.4.2 – Renderização e FPS

A seguir são apresentados os resultados de FPS obtidos no navegador Chrome, para todos os testes realizados no PC-1 e no PC-2, conforme o quadro 9 do apêndice I. O gráfico 01 se refere aos 9/15 testes realizados no PC-1:

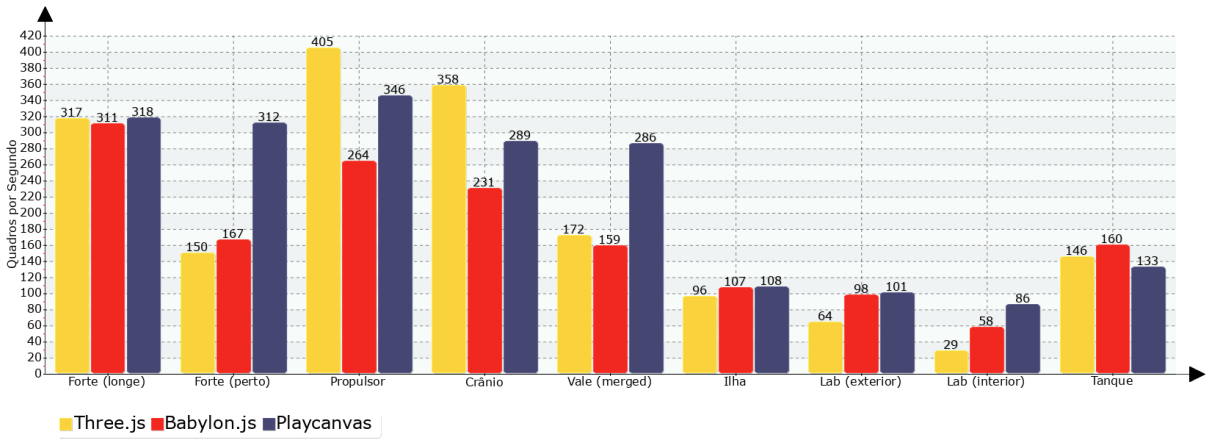

Gráfico 01 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC1 - Google Chrome – Parte 1/2

Fonte: Elaborado pelos autores (2023)

O gráfico 02 contém os últimos 6 testes do PC-1:

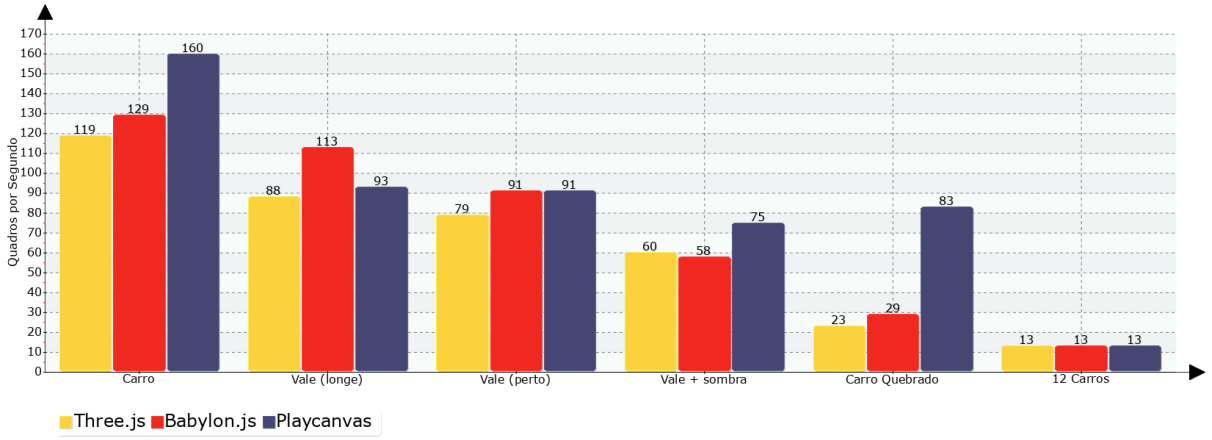

Gráfico 02 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC-1 - Google Chrome – Parte 2/2

Fonte: Elaborado pelos autores (2023)

O gráfico 03 mostra os 9/16 testes realizados no PC-2:

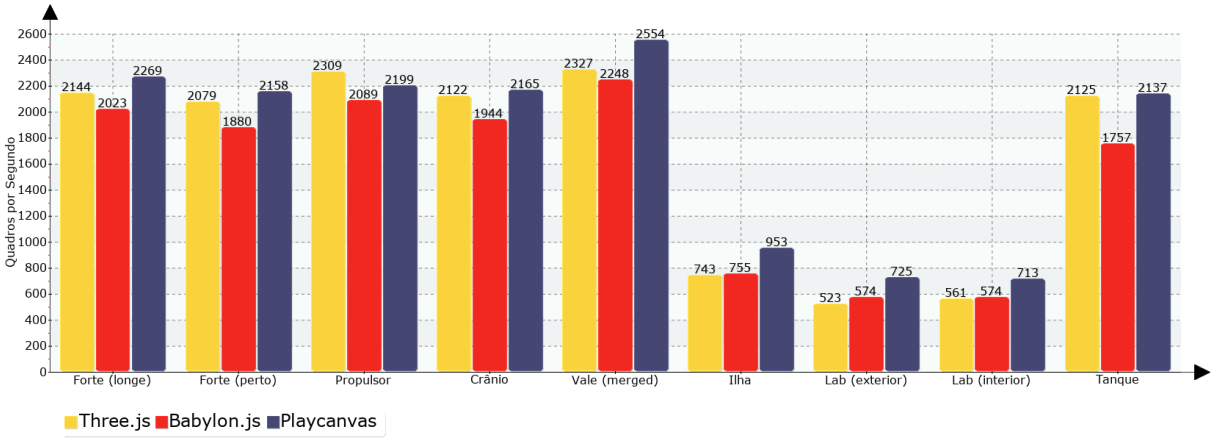

Gráfico 03 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC-2 - Google Chrome – Parte 1/2

Fonte: Elaborado pelos autores (2023)

Por fim, no gráfico 04, os 7 últimos testes realizados no PC-2. Nota-se a inclusão da cena do deserto como último teste, que, sendo a mais pesada, não pode ser renderizada pelo PC-1:

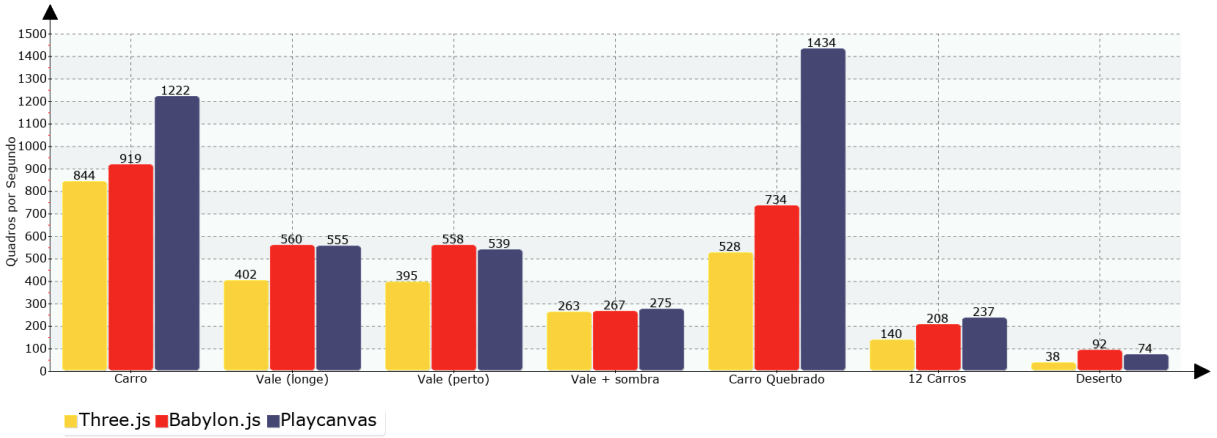

Gráfico 04 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC-2 - Google Chrome – Parte 2/2

Fonte: Elaborado pelos autores (2023)

A primeira conclusão que se pode tirar a partir dos resultados obtidos é a superioridade geral da Playcanvas perante as demais bibliotecas, estando em primeiro lugar na maioria dos testes realizados nos dois computadores, sendo que a margem de diferença se torna mais pronunciada quando se considera as cenas que contêm maior número de objetos renderizados.

A biblioteca Babylon.js fica em segundo lugar, apresentando consistentemente resultados acima da Three.js, com exceção de alguns testes de cenas com poucos objetos, como o Crânio e o Propulsor de Íons, e daqueles em que o FPS alcança valores acima de 2000, casos em que a diferença já não é mais relevante.

Quando se trata dos dados obtidos no Microsoft Edge no PC-2 (apêndice N), as estatísticas se tornam mais ambíguas, sendo que a Babylon.js passa ao primeiro lugar em boa parte dos cenários, mas ainda com uma margem de diferença menos significativa em relação a Playcanvas.

4.4.3 – 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 dos testes renderizados. O uso de CPU foi obtido pelos valores medianos observados no painel *Devtools*, enquanto o gasto de RAM, que se estabilizou após um determinado período de execução, apresentou um valor estático, sendo este número coletado como valor final. Foram utilizadas as mesmas cenas dos testes de FPS, mas sem variação da distância da câmera. Os dados de memória RAM não tendem a variar a depender do computador utilizado, e por isso foram coletados apenas no PC-2.

No gráfico 05 são apresentados os dados de uso de CPU coletados no PC-1:

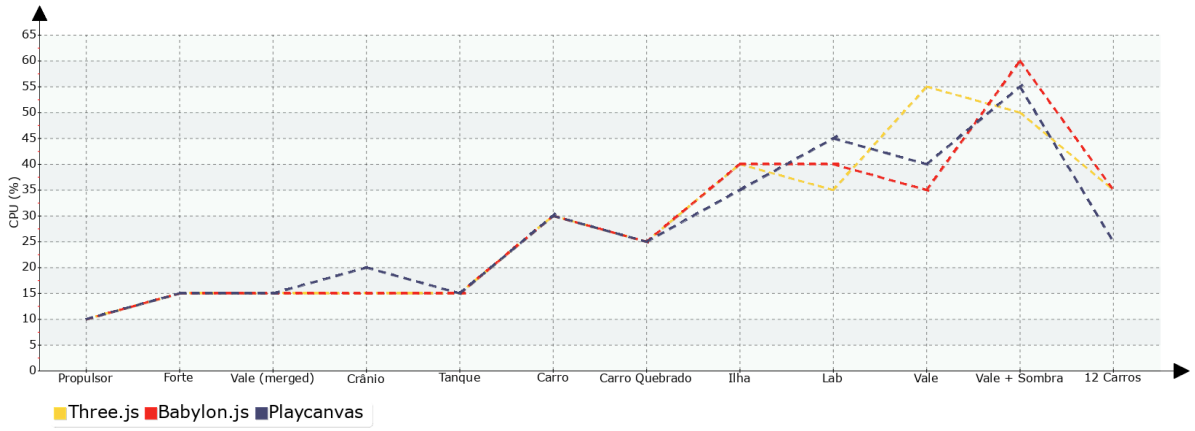

Gráfico 05 - Uso de CPU de cada biblioteca por cena renderizada – PC-1 – Google Chrome
Fonte: Elaborado pelos autores (2023)

O gráfico 06 possui os dados de uso de CPU para o PC-2:

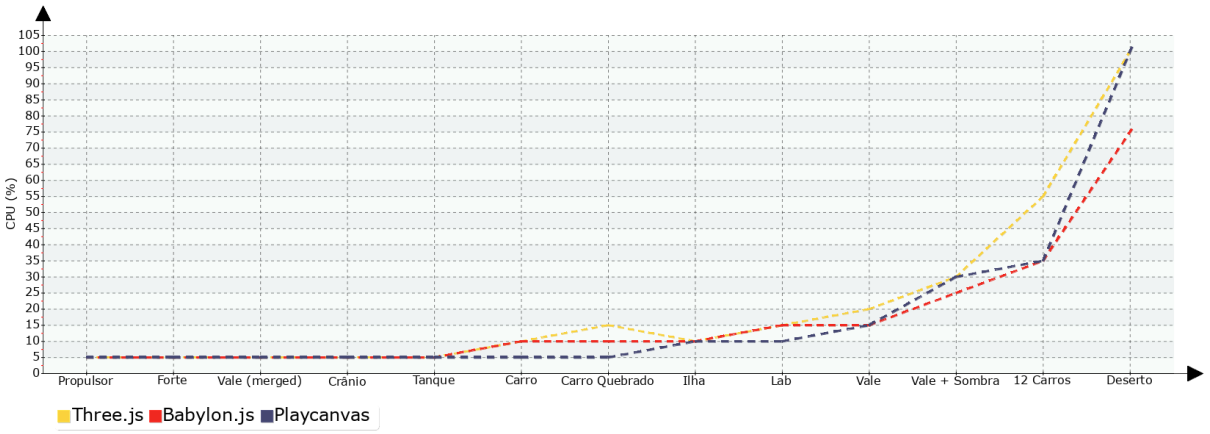

Gráfico 06 - Uso de CPU de cada biblioteca por cena renderizada – PC-2 – Google Chrome
Fonte: Elaborado pelos autores (2023)

É evidente que não houve muita diferença no uso de CPU por parte das três bibliotecas, havendo apenas pequenas margens de diferença na maioria dos testes. O que se destaca é a cena do deserto, no qual a Babylon.js foi a única que conseguiu utilizar 75% em contraste com os 100% das Playcanvas e da Three.js.

A seguir o gráfico 07, que trata do uso de memória RAM:

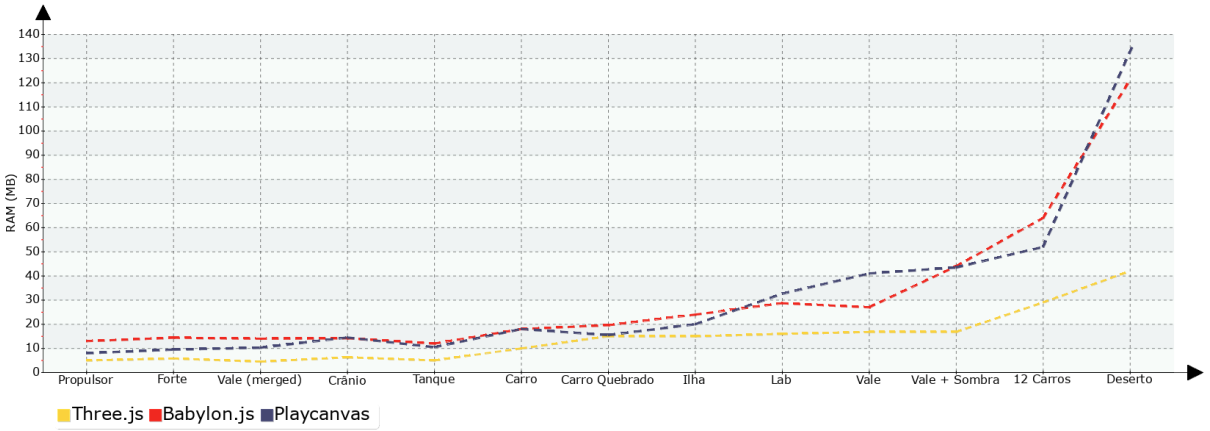

Gráfico 07 - Consumo de RAM de cada biblioteca por cena renderizada – PC-2 – Google Chrome
Fonte: Elaborado pelos autores (2023)

Ao analisar os dados, vê-se que a biblioteca Three.js possui o menor consumo de memória RAM entre elas, chegando a obter uma vantagem de até 2x ou 3x nas

cenas que possuem grande contagem de objetos. Isso provavelmente se deve aos sistemas de colisão que as bibliotecas Playcanvas e Babylon.js possuem, que as obrigam a manter os dados dos vértices do GPU também na RAM para que possam realizar os testes de colisão. Caso o desenvolvedor que utilize a Three.js também precise de um sistema de colisão, a utilização de uma biblioteca complementar possivelmente vai consumir recursos de forma a nivelar os resultados.

4.4.4 – Tempo de Inicialização e Carregamento de Arquivos

A seguir são apresentados os tempos de inicialização e de carregamento das bibliotecas. O tempo de inicialização corresponde ao tempo inicial para a biblioteca carregar no navegador, enquanto o tempo de carregamento se refere ao tempo de carregar os arquivos de objetos 3D para a renderização. Os testes foram realizados com arquivos em formato GLB (GLTF binário) múltiplas vezes a fim de obter a média dos resultados.

O gráfico 08 mostra os tempos obtidos no PC-1:

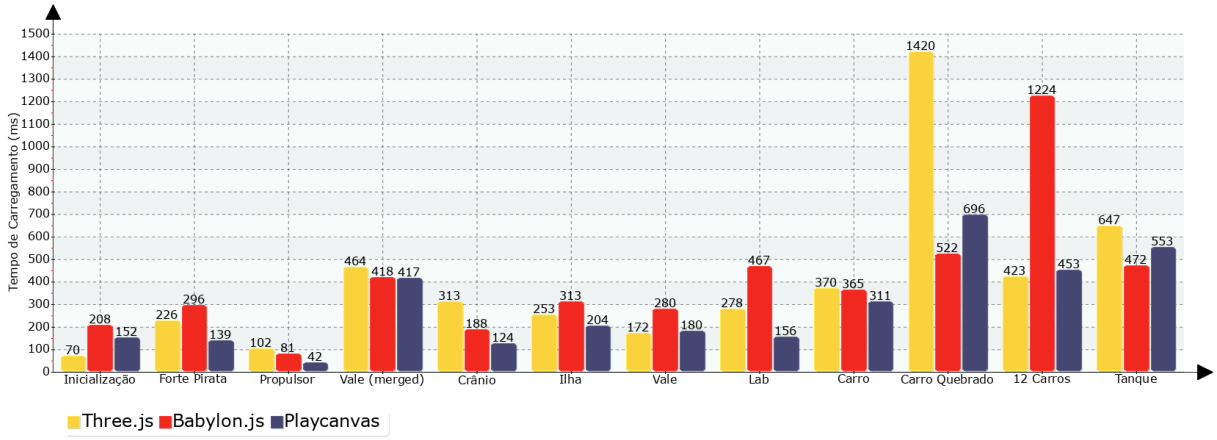

Gráfico 08 - Tempo de inicialização e carregamento de cada biblioteca por cena renderizada – PC-1 – Google Chrome

Fonte: Elaborado pelos autores (2023)

O gráfico 09 contém os tempos coletados no PC-2:

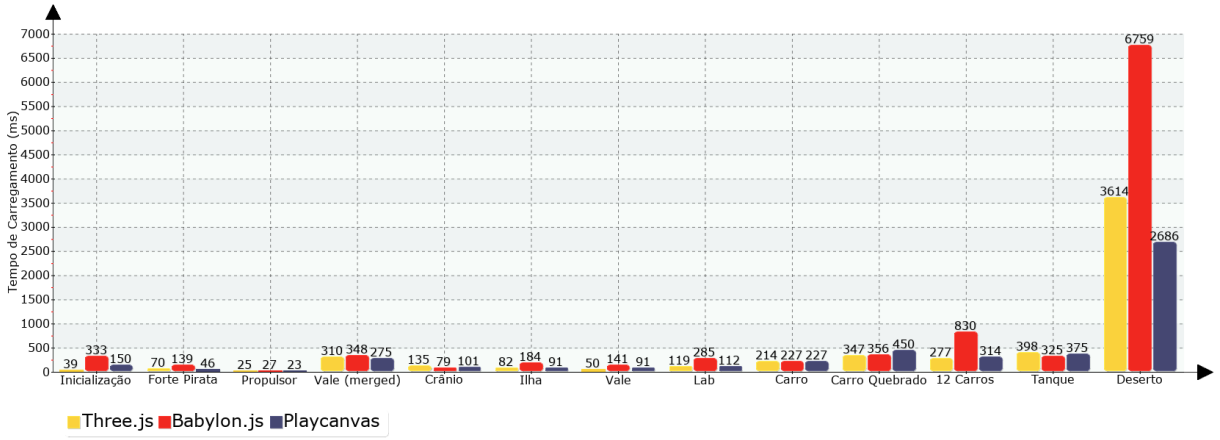

Gráfico 09 - Tempo de inicialização e carregamento de cada biblioteca por cena renderizada – PC-2 – Google Chrome

Fonte: Elaborado pelos autores (2023)

É possível ver que a biblioteca Three.js possui o menor tempo de inicialização inicial, e a Babylon.js, o maior. Quando ao tempo de carregamento, pode-se ver que há variação considerável das colocações, sendo que a Playcanvas apresentou resultados melhores no computador mais fraco, enquanto tanto a Three.js como a Babylon.js apresentaram alguns gargalos, particularmente nos modelos do Carro Quebrado, dos 12 Carros e do Deserto. Mas exceto nos casos em que a aplicação esteja constantemente carregando modelos, a diferença no tempo de carregamento não deve impactar consideravelmente a execução do programa.

4.4.5 – Problemas Encontrados

Ao realizar os testes de renderização, alguns defeitos e falhas foram encontrados, podendo ser visualizados nas figuras do apêndice M.

- Na renderização de cenas com sombras ativadas pela biblioteca Three.js, a cena ficou sem cores e repleta de artefatos visuais, conforme figura 16. Foi observado somente no navegador Google Chrome e no PC-1.
- A renderização da cena do propulsor de íons pela biblioteca Playcanvas, gerou algumas cores incorretas conforme figura 17.
- Na renderização do laboratório, ambas bibliotecas Three.js e Babylon.js geraram

estranhos artefatos na textura dos objetos, enquanto a Playcanvas produziu uma textura completamente regular (figura 18).

4.5 - Tamanho da Biblioteca

O gráfico 10 trata de mostrar o tamanho em memória de cada biblioteca. O gráfico a esquerda trata do tamanho do código-fonte, em seu tamanho total, e depois das transformações de minificação e compressão, respectivamente, enquanto o da direita trata do tamanho de download dos repositórios públicos das bibliotecas, sendo que o valor “Principal” indica o repositório principal, e o “Total”, a soma dos repositórios principais com os extras que a biblioteca oferece oficialmente:

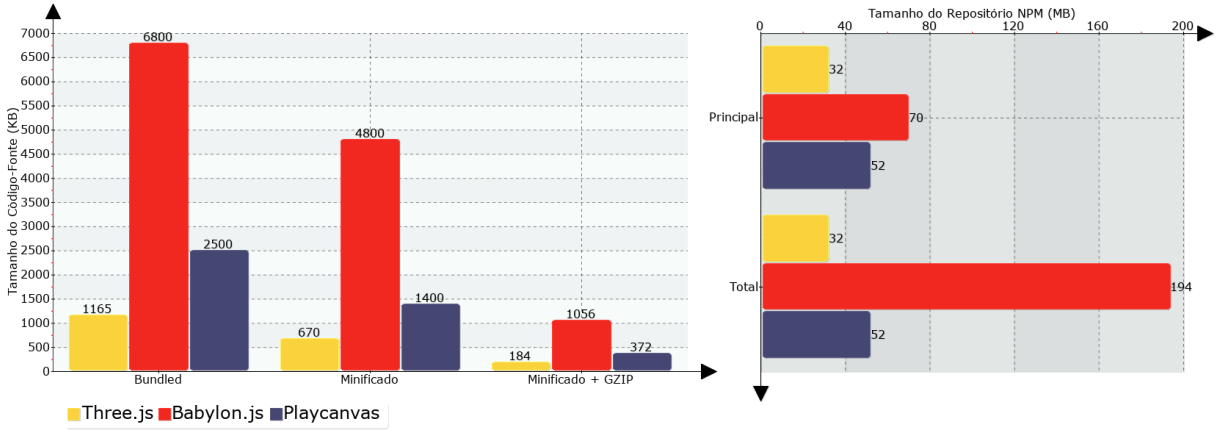

Gráfico 10 - Tamanho do código-fonte (esquerda), e dos repositórios (direita) por biblioteca
Fonte: Elaborado pelos autores (2023)

Ao analisar o tamanho do código-fonte, percebe-se que a biblioteca Babylon.js é significativamente mais pesada que as outras, o que 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 de resolução Full HD, reduzindo drasticamente tais possíveis problemas. Se há limitação na infraestrutura ou necessidade de reduzir a carga na rede do servidor, as bibliotecas Playcanvas ou Three.js são mais apropriadas.

5 – Conclusão

O objetivo deste trabalho foi executar uma análise comparativa de bibliotecas JavaScript que possibilitem ao seu utilizador integrar gráficos 3D a suas aplicações WEB com facilidade. Através das diversas métricas de software utilizadas foi possível obter um panorama das capacidades das diversas bibliotecas, e pelos testes práticos, realizados com as ferramentas de *benchmark*, foi possível comparar a eficiência computacional de cada uma delas em diversos cenários.

Pelos dados obtidos, observa-se que a biblioteca Playcanvas é a que possui melhor desempenho computacional, além de possuir muitas funcionalidades extras que servem para o desenvolvimento de jogos. Também possui algumas praticidades, sendo a única capaz de gerar *lightmaps* sem um programa externo, possui um poderoso editor gráfico e oferece hospedagem gratuita no seu website. É a mais adequada quando o objetivo se trata de obter a melhor velocidade possível.

Já a Babylon.js fica em segundo lugar no quesito desempenho, mas pode oferecer gráficos mais complexos, com efeitos de pós-processamento avançados que não estão disponíveis na Playcanvas, além de possuir mais propriedades disponíveis nos materiais baseados em física que proporcionam superior qualidade gráfica. Também possui desenvolvimento mais ativo e pode se corrigir e atualizar mais frequentemente que a Playcanvas. Como lado negativo, é a biblioteca mais pesada que impõe maior carga no servidor.

Já a biblioteca Three.js obteve os piores resultados nos testes de desempenho, mas é a biblioteca mais leve entre as três, possuindo o menor tamanho em memória e menor consumo de RAM. Também possui menos funcionalidades úteis no desenvolvimento de muitas aplicações em comparação com as outras, o que pode ser remediado com *plugins.* Conclui-se assim que *é* recomendada para enriquecer páginas WEB com efeitos visuais extras, mas menos útil no desenvolvimento de aplicações completas. Também é a única com amplo suporte a diferentes formatos de arquivo, utilizados em certos softwares de modelagem proprietários ou de uso mais específico, podendo ser utilizada quando é preciso dar suporte a esses formatos de arquivo.

Uma sugestão para trabalhos futuros é a comparação de bibliotecas desenvolvidas para WebAssembly, um padrão que permite o desenvolvimento em outras linguagens de programação nos navegadores. Pode-se também avaliar outras ferramentas como Unity e Game Maker, *game engines* que conseguem exportar uma aplicação desktop para a WEB. Outra proposta é realizar uma comparação da renderização 3D em aparelhos móveis, como smartphones ou tablets, a fim de avaliar como o desempenho da renderização 3D é impactado neste tipo de dispositivo.

Referências

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

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

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

CAMERAS. Blender, 2023. Disponível em: https://docs.blender.org/manual/en/3.6/render/cameras.html. Acesso em: 10 de outubro de 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 de outubro de 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 de abril de 2023.

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

ESBUILD: An extremely fast bundler for the web. Esbuild, [s.d]. Disponível em: https://esbuild.github.io/. Acesso em: 21 de outubro de 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 de outubro de 2023.

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

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

HISTORY of the Web. Web Foundation, c2008-2022. Disponível em: https://webfoundation.org/about/vision/history-of-the-web/. Acesso em: 16 de março de 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 de outubro de 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 de outubro de 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 de outubro de 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 de setembro de 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 de outubro de 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 de abril de 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 de março de 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 de outubro de 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 de abril de 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 de setembro de 2023.

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

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

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

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

SHAHBAZI, Nazanin. *Exploring the World of 3D Textures: A Comprehensive Guide.* Pixtune, c2023. Disponível em: https://pixune.com/blog/3d-texturing/. Acesso em: 26 de setembro de 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 de setembro de 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 de abril de 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 de setembro de 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 de abril de 2023.

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

18 de maio de 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 de setembro de 2023.

*TOP Browser Market Share***.** SimilarWeb, 2023. Disponível em: https://www.similarweb.com/browsers/. Acesso em: 06 de junho de 2023.

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

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

WEBGL 2.0. Can I use?, 2023. Disponível em: https://caniuse.com/webgl2. Acesso em: 24 de outubro de 2023.

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

Apêndices

Apêndice A – Bibliotecas de Gráficos 3D Disponíveis

Nome	Licensa	Localização	Linguagem	Última Atualização	Estrelas Github
Three.js	MIT	Github / Site / NPM	JavaScript	Ativo	94.718
Babylon.js	Apache License 2.0	Github / Site / NPM	JavaScript	Ativo	21.442
Playcanvas	MIT	Github / Site / NPM	JavaScript	Ativo	8.664
WebGLStudio.js	MIT	Github / Site	JavaScript	03/08/2020	5.036
ClayGL	BSD-2-Clause license	Github / Site / NPM	JavaScript	31/10/2021	2.679
OpenJSCAD	MIT	Github / Site / NPM	JavaScript	Ativo	2.330
Xeogl	MIT	Github / Site / NPM	JavaScript	14/05/2020	1.100
Xeokit SDK	AGPL V3	Github / Site / NPM	JavaScript	Ativo	601
Webgl-operate	MIT	Github / Site / NPM	TypeScript	12/12/2022	163
RedGL	MIT	Github / Site	JavaScript	17/08/2022	152
Zogra Render	MIT	Github / NPM	TypeScript	16/02/2023	21
CopperLicht	CopperLicht License	Site	JavaScript	-	-

Quadro 01 - Lista de bibliotecas JavaScript para renderização de gráficos 3D encontradas na WEB
Fonte: Elaborado pelos autores (2023)

Apêndice B – Desenvolvimento do Projeto por Biblioteca

Métricas Recentes	Biblioteca		
	Three.js	Babylon.js	Playcanvas
***Pull Requests* (requisitados)**	10	5	4
***Pull Requests* (aceitos)**	120	118	66
***Issues* (abertos)**	15	6	22
***Issues* (fechados)**	46	19	20
Autores	40	20	14
Commits	134	221	68

Tabela 01 - Desenvolvimento de cada biblioteca no período de um mês
Fonte: Dados obtidos no Github referente ao dias entre 5 de setembro de 2023 e 5 de outubro de 2023

Métricas Totais	Biblioteca		
	Three.js	Babylon.js	Playcanvas
***Pull Requests* (requisitados)**	145	8	21
***Pull Requests* (aceitos)**	14.627	11.371	3.692
***Issues* (abertos)**	386	93	525
***Issues* (fechados)**	11.550	2.836	1.422
Contribuidores	1.736	485	130
Commits	42.170	41.403	10.902
Estrelas	94.810	21.464	8.671
Usado Por	228.190	4.648	1

Tabela 02 - Desenvolvimento de cada biblioteca desde criação
Fonte: Dados obtidos no Github no dia 5 de outubro de 2023

Apêndice C – Funcionalidades – Propriedades PBR

Propriedade PBR	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas
Albedo	Cor base do material ao ser iluminado	Mapa	Mapa	Mapa
Ambient Occlusion	Sombras suavizadas nas partes obstruídas pela luz	Mapa	Mapa	Mapa
Anisotropic	Distorção da luz refletida		Mapa	Escalar

Propriedade PBR	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas
Attenuation	Como o objeto absorve a luz a medida que se distancia do observador	Escalar		Escalar
ClearCoat	Camada fina e brilhante em materiais ásperos	Mapa	Mapa	Mapa
Decal	Detalhes adicionais, como uma pichação na parede		Mapa	
Detail	Detalhes adicionais na textura quando próximos da câmera		Mapa	Mapa
Displacement	Elevação severa da superfície, afetando as sombras	Mapa	Mapa	Mapa
Emissive	Cor base em ambientes escuros	Mapa	Mapa	Mapa
Environment/ Reflection	Reflexão do plano de fundo (*Skybox)*	Mapa	Mapa	Mapa
Iridescence	Faixas de cores brilhantes que mudam com o ângulo	Mapa	Mapa	Mapa
LightMap	Parte iluminada do objeto, calculado a partir das fontes de luzes da cena	Mapa	Mapa	Mapa
Metalness	Reflexão dos objetos em volta, como um metal	Mapa	Mapa	Mapa
Normal	Elevação leve da superfície, sem afetar as sombras	Mapa	Mapa	Escalar
Opacity	Transparência	Mapa	Mapa	Mapa

Propriedade PBR	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas
Parallax	Efeito de profundidade em superfícies com relevos		Mapa	
Refraction	Como a luz atravessa o objeto e muda de direção	Escalar	Mapa	Mapa
Rough / Gloss	Aspereza da superfície	Mapa	Mapa	Mapa
Sheen	Simula a textura de roupas ou tecidos	Mapa	Mapa	Mapa
Specular	Ponto brilhante das luzes refletidas por um objeto	Mapa	Mapa	Mapa
Thickness	Espessura do objeto, que afeta a reflexão	Parcial	Mapa	Mapa
Transmission	Utilizada para aumentar a reflexão em objetos transparentes	Mapa	Escalar	

Quadro 02 - Funcionalidades de propriedades dos materiais PBR disponíveis por biblioteca – "Mapa" indica que é configurável por textura ou pixel, e "Escalar" por intensidade (0,0 a 1,0) ou cor RGB.

Fonte: Elaborado pelos autores a partir de informações disponíveis em Playcanvas Manual, Babylon.js Documentation e Three.js Docs (2023)

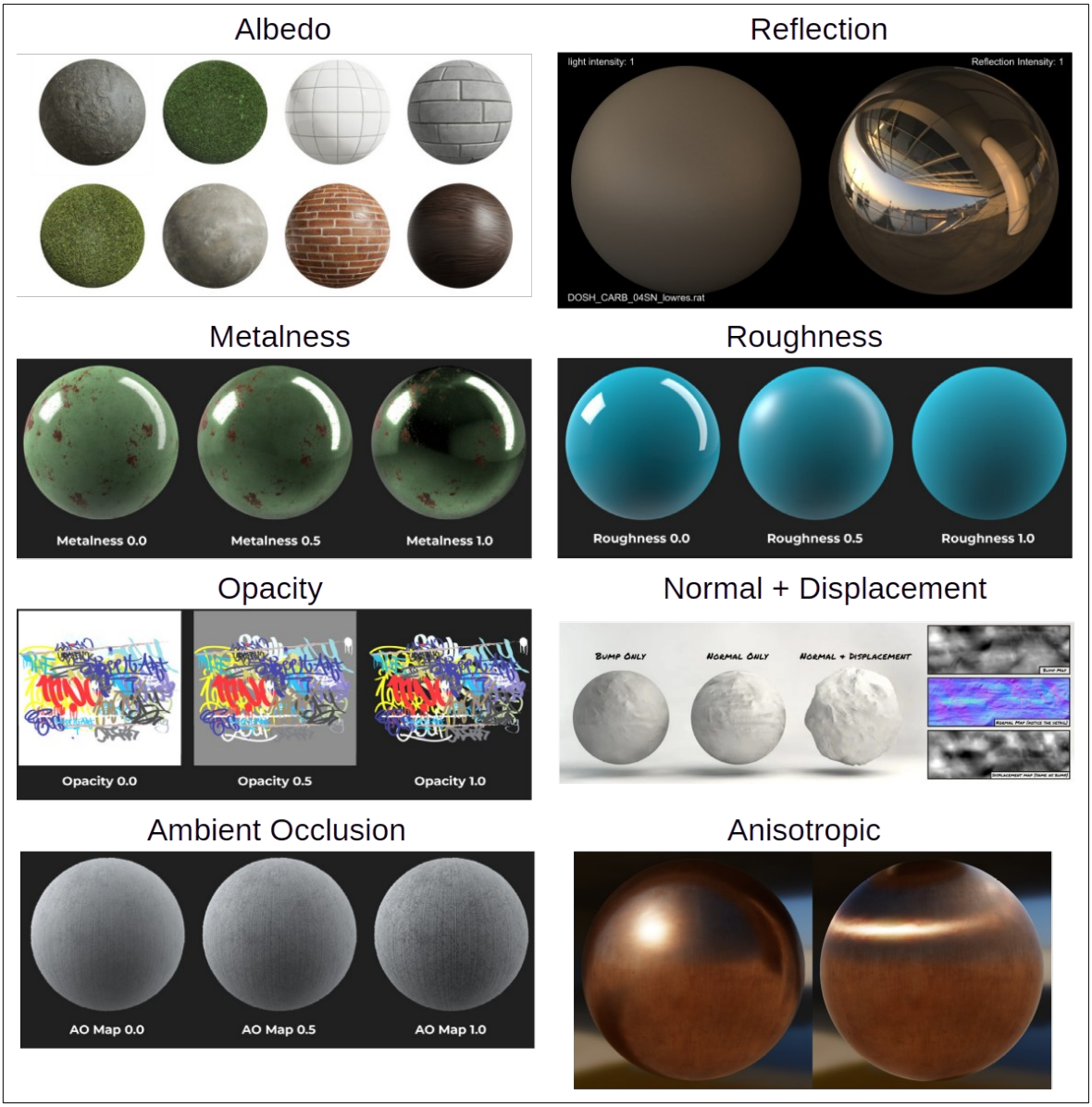

Figura 05 - Visualização das propriedades dos materiais PBR – Parte 1/3
Fonte: Compilação dos autores (2023)[5]

5 Compilado construído a partir de imagens disponíveis nos sites VISAO, SideFX, A23D, CGDirector, Filament Project e Houdini Docs.

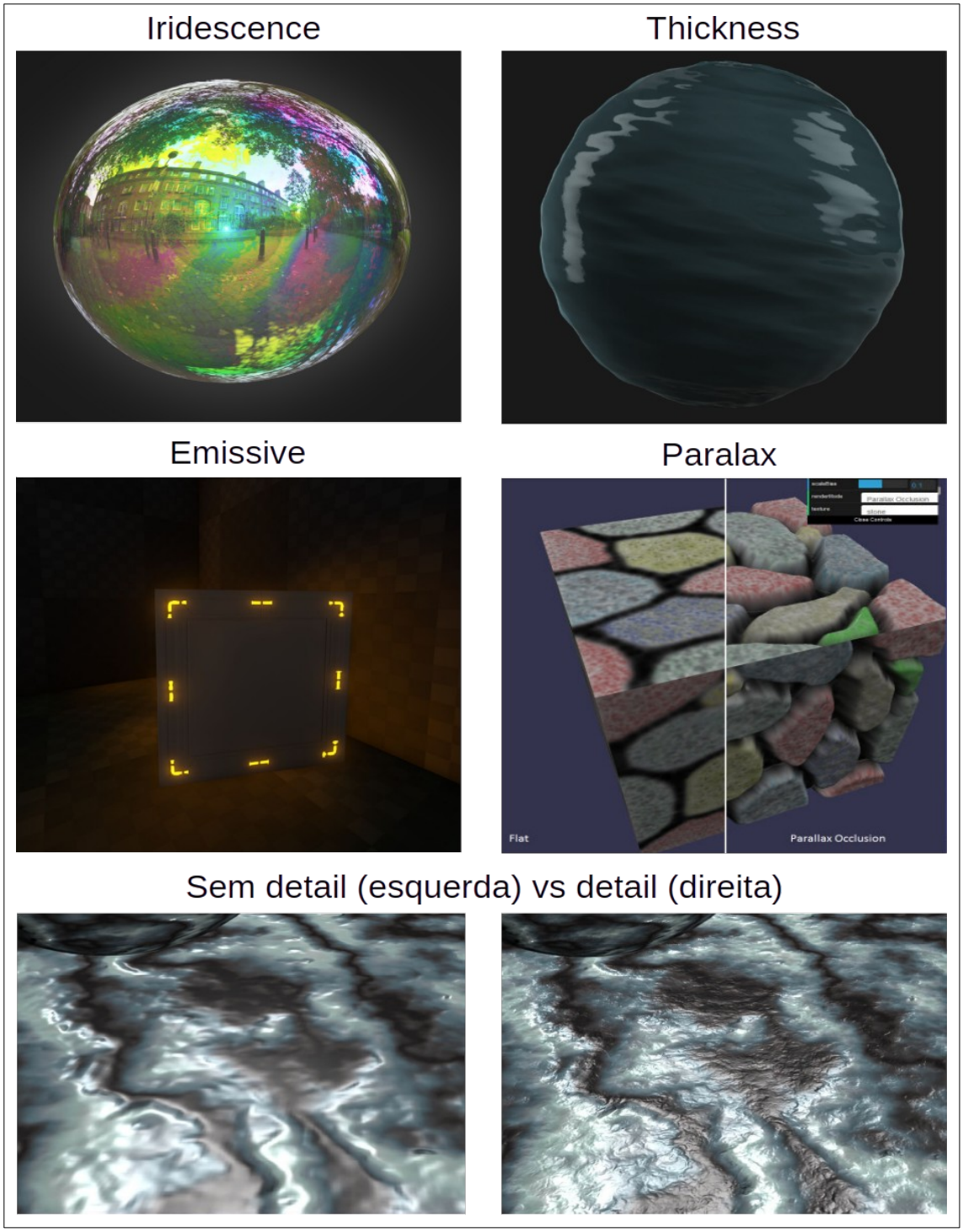

Figura 06 - Visualização das propriedades dos materiais PBR – Parte 2/3

Fonte: Compilação dos autores (2023)[6]

6 Compilado construído a partir de imagens disponíveis nos sites ArtStation, A23D, Unreal Engine Docs e Babylon.js Documentation.

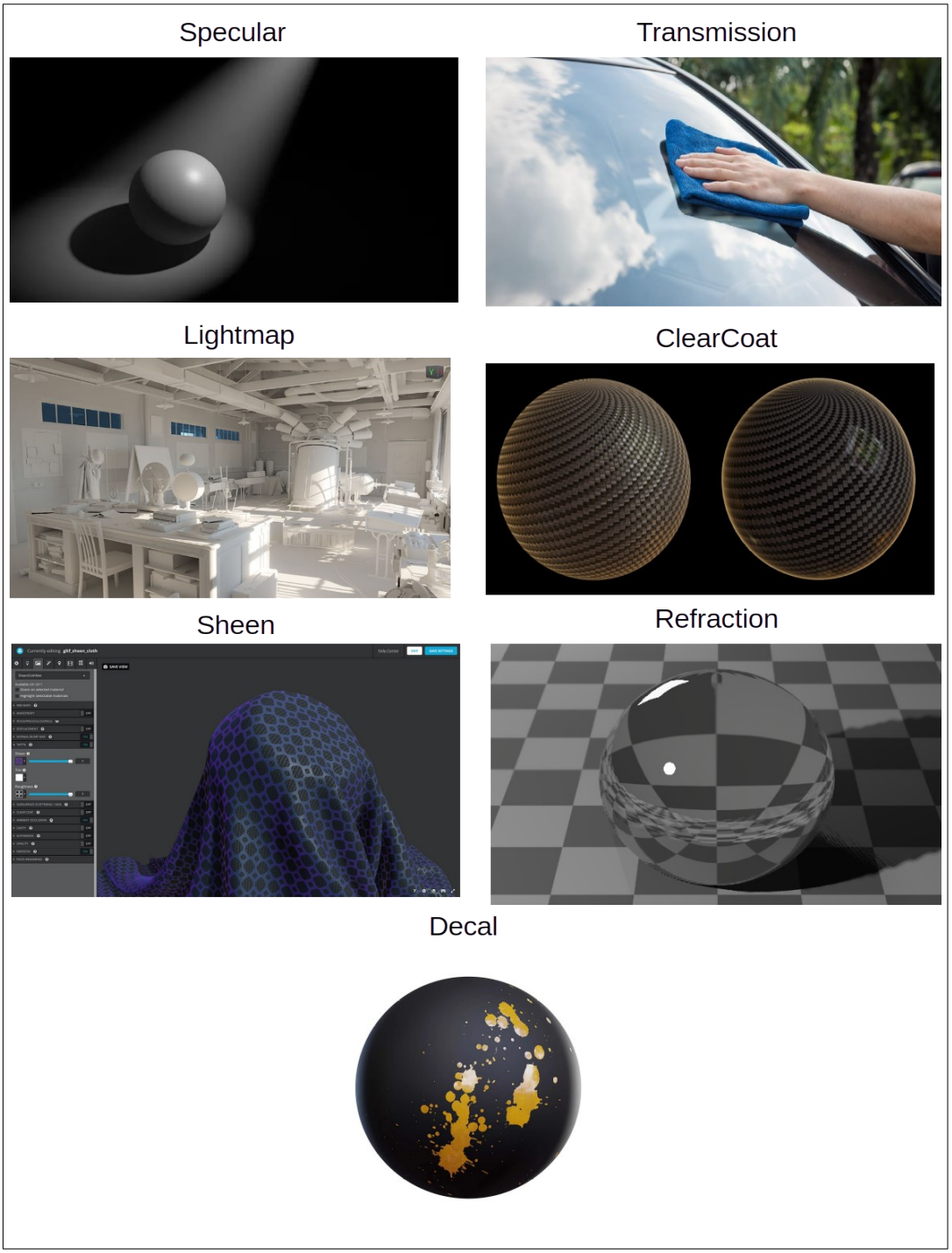

Figura 07 - Visualização das propriedades dos materiais PBR – Parte 3/3

Fonte: Compilação dos autores (2023)[7]

7 Compilado construído a partir de imagens disponíveis nos sites RayceArtist, ProAuto, SketchFab, Blender3DArchitect, Uni Engine, Kitware e CGAxis.

Apêndice D – Funcionalidades – Efeitos de pós-processamento

Efeitos	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas
Anti-Aliasing (FXAA)	Redução de serrilhado aproximado e rápido	X	X	X
Anti-Aliasing (MSAA)	Redução de serrilhado supereficaz.	*Plugin*	X	
Anti-Aliasing (SMAA)	Redução de serrilhado mais eficaz que FXAA	X		
Anti-Aliasing (TAA)	Redução de serrilhado com estabilidade (eficiente em movimentação)	X		
Bloom	Brilho em partes brilhantes de imagem	X	X	X
Blur	Desfoca certos objetos	*Plugin*	X	
Chromatic Aberration	Dispersão de cores ao longo das bordas dos objetos (falha na câmera)	*Plugin*	X	
Color Curves	Ajuste de cores como matiz, saturação e brilho	X	X	X
Depth of Field	Ofusca o fundo da imagem e foca os objetos da frente	*Plugin*	X	
Fog	Efeito de neblina nas áreas mais distantes de cena	X	X	X
Lens Distortion	Distorção da imagem pelo formato da lente	*Plugin*	X	

Efeitos	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas
Lookup Tables	Mapear cores para outras a partir de valores pré-definidos	*Plugin*	X	
Motion Blur	Desfoca objetos em movimento		X	
SSAO	Uma forma eficiente de calcular o *ambient occlusion*, as sombras em detalhes da superfície de objetos)	X	X	X
SSR	Sutis reflexões em objetos como poças d'água	X	X	
Tone Mapping	Mapeia as cores para valores HDR	X	X	X
Vignette	Sombreamento das bordas da imagem	X	X	X

Quadro 03 - Funcionalidades de pós-processamento disponíveis por biblioteca - "Plugin" necessita de biblioteca externa para provisão

Fonte: Elaborado pelos autores a partir de informações disponíveis em Babylon.js Documentation, Three.js Docs, Playcanvas API Docs e Unity 3D Docs (2023)

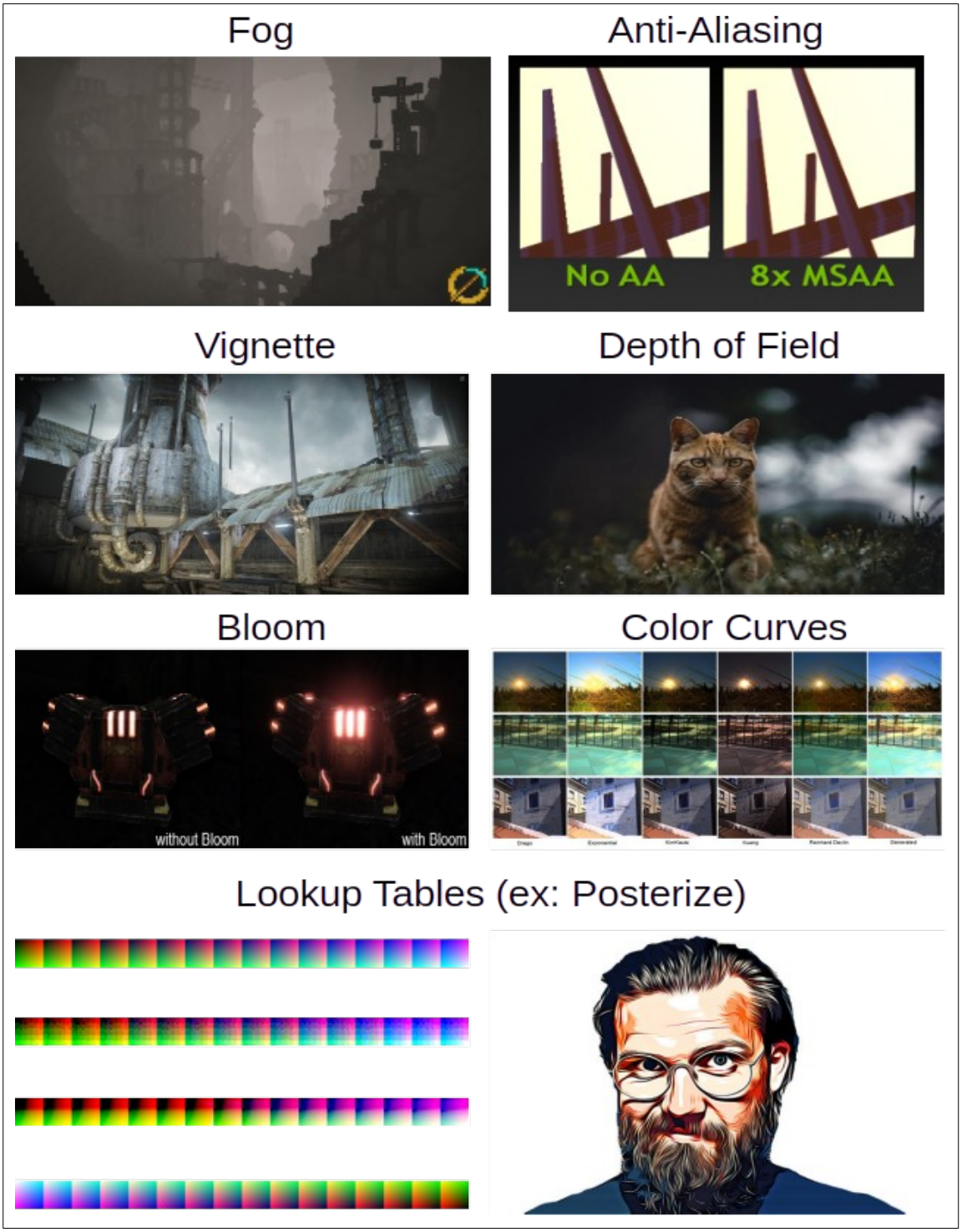

Figura 08 - Visualização dos efeitos de pós-processamento – Parte 1/2
Fonte: Compilação dos autores (2023)[8]

8 Compilado construído a partir de imagens disponíveis nos sites Babylon.js Documentation, Unreal Engine Docs, Batocera Wiki, Mc Middle Earth, LightStalking, Research Gate e Creative Fabrica.

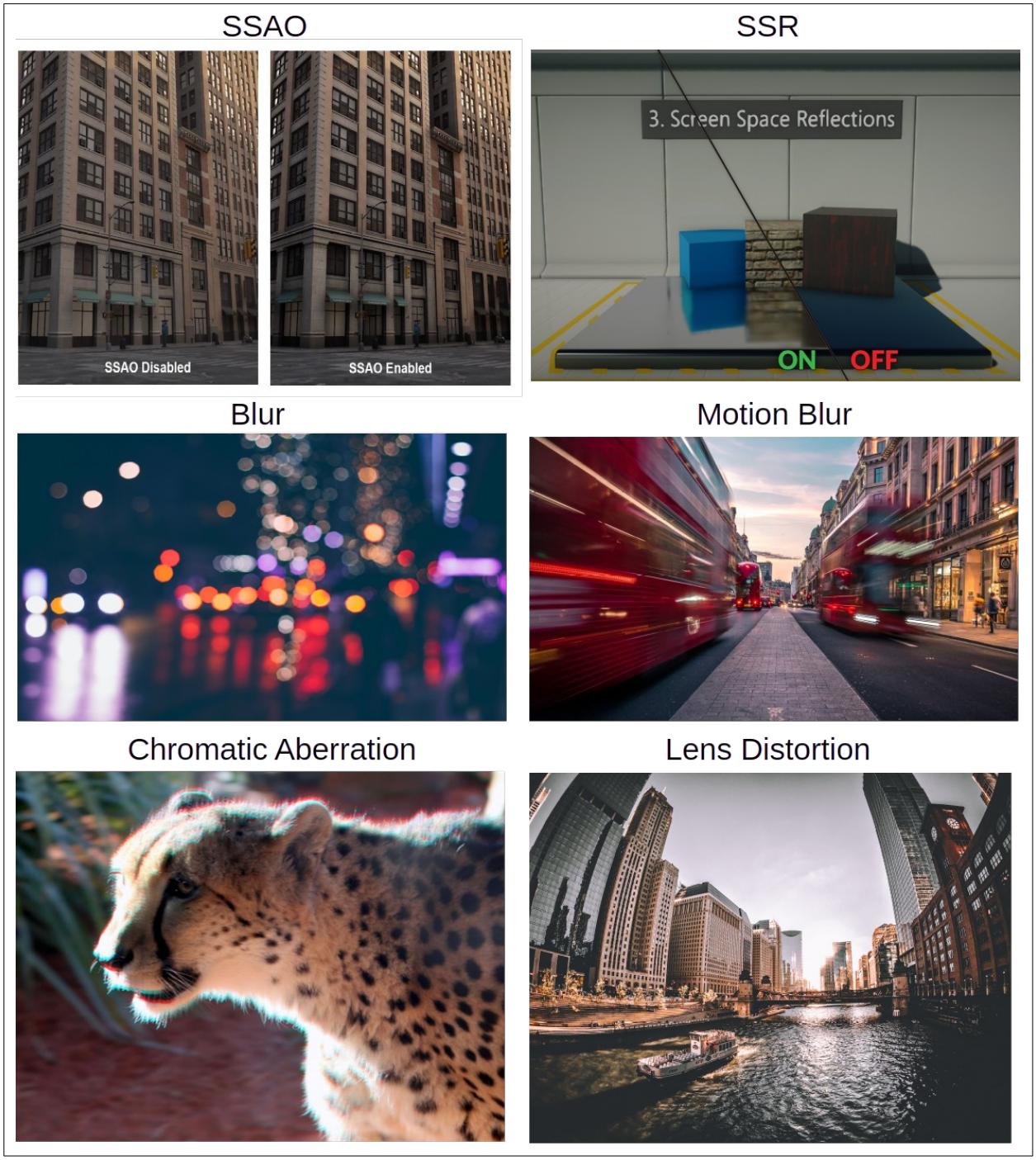

Figura 9 - Visualização dos efeitos de pós-processamento – Parte 2/2

Fonte: Compilação dos autores (2023)[9]

9 Compilado construído a partir de imagens disponíveis nos sites Epic Games Dev Community, Flax Engine Docs, Motion Array, Unsplash, Optography e Shop Moment.

Apêndice E – Funcionalidades – Filtros de Sombras

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

Quadro 04 - Suporte aos diferentes efeitos de sombra por biblioteca

Fonte: Elaborado pelos autores (2023)

Apêndice F – Funcionalidades – Utilidades de Otimização

Técnica de Otimização	Biblioteca		
	Three.js	Babylon.js	Playcanvas
Instanced Rendering	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	

Quadro 05 - Técnicas de otimização suportadas pelas bibliotecas

Fonte: Elaborado pelos autores (2023)

Apêndice G – Funcionalidades – Formatos de Arquivos 3D para Importação

Formato	Biblioteca		
	Three.js	Babylon.js	Playcanvas
GLTF / GLB (open source)	X	X	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

Quadro 06 - Formatos de arquivos de modelos 3D suportados por biblioteca

Fonte: Elaborado pelos autores (2023)

Apêndice H – Outras Funcionalidades e Ferramentas Extras por Biblioteca

Funcionalidade	Biblioteca		
	Three.js	**Babylon.js**	**Playcanvas**
Sistema de Animação	X	X	X
Sistema de Partículas	*Plugin*	X	X
Sistema de Física		*Plugin*	*Plugin*
Sistema de Áudio	X	X	X
Gerador de *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
Suporte a WebGL2	X	X	X
Suporte a Realidade Virtual	X	X	X

Quadro 07 - Funcionalidades extras disponíveis por biblioteca - "*Plugin*" necessita de biblioteca externa para provisão

Fonte: Elaborado pelos autores (2023)

Apêndice I – Descrição dos Modelos 3D e Testes de Desempenho

Rótulo	Contagem de Triângulos	Contagem de Objetos
Propulsor de Íons	52.072	6
Laboratório	83.665	608
Forte Pirata	90.334	7
Ilha	139.557	393
Crânio	1.126.158	26
Carro	1.544.351	140
Vale	2.076.694	1228
Vale (*merged*)	2.076.694	1
Tanque de Guerra	3.514.247	21
Carro Quebrado	10.390.252	206
Deserto	42.074.389	10.662

Quadro 08 - Arquivos de modelos 3D carregados nos testes
Fonte: Elaborado pelos autores (2023)

Teste	Modelo 3D Renderizado	Detalhes Adicionais
Forte (perto)	Forte Pirata	Câmera próxima – 4 Luzes omni – Sombras desligadas – PC 1 e 2
Forte (longe)	Forte Pirata	Câmera distante – 4 Luzes omni – Sombras desligadas – PC 1 e 2
Propulsor	Propulsor de Íons	1 Luz direcional – Sombras desligadas – PC 1 e 2
Lab (interior)	Laboratório	Câmera no interior do edifício – 1 Luz direcional / 14 luzes omni – Sombras desligadas – PC 1 e 2
Lab (exterior)	Laboratório	Câmera fora do edifício – 1 Luz direcional / 14 luzes omni – Sombras desligadas – PC 1 e 2
Ilha	Ilha	1 Luz direcional – Sombras desligadas – PC 1 e 2
Crânio	Crânio	Cena com Animação – 1 Luz direcional – Sombras desligadas – PC 1 e 2
Vale (perto)	Vale	Câmera próxima – 1 Luz direcional – Sombras desligadas – PC 1 e 2
Vale (longe)	Vale	Câmera distante – 1 Luz direcional – Sombras desligadas – PC 1 e 2
Vale (*merged*)	Vale (*merged*)	Câmera distante – 1 Luz direcional – Sombras desligadas – PC 1 e 2
Vale (sombra)	Floresta	Câmera distante – 1 Luz direcional – Sombras ligadas – PC 1 e 2
Carro	Carro	1 Luz direcional – Sombras desligadas – PC 1 e 2
Carro Quebrado	Carro Quebrado	1 Luz direcional – Sombras desligadas – PC 1 e 2
Tanque	Tanque de Guerra	1 Luz direcional – Sombras desligadas – PC 1 e 2

Teste	Modelo 3D Renderizado	Detalhes Adicionais
12 Carros	Carro	12 Carros clonados – 1 Luz direcional – Sombras desligadas – PC 1 e 2
Deserto	Deserto	1 Luz direcional – Sombras desligadas – PC 2
Deserto + sombra	Deserto	1 Luz direcional – Sombras ligadas – PC 2

Quadro 09 - Cenas para os testes de desempenho
Fonte: Elaborado pelos autores (2023)

Apêndice J – Painel *Devtools*

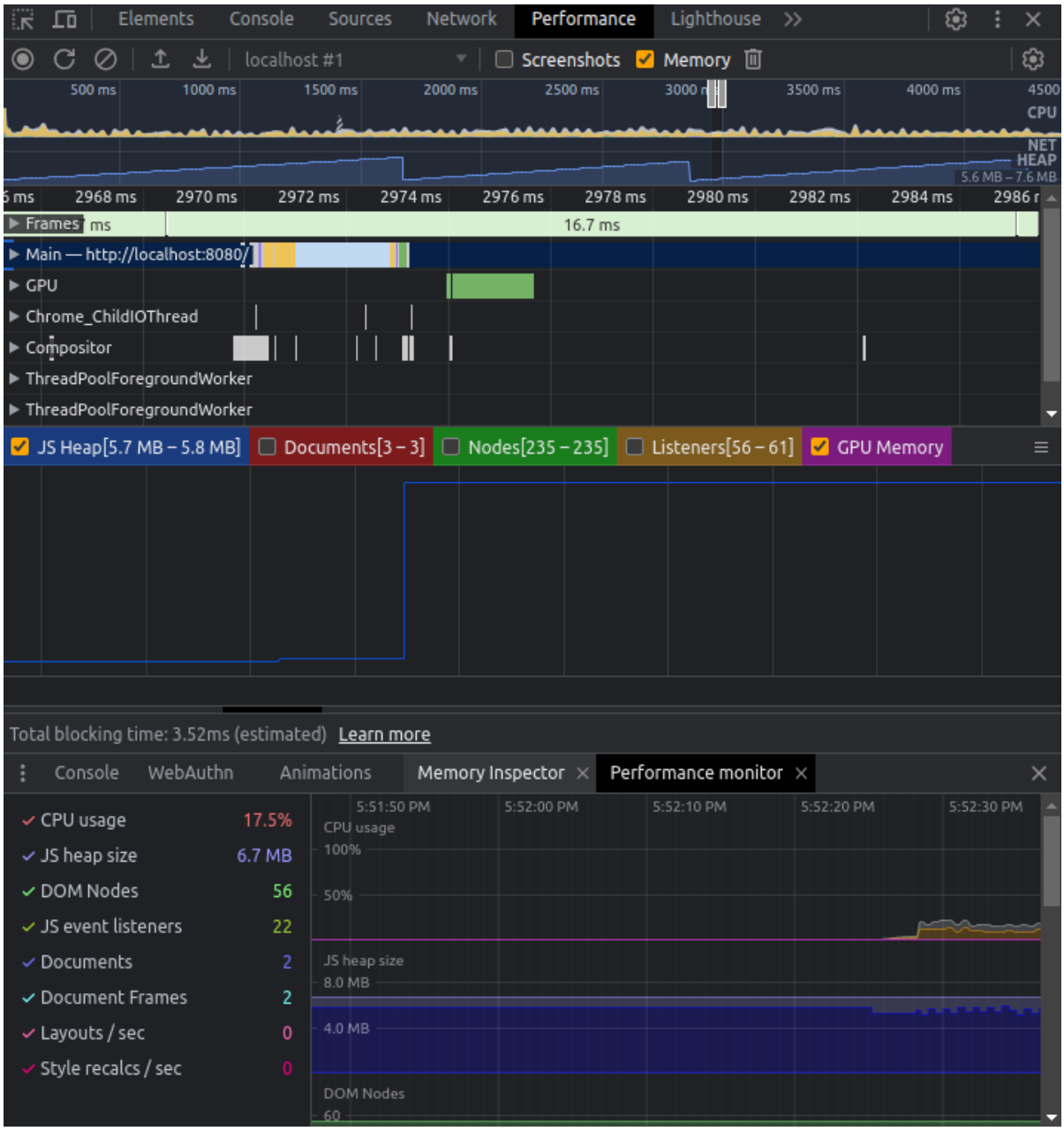

Figura 10 - Painel *Devtools* do Google Chrome
Fonte: Elaborado pelos autores (2023)

Apêndice K – Painel de Gerenciamento dos testes

Figura 11 - Interface gráfica do painel de gerenciamentos dos testes de desempenho realizados no navegador

Fonte: Elaborado pelos autores (2023)

Apêndice L – Visualização dos Modelos 3D

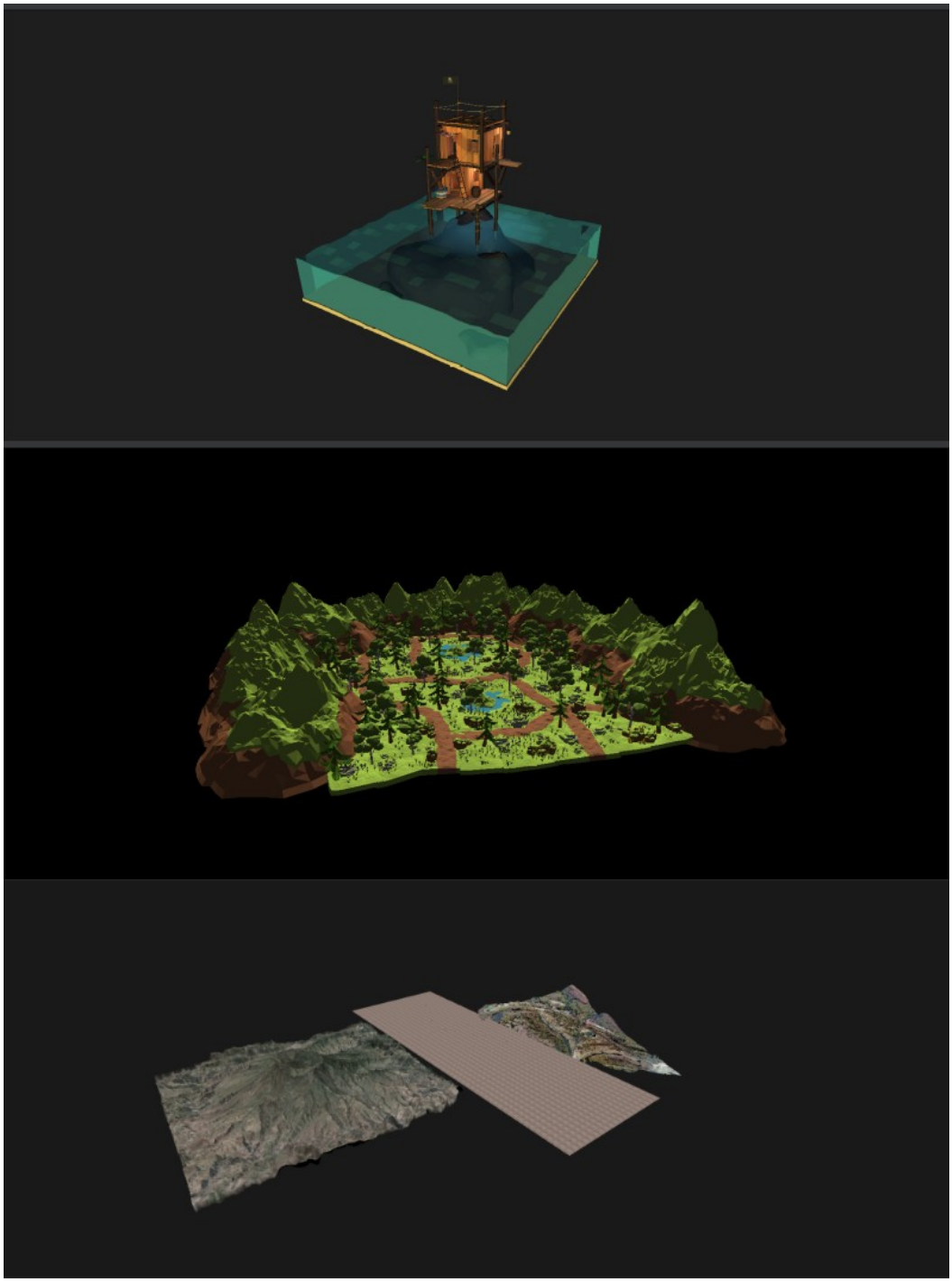

Figura 12 - Modelos 3D renderizados – Forte Pirata, Vale e Deserto
Fonte: Elaborado pelos autores (2023)

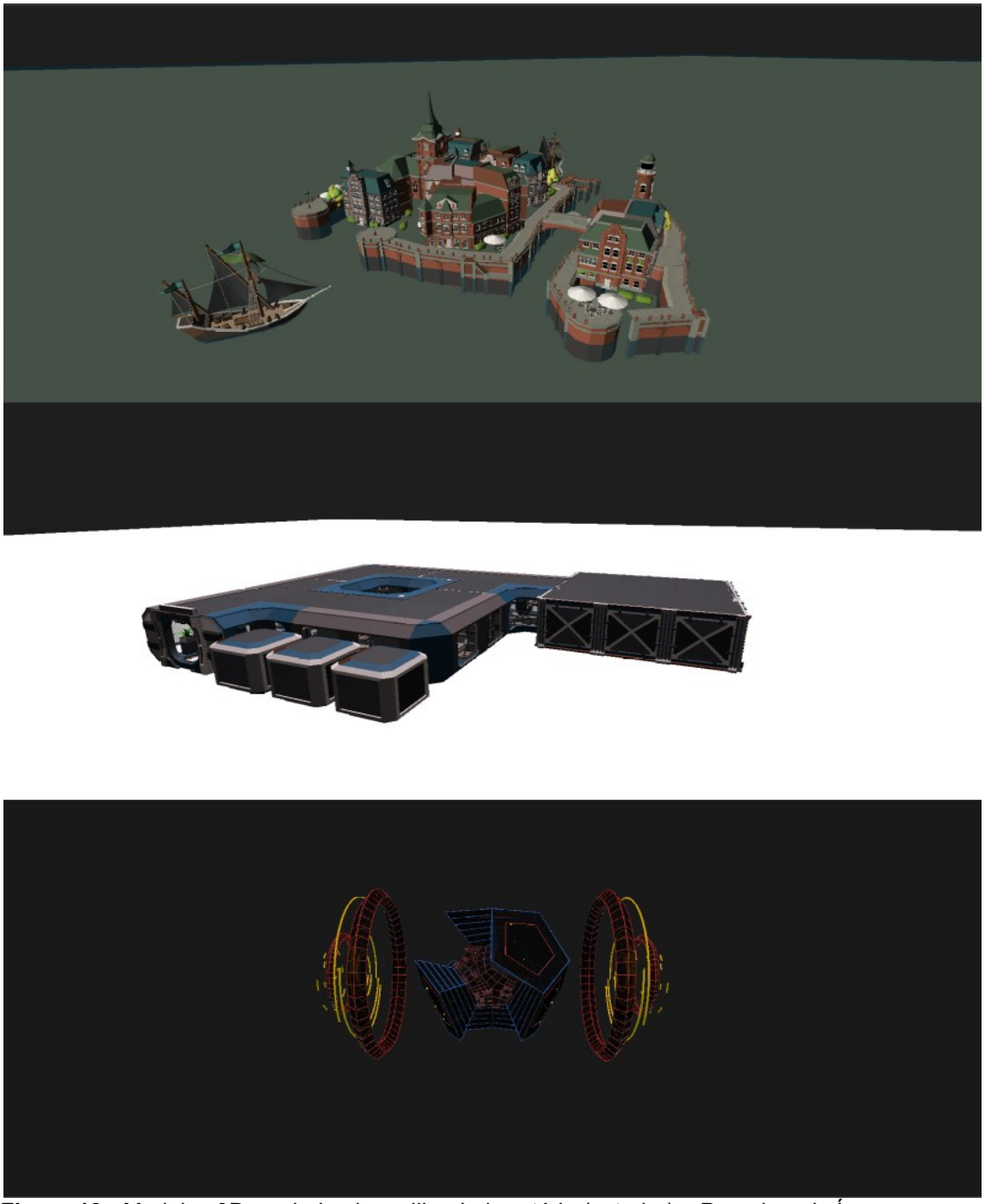

Figura 13 - Modelos 3D renderizados – Ilha, Laboratório (exterior) e Propulsor de Íons
Fonte: Elaborado pelos autores (2023)

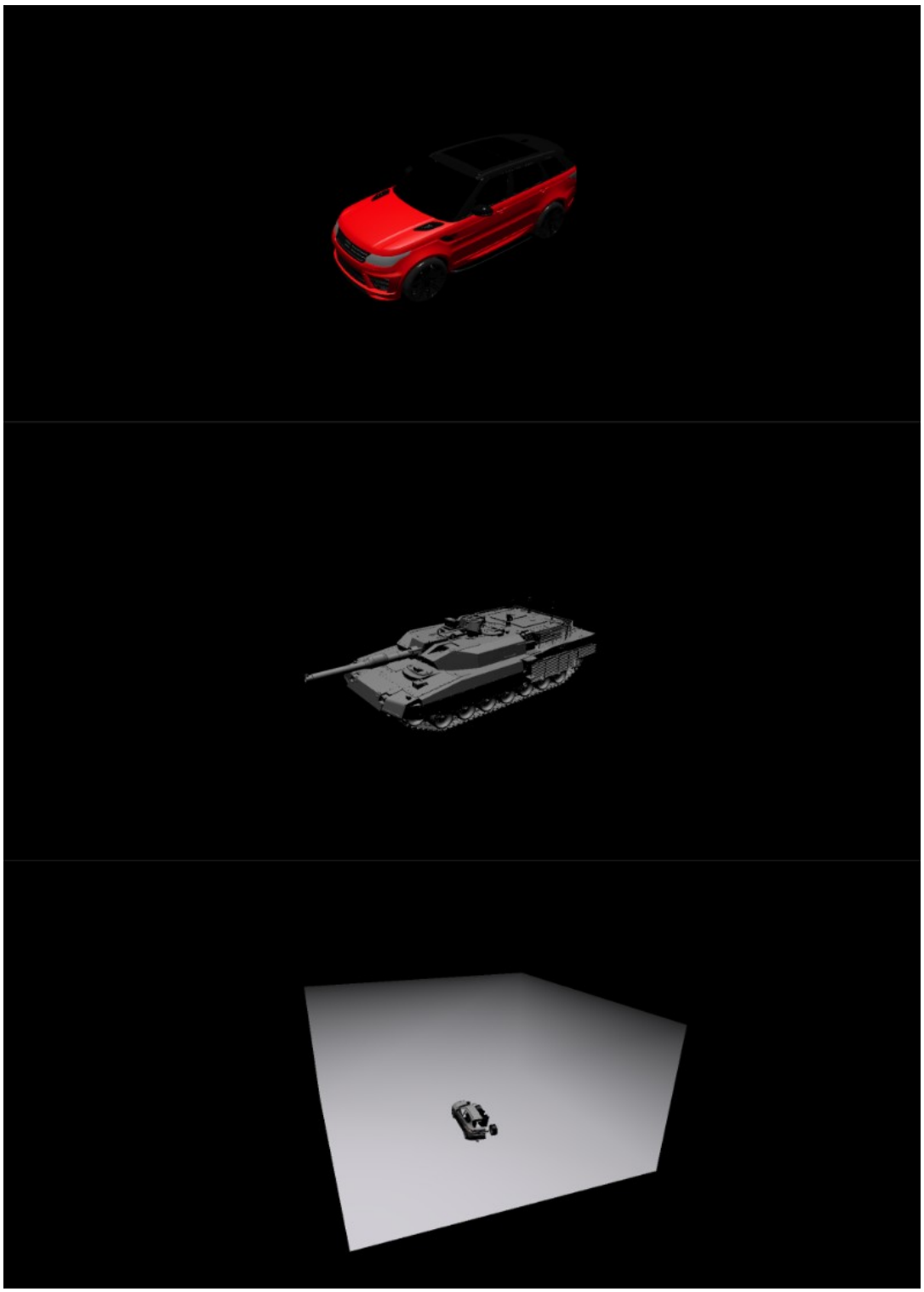

Figura 14 - Modelos 3D renderizados – Carro, Tanque de Guerra e Carro Quebrado
Fonte: Elaborado pelos autores (2023)

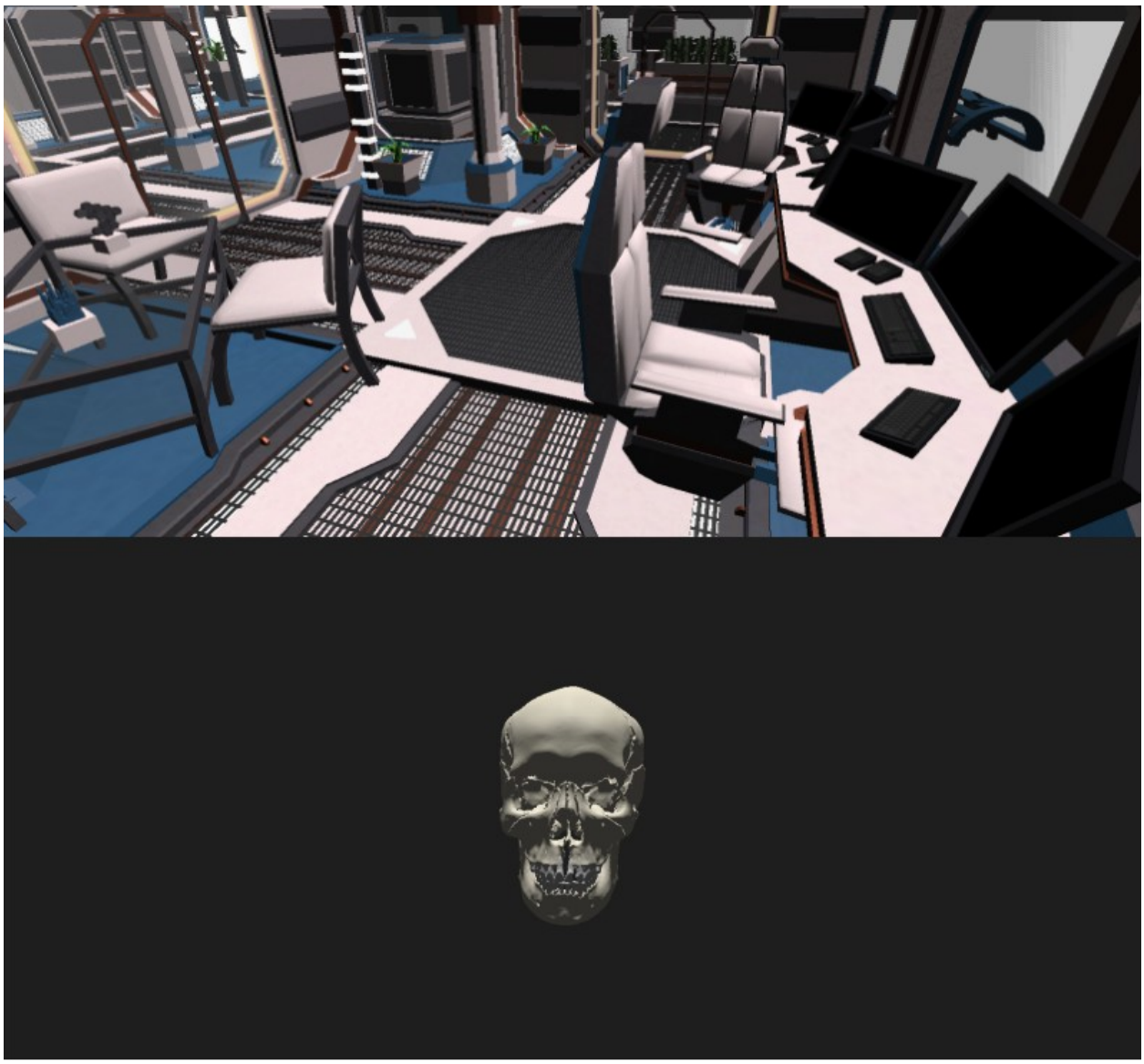

Figura 15 - Modelos 3D renderizados – Laboratório (interior) e Crânio
Fonte: Elaborado pelos autores (2023)

Apêndice M – Visualização dos Bugs e Falhas Visuais

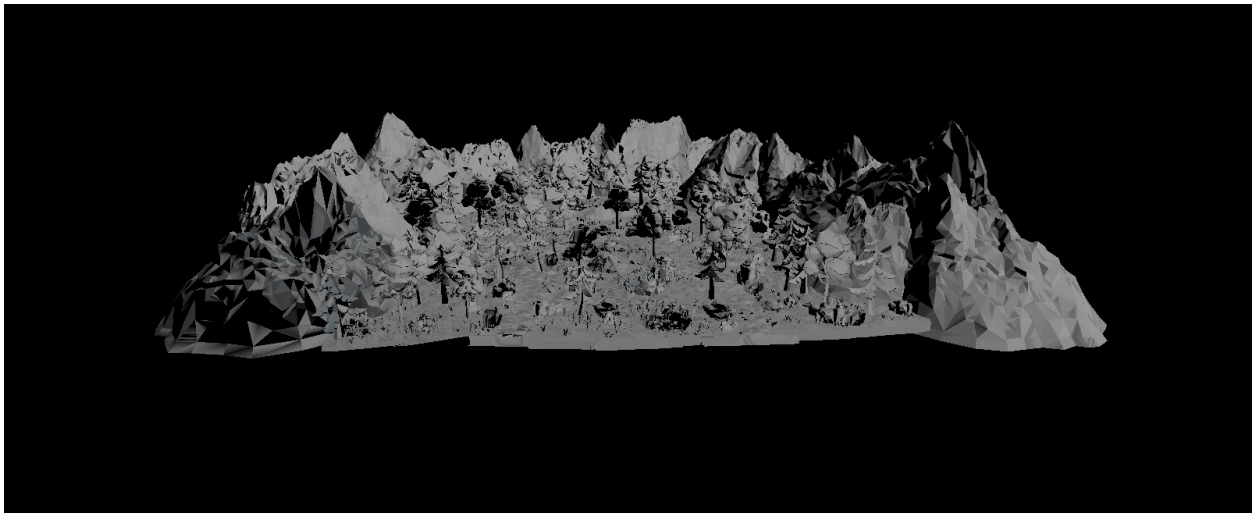

Figura 16 - Falha visual na renderização do teste "Vale + Sombra" - Three.js – PC-1 – Google Chrome
Fonte: Elaborado pelos autores: (2023)

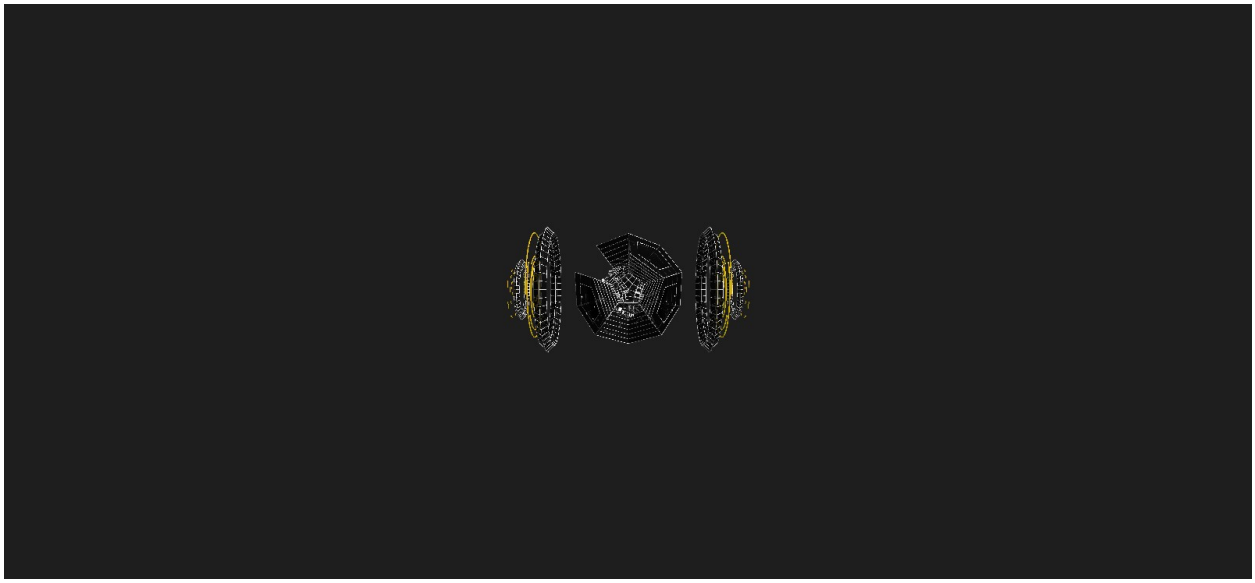

Figura 17 - Falha visual nas cores na renderização do teste "Propulsor" - Playcanvas (Editor) - PC-1 e PC-2 – Todos os navegadores
Fonte: Elaborado pelos autores: (2023)

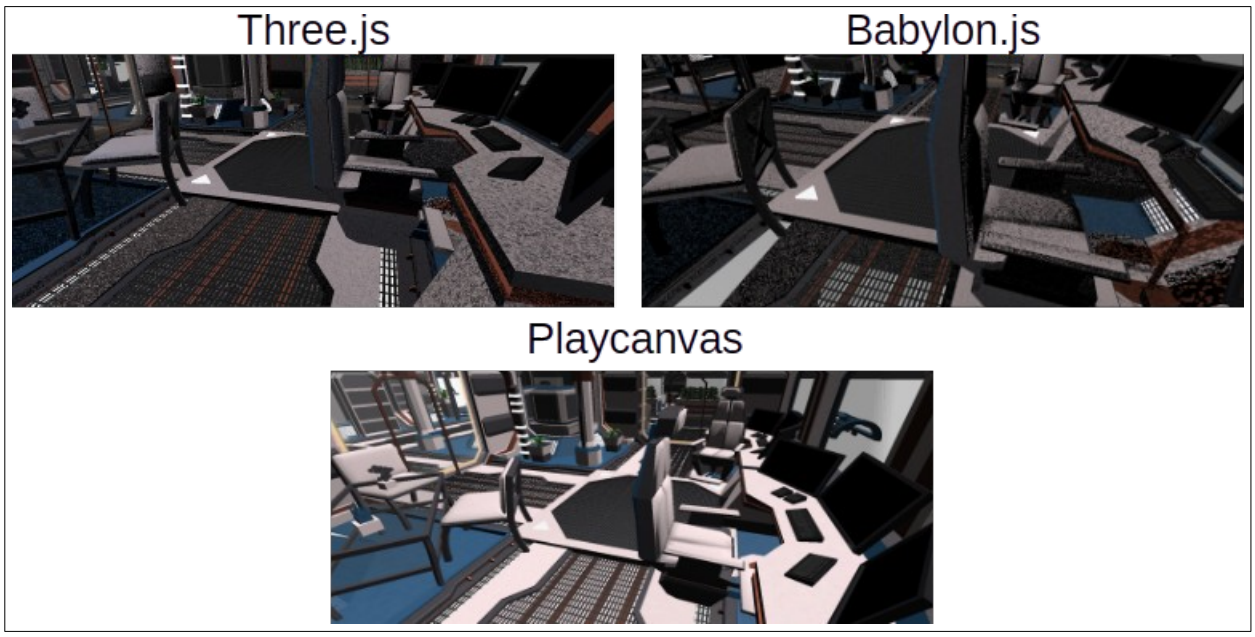

Figura 18 - A renderização do teste "Lab (interior)" apresenta estranhos artefatos quando renderizados pelas bibliotecas Three.js e Babylon.js – PC-1 e PC-2 – Todos os navegadores

Fonte: Elaborado pelos autores (2023)

Apêndice N – Dados de FPS coletados no Microsoft Edge

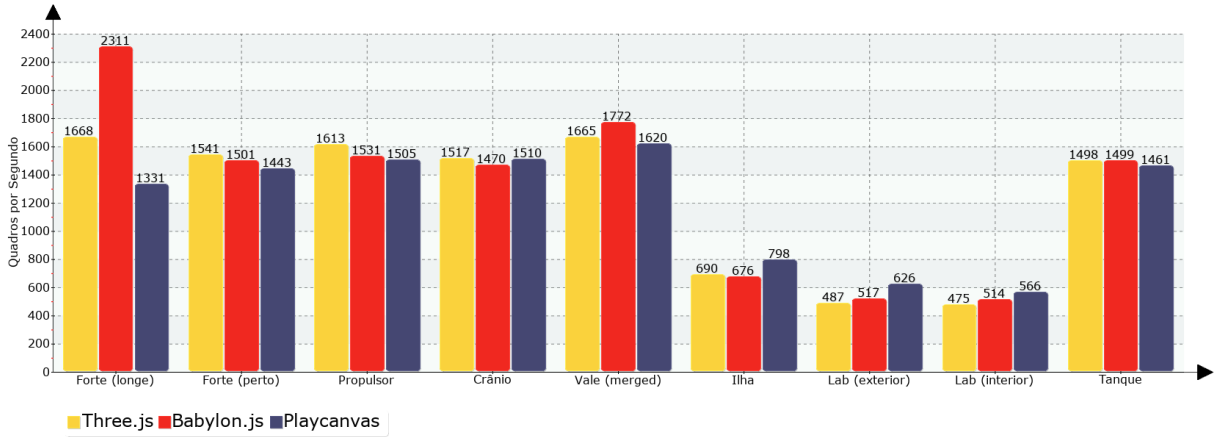

Gráfico 11 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC-2 – Microsoft Edge – Parte 1/2

Fonte: Elaborado pelos autores (2023)

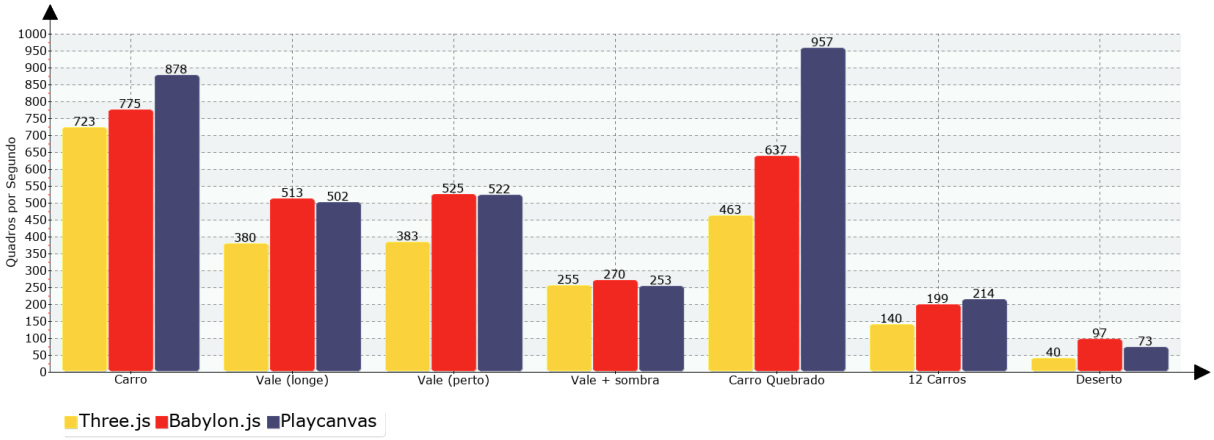

Gráfico 12 - Taxa de FPS médio de cada biblioteca por cena renderizada – PC-2 – Microsoft Edge – Parte 2/2
Fonte: Elaborado pelos autores (2023)