

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 coletado uma amostra de 3 bibliotecas através de uma criteriosa seleção prévia, sendo elas Three.js, Babylon.js, 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; Javascript; WebGL; Websites.

ABSTRACT

With the evolution of hardware technologies, websites have become increasingly complex and interactive, and today, in addition to text and images, it is possible to integrate three-dimensional graphics into browsers. The aim of this project was to compare various Javascript language libraries designed to help develop 3D applications in the web environment. For the analysis, a sample of 3 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; Javascript; WebGL; Websites.

* Rede de Ensino Doctum – Unidade Caratinga – mateusfcosta2002@gmail.com – Graduando em Ciência da Computação

** Rede de Ensino Doctum – Unidade Caratinga – aluno.paulo.martins@doctum.edu.br – Graduando em Ciência da Computação

*** Rede de Ensino Doctum – Unidade Caratinga – prof.jefte.ferreira@doctum.edu.br – Especialista em Ciência da Computação – Jefte de Lima Ferreira

1 - Introdução

Uma das áreas mais proeminentes da computação 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 da API gráfica OpenGL, funcionando nativamente no navegador, permitindo assim o desenvolvimento de gráficos 3D com aceleração de hardware em websites. (GDAD-S-RIVER, 2017)

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

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

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

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

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

Por fim, ao obter os dados e analisar as diferentes características de cada biblioteca, ressaltando seus pontos positivos e negativos, será possível tirar uma conclusão de suas diferentes especialidades e finalidades, que pode servir de ajuda aos desenvolvedores que desejam utilizar uma biblioteca gráfica em seus websites.

2 – Referencial Teórico

Este referencial teórico busca explicar os principais conceitos relacionados a computação gráfica. A seção 2.1 contém uma introdução ao tema, a seção 2.2 explica mais detalhadamente os gráficos 3D. Na seção 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 (mecânico, civil, elétrico), simulações e animações, arte e comércio, cartografia, e muitos outros. (PINHO, [s.d])

2.3 - Gráficos 3D

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

Para determinar o local nos quais os pixels serão desenhados, um sistema de coordenadas é necessário. No caso de gráficos 3D, o OpenGL utiliza um sistema conhecido como *destro*, no qual as coordenadas começam no quanto inferior

esquerdo, apontando para cima e para a direita, e a terceira dimensão, o Z, aponta para a frente da tela. (VRIES, 2020).

Para realizar as transformações de 3D para 2D, uma câmera é utilizada, que consiste num objeto invisível que permite a renderização e visualização da cena. As câmeras podem ser divididas em dois principais tipos, perspectiva e ortográfica. A câmera perspectiva reflete a visão do mundo real, no qual os objetos mais distantes tendem ao centro do campo de visão, já a câmera ortográfica permite visualizar os objetos sem distorção e no mesmo tamanho independente da distância, permitindo um melhor alinhamento de objetos e visualização para comparação de tamanhos, sendo mais utilizada na modelagem. (BLENDER, c2023)

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

O *Vertex Shader* é o programa da GPU responsável por realizar estas transformações. Primeiro, o objeto, chamado de *model*, é transformado no seu espaço local, no qual são aplicados a rotação e escalonamento. Depois, tem sua posição determinada na cena, seguido da sua posição na câmera e na tela do computador, no qual são descartadas (*Clip Space*) tudo aquilo que estiver além dos limites do ecrã do dispositivo. (VRIES, 2020). A figura abaixo demonstra este processo:

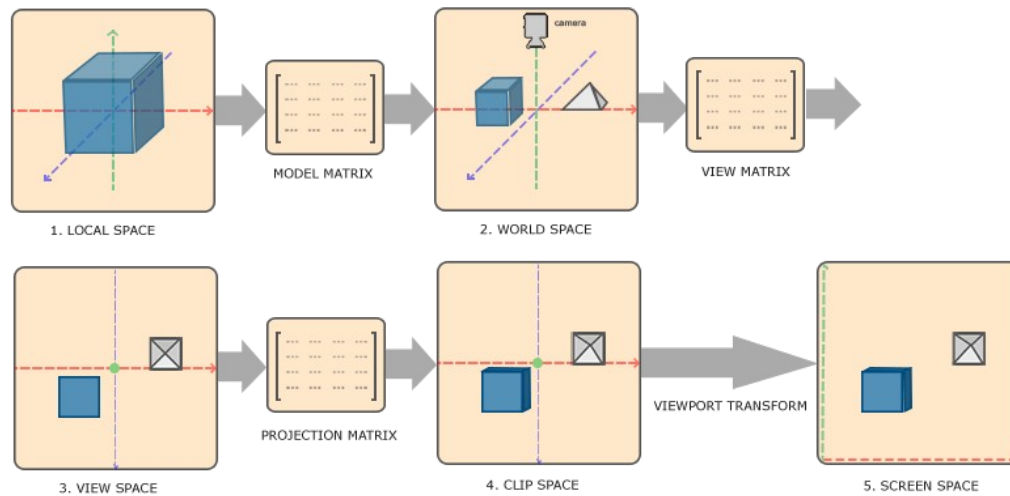


Figura 01 - Transformação de um objeto 3D nos diferentes espaços geométricos

Fonte: LearnOpenGL (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. Ainda, o *Fragment Shader* é responsável por aplicar a cor ao objeto, utilizando-se de cores, texturas e fontes de luz. Por fim, são realizados os testes de profundidade, e a partir deles pode-se destacar objetos obstruídos no plano de fundo, além de aplicar efeitos de transparência e mistura de cores. O processo de passar um objeto através dessas diversas etapas de processamento é conhecido como *Rendering Pipeline*. (VRIES, 2020)

2.4 - Geometrias e Materiais

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

¹ Disponível em: https://learnopengl.com/book/book_pdf.pdf. Acesso em: 20 out. 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 é utilizada para modelar e representar objetos tridimensionais (TIIGIMÄGI, 2021). Qualquer tipo de objeto, como pontos, linhas, cubos, esferas, edifícios, terrenos e animais pode ser construído e renderizado computacionalmente através das malhas poligonais.

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

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

2.5 - Iluminação e Sombra

A iluminação é a parte da modelagem 3D que lida com a configuração de simulação de fontes de luz em cenas tridimensionais, permitindo a visão mais clara de seus componentes. Essa é uma parte importante, pois tem grande impacto na

renderização final da cena, e uma iluminação mal feita pode fazer com que os modelos não apresentem muito bem todos os seus detalhes. (NAGHDI, 2020)

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

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

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

Pelo PBR, os materiais se tornam muito mais configuráveis, fornecendo atributos 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 aspero); *parallax* (acentuação de relevo); *decals* (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 abaixo 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 resultados da ausência de luz devido à oclusão de algum objeto, e

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

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

No entanto, no processo de geramento de sombras é muito comum que estas se tornem pixeladas ou tenham um grande serrilhado. Para resolver este problema, 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 custo de processamento; CSM (*Cascaded Shadow Mapping*), usado junto aos outros, é uma técnica de otimização; VSM (*Variance Shadow Mapping*), uma alternativa ao PCF que gera resultados semelhantes ao PCF com considerável ganho de desempenho. (BABYLON, 2023; MYERS, 2007)

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

demonstra como o método de *path tracing* consegue capturar melhor as cores do ambiente em relação ao *Ray Tracing* comum.

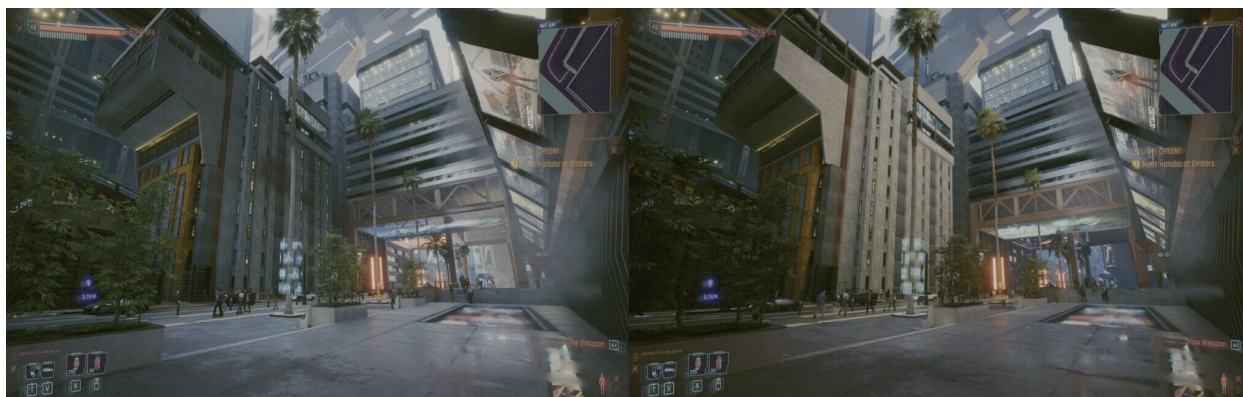


Figura 03 - *Ray Tracing* (esquerda) versus *Path Tracing* (direita).

Fonte: Hardware Times (2023)³

2.6 - Pós-Processamento, Sistema de Animação e Partículas

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

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

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

Uma cena gráfica também pode conter um plano de fundo conhecido como *skybox*, uma técnica de criação de cenas 3D com o intuito de melhorar sua ambientação e diminuir o custo de renderização. Ela consiste em criar uma caixa texturizada ao redor da cena para criar um plano de fundo do ambiente à cena sem ter que modelar e renderizar grandes áreas, melhorando o desempenho e velocidade de renderização da cena. (Yana Krasnolutska, 2023)

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

3- Metodologia

O objetivo deste projeto é exploratório ao realizar uma comparação entre diferentes bibliotecas, utilizando de dados quali-quantitativos. A seção 3.1 trata de explicar quais bibliotecas serão utilizadas para a comparação e o porquê. A seção 3.2 explica as métricas que serão utilizadas na comparação, sua relevância na análise e a seção 3.3 demonstra as ferramentas e métodos utilizadas para a coleta dos resultados.

3.1 - Escolha de Bibliotecas

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

3.1.1 - Bibliotecas Disponíveis

O quadro do apêndice A trata de listar e descrever as bibliotecas de linguagem Javascript encontradas na Web feitas para a renderização de gráficos 3D no navegador. As colunas indicam o nome da biblioteca, sua licença, linguagem em que foi desenvolvida, e sua última atualização no site de repositórios Github, cujo valor “Ativo” indica uma biblioteca que está em constante desenvolvimento. São elas: Three.js, Babylon.js, RedGL, WebGL-Operate, Zogra Renderer, ClayGL, Xeogl, Litescene, Playcanvas, CopperLicht, Xeokit SDK e OpenJSCAD.

3.1.2 - Critérios de Seleção

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

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

3.1.2.1 - Código-fonte Aberto

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

3.1.2.2 - Documentação Disponível

A documentação de uma biblioteca pública é necessária para que os desenvolvedores possam utilizá-la corretamente. A API (*Application Programming Interface*) listando todos os módulos, classes e funções de cada biblioteca deve estar disponível com comentários a fim de que suas funcionalidades e uso sejam claros para o desenvolvedor. Também deve haver 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-aberto, altamente popular e com amplo suporte, eficiente, capaz de armazenar ampla informação de modelos 3D (SMARTPIXELS, [s.d]), será utilizado para a realização dos testes de desempenho. Dessa forma, todas as bibliotecas precisam ter compatibilidade com modelos codificados nesse tipo de arquivo.

3.1.2.4 - Funcionalidades Presentes

Na renderização de gráficos 3D, existem diversos domínios com propósitos diferentes. Uma vez que o seguinte trabalho trata de selecionar bibliotecas feitas para a renderização de gráficos realísticos em tempo real, é necessário utilizar as bibliotecas que foram desenvolvidas para este propósito. As funcionalidades necessárias para

cumpri-lo são: sistema de renderização baseado em física, sistema de luzes com sombras, e sistema de animação esquelética.

3.1.2.5 – Projeto em Desenvolvimento

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 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

O Three.js é uma biblioteca open-source para a linguagem JavaScript que permite aos desenvolvedores criar cenas 3D de forma organizada e renderizá-las diretamente do navegador WEB. Ela tem se tornado uma das bibliotecas de desenvolvimento de gráficos 3D mais populares da atualidade, sendo principalmente utilizada para o desenvolvimento de jogos online, demonstrações e modelos. (BOSNJAK, 2018)

O Three.js foi lançado por Ricardo Cabello em 2010 na plataforma GitHub. Originalmente estava sendo desenvolvido em *ActionScript*, mas em 2009 foi portado para JavaScript, removendo a necessidade de ser previamente compilado pelo desenvolvedor. Desde os treze anos de atividade desde seu lançamento, o Three.js recebeu inúmeras contribuições e atualizações de diversos desenvolvedores, chegando a exceder 25.000 *commits*. (LILLY021, 2022)

É uma ótima escolha tanto para desenvolvedores experientes quanto para desenvolvedores novos no desenvolvimento 3D, sendo significativamente mais fácil de

ser utilizado que o WebGL puro. Sendo especialmente feito para desenvolver cenas e animações 3D sem se preocupar com sua interação com o hardware. Para utilizar o Three.js é necessário que o desenvolvedor possua pelo menos conhecimento básico em programação em JavaScript e HTML, além de saber CSS e seus seletores. (THREE.JS, 2023).

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

3.1.3.2 - Babylon.js

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

Originalmente foi criada como um projeto *open-source* pelo engenheiro da Microsoft David Catuhe e posteriormente se tornou seu trabalho em tempo integral. Conduzindo seu time para desenvolver a mais simples e poderosa ferramenta de renderização WEB, em 2015 o Babylon.js já demonstrava suas grandes capacidades, conquistando a atenção de seus concorrentes. (IRWIN, 2021)

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

O Babylon.js, além da engine, possui diversas ferramentas auxiliares para ajudarem no desenvolvimento, como o *playground*, que permite que se escreva código e o visualize em tempo real ao lado, um 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).

3.1.3.4 - Playcanvas

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

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

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

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

O playcanvas, ao contrário das outras ferramentas, tem 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 *hosting* no website da biblioteca.

3.2 - Métricas de Software

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

A seção 3.2.1 trata das métricas qualitativas utilizadas para a a comparação, seguida das métricas quantitativas, na seção 3.2.2.

3.2.1 - Métricas Qualitativas

3.2.1.1 - Qualidade da Documentação

A qualidade de uma documentação é um aspecto crucial a ser avaliado pelo desenvolvedor que pretende consumir uma biblioteca como API em seu projeto. A documentação permite que desenvolvedores conheçam as funcionalidades presentes da biblioteca, e entendam a sua utilização em código. Assim sendo, serão analisados a qualidade da documentação disponível para cada uma das bibliotecas avaliadas. Aspectos a serem considerados incluem: comentários e explicações da documentação API, guia e manual do usuário, demonstrações em código-fonte das funcionalidades, presença de projetos completos desenvolvidos, e por fim as redes sociais ou fóruns que a biblioteca possa possuir para suporte.

3.2.2 - Métricas Quantitativas

3.2.2.1 - Desenvolvimento do Projeto

Um fator de qualidade de uma biblioteca é o engajamento de comunidade responsável por seu desenvolvimento. O quanto uma biblioteca possui seu código-

fonte alterado para a adição de funcionalidades e correção de bugs é um fator a ser considerado antes da inclusão da mesma em um projeto. Assim, pode-se avaliar, a partir de dados disponíveis nos seus repositórios no Github: data da última atualização, quantidade total e recente de *commits* (atualizações e mudanças feitas em arquivos), quantidade total e recente de *pull requests* (atualizações sendo adicionadas na versão de produção), quantidade total e recente de *issues* (mudanças requisitadas pela comunidade ou *bugs* identificados), quantidade de projetos que utilizam a biblioteca e quantidade de contribuidores do projeto.

3.2.2.2 - Funcionalidades Presentes

Computação gráfica envolve muito mais do que apenas carregar modelos e dispô-los da tela do dispositivo. Uma biblioteca responsável por abstrair as funções do *OpenGL* pode oferecer outras funções além de uma simples renderização 3D, e pode possuir mais funcionalidades capazes de gerar gráficos mais realistas, ou também oferecer funções extras e customizações ao programador, permitindo-lhe que tenha melhor controle sobre sua aplicação e que desenvolva diversos tipos de aplicativos mais facilmente. Algumas das funcionalidades possíveis de serem avaliadas são:

- Configuração e propriedades dos materiais PBR disponíveis, efeitos de pós-processamento e filtros de sombras suportados, que enriquecem a riqueza dos gráficos gerados.
- Utilidades de renderização e otimização, tais como: *Instanced Rendering*, a capacidade de mandar vértices de vários objetos do mesmo para a GPU e renderizá-los numa única chamada; combinação de malhas, que consiste em combinar objetos, igualmente renderizando-o numa única chamada; *Level of Details*, que permite reduzir a qualidade dos modelos quando distantes da câmera; *Basis Texture Compression*, o suporte a um formato de textura otimizado para a GPU, que reduz consideravelmente o uso de memória gráfica.
- Funcionalidades e ferramentas extras, como: Sistema de física e colisão; áudio, suporte a realidade virtual, exportação de cenas; editor gráfico de cenas;

depurador gráfico de cenas, no qual é possível visualizar os atributos de todos os objetos durante a execução da aplicação; sistema de criação de GUI (interface gráfica) 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 suportados de modelos 3D para importação.
- Suporte a versão 2 do WebGL.

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

3.2.2.3 - Desempenho

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

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

Para que isso seja feito, serão desenvolvidos testes envolvendo cada uma das bibliotecas, nos quais modelos 3D em formato GLTF serão carregados para a renderização de uma cena. Serão feitos diversos testes a fim de obter um resultado diversificado, utilizando cenas com variações na quantidade objetos e complexidade dos objetos, na distância da câmera e com sombras desativadas e ativadas.

3.2.2.4 - Tamanho da Biblioteca

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

- Memória ocupada total dos arquivos do repositório público da biblioteca e de suas dependências ao serem baixados por um gerenciador de pacotes. Aqui também serão também avaliados repositórios extras, que estão disponíveis em algumas bibliotecas como extensão.
- Memória ocupada do código-fonte da biblioteca após juntado num único arquivo por um programa *bundler*, como o Esbuild, 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 sites oficiais dos desenvolvedores da biblioteca. Na seção 3.2.2 explica-se como medir com precisão tamanho ocupado por uma biblioteca utilizando o gerenciador de pacotes *NPM*, o *bundler* Esbuild e os algoritmos de compressão. A seção 3.3.3 trata de explicar como os navegadores e as funções nativas da linguagem Javascript podem ser utilizados para a coleta e análise de dados de desempenho, e na seção 3.3.4 detalha-se a configurações dos computadores utilizados nos testes de desempenho.

3.3.1 - Documentação e Repositórios

Dos critérios de comparação, alguns podem ser avaliados facilmente através de uma pesquisa em seus sítios oficiais. O desenvolvimento do projeto possui estatísticas diretamente disponíveis seu repositório de código Github, enquanto as funcionalidades

presentes, os formatos de arquivo suportados e qualidade da documentação podem ser avaliados a partir das informações disponíveis site oficial da biblioteca e/ou no seu repositório.

3.3.2 - Ferramentas de Medição de Tamanho

3.3.2.1 - Gerenciador de pacotes NPM

O gerenciador de pacotes NPM pode ser definido como:

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

Quando uma biblioteca é baixada pelo NPM num diretório de um projeto, é gerado um subdiretório para 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 de analisar qual o tamanho dos arquivos que realmente serão requisitados pelo navegador ao desenvolver uma aplicação que utiliza esta biblioteca. Um *bundler*, como o Esbuild, é capaz de pegar todos os arquivos de código-fonte utilizadas por uma aplicação e juntá-lo num único arquivo, tornando mais fácil a medição do tamanho real em memória que a biblioteca irá utilizar. Também é possível aplicar o algoritmo de minificação, que reduz consideravelmente o tamanho final do arquivo gerado. (ESBUILD, [s.d])

3.3.2.3 - Compressão DEFLATE

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

3.3.3 - Mensuradores de Desempenho

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

3.3.3.1 - Navegadores

As bibliotecas deste projeto foram desenvolvidas para serem utilizadas no contexto de uma página WEB, e assim, requerem um navegador para serem executados no computador. O navegador Google Chrome foi selecionado como o base 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. A

figura abaixo demonstra o mensurador de CPU e RAM no canto inferior da tela:

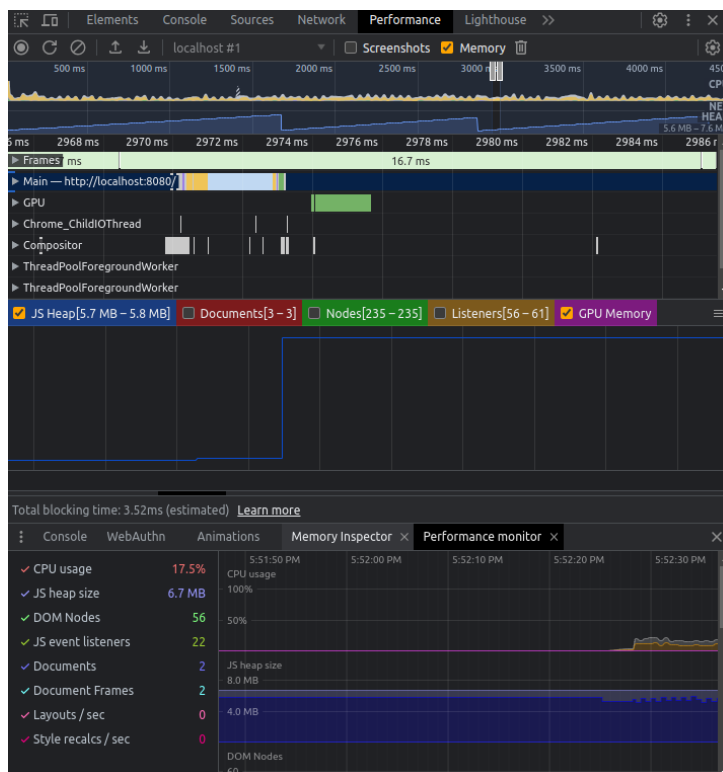


Figura 04 - Painel Devtools do Google Chrome

Fonte: Elaborado pelos autores (2023)

3.3.3.3 - Funções Javascript

Na coleta dos dados de tempo de inicialização e tempo de carregamento de arquivos de modelos 3D, pode-se utilizar a função *performance.now* da linguagem Javascript no início e no fim da cronometragem e tirar sua diferença em milissegundos.

Para obter o tempo de renderização por quadro, é necessário utilizar a função *requestIdleCallback* no final de cada quadro, que recebe uma função como argumento que será 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 a do monitor, de modo que se

utilizasse função *performance.now* para calcular a diferença entre os quadros não seria possível captar valores excedentes a este limite. Por fim, divide-se 1000 pelo tempo obtido, resultando no FPS (quadros por segundo). O processo será repetido num período de 30 segundos, e no fim será calculado a média dos valores gerados.

3.3.4 - Ambiente de Desenvolvimento

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

- PC-1: processador Intel Core i3-4170 CPU (4 CPUs) ~3.7GHz, de placa de vídeo Mesa Intel HD Graphics 4400 (1.5 Gib), de memória RAM 3.7 GiB DDR3, e sistema operacional Arch Linux Kernel 6.1.26-1-lts, e Monitor 1600x900 60hz.
- PC-2: processador AMD Ryzen 5 5500 (12 CPUs) ~3.6GHz, de placa de vídeo NVIDIA GeForce GTX 1660 SUPER, de memória RAM 16384MB DDR4, e sistema operacional Windows 11 Home 64 bits, e Monitor 2560x1440 75hz.

4 – Resultados

A seguir são apresentados os dados coletados conforme explicado no capítulo anterior. A seção 4.1 trata da métrica qualitativa da documentação, e a seção 4.2 das métricas quantitativas: o desenvolvimento do projeto, funcionalidades presentes, desempenho e tamanho da biblioteca.

4.1 - Análise das Métricas Qualitativas

4.1.1 - Qualidade da Documentação

4.1.1.1 - Three.js

A Biblioteca Three.js além de sua API Docs comentada possui um amplo manual de usuário, demonstrando suas funcionalidades principais com exemplos e explicações detalhadas. Também possui uma página no seu site com amplos projetos

desenvolvidos pela comunidade. Ainda, por ser uma biblioteca popular, conta com diversos tutoriais disponíveis em outras plataformas desenvolvidos por terceiros. Redes sociais: fórum próprio, Discord e Twitter.

4.1.1.2 - Babylon.js

A Biblioteca Babylon.Js possui sua API Docs plenamente comentada. Possui também um guia de usuário amplo demonstrando praticamente todas as funcionalidades e configurações de Biblioteca, além de muitos projetos completos desenvolvidos pelos desenvolvedores e comunidades. Redes sociais: fórum próprio, Discord, Medium, Youtube e Twitter.

4.1.1.3 - Playcanvas

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

4.2 - Análise das Métricas Quantitativas

4.2.1 - Desenvolvimento do Projeto

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

Métricas Recentes	Biblioteca		
	Three.js	Babylon.js	Playcanvas
<i>Pull Requests (Requisitados)</i>	10	5	4
<i>Pull Requests (Aceitos)</i>	120	118	66
<i>Issues (Abertos)</i>	15	6	22
<i>Issues (Fechados)</i>	46	19	20
<i>Autores</i>	40	20	14
<i>Commits</i>	134	221	68

Quadro 01 - Desenvolvimento de cada biblioteca no período de um mês

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

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

Métricas Totais	Biblioteca		
	Three.js	Babylon.js	Playcanvas
<i>Pull Requests (Requisitados)</i>	145	8	21
<i>Pull Requests (Aceitos)</i>	14.627	11.371	3.692
<i>Issues (Abertos)</i>	386	93	525
<i>Issues (Fechados)</i>	1.1550	2.836	1.422
<i>Contribuídores</i>	1.736	485	130
<i>Commits</i>	42.170	41.403	10.902
<i>Estrelas</i>	94.810	21.464	8.671
<i>Usado Por</i>	228.190	4.648	1

Quadro 02 - Desenvolvimento de cada biblioteca desde criação

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

4.2.2 - Funcionalidades Presentes

4.2.2.1 - Propriedades dos Materiais PBR Disponíveis

O quadro 2 do apêndice B demonstra as propriedades dos materiais PBR disponíveis em cada biblioteca. Um campo marcado com “Mapa” indica que a propriedade é configurável através de uma textura, ou seja, por pixel, caso seja

marcado com “Escalar” significa que a propriedade só pode ser configurada em forma de intensidade (um número de 0.0 a 1.0) ou em forma de coloração geral (um valor RGB), enquanto o campo vazio implica a ausência total da propriedade.

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

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

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



Figura 05 - Propriedade *parallax*

Fonte: Blender Forum (2019)⁴

⁴ Disponível em: <https://blender.community/c/rightclickselect/t9bbbc/?sorting=hot>. Acesso em: 22 out. 2023.

4.2.2.2 - Efeitos de Pós-Processamento

O quadro disponível no apêndice C contém os dados em relação a alguns dos efeitos de pós-processamento mais populares. Um campo marcado com “X” significa o filtro está disponível nos repositórios oficiais da biblioteca, enquanto os campos marcados com “Plugin” significam que a funcionalidade está disponível pela extensão de um terceiro. 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.

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

Em último lugar se apresenta a biblioteca Playcanvas, que possui a menor quantidade de efeitos de pós-processamento em relação as outras, que, além da falta dos efeitos de *anti-aliasing* mais avançados e o efeito de *blur*, destaca-se a ausência dos efeitos de SSR e *depth of field*, que são capazes de mudar significativamente a ambientação de uma cena.

4.2.2.3 - Efeitos de Sombras Suportados

O quadro abaixo contém a lista dos diferentes filtros de sombras suportados em cada biblioteca. Os filtros, com exceção do CSM, que é uma técnica de otimização, são excludentes entre si, e podem ser escolhidos a depender da preferência do usuário por maior desempenho ou qualidade gráfica.

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

Quadro 03 - Suporte aos diferentes efeitos de sombra por cada biblioteca

Fonte: Elaborado pelos autores (2023)

4.2.2.3 - Utilidades de Otimização

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

Técnica de Otimização	Biblioteca		
	Three.js	Babylon.js	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 04 - Técnicas de otimização suportadas pelas bibliotecas

Fonte: Elaborado pelos autores (2023)

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

4.2.2.4 - Formatos de Arquivos 3D para Importação Suportados

O quadro abaixo trata dos diferentes formatos de arquivo que cada biblioteca suporta para a importação de objetos 3D. Os formatos GLTF, COLLADA, OBJ e FBX são formatos de propósito geral, enquanto o MMD e 3DS são vinculados a programas

de modelagem proprietários, e os outros formatos são utilizados em certos domínios de aplicações, como CADs, modelagem de avatares, dados de pontos (utilizados em robótica, realidade aumentada), dados de moléculas e Lego. O último campo se refere a capacidade em carregar arquivos comprimidos pela biblioteca DRACO.

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 Bank)	X		
Ldraw (LEGO)	X		
Compressão de Malha DRACO	X	X	X

Quadro 05 - Formatos de arquivos de modelos 3D suportados por cada biblioteca

Fonte: Elaborado pelos autores (2023)

Em geral, para construção de aplicações WEB, o formato GLTF 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 pelo Three.js.

4.2.2.5 - Funcionalidades e Ferramentas Extras Presentes

A tabela 9 trata de listar diversas funcionalidades extras que cada biblioteca implementa e que possuem diferentes utilidades a depender do domínio da aplicação desenvolvida.

Nota-se que funcionalidades úteis no desenvolvimento de jogos, como sistema de partículas, áudio, física (através da integração com um plugin externo) e *NavMesh*

estão presentes nas bibliotecas Playcanvas e Babylon.js. Já a biblioteca Three.js requer o uso de *plugins* para fornecer tais funcionalidades – com exceção do áudio, que funciona nativamente -. Não obstante, não possui integração com um sistema de física, cabendo ao desenvolvedor da aplicação implementar suas próprias integrações para conseguir obter tal utilidade. A necessidade de muitos *plugins* também implica na limitação do editor gráfico da Three.js em relação aos outros, uma vez que não é capaz de integrar suas funcionalidades como se fossem nativas.

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

4.2.3 - Testes de Desempenho

Este subcapítulo é dedicado a demonstrar as estatísticas relacionadas aos testes de desempenho realizadas em cada biblioteca. O quadro 10 do apêndice E informa a quantidade de triângulos e objetos por cada cena renderizada, enquanto o quadro 11 explica os detalhes específicos dos testes realizados. A visualização das imagens geradas pode ser encontrada no apêndice G.

Neste capítulo estão disponíveis as estatísticas coletadas no navegador Chrome no PC-1 e PC-2. Os dados coletados no Mozilla Firefox foram descartados por apresentarem notórias imprecisões. Dados referentes ao FPS coletados no Microsoft Edge e no PC-2 estão disponíveis no apêndice I.

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 X do apêndice H), razão pela o teste foi realizado no Brave, um navegador semelhante ao Chrome, que utiliza a mesma engine *Chromium* (KEIZER,

2021).

A biblioteca Playcanvas, possuindo dois modos de desenvolvimento, o editor gráfico e o modo “*engine only*”, no qual se utiliza apenas código-fonte, foi testada dos dois jeitos, possuindo os rótulos nos gráficos “Playcanvas (Editor)” e “Playcanvas (Engine)”, respectivamente.

4.3.3.1 – Renderização e FPS

O gráfico abaixo contém os dados de FPS médio realizados no PC-1 e no navegador Chrome para todos os 11 testes realizados no PC-1, conforme informações disponíveis no quadro 11 do apêndice E:

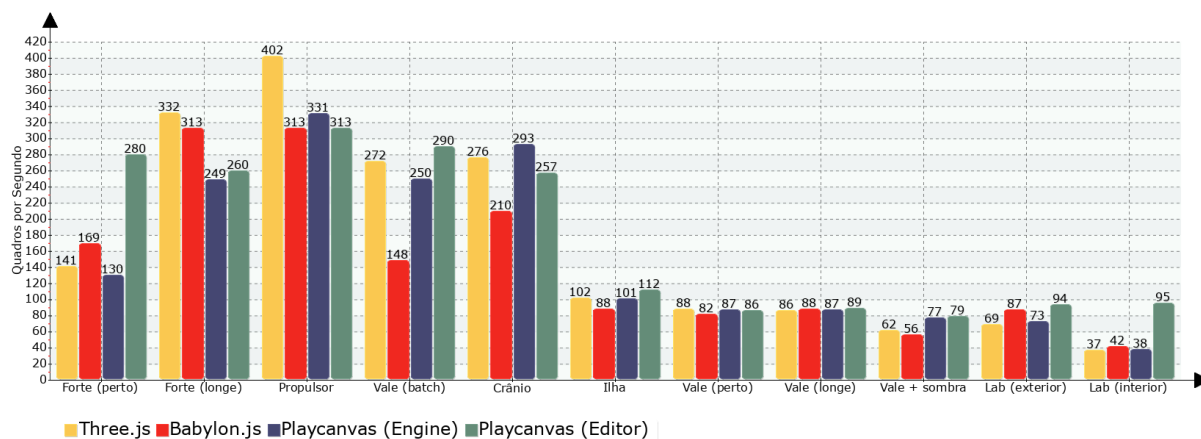


Gráfico 01 - Taxa de FPS de cada biblioteca por cena renderizada – PC-1 - Google Chrome

Fonte: Elaborado pelos autores (2023)

Os dois gráficos a seguir mostram os testes no PC-2. Além dos 11 testes já realizados no PC-1, foi incluído a renderização da cena do deserto, que demanda considerável poder computacional e não pode ser renderizada nas configurações do PC-1. Abaixo segue o gráfico com os 6 primeiros testes:

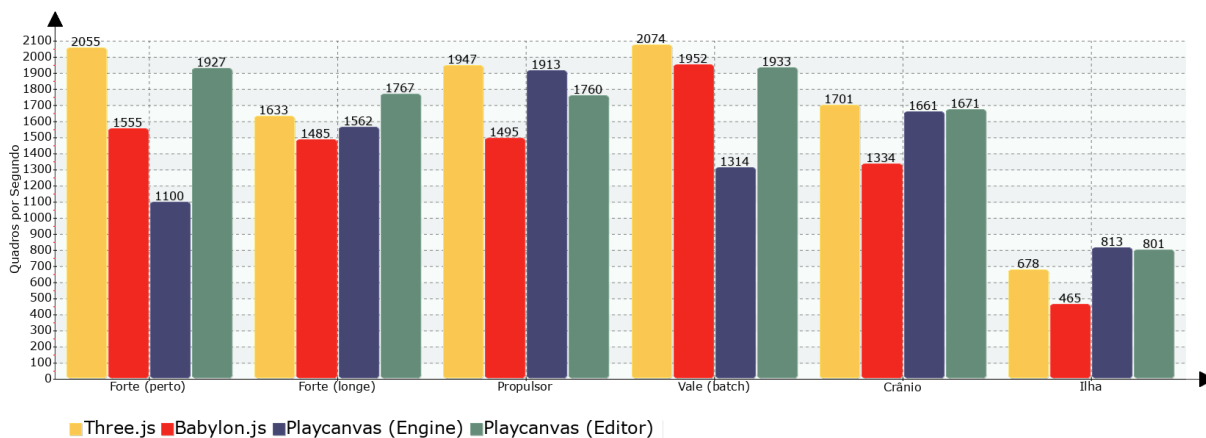


Gráfico 02 - Taxa de FPS de cada biblioteca por cena renderizada – PC-2 - Google Chrome – Parte 1/2
Fonte: Elaborado pelos autores (2023)

Abaixo os 7 últimos testes realizados no PC-2:

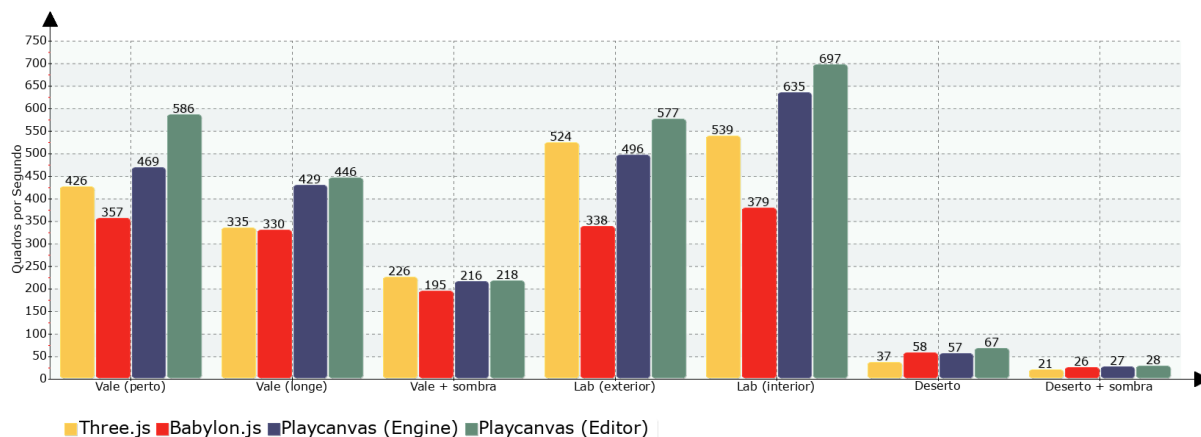


Gráfico 03 - Taxa de FPS de cada biblioteca por cena renderizada – PC-2 - Google Chrome – Parte 2/2
Fonte: Elaborado pelos autores (2023)

É possível observar que a biblioteca Three.js obteve vantagem em relação as outras duas na renderização das cenas mais simples, evidenciado pelo dados referentes às cenas do forte pirata e do propulsor de íons, mas ficou muito atrás das outras na cena mais complexa do deserto.

Enquanto isso, a Playcanvas, demonstrou os melhores resultados na renderização das cenas mais complexas do vale, laboratório, ilha e deserto, sendo a biblioteca com resultados mais robustos dentre as três. Também é notório a

superioridade em se utilizar o editor em relação ao modo “*engine only*”.

Já a biblioteca Babylon.js manteve um desempenho razoável na maioria das cenas, obtendo bons resultados no PC-1, mas ficando atrás em muitas cenas no PC-2. No entanto, no único teste em que os valores de FPS para todas as bibliotecas caíram para números abaixo do ideal, a cena do deserto, manteve-se com bons resultados ao lado da Playcanvas.

A utilização do Microsoft Edge não impacta significativamente as conclusões tiradas acima, conforme apêndice J.

4.3.3.2 – Uso de CPU e memória RAM

A seguir são apresentados os dados do uso de CPU e de gasto de memória RAM dos programas pelas bibliotecas em cada uma dos testes renderizados. O uso de CPU foi obtido pelos valores medianos observados no painel *Devtools*, enquanto o gasto de RAM, que estabilizou-se após um determinado período de execução, apresentou um valor estático, sendo este número coletado como valor final. Foram utilizados as mesmas cenas do testes de FPS, mas sem variar a distância da câmera, havendo apenas 8 testes para o PC-1, e 9 para o PC-2.

A seguir são apresentados os dados de uso de CPU coletados no PC-1:

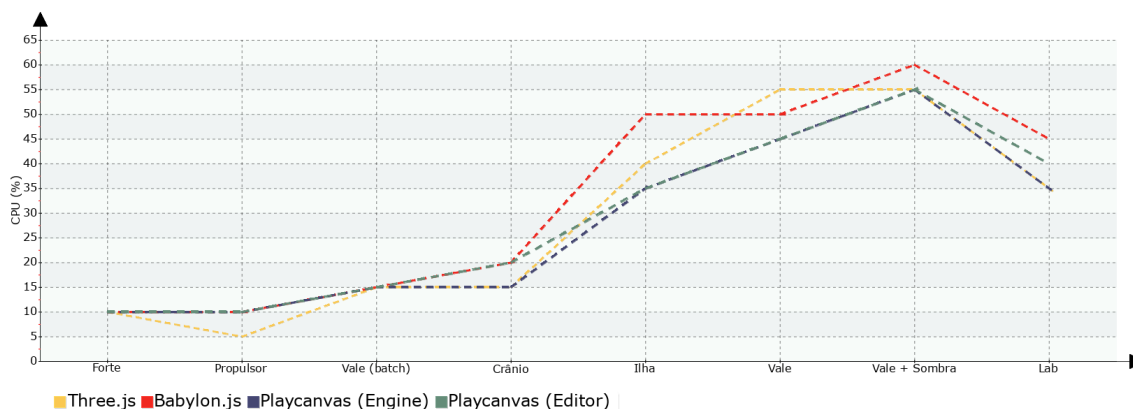


Gráfico 04 - Uso de CPU de cada biblioteca por cena renderizada – PC-1 – Google Chrome

Fonte: Elaborado pelos autores (2023)

Abaixo têm-se os dados de uso de CPU para o PC-2:

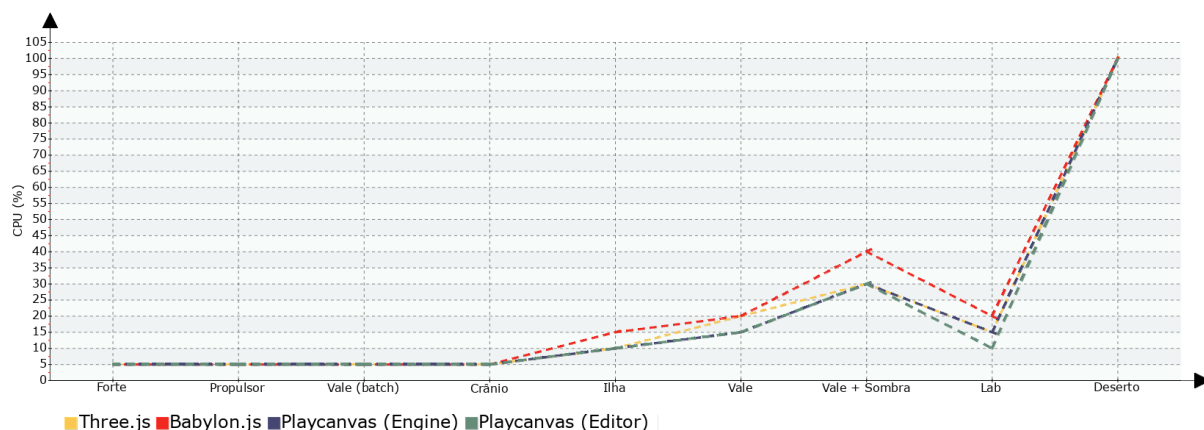


Gráfico 05 - Uso de CPU de cada biblioteca por cena renderizada – PC-2 – Google Chrome

Fonte: Elaborado pelos autores (2023)

Nas cenas mais simples é evidente que não houve diferença no uso de CPU por parte das três bibliotecas. Mas ao analisar as cenas mais complexas do vale, do laboratório e da ilha, tanto no PC-1 como no PC-2, vê-se que a biblioteca Babylon.js gastou até 15% de CPU a mais que as outras na cena da ilha no PC-1, e 10% no PC-2 na renderização de sombras na floresta. Todas elas gastaram 100% na renderização da cena do deserto.

A seguir são apresentados os dados do consumo de RAM para o PC-1:

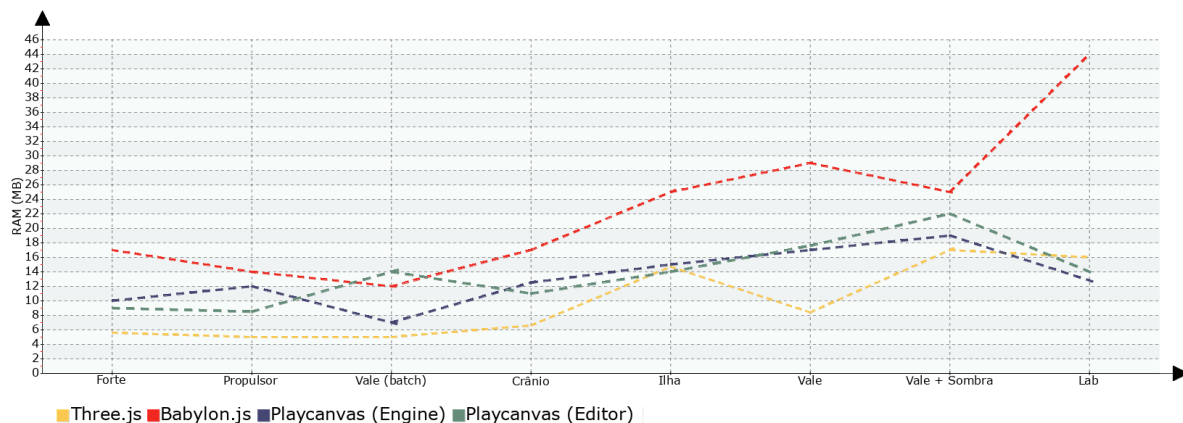


Gráfico 06 - Consumo de RAM de cada biblioteca por cena renderizada – PC-1 – Google Chrome

Fonte: Elaborado pelos autores (2023)

Abaixo têm-se os dados do consumo de RAM para o PC-2:

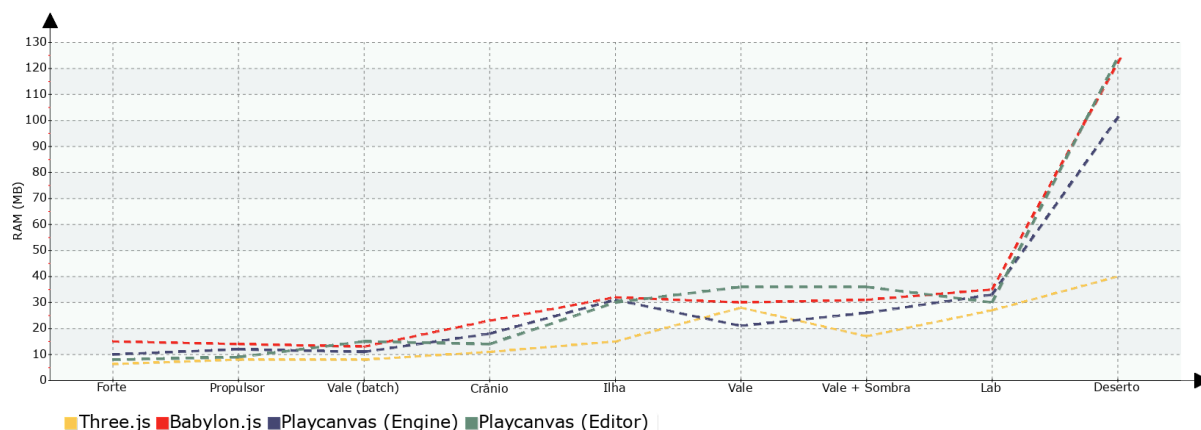


Gráfico 07 - Consumo de RAM de cada biblioteca por cena renderizada – PC-2 – Google Chrome

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, seguida logo da Playcanvas com um consumo ligeiramente maior, com exceção da cena do deserto, no qual gastou 100MB em relação aos 40MB da Three.js. Já a Babylon.js utilizou consistentemente mais memória que as outras em todos os casos.

4.3.3.4 – 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, independente do que será feito depois, 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), com exceção para a biblioteca Playcanvas em modo de editor, que ao exportar as cenas, gera um arquivo de texto *json* (*Javascript Object Notation*).

Abaixo segue o gráfico dos tempos obtidos no PC-1:

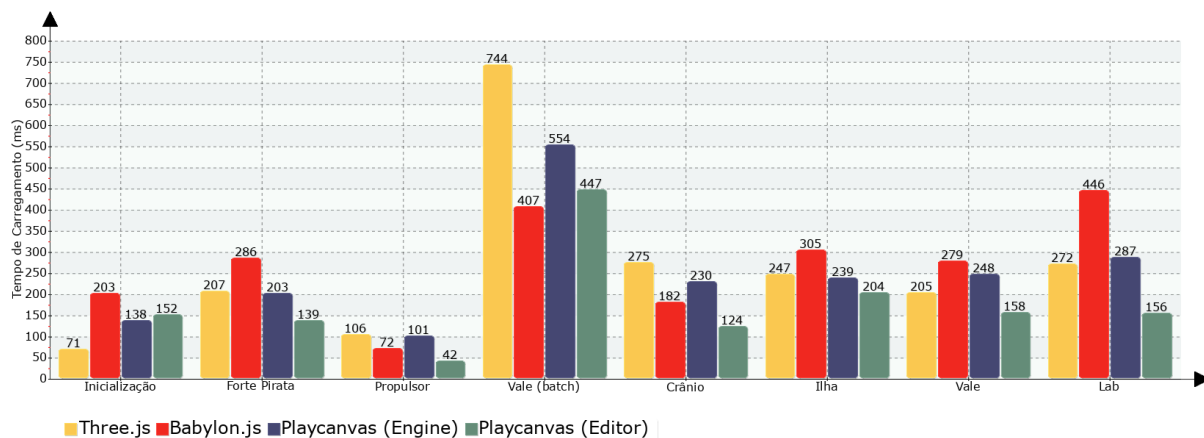


Gráfico 08 - Tempo de inicialização e carregamento de cada biblioteca por cena renderizada – PC-1 – Google Chrome

Fonte: Elaborado pelos autores (2023)

A seguir têm-se os dados coletados no PC-2:

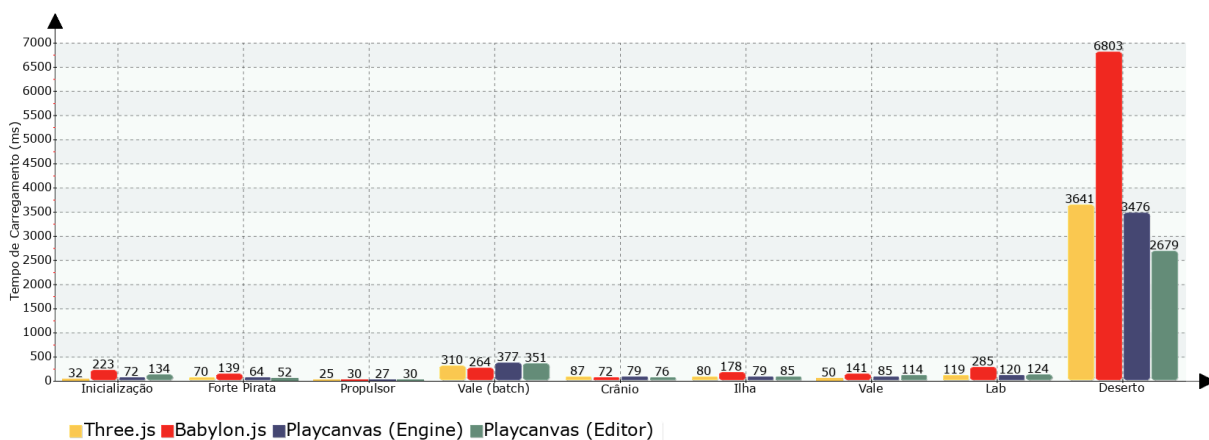


Gráfico 09 - Tempo de inicialização e carregamento de cada biblioteca por cena renderizada – PC-2 – Google Chrome

Fonte: Elaborado pelos autores (2023)

É possível ver que a biblioteca Three.js possui o menor tempo de inicialização, enquanto a Playcanvas no modo editor consegue carregar a maioria das cenas mais rapidamente que o restante, sobretudo no PC-1. A biblioteca Babylon.js, por outro lado, apresentou pior tempo de inicialização e de carregamento de arquivos, sendo 2x mais lenta no carregamento do deserto.

Em aplicações mais complexas, a demora da Babylon.js em relação às outras é possível causa de desistência do usuário final, que se incomodará com o excessivo

tempo de carregamento.

4.3.3.4 – Problemas Encontrados

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

- Na renderização de cenas com sombras ativadas pela biblioteca Three.js, a cena ficou sem cores e repleta de artefatos visuais, conforme figura X do apêndice H. Foi observado somente no navegador Google Chrome e no PC-1.
- A renderização da cena do propulsor de íons, por parte da biblioteca Playcanvas no modo editor gráfico, gerou algumas cores incorretas conforme figura Y do apêndice H.
- 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 O do apêndice H).

4.2.4 - Tamanho da Biblioteca

O gráfico abaixo trata de mostrar o tamanho em memória de cada biblioteca. O gráfico a esquerda trata do tamanho do código-fonte, 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.

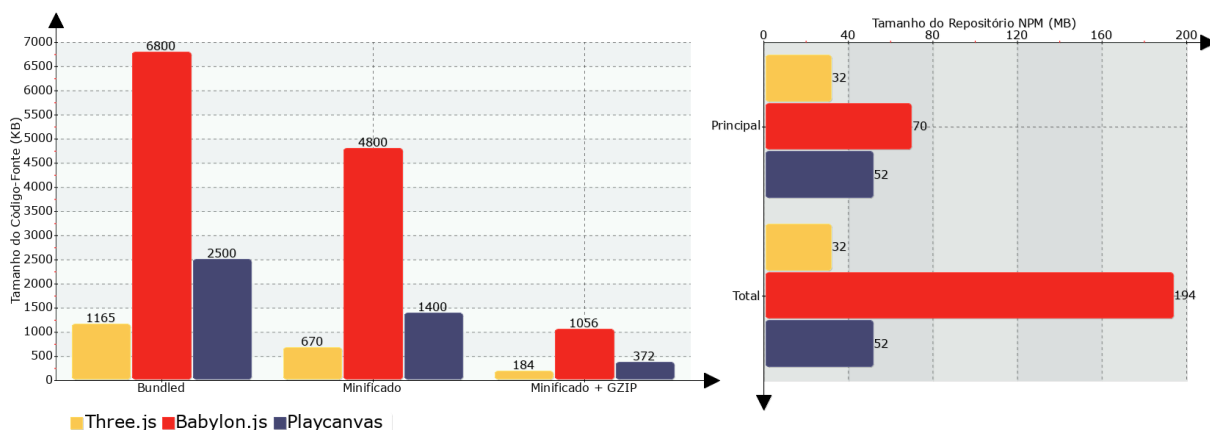


Gráfico 10 - Tamanho do código-fonte (esquerda), e dos repositórios (direita) por biblioteca

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 em resolução FULL HD, reduzindo drasticamente tais possíveis problemas.

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 geral 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 Three.js é mais leve, ocupando pouco espaço em memória e gastando poucos recursos de RAM, além de poder ser estendida com *plugins* que fornecem as funcionalidades que lhe faltam em relação às outras. Por isso, é recomendada quando o objetivo é enriquecer páginas da WEB com efeitos visuais 3D. Além disso, se mostrou a única com amplo suporte a diferentes

formatos de arquivo, que são utilizados em certos softwares de modelagem proprietários ou que possuem uso muito específico, sendo recomendada quando o suporte a este tipo de arquivo é necessário. Sendo a biblioteca mais popular dentre elas, também possui bom desempenho e é recomendada para qualquer tipo de aplicação no geral, no entanto, se mostrou muito mais lenta que as outras num dos testes realizados.

Já as bibliotecas Playcanvas e Babylon.js possuem mais funcionalidades quando se tratam de desenvolvimento de jogos. A primeira leva grande vantagem por ter obtido os melhores resultados de FPS entre todas bibliotecas analisadas, além de gastar menos espaço em memória e RAM em relação a segunda, sendo altamente recomendada quando o objetivo é obter o máximo de desempenho possível. Também é útil do ponto de vista da praticidade, sendo a única capaz de gerar *lightmaps* sem o uso de um programa externo, e possuindo um editor em interface gráfica extremamente poderoso, além de oferecer *hosting* gratuito no seu website para projetos construídos em seu editor. Enquanto isso, a Babylon.js 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.

Referências

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

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

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

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

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

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

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

CAULFIELD, Brian. **O que é Path Tracing?**. Blog NVIDIA, 2022. Disponível em: <https://blog.nvidia.com.br/2022/05/10/o-que-e-path-tracing/>. Acesso em: 16 out. 2023.

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

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

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

Free Software Foundation. **GNU Gzip: General File (de)compression**. GNU, c2009-2023. Disponível em: <https://www.gnu.org/software/gzip/manual/gzip.html>. Acesso em: 21 out. 2023.

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

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

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

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

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

IRWIN, Emma. **Microsoft Open Source success story—Babylon**. Microsoft, 2021.

Disponível em: <https://cloudblogs.microsoft.com/opensource/2021/02/22/microsoft-open-source-success-story-babylon/>. Acesso em: 25 mai. 2023.

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

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

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

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

MARTINDALE, Jon. **Ray tracing vs. path tracing — which is the best dynamic lighting technique?**. DigitalTrends, 2022. Disponível em: <https://www.digitaltrends.com/computing/ray-tracing-vs-path-tracing>. Acesso em: 16 out. 2023.

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

MFUJI09 etc al. **MIME Types (IANA media types)**. MDN Web Docs, c1998-2023. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Acesso em: 16 mar. 2023.

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

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

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

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

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

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

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

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

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

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

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

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

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

SILVA, Luiz . **O que é geometria?**. Brasil Escola, [s.d]. Disponível em: <https://brasilecola.uol.com.br/o-que-e/matematica/o-que-e-geometria.htm>. Acesso em: 24 set. 2023.

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

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

photorealistic-3d-graphics. Acesso em: 27 set. 2023.

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

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

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

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

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

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

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

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

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

Apêndices

Apêndice A – Bibliotecas de Gráficos 3D Disponíveis

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

Quadro 06 - Lista de bibliotecas Javascript para renderização de gráficos 3D encontradas na WEB

Fonte: Elaborado pelos autores (2023)

Apêndice B – Funcionalidades – Propriedades PBR

Propriedade PBR	Descrição	Biblioteca		
		Three.js	Babylon.js	Playcanvas

<i>Albedo</i>	Cor base do material ao ser iluminado	Mapa	Mapa	Mapa
<i>Metalness</i>	Reflexão dos objetos em volta, como um metal	Mapa	Mapa	Mapa
<i>Rough / Gloss</i>	Aspereza da superfície	Mapa	Mapa	Mapa
<i>Refraction</i>	Como a luz atravessa o objeto e muda de direção	Escalar	Mapa	Mapa
<i>Environment/ Reflection</i>	Reflexão do plano de fundo (<i>Skybox</i>)	Mapa	Mapa	Mapa
<i>Normal</i>	Elevação leve da superfície, sem afetar as sombras	Mapa	Mapa	Escalar
<i>Emissive</i>	Cor base em ambientes escuros	Mapa	Mapa	Mapa
<i>Opacity</i>	Transparência	Mapa	Mapa	Mapa
<i>LightMap</i>	Parte iluminada do objeto, calculado a partir das fontes de luzes da cena	Mapa	Mapa	Mapa
<i>Detail</i>	Detalhes adicionais na textura quando próximos da câmera		Mapa	Mapa
<i>Iridescence</i>	Faixas de cores brilhantes que mudam com o ângulo	Mapa	Mapa	Mapa
<i>Anisotropic</i>	Distorção da luz refletida		Mapa	Escalar
<i>Sheen</i>	Simula a textura de roupas ou tecidos	Mapa	Mapa	Mapa
<i>Specular</i>	Ponto brilhante das luzes refletidas	Mapa	Mapa	Mapa
<i>Transmission</i>	Utilizada para aumentar a reflexão em objetos transparentes	Mapa	Escalar	
<i>ClearCoat</i>	Camada fina e brilhante em materiais ásperos	Mapa	Mapa	Mapa
<i>Parallax</i>	Efeito de profundidade em superfícies com relevos		Mapa	
<i>Decal</i>	Detalhes adicionais, como uma pixação numa parede		Mapa	

<i>Ambient Occlusion</i>	Sombras suavizadas nas partes obstruídas pela luz	Mapa	Mapa	Mapa
<i>Displacement</i>	Elevação severa da superfície, afetando as sombras	Mapa	Mapa	Mapa
<i>Attenuation</i>	Como o objeto absorve a luz a medida que se distancia do observador	Escalar		Escalar
<i>Thickness</i>	Espessura do objeto, que afeta a reflexão	Parcial	Mapa	Mapa

Quadro 07 - Funcionalidades de propriedades de materiais PBR disponíveis por biblioteca – "Mapa" indica configuração por pixel, "Escalar" indica configuração por intensidade única ou cor RGB.

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

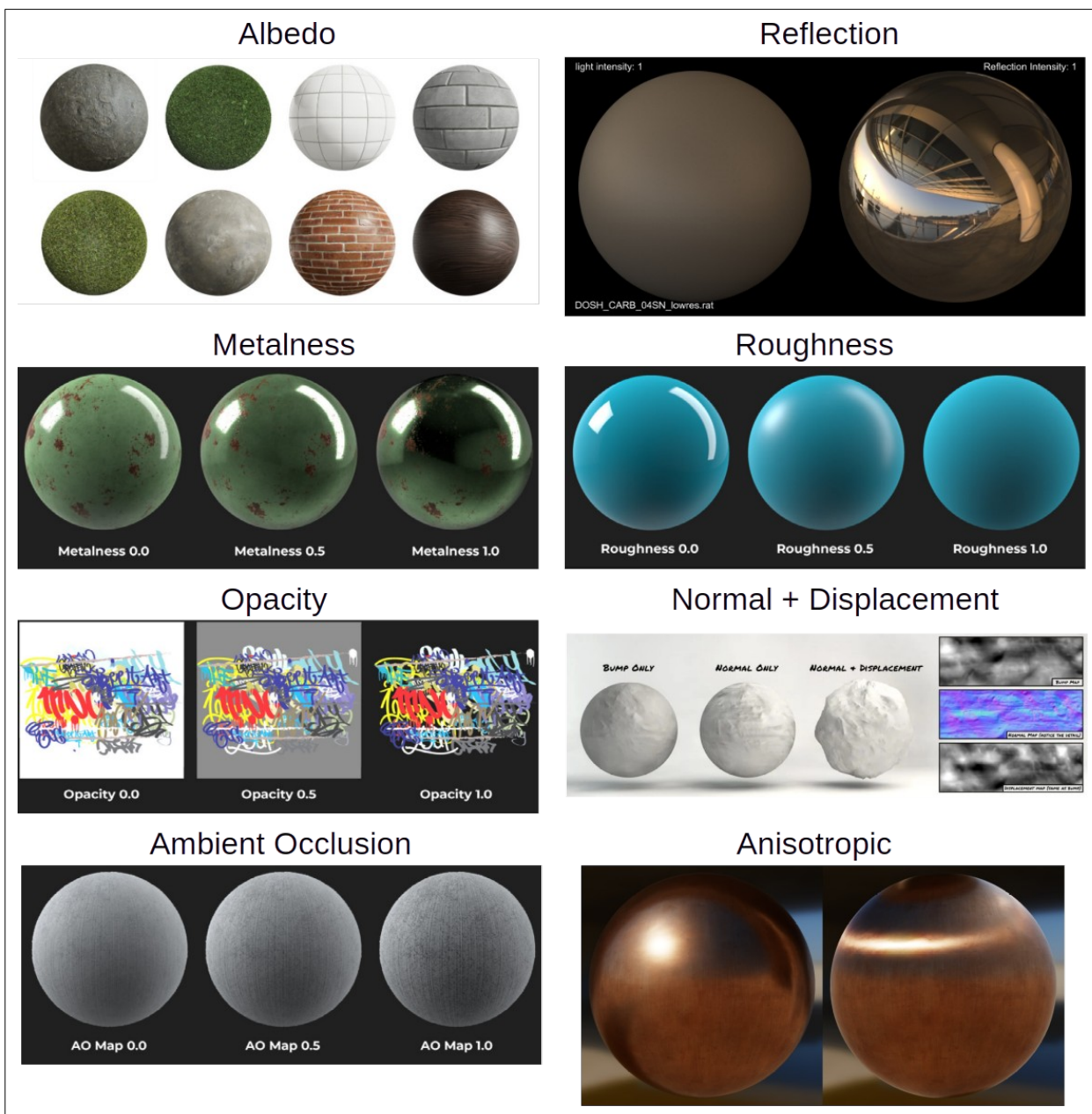


Figura 06 - Visualização das propriedades dos materiais PBR – Parte 1/3

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.

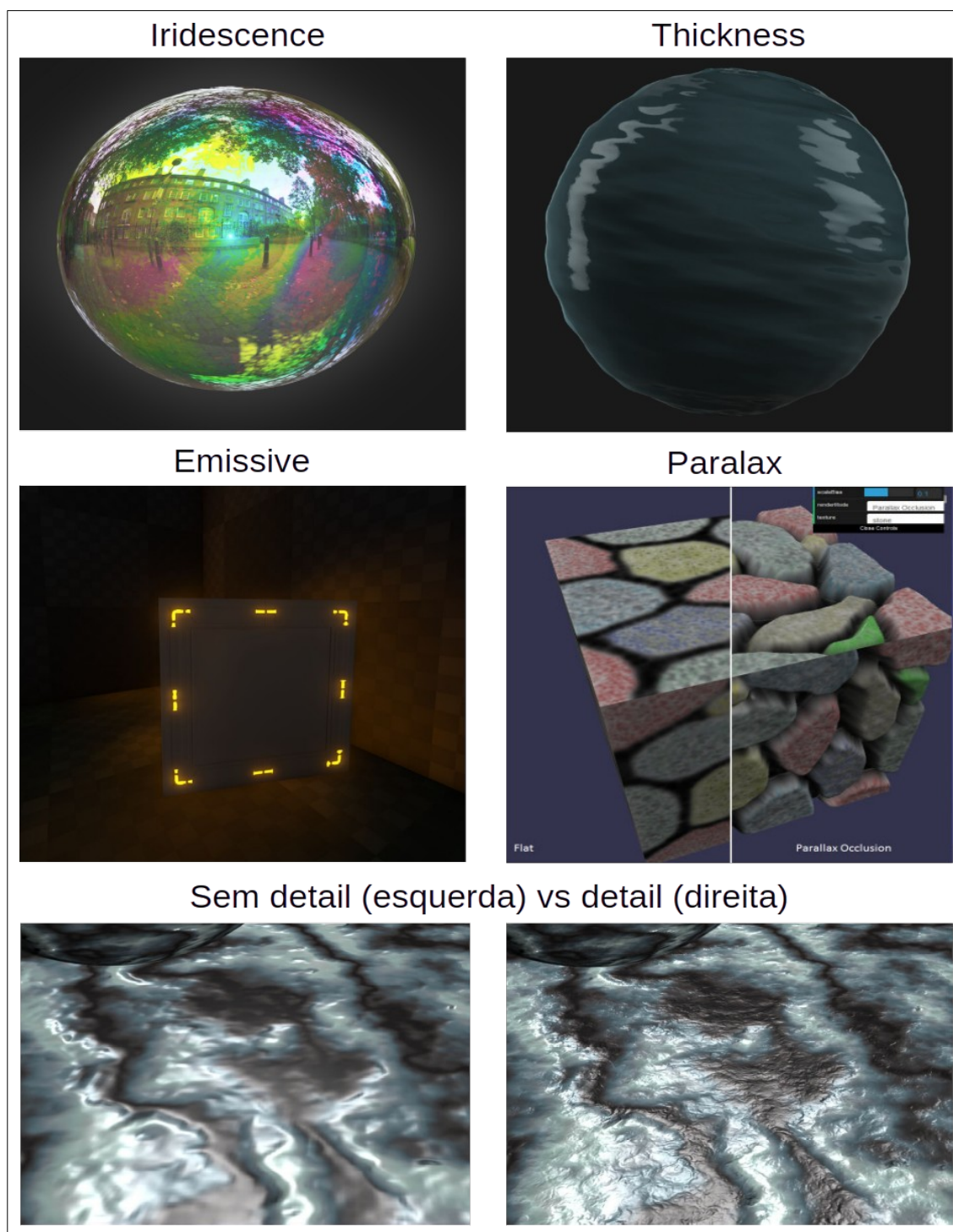


Figura 07 - Visualização das propriedades dos materiais PBR – Parte 2/3

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.

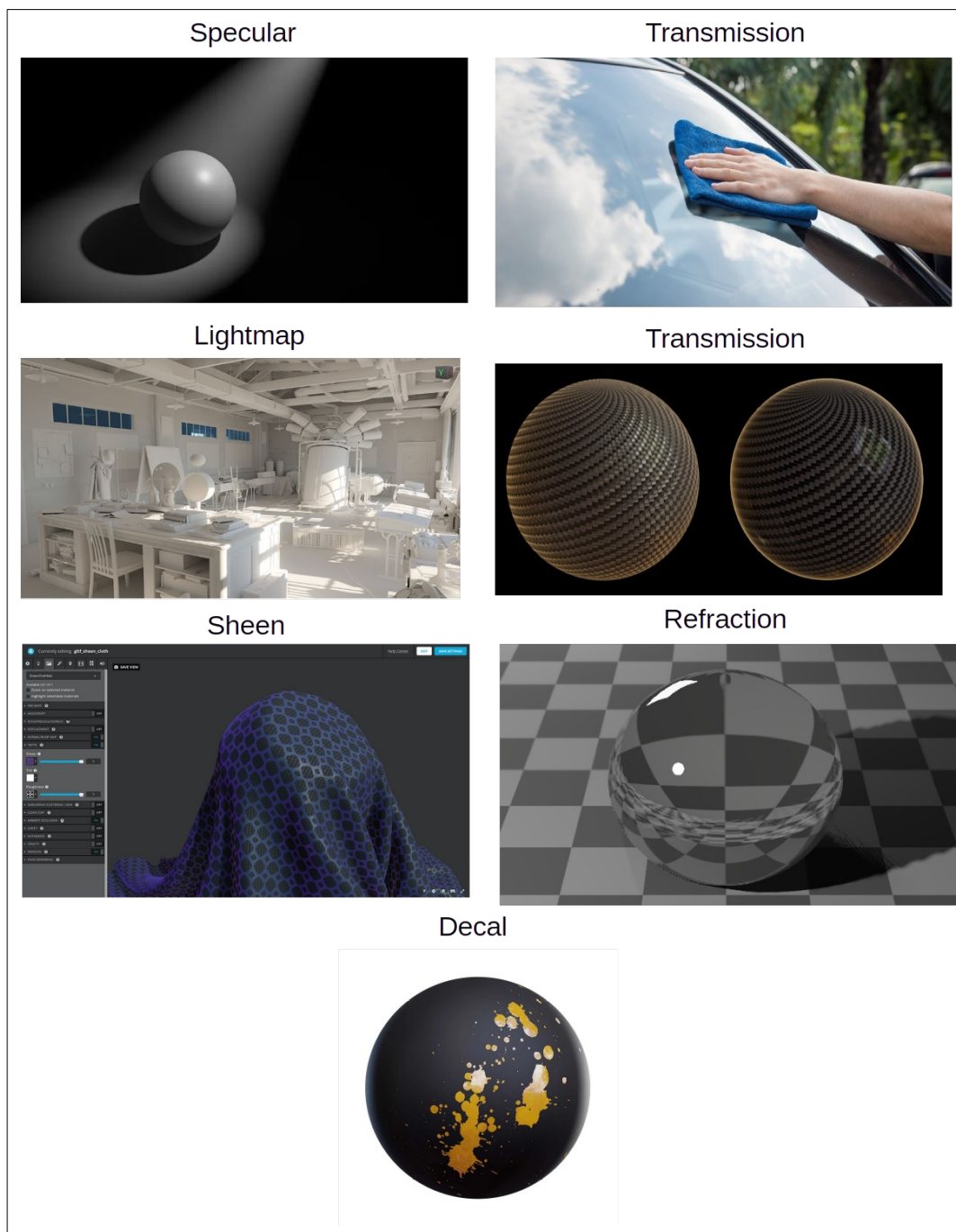


Figura 08 - Visualização das propriedades dos materiais PBR – Parte 3/3

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 C – Funcionalidades – Efeitos de pós-processamento

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

	(falha na câmera)			
Lens Distortion	Distorção da imagem pelo formato da lente	Plugin	X	
Lookup Tables	Mapear cores para outras a partir de valores pré-definidos	Plugin	X	

Quadro 08 - 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 e Unity 3D Docs (2023)

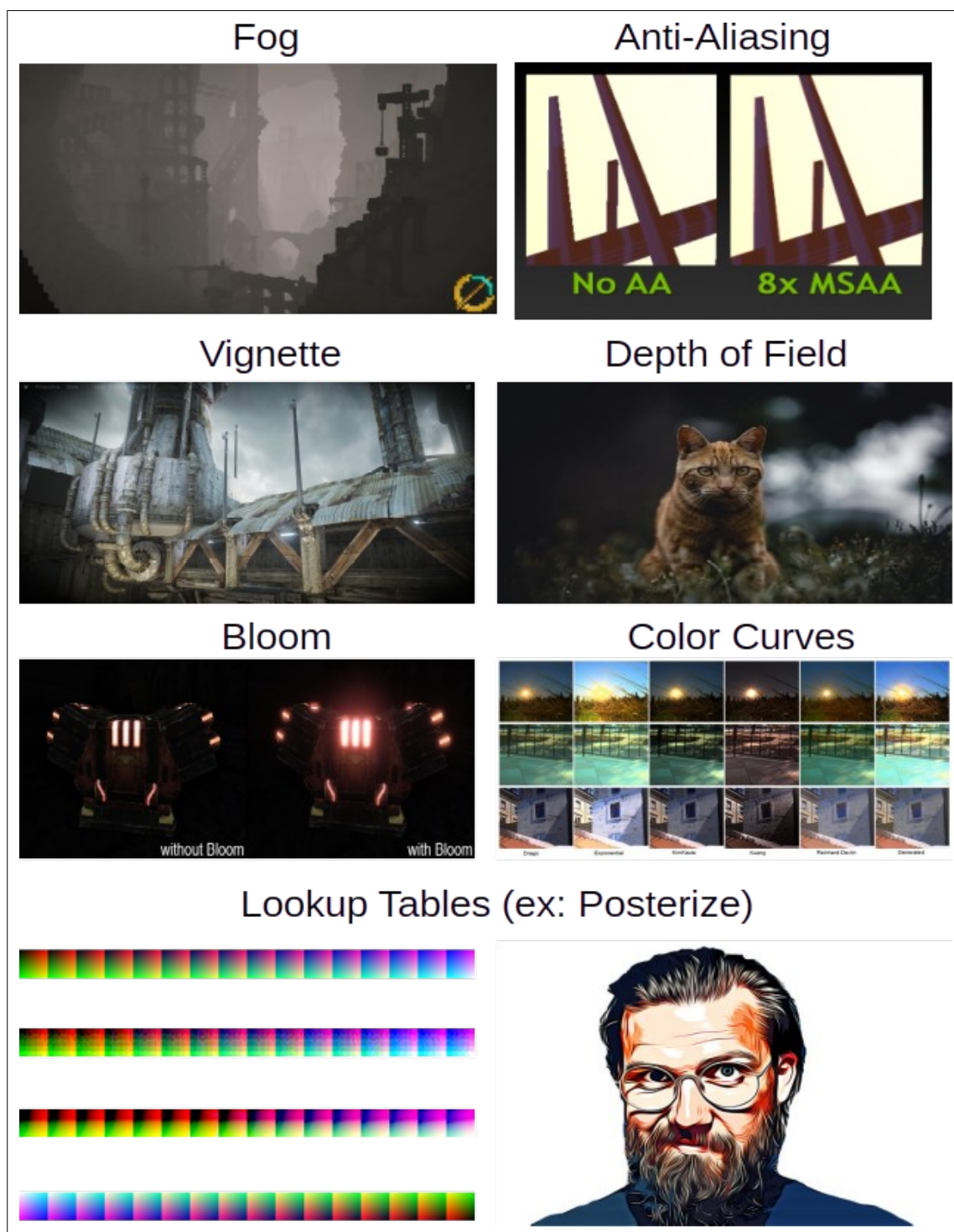


Figura 09 - Visualização das efeitos de pós-processamento – Parte 1/2

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.

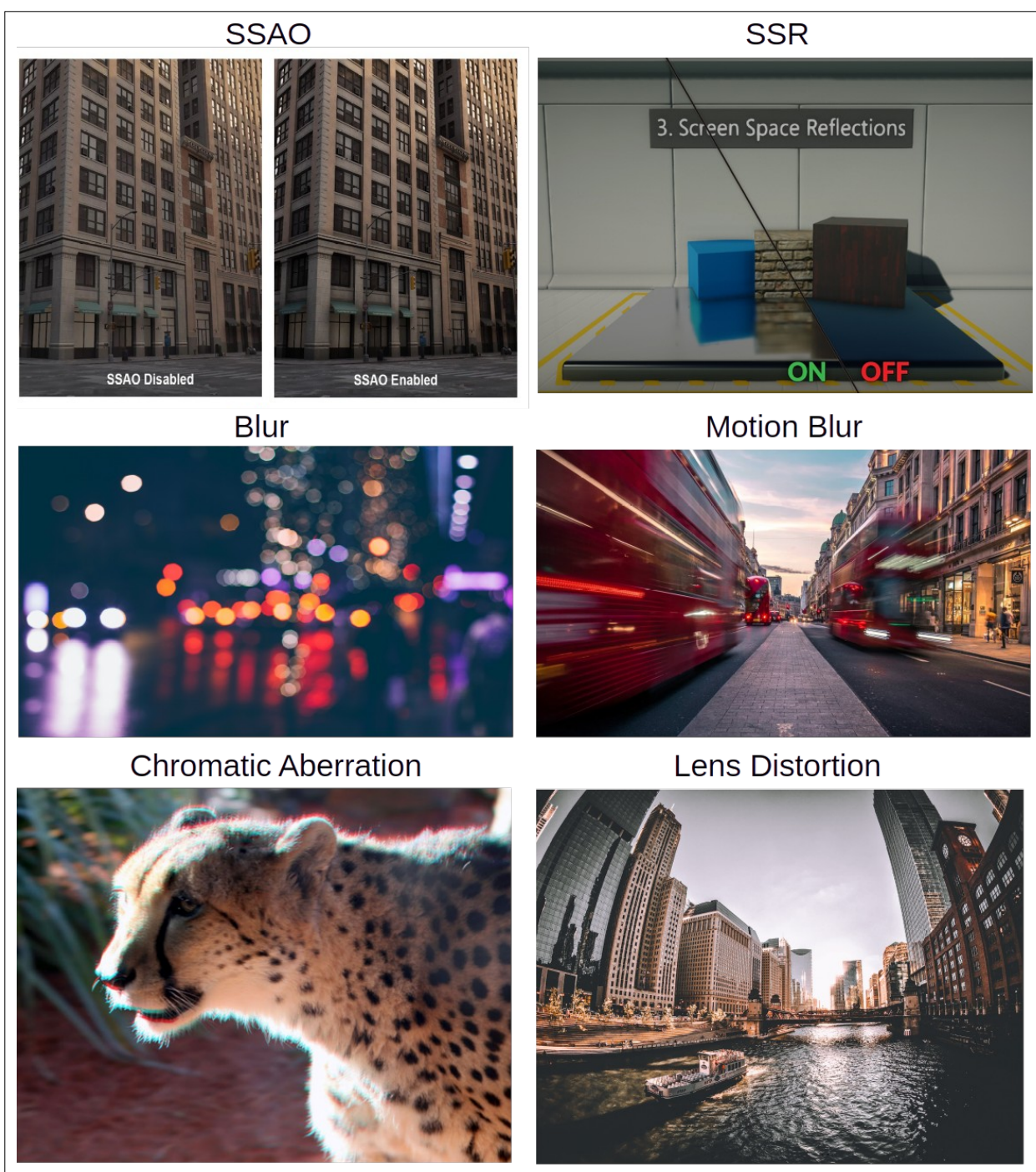


Figura 10 - Visualização das efeitos de pós-processamento – Parte 2/2

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 D – 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 Lightmaps			X
<i>NavMesh</i>	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 09 - Funcionalidades extras disponíveis por biblioteca - "Plugin" necessita de biblioteca externa para provisão

Fonte - Elaborado pelos autores (2023)

Apêndice E – Descrição dos Modelos e dos Testes de Desempenho

Rótulo	Contagem de Triângulos	Contagem de Objetos
Forte Pirata	90.334	7
Propulsor de Íons	52.072	6
Laboratório	83.665	608
Ilha	139.557	393
Crânio	1.126.158	26
Vale	2.076.694	1228
Vale (batch)	2.076.694	1
Deserto	42.074.389	10.662

Quadro 10 - Modelos 3D carregados nos testes - Número de triângulos e poligonais por cena

Fonte: Elaborado pelos autores (2023)

Rótulo	Modelo 3D utilizado	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 (batch)	Vale (batch)	1 Luz direcional – Sombras desligadas – PC 1 e 2
Vale (sombra)	Floresta	1 Luz direcional – Sombras ligadas – PC 1 e 2
Deserto	Deserto	1 Luz direcional – Sombras desligadas – PC 2
Deserto + sombra	Deserto	1 Luz direcional – Sombras ligadas – PC 2

Quadro 11 - Cenas para os teste de desempenho - Rótulo - Modelo 3D utilizado – Detalhes adicionais

Fonte: Elaborado pelos autores (2023)

Apêndice F – Painel de Gerenciamento dos testes

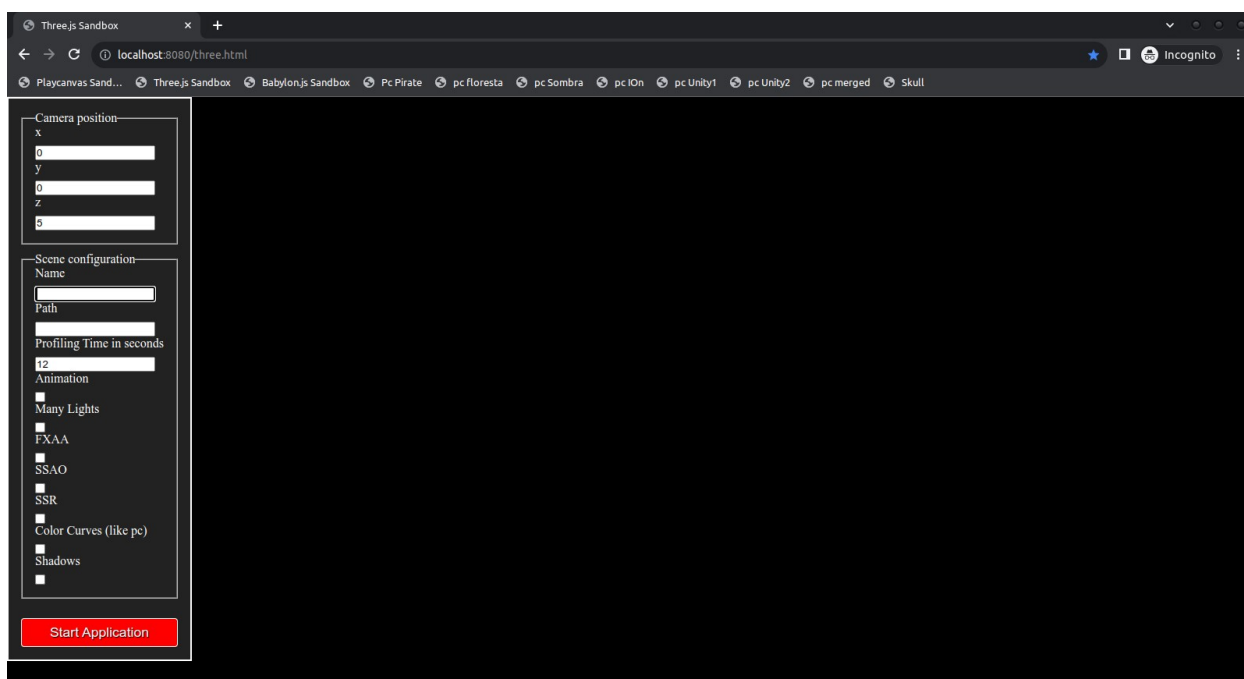


Figura 11 - Interface gráfica do painel de gerenciamentos dos testes de desempenho realizados no navegador

Fonte: Elaborado pelos autores (2023)

Apêndice G – Visualização das Cenas 3D

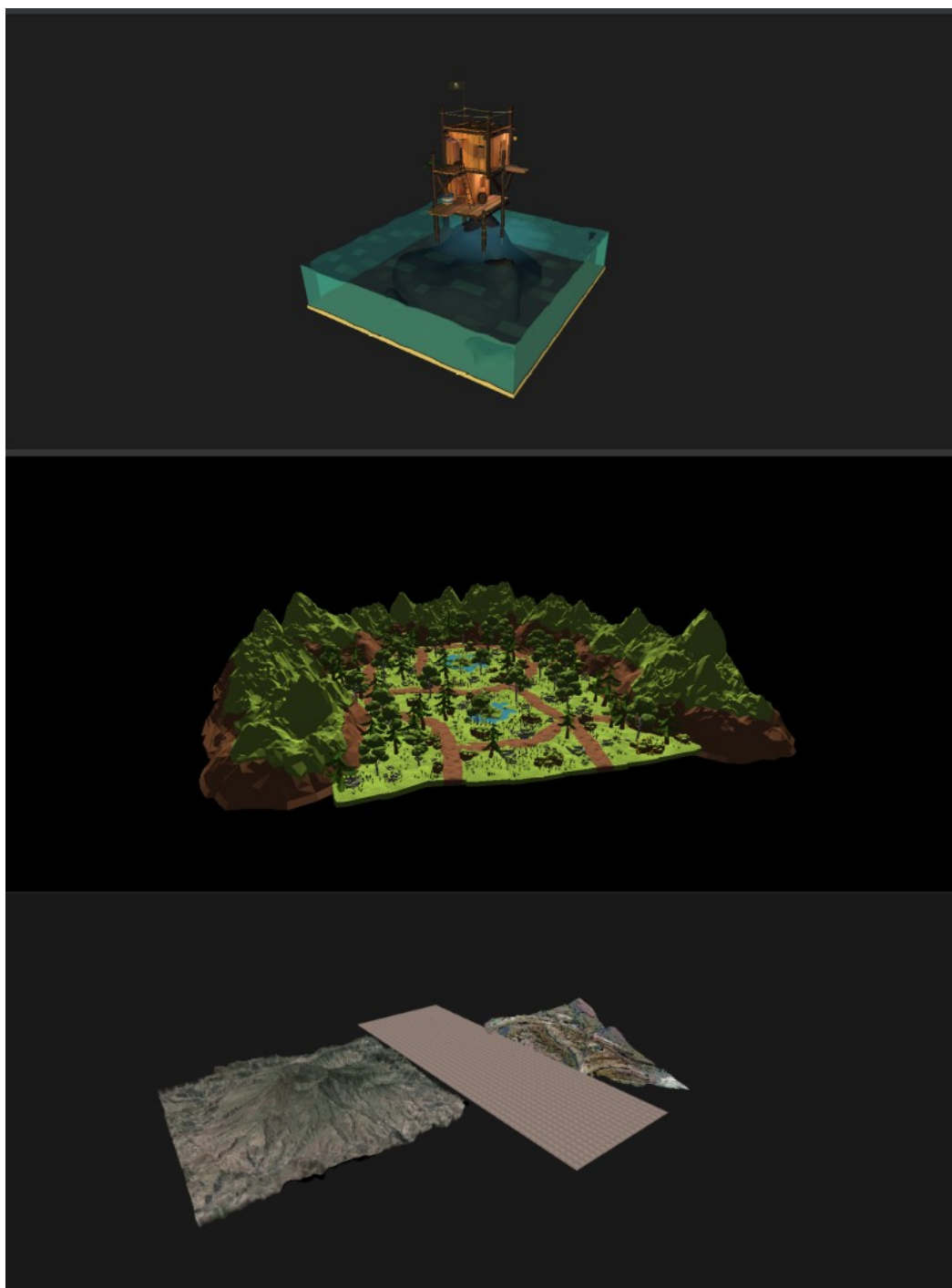


Figura 12 - Modelos 3D renderizados – Forte Pirata, Vale e Deserto

Fonte: Elaborado pelos autores (2023)

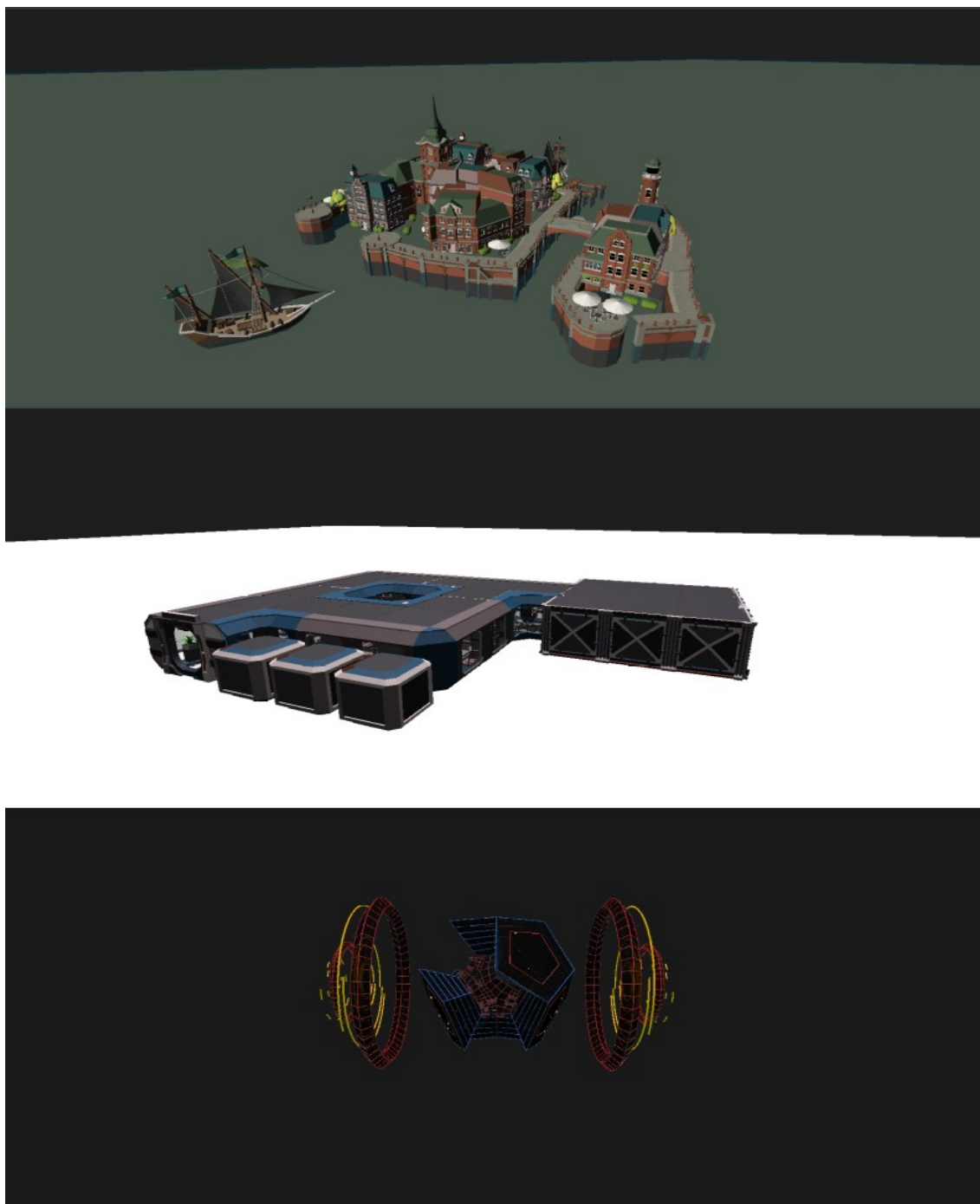


Figura 13 - Modelos 3D renderizados – Ilha, Laboratório (exterior) e Propulsor de Íons
Fonte: Elaborado pelos autores (2023)

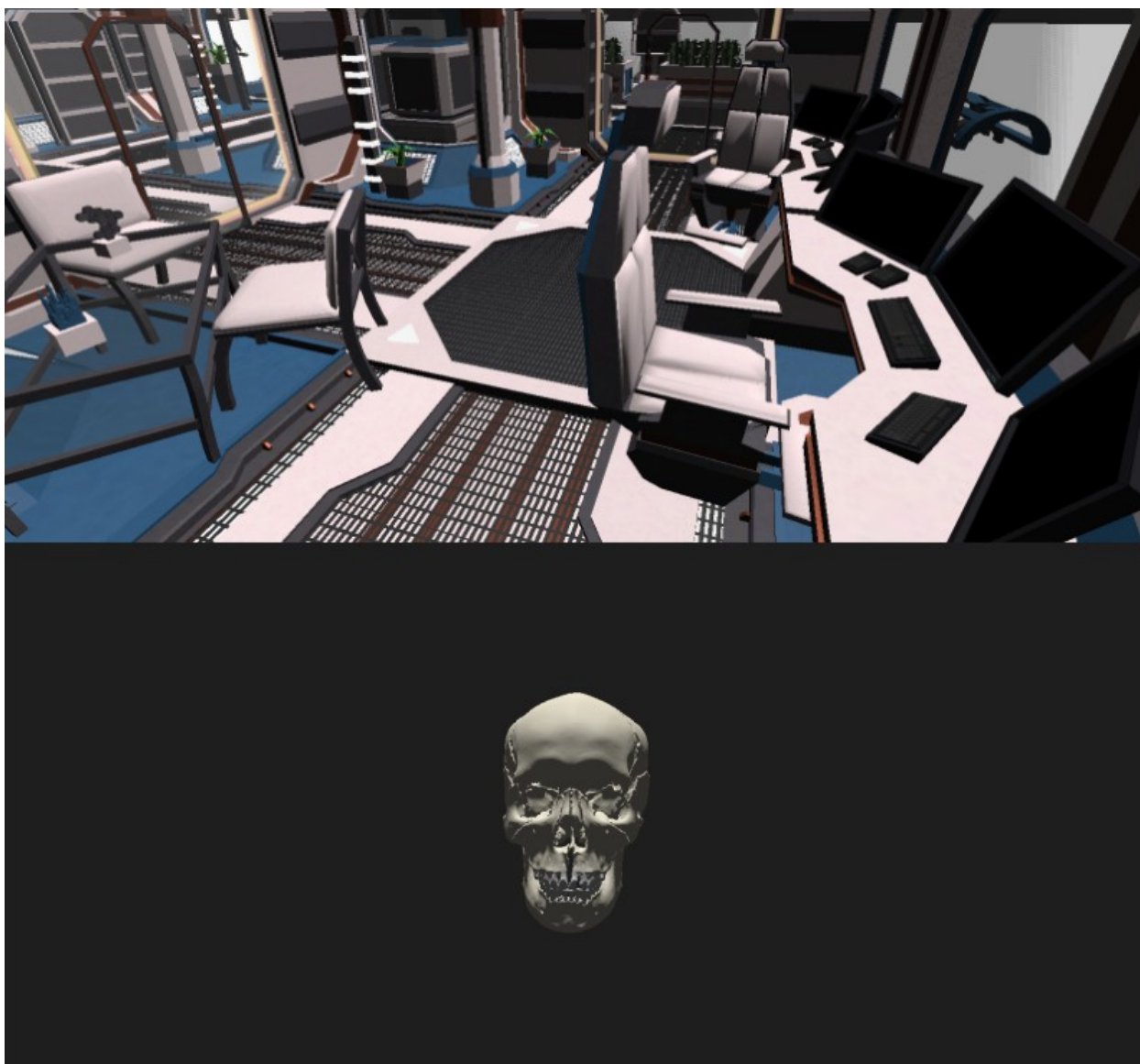


Figura 14 - Modelos 3D renderizados – Laboratório (interior) e Crânio
Fonte: Elaborado pelos autores (2023)

Apêndice H – Visualização dos Bugs e Falhas Visuais

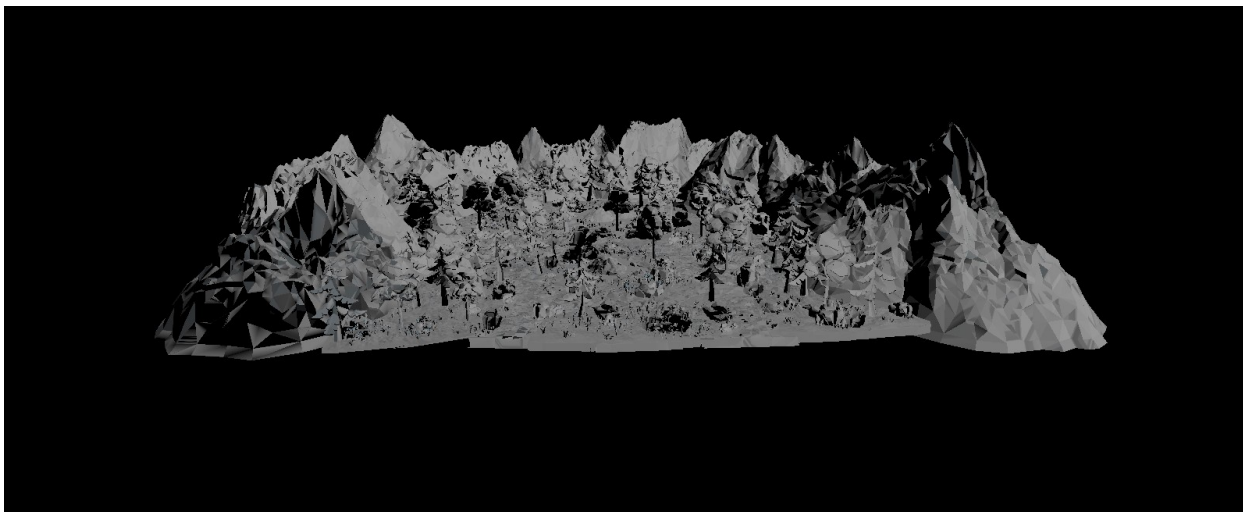


Figura 15 - Falha visual na renderização do teste "Vale + Sombra" - Three.js – PC-1 – Google Chrome

Fonte: Elaborado pelos autores: (2023)

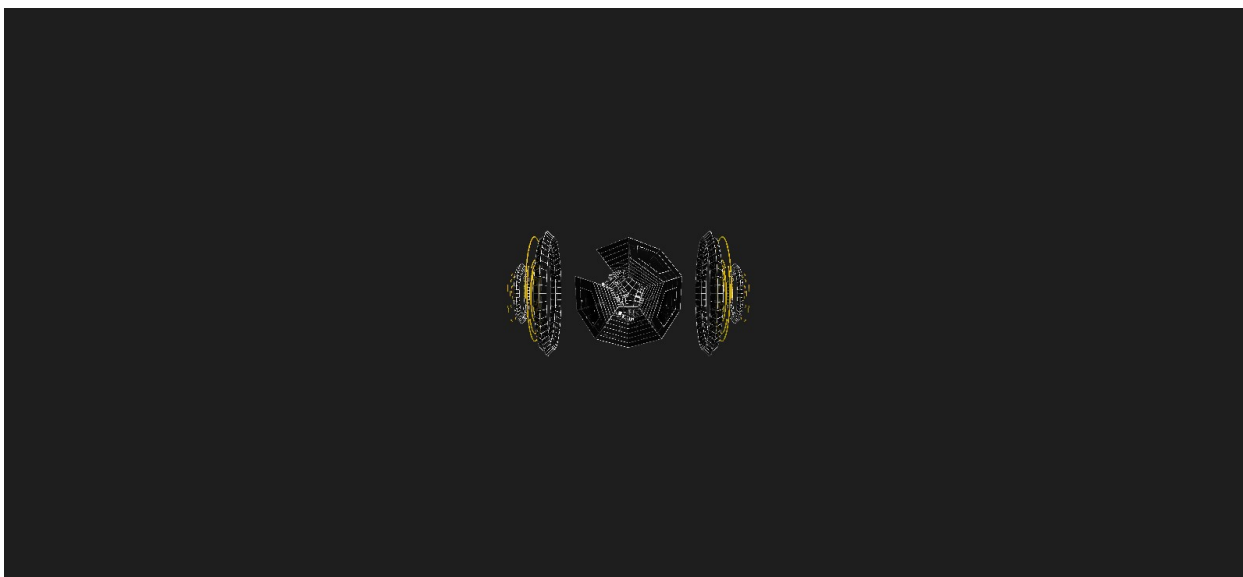


Figura 16 - Falha visual nas cores na renderização do teste "Propulsor" - Playcanvas (Editor) - PC-1 e PC-2 – Todos os navegadores

Fonte: Elaborado pelos autores: (2023)

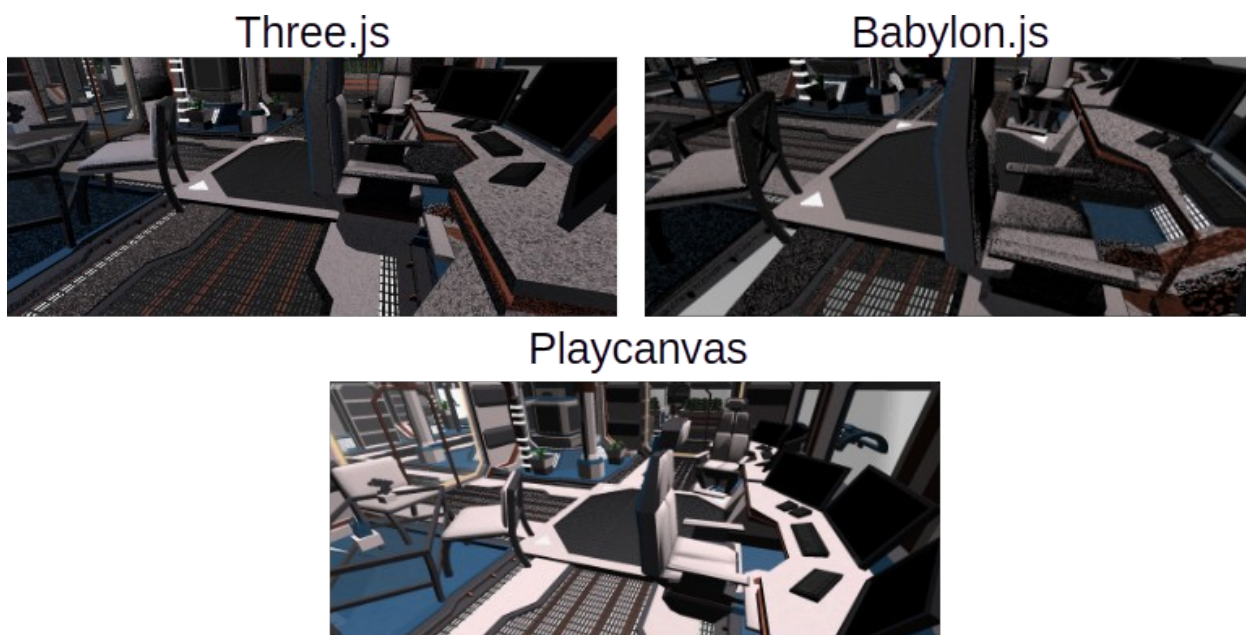


Figura 17 - A renderização do teste "Lab (interior)" apresenta estranhos artefatos quando renderizados pelas bibliotecas Three.js e Babylon.js – PC-1 e PC-2 – Todos os navegadores

Fonte: Elaborado pelos autores (2023)

Apêndice I – Dados de FPS coletados no Microsoft Edge