Análise Comparativa de Desempenho em FPS e Consumo de Memória RAM de Modelos 3D: Casos de Estudo com as Bibliotecas ThreeJS e BabylonJS

Hallyson Nobres Fernandes[1], **Ricardo Theis Geraldi (Orientador)**[1]

[1]Faculdade de Computação (FACOM) – Universidade Federal de Mato Grosso do Sul (UFMS)
79070-900 – Campo Grande – MS – Brasil

hallyson_nobres@ufms.br, ricardo.geraldi@ufms.br

Abstract. *With the growing use of 3D models on the Web, it becomes important to evaluate the performance of the libraries responsible for this rendering in order to verify the feasibility of their use in hardware-limited scenarios. This work aims to compare the performance between two JavaScript libraries, ThreeJS and BabylonJS, in the context of rendering three-dimensional models. The metrics used were Frames Per Second (FPS) and RAM memory consumption, as they represent the smoothness of rendering and the efficiency in the use of browser resources, which are essential aspects in 3D applications. As a result of the study, it was observed that ThreeJS presented superior performance in terms of FPS, while BabylonJS showed an advantage by avoiding high peaks in memory consumption, proving to be more efficient in this aspect.*

Resumo. *Com o crescimento do uso de modelos 3D na Web, torna-se importante avaliar o desempenho das bibliotecas responsáveis por essa renderização, a fim de verificar a viabilidade de seu uso em cenários de hardware limitado. Este trabalho tem como objetivo comparar o desempenho entre duas bibliotecas JavaScript, ThreeJS e BabylonJS, no contexto de renderização de modelos tridimensionais. As métricas utilizadas foram a quantidade de Frames por Segundo (FPS) e o consumo de memória RAM, por representarem a fluidez da renderização e a eficiência no uso de recursos do navegador, aspectos essenciais em aplicações 3D. Como resultado do trabalho, foi possível observar que o ThreeJS apresentou um desempenho superior em relação ao FPS, enquanto o BabylonJS demonstrou vantagem ao evitar picos elevados no consumo de memória, mostrando-se mais eficiente nesse aspecto.*

1. INTRODUÇÃO

Os modelos tridimensionais (3D) têm demonstrado uma crescente importância conforme se tornam cada vez mais sofisticados. Atualmente, os modelos 3D representam objetos da vida real com alta fidelidade, o que aumenta sua utilização em diversas áreas e traz novas possibilidades para os meios digitais. Consequentemente, a demanda por profissionais capacitados também vem crescendo, podendo ser uma boa área para quem busca novas oportunidades [Nishanbaev 2020].

Um modelo 3D é uma representação digital de um objeto, pessoa ou lugar em três dimensões (altura, largura e profundidade) e é amplamente utilizado em jogos digitais,

filmes, impressões 3D, dentre outros [Laviola Jr and Zeleznik 2005]. Com o desenvolvimento das técnicas de modelagem, modelos mais realistas e detalhados foram surgindo e também a necessidade dos modelos serem mais eficientes para entregarem maior fluidez durante as suas apresentações e até mesmo garantirem viabilidades técnicas. Pensando nisso, o desempenho de bibliotecas gráficas como BabylonJS e ThreeJS é crucial para manter uma alta taxa de Frames por Segundo (FPS) em apresentações de modelos 3D [Laviola Jr and Zeleznik 2005].

Com o crescimento do uso dos modelos 3D, cresceu a preocupação com os requisitos computacionais para se ter uma renderização fluída e agradável dos modelos. Um dos motivos dessa preocupação é o surgimento de modelos de grande escala, como réplicas de cidades e patrimônios históricos, que apresentam um nível elevado de detalhes [Trubka 2016]. A complexidade desses modelos eleva de forma significativa o custo computacional, exigindo cada vez mais capacidade de processamento dos computadores [Chen 2011].

As bibliotecas de renderização gráfica têm buscado maior otimização e eficiência, não apenas para melhorar o desempenho da renderização, mas também para possibilitar a exibição adequada de modelos 3D mais complexos. Nesse contexto, uma das métricas mais utilizadas para medir a fluidez da visualização é o FPS e para a eficiência dos modelos, um dado importante é o consumo de memória, medido em Megabytes (MB) [Sheng et al. 2017].

Diante desse cenário, serão analisadas duas das bibliotecas mais populares de renderização de modelos 3D em JavaScript: BabylonJS e ThreeJS. A comparação entre elas será realizada por meio da avaliação do desempenho considerando o uso de memória e quantidade de FPS atingidos durante a renderização dos modelos 3D [Söderberg 2020].

As métricas para as comparações foram obtidas através da função do JavaScript "requestAnimationFrame" para determinar os valores médios de FPS e para o consumo de memória foram utilizados os atributos da ferramenta Performance do navegador Google Chrome. Após obter esses valores, foi possível realizar uma análise do comportamento de cada biblioteca nas métricas avaliadas que será discutida nos capítulos seguintes [Söderberg 2020].

2. REVISÃO DA LITERATURA

Nesta seção são apresentadas definições e conceitos fundamentais para a compreensão deste trabalho, como Modelos Tridimensionais, Frames por segundo, Bibliotecas JavaScript, BabylonJS e ThreeJS, além de considerações importantes sobre suas aplicações no contexto da Web.

2.1. Modelos Tridimensionais (3D)

Desde a década de 1980, modelos tridimensionais já vinham sendo divulgados na internet, inicialmente de forma gratuita, com o objetivo de promover sua popularização. Com o tempo, passaram a surgir modelos 3D pagos, geralmente com qualidade superior aos gratuitos disponíveis. Esse valor agregado dos modelos 3D impulsionou ainda mais o desenvolvimento e a produção dos mesmos, bem como o avanço das tecnologias utilizadas em sua criação [Groenendyk 2016].

Entre 2005 e 2010, a demanda por modelos 3D concentrou-se em duas principais funções: (i) o desenvolvimento de modelos 3D para animações em vídeo; e (ii) a aplicação desses modelos em projetos CAD (*Computer-Aided Design*) direcionados à indústria, nos quais os modelos eram concebidos para se tornarem peças reais produzidas por máquinas. Esse período foi essencial para o amadurecimento das tecnologias disponíveis, impulsionado pela relevância dos modelos industriais [Groenendyk 2016]. Paralelamente, surgiram iniciativas com outros propósitos, como o site Plastic Boy (2024), que comercializa modelos anatômicos humanos e mantém suas atividades até os dias atuais [Plastic Boy 2024].

A partir de 2010, uma nova demanda surgiu com a popularização das impressoras 3D. Isso levou muitas pessoas a procurarem por modelos utilitários prontos para impressão doméstica. Esse movimento evidenciou ainda mais o valor dos modelos 3D, que passaram a ser utilizados para criar soluções únicas, muitas vezes inexistentes no mercado convencional. A comunidade passou, então, a compartilhar, comercializar e valorizar essas criações, aumentando a rentabilidade de quem produz modelos personalizados [Groenendyk 2016].

Um dos principais problemas enfrentados na comercialização dos modelos 3D era o processo de visualização, que ainda era limitado à interfaces pouco intuitivas. Para visualizar os modelos era necessário utilizar softwares específicos e lidar com linguagens técnicas pouco difundidas, o que criava uma barreira significativa. Mesmo com os avanços obtidos até então, ainda não era trivial conseguir visualizar um modelo de forma prática e rápida antes de adquiri-lo [Groenendyk 2016].

Essa realidade começou a mudar em 2011 com o surgimento da *Web Graphics Library* (WebGL), que possibilitou a incorporação de modelos 3D diretamente nos navegadores. Esse avanço só foi possível graças à evolução do hardware e ao aumento da largura de banda das conexões de internet, que permitiram o carregamento e a manipulação de conteúdos gráficos mais complexos [Nishanbaev 2020].

Neste contexto, a fim de compreender definições técnicas sobre os modelos 3D, é importante recorrer a definições técnicas de autores da área. Para Watt (1999), o principal objetivo da computação gráfica tridimensional é criar uma imagem bidimensional capaz de representar uma cena ou objeto tridimensional, utilizando uma descrição ou modelo desse mesmo objeto [Watt 1999]. Nesse contexto, LaViola (2005) define Modelo 3D como: "um modelo digital de um objeto ou ambiente em três dimensões, produzido em um software de computador baseado nos dados fornecidos" [Laviola Jr and Zeleznik 2005].

Os modelos tridimensionais podem ser criados de diversas formas: (i) por meio de programas CAD; (ii) com dispositivos como telêmetros a laser; ou (iii) utilizando digitalizadores tridimensionais. Essas formas são criadas manualmente ou de maneira semi-automatizada e, à medida que a complexidade dos modelos 3D aumentam, surgem métodos mais automatizados para facilitar sua criação [Watt 1999].

Algumas vantagens que os modelos 3D têm sobre imagens 2D estáticas é a possibilidade de rotacioná-los e visualizá-los sob diferentes ângulos. No campo da genética, por exemplo, modelos 3D são utilizados para representar genes com maior precisão em comparação às imagens estáticas [Gerth et al. 2007].

Além dessa aplicação, os modelos tridimensionais também são amplamente utilizados na visualização de mapas geográficos. Eles podem ser utilizados para planejamento urbano, apoio a resultados quantitativos e análises estratégicas [Trubka 2016]. A utilização de mapas 3D melhora a experiência dos usuários, proporcionando interações mais precisas, o que facilita a localização de pontos específicos e a navegação em ambientes complexos [Chen 2011].

Apesar de suas vantagens, a criação de modelos 3D pode ser um processo complexo e pode demandar um alto nível de dedicação. A modelagem manual exige experiência para garantir precisão e manter fidelidade à realidade [Chen 2011].

Além das dificuldades de criação dos modelos, sua renderização nos navegadores também pode ser dificultosa, principalmente, em processar e exibir grandes volumes de dados gráficos simultaneamente. Uma das consequências desse cenário resulta em baixa fluidez na visualização dos modelos [Ye et al. 2023].

Para monitorar e aprimorar o desempenho da renderização dos modelos, passou-se a medir a taxa de frames por segundo e a adotar outras soluções como a utilização da WebGL e algoritmos de otimização gráfica [Ye et al. 2023].

A WebGL trouxe uma revolução ao permitir a renderização 3D na web sem a necessidade de plugins, sendo suportada nativamente pelos principais navegadores. No entanto, por ser uma API de baixo nível, apresentava uma curva de aprendizado acentuada. Para contornar essa limitação, surgiram diversos frameworks (como ThreeJS e BabylonJS) que simplificaram seu uso e aceleraram o desenvolvimento de aplicações com modelos 3D na web [Nishanbaev 2020].

2.2. Frames Por Segundo

Frames por segundo é a taxa que representa quantas imagens são exibidas por segundo. Essa quantidade está diretamente ligada à sensação de interatividade, pois, quando há pouca fluidez, torna-se perceptível a transição entre cada nova imagem. De acordo com Akenine-Möller (2018), a partir de 6 FPS já é possível observar um início de continuidade das imagens, embora ainda com evidente descontinuidade. Com 30 FPS, a qualidade de visualização é considerada adequada para a maior parte das aplicações. Valores próximos a 60 FPS indicam um nível de alta responsividade na atualização das imagens. Em cenários que demandam elevada responsividade, como determinados jogos ou aplicações de interação com o mundo real, busca-se chegar o mais próximo do limite de hardware do usuário, que pode incluir taxas de 90, 120, 240, 360 FPS ou mais, a depender da capacidade do dispositivo [Akenine-Möller et al. 2018].

Taxas mais altas de FPS indicam uma maior quantidade de imagens exibidas por segundo. Essa taxa está diretamente relacionada ao tempo de inferência, que representa o intervalo necessário para o processamento de uma nova imagem. Assim, quanto menor o tempo de inferência, mais imagens podem ser geradas por segundo e, consequentemente, maior tende a ser o FPS [Rangkuti et al. 2023].

No contexto de modelos 3D na web, o FPS é uma unidade de medida fundamental para o monitoramento do desempenho na renderização. Ele pode ser utilizado para comparar a performance de diferentes modelos 3D gerados e executados por meio de bibliotecas JavaScript. As bibliotecas ThreeJS e BabylonJS renderizam modelos tridimensionais

diretamente nos navegadores e serão detalhadas na próxima seção [Sheng et al. 2017].

2.3. Bibliotecas Javascript

As bibliotecas desenvolvidas na linguagem JavaScript representam um conjunto de funcionalidades previamente implementadas que visam facilitar o desenvolvimento de aplicações, permitindo a reutilização de soluções já consolidadas. Essas funcionalidades vão desde a manipulação e formatação de dados, até recursos mais avançados, como métodos que auxiliam na criação de modelos 3D [Akenine-Möller et al. 2018].

Entre essas bibliotecas, existem diversas que possibilitam a utilização e criação de modelos 3D para visualização dinâmica na web, como BabylonJS, ThreeJS, CesiumJS, sceneJS, OSGJS, CannonJS, entre outras [Nishanbaev 2020].

Dentre as bibliotecas disponíveis, Three.js e Babylon.js se destacam por serem muito populares e amplamente utilizadas para a criação e exibição de modelos 3D em navegadores. Ambas oferecem os recursos mais atuais e contam com comunidades ativas que contribuem continuamente para o seu desenvolvimento [Nishanbaev 2020].

2.3.1. BabylonJS

Lançada em 2013 pela Microsoft, BabylonJS é uma biblioteca de renderização 3D desenvolvida em JavaScript, voltada para a criação de gráficos imersivos e interativos na web. A ferramenta permite o desenvolvimento de diversos tipos de aplicações tridimensionais, desde jogos até modelos urbanos complexos, oferecendo suporte a múltiplos formatos de arquivos e funcionalidades modernas de renderização 3D [BabylonJS Documentation 2024].

A biblioteca também é compatível com tecnologias como a WebGL, o que proporciona um desempenho otimizado mesmo em computadores com hardware modesto e em dispositivos móveis. Além disso, BabylonJS oferece suporte para aplicações em diversas áreas como desenvolvimento de jogos, visualização de dados, simulações, realidade aumentada (RA), realidade virtual (RV), dentre outras [BabylonJS Documentation 2024].

A Figura 1 apresenta um exemplo de modelo 3D implementado com BabylonJS, que ilustra a capacidade da biblioteca em produzir e renderizar modelos 3D com alto nível de qualidade e complexidade.

A Figura 1 mostra a representação de uma moto com um elevado grau de fidelidade em relação ao modelo real e com uma alta qualidade gráfica.

Inicialmente, foram executados os primeiros passos da documentação no próprio ambiente de desenvolvimento *online* da biblioteca, chamado de Babylon Playground. Por exemplo, a Figura 2 apresenta o código-fonte escrito seguindo os primeiros passos da documentação do Babylon Playground e seu resultado.

Na Figura 2, é possível observar que algumas etapas podem ser seguidas para criar um modelo 3D com a biblioteca. Primeiramente, cria-se uma cena inicialmente vazia; em seguida, adiciona-se uma câmera, responsável pelo ângulo de visualização do modelo; e, a iluminação (uma luz) que funciona como uma lanterna direcionada para um ponto, fundamental para gerar sombras nos objetos. Após a configuração desse ambiente 3D, é

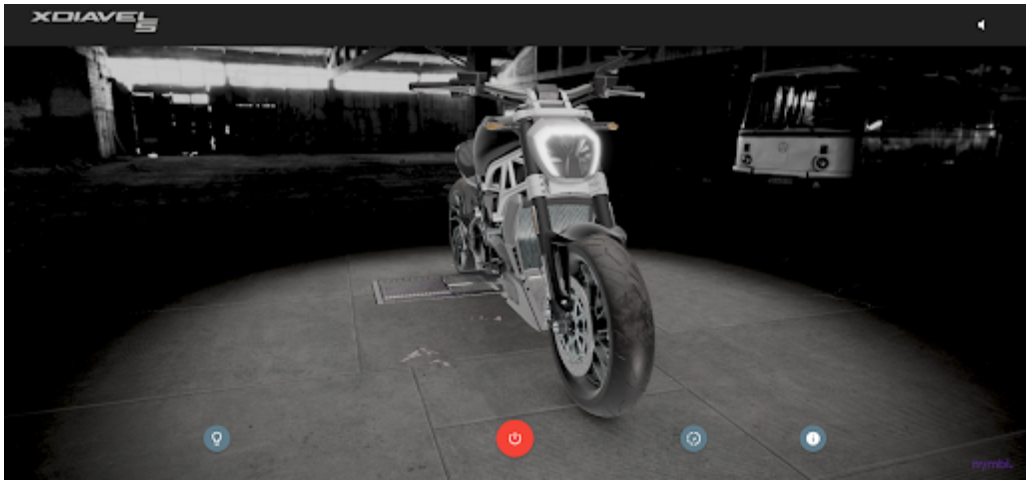

Figura 1. Modelo 3D de uma moto implementada com BabylonJS (adaptado de Ducati XDiavel (2024)) [Ducati XDiavel 2024].

```
var createScene = function () {
    // This creates a basic Babylon Scene object (non-mesh)
    var scene = new BABYLON.Scene(engine);

    // This creates and positions a free camera (non-mesh)
    var camera = new BABYLON.FreeCamera("camera1", new BABYLON.Vector3(0, 5, -10), scene);

    // This targets the camera to scene origin
    camera.setTarget(BABYLON.Vector3.Zero());

    // This attaches the camera to the canvas
    camera.attachControl(canvas, true);

    // This creates a light, aiming 0,1,0 - to the sky (non-mesh)
    var light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0, 1, 0), scene);

    // Default intensity is 1. Let's dim the light a small amount
    light.intensity = 0.7;

    // Our built-in 'ground' shape.
    var ground = BABYLON.MeshBuilder.CreateGround("ground", {width: 6, height: 6}, scene);
    let groundMaterial = new BABYLON.StandardMaterial("Ground Material", scene);
    ground.material = groundMaterial;
    let groundTexture = new BABYLON.Texture(Assets.textures.checkerboard_basecolor_png.path, scene);
    groundMaterial.diffuseColor = BABYLON.Color3.Red();
    ground.material.diffuseTexture = groundTexture;

    BABYLON.SceneLoader.ImportMesh("", Assets.meshes.Yeti.rootUrl, Assets.meshes.Yeti.filename, scene, function(newMeshes){
        newMeshes[0].scaling = new BABYLON.Vector3(0.1, 0.1, 0.1);
    });

    return scene;
};
```

Figura 2. Exemplo do código inicial com BabylonJS no Babylon Playground (Adaptado de BabylonJS First Steps (2024)) [BabylonJS First Step 2024].

inserido um “chão”, que servirá de base para o objeto 3D principal. Por fim, adiciona-se o objeto escolhido pela documentação: um Yeti. Todos esses componentes contribuem para a criação do modelo final e possuem diversos parâmetros capazes de transformar completamente a cena [BabylonJS Documentation 2024]. O código-fonte da Figura 2 produz o seguinte modelo 3D.

O modelo 3D exibido na Figura 3 na primeira experiência com a biblioteca é útil para apresentar ao usuário uma primeira impressão de como usar a biblioteca e como funciona sua estruturação e sintaxe. A seguir, será detalhada a biblioteca ThreeJS.

Figura 3. Modelo 3D resultante do código do BabylonJS citado previamente (Adaptado de BabylonJS First Steps (2024)) [BabylonJS First Step 2024].

2.3.2. ThreeJS

O ThreeJS é uma biblioteca JavaScript criada em 2010 por Ricardo Cabello com o objetivo de trazer modelos 3D para a web. Inicialmente, a biblioteca foi projetada para utilizar vetores SVG (*Scalable Vector Graphics*) e imagens bitmap na renderização dos modelos. Entretanto, em 2011, com a chegada da WebGL, o foco da ThreeJS mudou para se tornar uma biblioteca mais voltada a simplificar o desenvolvimento de modelos 3D com WebGL [Dirksen 2015].

O WebGL é uma API de baixo nível para modelagens 3D, o que dificulta a criação de novos modelos. Para resolver esse problema, o ThreeJS simplificou a utilização do WebGL, o que fez com que houvesse um crescimento considerável na aceitação da comunidade para essa nova tecnologia [Dirksen 2015].

Referente às suas capacidades atuais, a biblioteca oferece uma ampla gama de recursos e funcionalidades, incluindo geometria, texturas, iluminação, sombras, animações, shaders personalizados, física e interações em tempo real. Esses recursos possibilitam a criação de projetos que vão desde visualizações 3D simples até games e simulações complexas [Three.js Documentation 2024].

Além disso, o ThreeJS é open-source, possui uma grande comunidade de desenvolvedores, que faz a biblioteca acompanhar as mais recentes atualizações das tecnologias de renderização 3D na web. Um exemplo disso é a integração com inteligência

artificial, como o ChatGPT, que pode atuar como instrutor para usuários da ThreeJS, auxiliando na solução de dúvidas, fornecendo instruções e até corrigindo erros em códigos [ThreeJS Mentor 2024].

Para exemplificar a capacidade do ThreeJS, a Figura 4 ilustra um exemplo de renderização de modelos 3D complexos, evidenciando sua capacidade de exibir modelos com alta qualidade e riqueza de detalhes.

Figura 4. Modelo 3D de um carro implementado com ThreeJS (Adaptado de Faraday Future (2024)) [Faraday Future 2024].

Em relação à utilização da biblioteca no desenvolvimento de modelos 3D, foi desenvolvido um código seguindo o passo a passo da própria documentação da biblioteca ThreeJS para visualizar um pouco da sua complexidade e executado localmente. Segue o exemplo do código inicial com ThreeJS na Figura 5:

É possível observar na Figura 5 que a estruturação da biblioteca ThreeJS possui alguns componentes que seguem a mesma lógica do BabylonJS, o que facilita a compreensão de termos comuns, como cena, câmera, entre outros.

Ambas as bibliotecas iniciam com a criação de uma cena vazia, seguida da adição de uma câmera, que define a posição a partir da qual o usuário visualizará o modelo. Em seguida, adiciona-se o renderizador da cena, cria-se o objeto geométrico utilizado nos primeiros passos, aplica-se a textura sobre o objeto e, assim, obtém-se o objeto 3D principal. Posteriormente, é implementada uma animação que faz o objeto girar sobre seu próprio eixo. Apesar de ambas as bibliotecas serem relativamente simples de compreender, a grande quantidade de parâmetros disponíveis pode gerar uma curva de aprendizado acentuada.

Ao executar o código apresentado na Figura 5, obteve-se o modelo 3D ilustrado na Figura 6, que consiste em um cubo em rotação. Embora esse exemplo do ThreeJS pareça mais simples que o do BabylonJS, ele representa apenas os primeiros passos descritos na documentação, servindo como base para compreender de forma inicial como a biblioteca manipula objetos 3D.

A partir dessa experiência inicial com ambas as bibliotecas, torna-se possível avançar para uma análise de desempenho, considerando aspectos técnicos relevantes para

main.js

```
import * as THREE from "three";

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(
  75,
  window.innerWidth / window.innerHeight,
  0.1,
  1000
);

const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setAnimationLoop(animate);
document.body.appendChild(renderer.domElement);

const geometry = new THREE.BoxGeometry(1, 1, 1);
const material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
const cube = new THREE.Mesh(geometry, material);
scene.add(cube);

camera.position.z = 5;

function animate() {
  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;

  renderer.render(scene, camera);
}
```

Figura 5. Exemplo do código inicial com ThreeJS (Adaptado de ThreeJS Creating Scene (2024)) [ThreeJS Creating Scene 2024].

o desenvolvimento de aplicações 3D na web, como o FPS e o consumo de memória.

No decorrer deste trabalho, será realizada uma comparação de desempenho entre as bibliotecas BabylonJS e ThreeJS, considerando a taxa de frames por segundo e o consumo de memória durante a renderização de dois modelos 3D. A partir dos resultados, espera-se identificar as principais vantagens e desvantagens de cada uma no contexto do desenvolvimento de aplicações web com recursos tridimensionais.

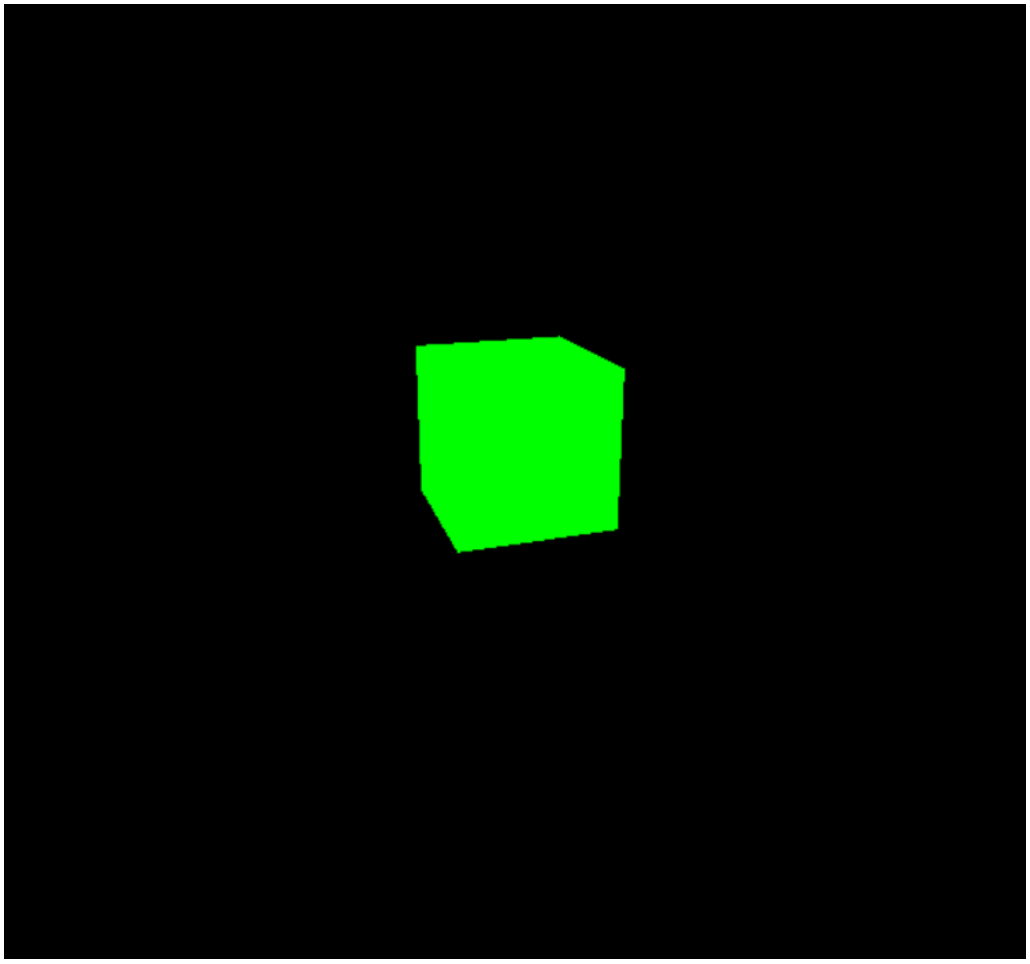

Figura 6. Modelo 3D resultante do código do ThreeJS citado anteriormente (Adaptado de ThreeJS Creating Scene (2024)) [ThreeJS Creating Scene 2024].

3. ESTRUTURAÇÃO DA PESQUISA

As etapas deste trabalho foram adaptadas para atender às necessidades da análise comparativa entre as bibliotecas JavaScript BabylonJS e ThreeJS, seguindo uma adaptação das etapas propostas pelo protocolo experimental de Wohlin et al. (2024) [Wohlin et al. 2024]. Assim, a pesquisa é organizada em três fases principais: Planejamento, Execução e Análise e Interpretação dos Resultados. A Figura 7 apresenta as etapas desta pesquisa.

3.1. Planejamento

O planejamento envolve atividades relacionadas à revisão da literatura com o objetivo de consolidar o conhecimento das áreas relacionadas à pesquisa. Nessa etapa, foram estudados o histórico e as definições dos modelos 3D, o conceito de FPS e sua aplicação dentro do contexto da modelagem tridimensional, além das bibliotecas em JavaScript utilizadas para manipular esses modelos. Também foram analisadas algumas ferramentas para medir o FPS dos modelos 3D nos navegadores e como quantificar o uso de memória dos mesmos.

Outra atividade fundamental do planejamento foi a construção inicial de modelos

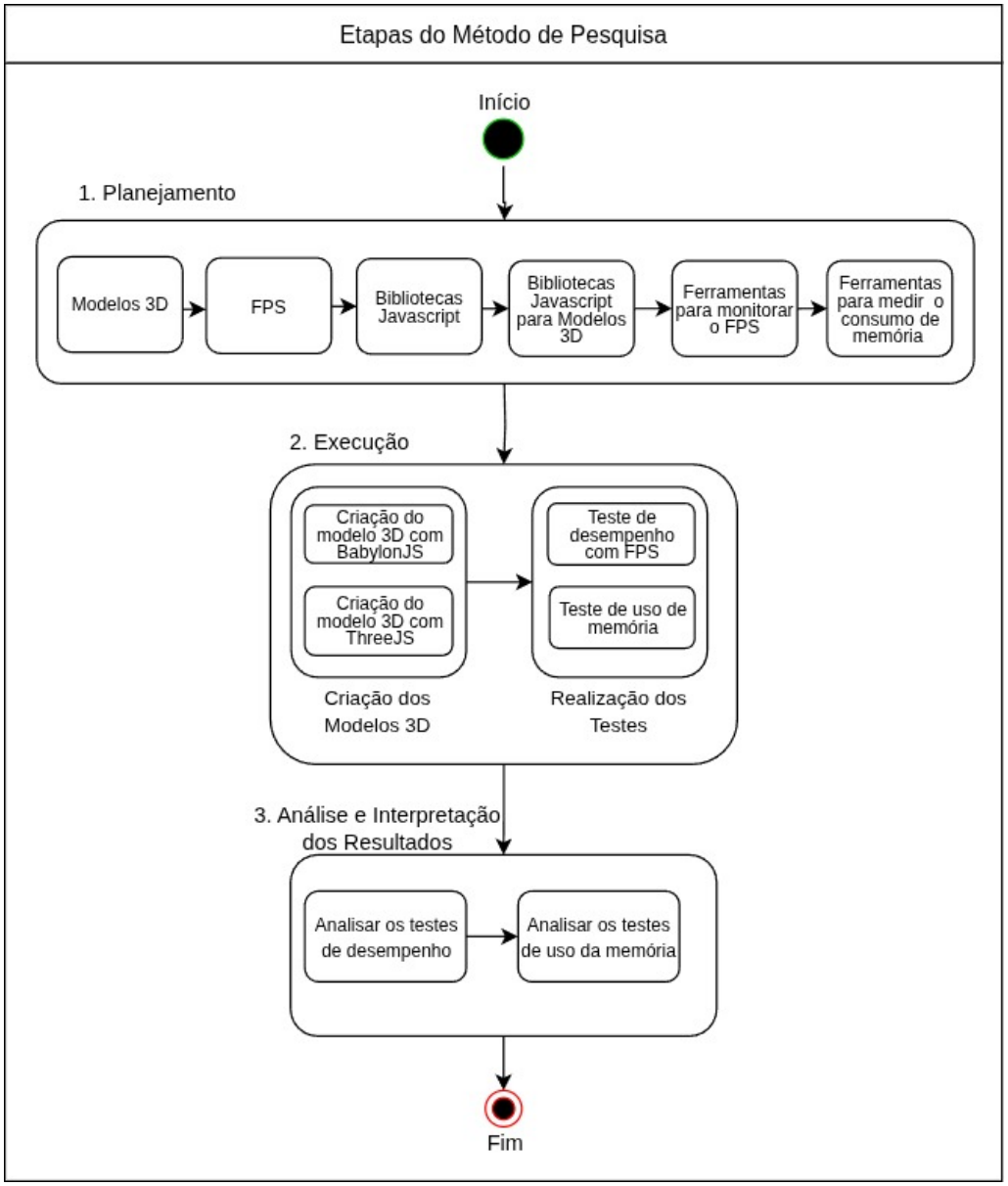

Figura 7. Etapas do Método de Pesquisa (Autoria Própria com etapas adaptadas de Wohlin) [Wohlin et al. 2024].

3D com as bibliotecas BabylonJS e ThreeJS. Esse primeiro contato prático permitiu compreender o funcionamento básico de cada biblioteca e identificar potenciais dificuldades para o comparativo.

3.2. Execução

Para a fase de execução dos testes de desempenho de FPS e memória, foram escolhidos dois modelos 3D gratuitos. O primeiro foi o Low Poly Tree Scene Free (2024), considerado um modelo de tamanho médio com 377,8 mil vértices, 256,7 mil triângulos, 2.713 malhas e tamanho de 51 MB [Low Poly Tree Scene Free 2024]. O segundo modelo escolhido foi o Hintze Hall (2024), considerado um modelo grande com 739 mil vértices, 1,3 milhão de triângulos, 29 malhas e tamanho de 234 MB [Hintze-Hall 2024]. Não foram

incluídos modelos 3D menores, porque a variação de FPS era quase imperceptível, o que inviabilizaria a análise de desempenho das bibliotecas.

Para a execução dos testes foi criado um arquivo HTML contendo apenas o essencial de CSS e JavaScript para exibir cada modelo em cada biblioteca individualmente com o mínimo de funcionalidades adicionais. Alguns recursos básicos das bibliotecas foram adicionados para possibilitar a visualização dos modelos, como:

- Luz ambiente e direcionada: adicionadas em ambas bibliotecas para que pudesse haver uma visualização agradável dos modelos.
- Câmeras: foram utilizadas as câmeras mais simples disponíveis, que não permitem interação do usuário para alterar a direção da câmera. Elas apenas se movem em círculo, possibilitando a visualização completa dos modelos. Essa escolha levou em consideração tanto a dificuldade de replicar os mesmos movimentos nos testes de execução quanto o fato de o ThreeJS utilizar uma biblioteca secundária (OrbitControls) para movimentação interativa, o que poderia influenciar nos resultados.
- Posicionamento das câmeras: devido às diferenças no ponto inicial de renderização em cada biblioteca, foram feitas adequações na posição das câmeras individualmente para melhorar a visualização dos modelos.

Nos arquivos de testes dos modelos, foram implementadas duas funções em JavaScript para registrar as métricas de FPS e de consumo de memória durante a renderização dos modelos 3D nos testes de desempenho.

O cálculo do FPS foi implementado diretamente em JavaScript, utilizando a diferença de tempo entre frames consecutivos através da função nativa “requestAnimationFrame” para poder estimar a taxa média de FPS. Essa abordagem foi escolhida para evitar o uso de bibliotecas externas que poderiam causar atrasos ou divergências nos valores retornados e impactar negativamente o resultado final.

O FPS foi metrificado por meio da quantidade de vezes que a função de animação do modelo era chamada. A contagem foi realizada de acordo com as chamadas de renderização do modelo a cada segundo e armazenadas em um array até ter 60 valores, correspondentes a 60 segundos de execução. O procedimento foi aplicado a cada biblioteca (BabylonJS e ThreeJS) e a cada modelo 3D de teste (Low Poly Tree e Hintze Hall).

Para a avaliação do consumo de memória, foi utilizada a funcionalidade de performance do Chrome (disponível apenas em navegadores Chromium), o atributo “performance.memory.usedJSHeapSize”. Esse atributo refere-se ao uso de memória RAM pelo JavaScript durante a execução do navegador. O valor obtido não refere-se ao consumo de memória apenas do modelo mas de todo o JavaScript presente na página que o exibe. Contudo, por não haver outros componentes ou funcionalidades na página, esse valor foi considerado como representativo do consumo de cada biblioteca. Ressalta-se que o objetivo principal é observar o comportamento desse valor entre as bibliotecas e não o real consumo de memória de cada modelo em si.

Os arquivos de teste foram executados localmente a partir do comando “npx vite”. Essa abordagem foi necessária para permitir a conexão com as bibliotecas de renderização

de modelos 3D sem restrições de CORS (*Cross-Origin Resource Sharing*), que ocorreram ao tentar importar as bibliotecas sem um servidor local.

A execução dos testes foi realizada em um computador desktop com processador Ryzen 7 5700G, 48 GB de memória RAM, sistema operacional Ubuntu 24.04.3 LTS e um monitor de 120 Hz (Figura 8). Durante a execução dos testes, foi mantida apenas a aplicação rodando no VS Code, com o computador em modo de desempenho e o navegador Google Chrome apenas com a aba do modelo aberta. Assim, para a execução dos testes de execução dos modelos 3D, foi utilizada a versão 8.28.0 do BabylonJS e a versão r180 do ThreeJS, ambas as versões mais recentes disponíveis na data de 25/09/2025.

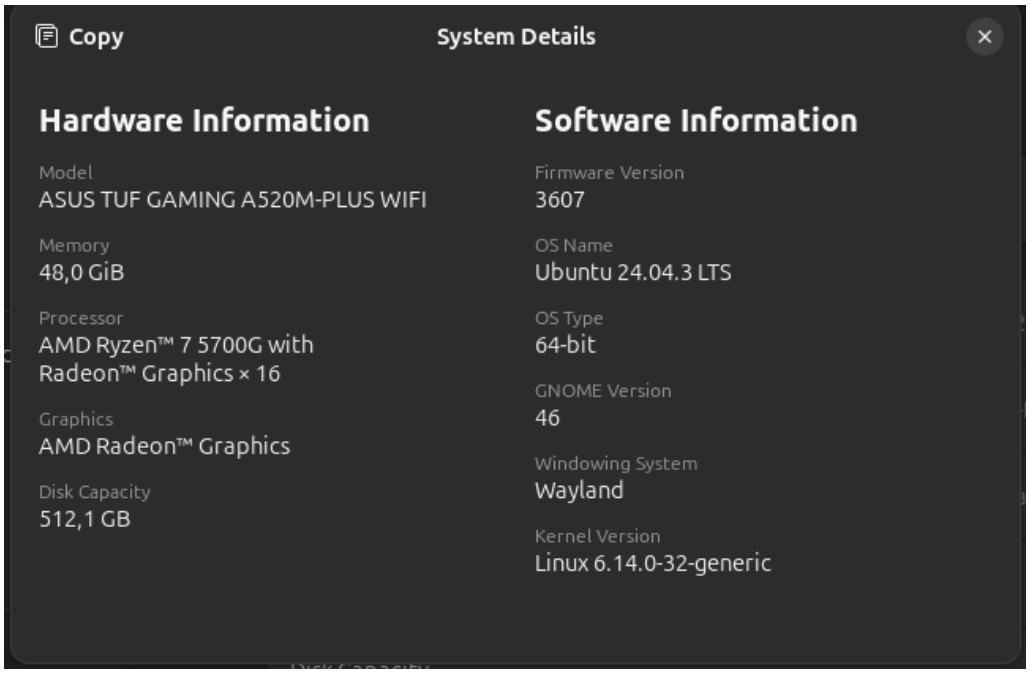

Figura 8. Configurações do computador utilizado para testes, fonte: autoria própria.

4. RESULTADOS E DISCUSSÃO

Cada teste de execução foi repetido três vezes a fim de garantir maior consistência dos resultados. A partir das medições obtidas, foram calculadas estatísticas descritivas como média, mediana, desvio padrão, valor mínimo e valor máximo. Para lidar com a variação natural entre execuções, as métricas foram consolidadas da seguinte forma:

- Média e desvio padrão: foram calculados separadamente em cada teste, e o valor final obtido pela média dos três resultados.
- Mediana: foi calculada individualmente em cada teste, e o valor final pela mediana das três medianas obtidas.
- Valores mínimo e máximo: eram considerados os menores e maiores valores registrados entre todas as execuções.

Essa decisão de utilizar a média das três execuções foi tomada para diminuir o impacto das variações entre as execuções e evitar perder os "outliers" que são fundamentais para o comparativo das bibliotecas, o que não seria possível visualizar caso a

escolha fosse da mediana. Os dados obtidos nas execuções foram agrupados por modelo (Low Poly Tree e Hintze Hall) e pelo tipo de teste de desempenho (FPS e consumo de memória), sendo analisados a seguir.

4.1. Modelo 3D Low Poly Tree

O modelo 3D Low Poly Tree possui as seguintes características relevantes para os testes do trabalho: tamanho do arquivo de 51 MB, 2.713 malhas, aproximadamente 377,8 mil vértices e 256,7 mil triângulos. Os testes realizados com esse modelo utilizando as bibliotecas ThreeJS e BabylonJS são apresentados a seguir [Low Poly Tree Scene Free 2024].

4.1.1. Desempenho do FPS

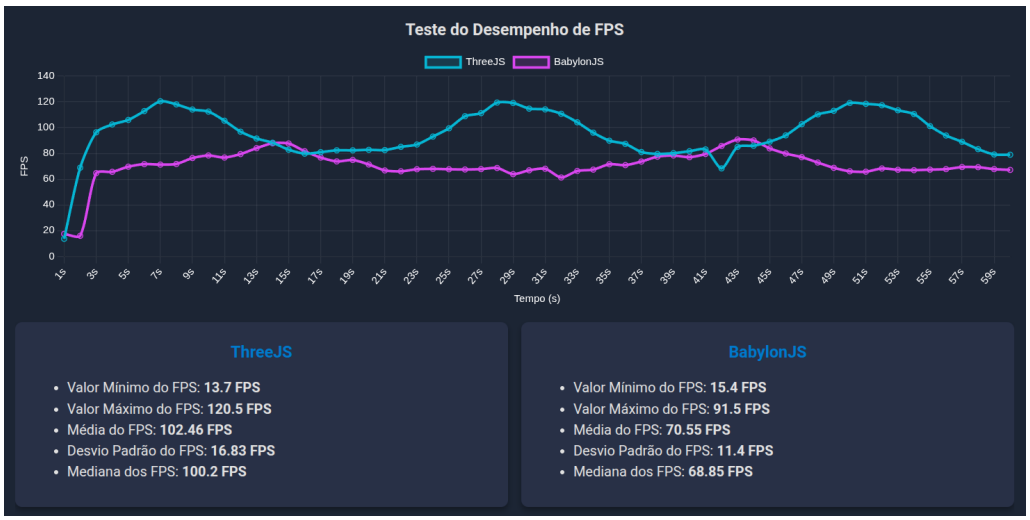

Figura 9. Desempenho do FPS no modelo Low Poly Tree, fonte: autoria própria.

Como podemos observar nos resultados obtidos na Figura 9, o ThreeJS apresentou uma média de 102,5 FPS enquanto o BabylonJS obteve uma média de 70,55 FPS.

Podemos observar no gráfico da Figura 9 que o ThreeJS obteve maior instabilidade nos valores de FPS tanto que o desvio padrão ficou em 16,83 FPS enquanto o BabylonJS teve 11,4 FPS de desvio. Em alguns momentos o ThreeJS obteve picos negativos e isso se dá por causa de uma parte do modelo que é mais densa e acaba exigindo mais recursos, tanto que os picos negativos ocorrem exatamente a cada 15 segundos, mostrando que a biblioteca tem dificuldade em manter a taxa de FPS alta em áreas mais densas.

Já no BabylonJS, ele se manteve estável a um valor base na faixa dos 60-80 FPS, enquanto o ThreeJS conseguiu chegar aos 120 FPS e apenas caiu para os 80 FPS em partes mais complexas. Um diferencial peculiar entre essas bibliotecas é que enquanto o ThreeJS se manteve em faixas mais altas com alguns picos negativos, o BabylonJS se manteve em uma faixa mais baixa com picos mais altos e uma estabilidade maior que a do ThreeJS. Um outro ponto importante a se visualizar na Figura 9 é que em 2 segundos de execução, o BabylonJS se encontrava com 40 FPS e o ThreeJS já estava a 110 FPS, o que demonstra uma inicialização um pouco mais rápida do ThreeJS.

Para efeito de verificar os dados apresentados, os valores dos outros dois testes realizados foram muito similares ao apresentado inicialmente, que é o teste com maior valor de média de ambas bibliotecas. A seguir estão os outros dois testes realizados com o modelo Low Poly Tree na Figura 10 e na Figura 11:

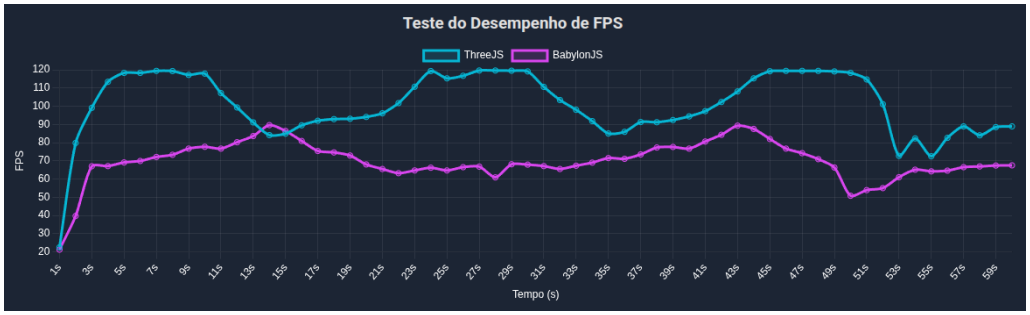

Figura 10. Segundo teste de FPS do modelo Low Poly Tree.

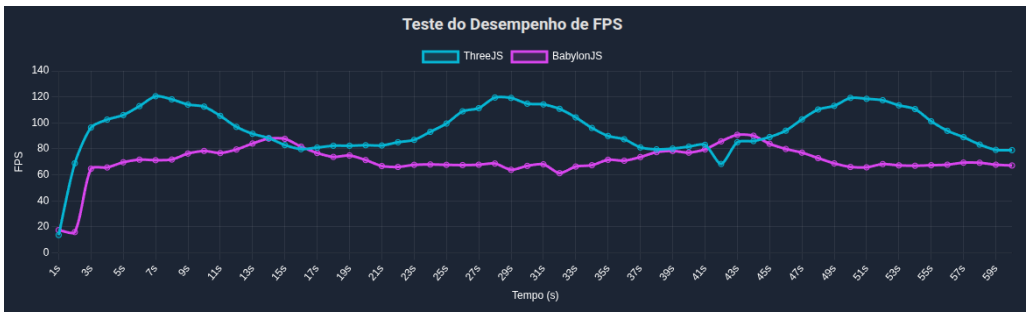

Figura 11. Terceiro teste de FPS do modelo Low Poly Tree.

Observando as Figura 10 e 11 dos outros dois testes, é possível visualizar que o BabylonJS se mantém mais estável. Porém, em uma faixa de FPS mais baixa que o ThreeJS.

4.1.2. Consumo de Memória

Analisando a Figura 12, o consumo de memória começou instável em ambas as bibliotecas, com valores entre 140 MB até 200 MB, o ThreeJS atingiu o pico de 221,98 MB enquanto que o BabylonJS chegou a 200 MB. É interessante observar que o comportamento das bibliotecas, após o período inicial, o BabylonJS, a partir dos 27 segundos já se manteve estável com um consumo de 165 MB e o ThreeJS demorou um pouco mais para ter declínio, mas aos 42 segundos obteve um consumo bem menor que o BabylonJS e ficou na faixa dos 65-100 MB, o que a longo prazo faria um grande efeito na média de consumo de memória.

Foram observados mais dois testes de consumo de memória para verificar se existe algum comportamento padrão das bibliotecas. Foi possível identificar alguns padrões como no ThreeJS que em ambos os testes obteve um pico de uso de memória e depois caiu para um valor de consumo de memória bem abaixo do BabylonJS, cerca de até 100 MB de diferença. Já o Babylon manteve um padrão de oscilação inicial e depois manteve um valor contínuo mas bem acima do ThreeJS.

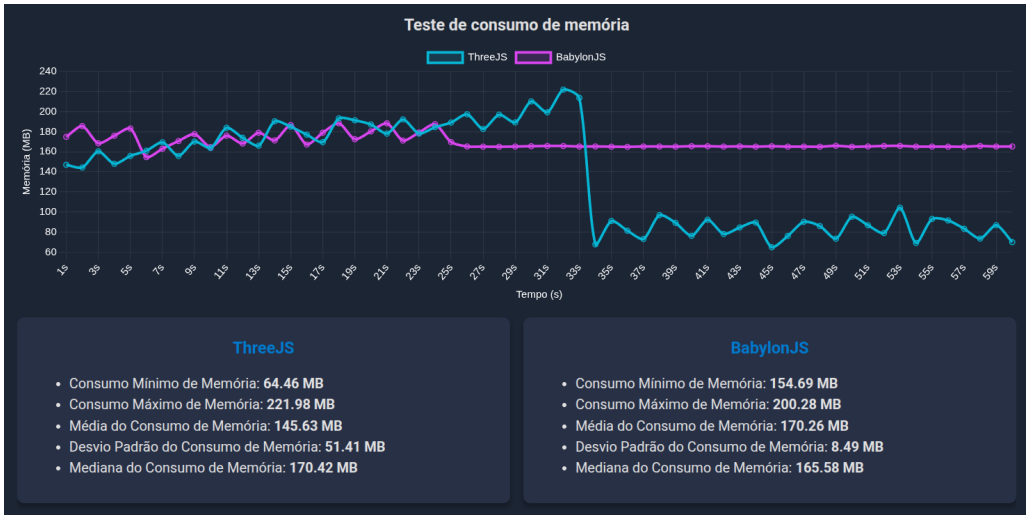

Figura 12. Consumo de memória do modelo Low Poly Tree, fonte: autoria própria.

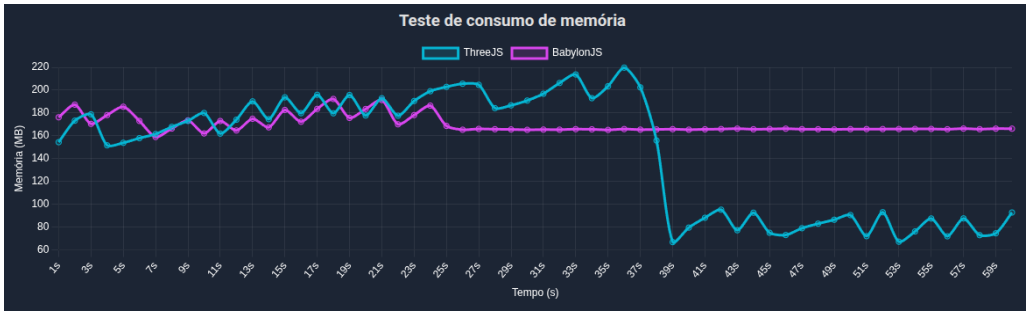

Figura 13. Segundo teste do consumo de memória do modelo Low Poly Tree.

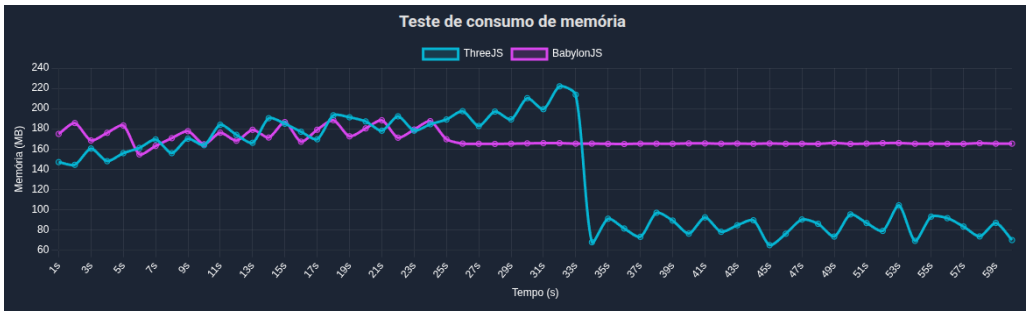

Figura 14. Terceiro teste do consumo de memória do modelo Low Poly Tree.

4.2. Modelo Hintze-Hall

O modelo 3D Hintze-Hall apresenta as seguintes características relevantes para os testes deste trabalho: tamanho do arquivo de 234 MB, 29 malhas, aproximadamente 739 mil vértices e cerca de 1,3 milhão de triângulos. Os testes realizados com esse modelo utilizando as bibliotecas ThreeJS e BabylonJS são apresentados a seguir[Hintze-Hall 2024].

4.2.1. Desempenho do FPS

Analisando o gráfico da Figura 15 com os valores dos FPS, é possível perceber que o modelo Hintze Hall foi fácil de ser executado pelas bibliotecas mesmo tendo um tamanho em Megabytes bem maior que o Low Poly Trees. Ambas bibliotecas obtiveram um desempenho excelente nesse modelo, mostrando que são capazes de exibir até mesmos modelos pesados e manter uma taxa de 120 FPS. A seguir, os outros dois testes executados para validar os dados obtidos no primeiro teste.

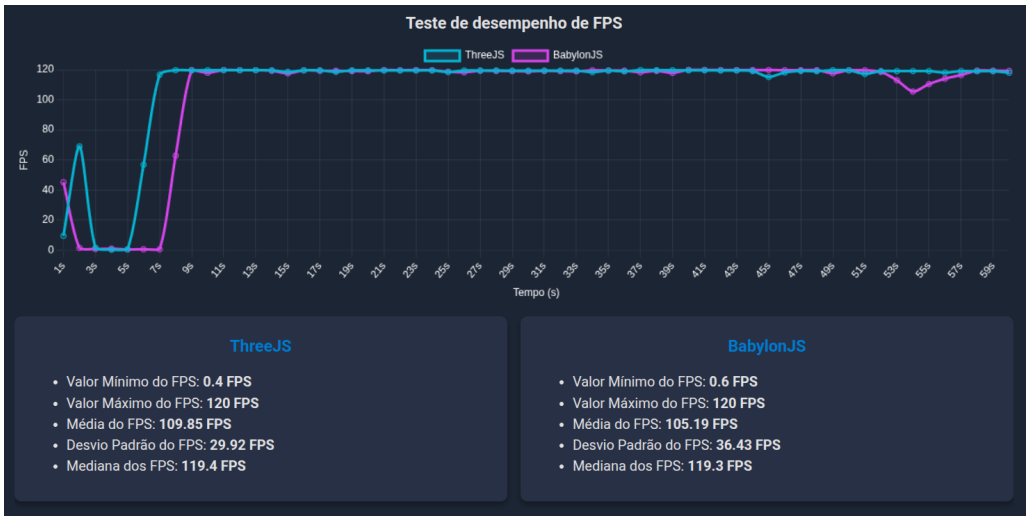

Figura 15. Desempenho do FPS no modelo Hintze Hall.

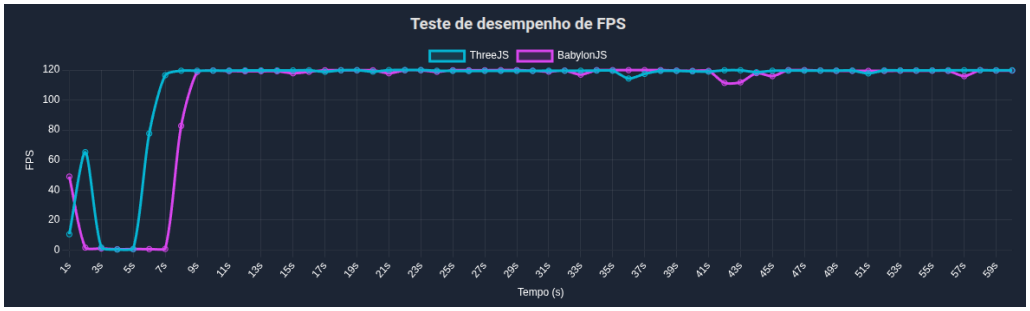

Figura 16. Segundo teste de FPS do modelo Hintze Hall.

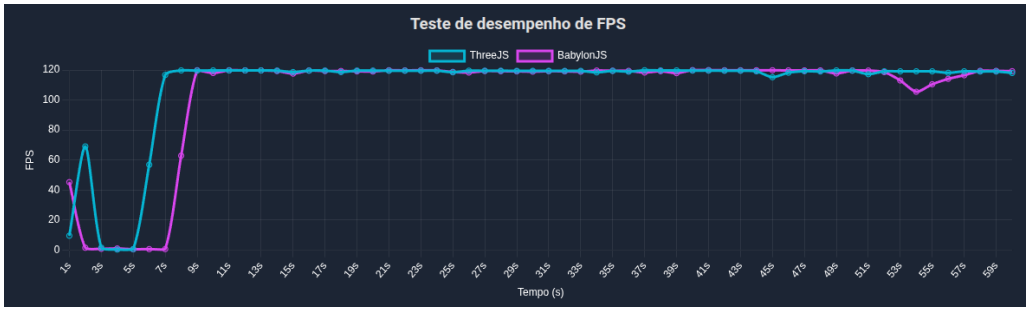

Figura 17. Terceiro teste de FPS do modelo Hintze Hall.

Como podemos visualizar na Figura 16 e na Figura 17, os outros dois testes foram muito semelhantes, o que confirma a análise realizada sobre o desempenho das bibliotecas.

4.2.2. Consumo de Memória

Figura 18. Consumo de memória do modelo Hintze Hall.

Na Figura 18, apesar do desempenho do FPS não ter variado muito nas bibliotecas, o consumo de memória apresentou um comportamento discutível. No começo, o ThreeJS apresentou um pico de consumo de memória bem alto, atingindo 724,53 MB em um modelo com tamanho de 234 MB. Após o pico inicial, manteve uma média de 500 MB, ainda mais alta que o BabylonJS, mas após mais ou menos 50 segundos de execução, apresentou um valor extremamente baixo de apenas 100 MB de consumo, o que pode ser crucial para a decisão de qual biblioteca utilizar.

O Babylon JS manteve um consumo de memória muito estável, com um desvio padrão de apenas 2,9 MB, sem oscilações, a uma média de 274,5 MB de consumo durante o teste.

Na Figura 19 e 20, estão os outros dois testes realizados para validar o comportamento das bibliotecas em relação ao consumo de memória.

Após analisar os testes, é possível confirmar a análise realizada sobre o consumo de memória das bibliotecas, que foi bem interessante de se observar por ter um comportamento bem diferente do usual, pelo menos por parte do ThreeJS que variou mais.

4.3. Análise dos resultados

A partir das análises realizadas sobre as bibliotecas nos modelos utilizados é possível analisar algumas situações:

- Tanto o BabylonJS quanto o ThreeJS se mostraram competitivos e otimizados na exibição de modelos 3D, mesmo com modelos grandes não tiveram grandes dificuldades.

Figura 19. Segundo teste do consumo de memória do modelo Hintze Hall.

Figura 20. Terceiro teste do consumo de memória do modelo Hintze Hall.

- Apesar do Hintze Hall apresentar um arquivo com mais megabytes que o Low Poly Tree, o hardware de teste teve mais dificuldades para alcançar maiores valores de FPS no modelo Low Poly Tree. Essa diferença de desempenho pode ter ocorrido devido à quantidade de malhas dos modelos, que foi o principal valor discrepante identificado ao analisar ambos modelos.
- Para decidir qual biblioteca é adequada, é necessário avaliar em quais parâmetros uma se sobressai da outra. No quesito desempenho, o ThreeJS apresentou os melhores resultados assim como o BabylonJS em modelos mais simples e em modelos mais complexos o ThreeJS se sobressaiu e obteve FPS mais altos.

Ao analisar o consumo de memória de cada biblioteca, deve-se atentar ao tipo de modelo que deseja exibir, a finalidade daquele modelo e quantos megabytes possui. Se for um modelo com tamanho reduzido, pode ser melhor utilizar o ThreeJS pois o mesmo irá apresentar um desempenho um pouco melhor que o BabylonJS. Agora, se a situação estiver em um contexto de hardware limitado e um modelo que desafia os limites de

memória, pode ser melhor considerar o BabylonJS para não haver um pico de memória que atinja o limite do hardware em uso e impossibilite a visualização do modelo.

Outra situação em que o ThreeJS pode ser vantajoso ocorre quando o modelo possui um tamanho intermediário em relação à capacidade do hardware em uso e precisa ser exibido por longos períodos de tempo. Nos testes de consumo de memória, o ThreeJS apresentou uma redução significativa após os primeiros 50 segundos de exibição contínua, o que justifica seu uso nessas situações.

5. CONCLUSÃO

Com essa pesquisa, foi possível fazer uma comparação mais atual das principais bibliotecas de renderização 3D em JavaScript e ter uma análise e observação preliminar de como está o cenário para a utilização de modelos 3D na web. Assim, pode-se analisar algumas diferenças no desempenho das bibliotecas BabylonJS e ThreeJS.

Algumas limitações da pesquisa estão relacionadas a variáveis que não se tem muito controle. Por exemplo, quando ocorrer testes em um novo modelo, sua quantidade de vértices, triângulos e malhas influenciam diretamente no desempenho. Outro ponto é a diferença entre navegadores que não foi apresentada, podendo haver divergências dos,0 testes, diferença de hardware e até mesmo de um uso real onde teriam mais abas abertas, podendo gerar um problema. A pesquisa teve como objetivo testar as bibliotecas de maneira neutra e imparcial, por isso, foram adotadas algumas medidas que podem acabar distanciando o ambiente de testes do ambiente real, como deixar apenas o navegador e uma só aba aberta, utilizar a versão mais recente de cada biblioteca que depende de cada aplicação atualizar as mesmas, podendo ser encontrados sistemas que as utilizam mas em versões antigas à data de execução dos testes de desempenho, dentre outras variáveis que podem afetar um teste real, como conexão de rede instável, etc.

Para trabalhos futuros, pode ser testado outros navegadores, dispositivos móveis, criar scripts para testar mais modelos, e tentar diversificar máquinas desktops para configurações mais simples. Isso ajudaria a verificar com mais precisão o desempenho das bibliotecas e analisar o seu consumo de memória nesses cenários diferentes do que foi apresentado nesse trabalho.

Referências

Akenine-Möller, T., Haines, E., and Hoffman, N. (2018). *Real-time Rendering*. A K Peters/CRC Press, 4th edition.

BabylonJS Documentation (2024). Learning the docs. Disponível em: `https://doc.babylonjs.com/journey/learningTheDocs`. Acesso em: 31 out. 2024.

BabylonJS First Step (2024). The very first step. Disponível em: `https://doc.babylonjs.com/journey/theFirstStep`. Acesso em: 31 out. 2024.

Chen, R. (2011). The development of 3d city model and its applications in urban planning. In *19th International Conference on Geoinformatics*, pages 1–5. Acesso em: 1 set. 2024.

Dirksen, J. (2015). *Learning Three.js - the JavaScript 3D Library for WebGL: Create Stunning 3D Graphics in Your Browser Using the Three.js JavaScript Library*. Packt Pub Ltd.

Ducati XDiavel (2024). Ducati xdiavel 3d webgl. Disponível em: https://nymbl3d.github.io/nymblWebGL/projects/04DucatiXDiavel/XDiavelDemo.html. Acesso em: 31 out. 2024.

Faraday Future (2024). Ff 91 vr test ride. Disponível em: https://vr.ff.com/us/. Acesso em: 31 out. 2024.

Gerth, V. E., Katsuyama, K., Snyder, K. A., Bowes, J. B., Kitayama, A., Ueno, N., and Vize, P. D. (2007). Projecting 2d gene expression data into 3d and 4d space. *Developmental Dynamics*, 236:913–1159. Acesso em: 15 ago. 2024.

Groenendyk, M. (2016). Cataloging the 3d web: the availability of educational 3d models on the internet. *Library Hi Tech*, 34(2). Acesso em: 2 out. 2024.

Hintze-Hall (2024). Hintze-hall vr tour. Disponível em: https://sketchfab.com/3d-models/hintze-hall-vr-tour-058b26fb31ed49df978de31af3dc091f. Acesso em: 31 out. 2024.

Laviola Jr, J. J. and Zeleznik, R. C. (2005). *3D User Interfaces: Theory and Practice*. Addison-Wesley Professional.

Low Poly Tree Scene Free (2024). Low poly tree scene free. Disponível em: https://sketchfab.com/3d-models/low-poly-tree-scene-free-89daa5e21f0d4f08a59dba0d566e88bd. Acesso em: 31 out. 2024.

Nishanbaev, I. (2020). A web repository for geo-located 3d digital cultural heritage models. *Digital Applications in Archaeology and Cultural Heritage*, 16. Acesso em: 15 ago. 2024.

Plastic Boy (2024). Anatomy 3d model. Disponível em: https://plasticboyanatomy.com/. Acesso em: 31 out. 2024.

Rangkuti, A., Athalaa, V., Indallah, F., and Febriansyah, F. (2023). Optimizing hand gesture recognition using cnn model supported by raspberry pi for self-service technology. *International Journal on Informatics Visualization*, 7(1). Acesso em: 15 ago. 2024.

Sheng, B., Zhao, F., Zhang, C., Yin, X., and Shu, Y. (2017). 3d rubik's cube — online 3d modeling system based on webgl. In *IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. Acesso em: 23 set. 2024.

Söderberg, J. (2020). Fps in an interactive 3d model viewer with webgl: A comparison of fps between the javascript libraries three.js and babylon.js. Bachelor's Thesis (Information Technology) – Högskolan i Skövde, Skövde, Sweden, Spring 2020.

ThreeJS Creating Scene (2024). Creating a scene 12 - three.js docs. Disponível em: https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene. Acesso em: 31 out. 2024.

Three.js Documentation (2024). Three.js documentation. Disponível em: https://threejs.org/docs/. Acesso em: 31 out. 2024.

ThreeJS Mentor (2024). Three.js mentor. Disponível em: https://chatgpt.com/g/g-jGjqAMvED-three-js-mentor. Acesso em: 31 out. 2024.

Trubka, R. (2016). A web-based 3d visualisation and assessment system for urban precinct scenario modelling. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:175–186. Acesso em: 15 ago. 2024.

Watt, A. (1999). *3D Computer Graphics*. Addison-Wesley, 3rd edition.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2024). *Experimentation in Software Engineering*. Springer Berlin, Heidelberg. Acesso em: 14 out. 2025.

Ye, L., Liu, G., Chen, G., Li, K., Chen, Q., Fan, W., and Zhang, J. (2023). 3d model occlusion culling optimization method based on webgpu computing pipeline. *Computer Systems Science and Engineering*, pages 2529–2545. Acesso em: 22 set. 2024.