

Mayara Gomes Silva  
Matrícula: 2012393

## **Visualizador 2D para dados sísmicos usando WebGL**

Rio de Janeiro - RJ  
Julho - 2021

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>2</b>
<b>1.1</b>	<b>Objetivo</b>	<b>2</b>
<b>1.2</b>	<b>Organização do Documento</b>	<b>2</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>4</b>
<b>2.1</b>	<b>WebGI</b>	<b>4</b>
2.1.1	Pipeline da WebGL	4
2.1.1.1	Aplicação WebGL	5
2.1.1.2	Shader de Vértice	6
2.1.1.3	Construção de Primitivas	7
2.1.1.4	Rasterização	7
2.1.1.5	Shader de Fragmento	8
2.1.1.6	Operação Por Fragmento	8
<b>3</b>	<b>REQUISITOS E ESPECIFICAÇÕES</b>	<b>9</b>
<b>3.1</b>	<b>Requisitos Funcionais</b>	<b>9</b>
<b>3.2</b>	<b>Requisitos Não Funcionais</b>	<b>9</b>
<b>3.3</b>	<b>Casos de Uso</b>	<b>10</b>
<b>4</b>	<b>DESENVOLVIMENTO</b>	<b>13</b>
<b>4.1</b>	<b>Plataformas e Tecnologias</b>	<b>13</b>
<b>4.2</b>	<b>Modelagem e Arquitetura</b>	<b>13</b>
<b>4.3</b>	<b>Testes</b>	<b>15</b>
<b>5</b>	<b>MANUAL DO USUÁRIO</b>	<b>18</b>
<b>5.1</b>	<b>Requisitos Mínimos</b>	<b>18</b>
<b>5.2</b>	<b>Interface Gráfica de Usuário</b>	<b>18</b>
<b>5.3</b>	<b>Principais Funções do Programa</b>	<b>18</b>
	<b>REFERÊNCIAS</b>	<b>21</b>

# 1 Introdução

A web foi inventada na década de 90 por Sir Tim Berners-Lee, quando pesquisava sobre um método que possibilitasse aos cientistas do mundo inteiro compartilhar suas pesquisas eletronicamente ([SILVA, 2019](#)). Atualmente a web nos proporciona o compartilhamento de informação, como também diversão, realização de transações e comunicação social.

A WebGL é a junção das tecnologias web e OpenGL, permitindo a visualização de dados bidimensionais e tridimensionais em um ambiente web. Essa visualização pode ser feita sem a necessidade de instalação de nenhum tipo de software ou plugin especial, bastando um navegador de Internet compatível, tal como Google Chrome e Mozilla Firefox. O WebGL permite a programação com desenvolvimento de funções em JavaScript para que seja possível a implementação de mecanismos de navegação pelos dados visualizados. Este recurso permite a visualização e análise mais detalhada das estruturas dos dados, assim o profissional pode analisar mais detalhadamente os dados.

O presente trabalho apresenta uma aplicação que permite que seus usuários visualizem dados sísmicos 2D. Para realizar esse processo foi utilizado a renderização de dados com WebGL e algumas tecnologias de HTML5, Javascript e bibliotecas em Python.

## 1.1 Objetivo

Desenvolver um visualizador de dados sísmico em web, navegar dentro do volume de dados permitindo a visualização e análise mais detalhada das estruturas dos dados e que possa ser facilmente incorporado em sistemas sísmicos.

## 1.2 Organização do Documento

Este trabalho apresenta a seguinte organização:

No Capítulo 2, Fundamentos Teóricos, apresenta conceitos fundamentais para o contexto e entendimento do trabalho, tais como explicações sobre WebGL.

No Capítulo 3, Requisitos e Especificações, mostra-se os Requisitos Funcionais e Não Funcionais coletados para melhor definir o programa e o diagrama de Casos de Uso contendo a descrição e pré-condições de cada caso de uso.

No Capítulo 4, Desenvolvimento, apresenta-se as plataformas e tecnologias utilizadas para desenvolver o programa, bem como seu diagrama de Classes seguindo o padrão de

arquitetura Model-View-Controller e diagramas de sequência para melhor entendimento de funcionalidades críticas e o relacionamento das classes envolvidas nessas funcionalidades.

No Capítulo 5, Manual do Usuário, apresenta-se os requisitos mínimos para a execução consistente do programa, explicações sobre cada componente da GUI e explicações de como usar o programa corretamente.

## 2 Fundamentação Teórica

Nessa seção, serão explicados os conceitos necessários para entender a tecnologia que será empregada ao decorrer de todo o trabalho.

### 2.1 WebGL

WebGL é um padrão web gráfico de baixo nível, que é gratuito, baseado em OpenGL ES SL. Permite a criação de shaders utilizando OpenGL ES, desenvolvendo aplicações gráficas em 2D e 3D utilizando JavaScript diretamente no navegador.

OpenGL é uma API padrão da indústria de desenvolvimento de computação gráfica, disponibilizada na década de 90. Uma de suas características é ser multiplataforma, o que permite seu uso em diversos tipos de sistemas, como aplicações de CAD, animação, processamento de vídeo, jogos e visualização científica ([RÖSSLER, 2012](#)).

O projeto inicial de OpenGL tinha funcionalidades especificadas por meio de funções fixas. Porém, com o desenvolvimento das tecnologias disponíveis nos hardwares, foram sendo implementadas extensões que permitiam ao OpenGL utilizar funcionalidades específicas de uma placa gráfica. Assim, surgindo a linguagem de programação desenvolvida para programação dos Shaders de vértices e de fragmento, a OpenGL Shading Language (OpenGL SL). E com a necessidade de expandir a API de ambientes desktop para também ambientes móveis, foi criada uma nova API denominada OpenGL ES ([JUNIOR, 2013](#)).

OpenGL e OpenGL ES são APIs para computadores desktop e dispositivos móveis respectivamente, porém, havia a necessidade de uma API para computação gráfica disponível para a Internet, então a WebGL foi desenvolvida a fim de suprir essa necessidade. ([RÖSSLER, 2012](#))

A versão mais atual e estável da WebGL é a 2.0, e tem suporte para vários navegadores, como o Mozilla, Chrome, Opera.

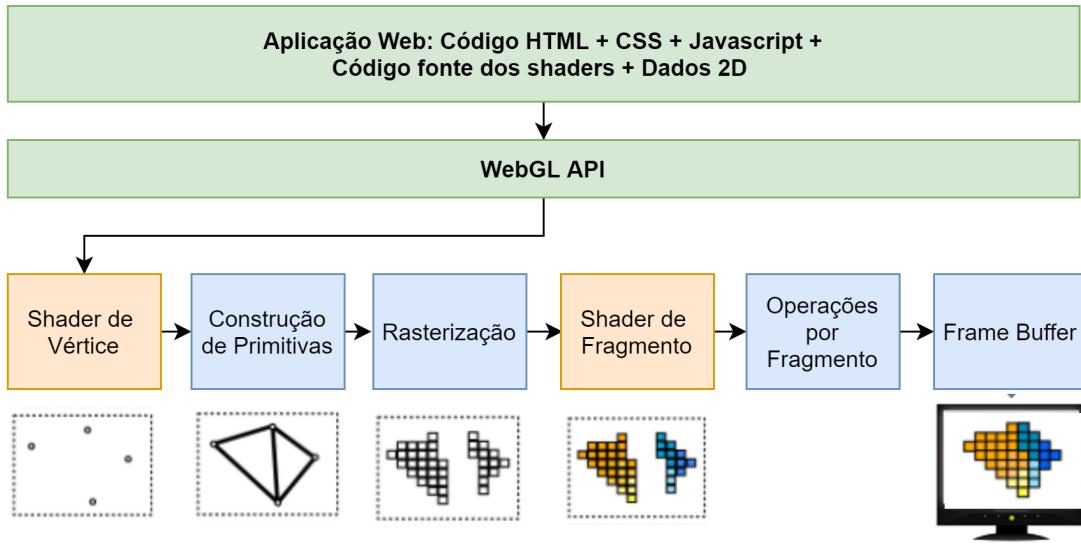
A WebGL oferece várias vantagens, por ser baseada em OpenGL e funcionar em várias plataformas. Tem compatibilidade para vários navegadores, é multi-plataforma, integração com o conteúdo HTML, utiliza a placa de vídeo do computador para acelerar o processamento dos gráficos e gerenciamento automático de memória. ([PARISI, 2014](#))

#### 2.1.1 Pipeline da WebGL

Para renderizar gráficos 2D ou 3D, é necessário seguir uma sequência de etapas, que são chamadas de pipeline de renderização. A Figura 1 ilustra o diagrama do pipeline

de aplicação desenvolvida em WebGL.

Figura 1 – Pipeline de aplicação desenvolvida com WebGL.



Fonte: ([ANYURU, 2012](#)) adaptado pelo autor

### 2.1.1.1 Aplicação WebGL

Uma aplicação web é composta de vários elementos, como o código HTML para a construção da página, o CSS para definir a aparência dos elementos do HTML e o código Javascript. Em uma aplicação WebGL, além desses elementos também contém o código fonte de seus shaders. A API WebGL proporciona a comunicação do HTML e o código Javascript, permitindo a execução dos shaders e a renderização dos objetos na aplicação Web. Para usar a WebGL, é necessário um contexto gráfico WebGL, que é um objeto JavaScript onde métodos implementam o lado JavaScript da API WebGL. Com o contexto podemos desenhar elementos gráficos em um elemento <canvas> do HTML.

Segundo [Parisi \(2012\)](#) aplicação WebGL consiste de alguns passos mínimos para a estrutura básica de uma renderização WebGL:

1. Criar um elemento canvas;
2. Obter um contexto de desenho no canvas;
3. Inicializar a Viewport;
4. Criar um ou mais buffers contendo os dados a serem renderizados;
5. Criar uma ou mais matrizes para definir a transformação dos buffers de vértice para espaço na tela;
6. Criar um ou mais Shaders para implementar o algoritmo de desenho;

7. Inicializar os Shaders com parâmetros;
8. Desenhar;

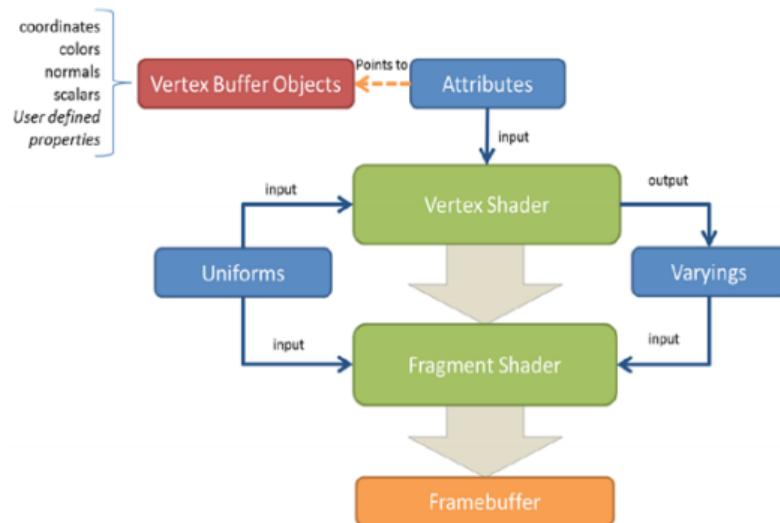
A WebGL precisa de dois tipos de shaders, o Vertex shader e o Fragment shader. Eles são escritos na linguagem de Shading do OpenGL ES, a GLSL, que foi projetada especialmente para gráficos. Nesses shaders, definimos como os vértices, transformações, materiais, luzes e câmera interagem entre si para criar uma imagem específica ([ANYURU, 2012](#)).

#### 2.1.1.2 Shader de Vértice

O shader de vértices é responsável por calcular as posições de cada vértice, e pode também calcular as coordenadas de textura. Esse shader é chamado uma vez por vértice e geralmente aplica transformações para transformar as posições dos vértices do espaço do modelo para o espaço da tela ([BARBEDO; KOENIGKAN; SANTOS, 2016](#)).

Esse shader precisa de dados para processar e depois envia esses dados processados para o próximo passo. A Figura 2 mostra como ocorre esse processo. Os dados são obtidos através de atributos (dados extraídos de buffers), e uniformes (valores constantes).

Figura 2 – API WebGL.



Fonte: ([JUNIOR, 2013](#))

Os dados são passados do código javascript para o shader através do Vertex Buffer Objects (VBOs). Cada VBO armazena dados de um atributo particular dos vértices (posição, cor, coordenadas de texturas, etc). Os Atributos são tipos de variáveis que apontam para os VBOs, e através dos atributos o código WebGL tem acesso a esses dados. Em cada Ciclo de Renderização, os valores dos atributos são diferentes ([ANYURU, 2012](#)).

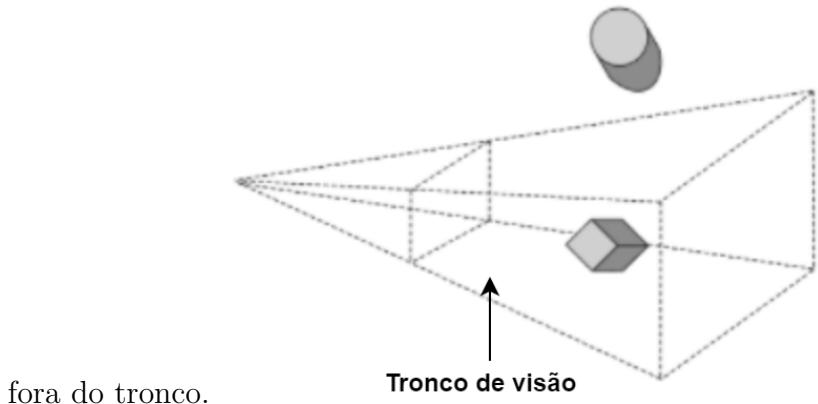
Os Uniforms são variáveis de entrada para os dois tipos de shader. Os uniformes são valores passados para o sombreador de vértice que permanecem os mesmos para todos os vértices durante todo o ciclo de renderização (PARISI, 2014).

O shader de vértice pode passar informações para o shader de fragmento quando necessário, utilizando as variáveis do tipo Varyings. Essa variável pode ser acessada pelo shader de fragmento com o mesmo nome que está no shader de vértice.

### 2.1.1.3 Construção de Primitivas

Após calcular a posição e outros detalhes de cada vértice, a próxima fase é a etapa de montagem primitiva, que podem ser linhas, pontos e triângulos. A partir das variações das primitivas qualquer coisa pode ser desenhada. Então, para cada elemento, webgl precisa decidir se o primitivo está dentro da região 3D que está visível na tela no momento. As primitivas dentro do tronco da visão são enviadas para a etapa de rasterização e as primitivas fora do tronco da visão são removidas (ANYURU, 2012). A figura 3 mostra o troco de visão e um exemplo de elementos que estão dentro e fora desse campo.

Figura 3 – Vista do tronco de visão com um cubo que está dentro e um cilindro que está



fora do tronco.

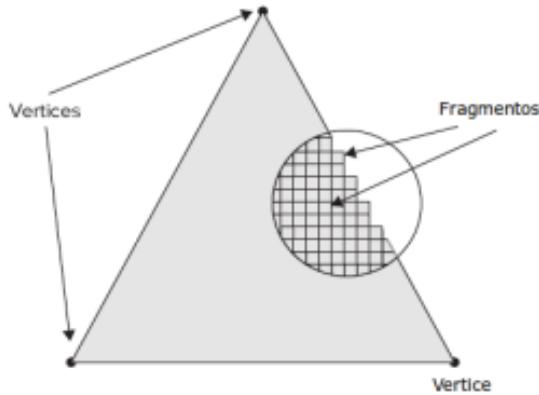
Fonte: (ANYURU, 2012) adaptado pelo autor

### 2.1.1.4 Rasterização

Nesta etapa as primitivas são convertidas em fragmentos. Esses fragmentos são os pixels que serão enviados ao shader de fragmento (PAZA, 2014).

A Figura 13 mostra o exemplo da rasterização de um triângulo. O triângulo é composto por três vértices a rasterização transforma a região entre esses vértices em vários pixels, e para cada pixel é definido uma cor, coordenada de textura e outros elementos necessários. Os valores são determinados pela interpolação a partir dos valores dos vértices (JUNIOR, 2013).

Figura 4 – Exemplo da rasterização de um triângulo.



Fonte: ([JUNIOR, 2013](#))

#### 2.1.1.5 Shader de Fragmento

O shader de fragmento é responsável por calcular uma cor para cada pixel da primitiva que está sendo desenhada no momento. O shader de fragmento é chamado uma vez por pixel, e cada vez que é chamado, deve definir sua variável de saída para alguma cor. Durante as operações alguns fragmentos podem ser descartados, logo não são renderizados ([ANYURU, 2012](#)).

Assim como o shader de vértices, o shader de fragmento também recebe dados. Podem ser por Uniformes (valores definidos pelo usuário) e por varyings (valores obtidos do shader de vértices).

#### 2.1.1.6 Operação Por Fragmento

Esta etapa consiste nas operações por fragmento, e contém várias subetapas. Cada fragmento fora do sombreador de fragmento pode modificar um pixel no buffer de desenho de maneiras diferentes, dependendo das condições e resultados das diferentes etapas nas operações por fragmento. Essas operações podem envolver profundidade, Dithering e outras ([ANYURU, 2012](#)).

Logo após o processamento de todos os fragmentos, uma imagem 2D é formada e exibida na tela.

# 3 Requisitos e Especificações

Nesta seção serão apresentados os principais requisitos e especificações para a criação da aplicação.

## 3.1 Requisitos Funcionais

A ferramenta é constituída dos seguintes requisitos funcionais descritos abaixo.

1. Leitura e carregamento dos dados: O sistema carrega os nomes dos dados de uma pasta e o usuário poderá selecionar um arquivo de um dado mostrado na tela, então ele é carregado para o programa.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

2. Mudança do mapa de cor: O usuário poderá selecionar uns dos mapas de cor que deseja visualizar os dados.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

3. Inversão das cores dos dados : O usuário poderá selecionar a opção de inversão de cores.

<b>Prioridade</b>	( ) Essencial	(X) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	( ) Médio	(X) Baixo

4. Interação na visualização dos dados: Uma vez que o usuário tenha carregado um arquivo, o usuário pode modificar a visualização do dado, como aumentar a escala, movimentar o dado com o mouse.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

## 3.2 Requisitos Não Funcionais

A aplicação possui os seguintes requisitos não funcionais.

1. Multiplataforma: O programa poderá ser executado em diferentes sistemas operacionais, tais como Linux e Windows, necessitando somente de um browser.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

2. Robustez: O programa deverá ser estável, ou seja, não deverá travar ou fechar inesperadamente.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

3. Internet: Por ser uma aplicação Web é necessário o uso de internet para acessar o sistema.

<b>Prioridade</b>	(X) Essencial	( ) Importante	( ) Desejável
<b>Esforço</b>	( ) Alto	(X) Médio	( ) Baixo

### 3.3 Casos de Uso

Nessa subseção é apresentado o diagrama de casos de uso, Figura 5, criado para a aplicação com base nos requisitos do sistema, assim como a especificação de cada um.

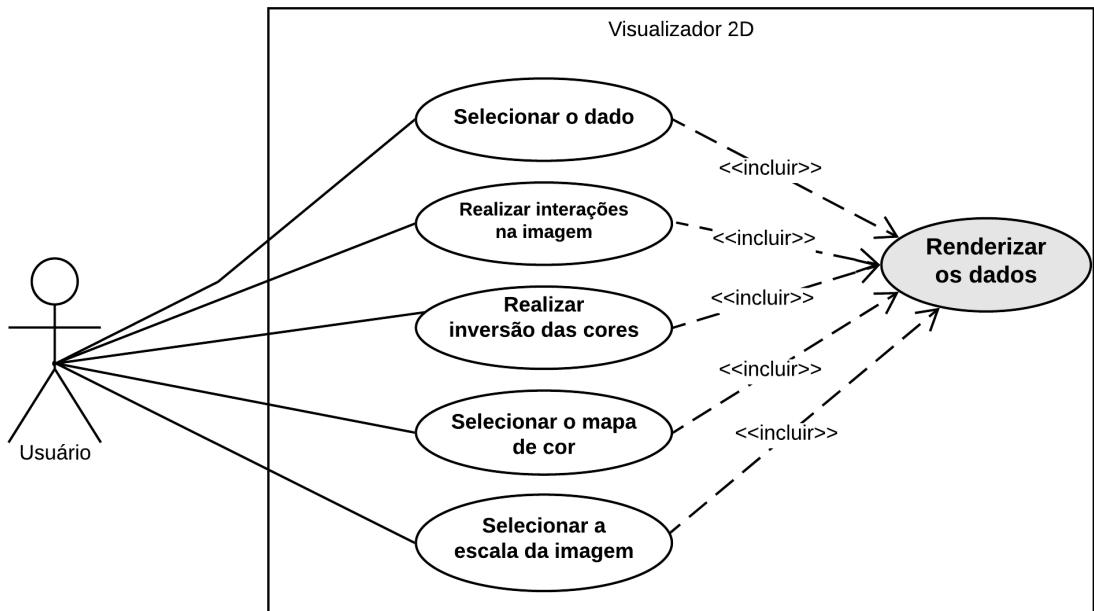


Figura 5 – Diagrama de Casos de Uso.

1. Selecionar o dado:

<b>Descrição</b>	
<b>Autor(es):</b>	O usuário.
<b>Descrição Sucinta:</b>	Permite que o usuário selecione o arquivo que deseja visualizar.
<b>Pré-condições:</b>	Ter dados para selecionar.
<b>Pós-condições:</b>	Mostrar o dado renderizado na tela.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O usuário clica no arquivo que deseja;</li> <li>2. O sistema carrega os dados desse arquivo da pasta;</li> <li>3. O sistema processa os dados;</li> <li>4. O sistema chama o caso de uso "Renderizar os dados";</li> <li>5. Os dados são renderizados na tela.</li> </ol>
<b>Cenário Alternativo:</b>	O sistema não consegue renderizar os dados: o sistema exibe uma mensagem ao seu usuário.

2. Renderizar os dados:

<b>Descrição</b>	
<b>Autor(es):</b>	Sistema.
<b>Descrição Sucinta:</b>	Renderiza os dados selecionados, para visualização.
<b>Pré-condições:</b>	Ter realizado o caso de uso "Selecionar o dado"
<b>Pós-condições:</b>	Mostrar uma imagem renderizada na tela.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O sistema recebe os dados;</li> <li>2. O sistema renderiza a imagem na tela;</li> </ol>
<b>Cenário Alternativo:</b>	O usuário não carrega nenhum arquivo: o sistema exibe uma mensagem ao seu usuário.

3. Realizar interações com o mouse:

<b>Descrição</b>	
<b>Autor(es):</b>	O usuário.
<b>Descrição Sucinta:</b>	Permite que o usuário movimente a imagem no canvas com a utilização do mouse.
<b>Pré-condições:</b>	Ter realizado o caso de uso "Selecionar o dado para".
<b>Pós-condições:</b>	Mostrar a imagem de acordo com a movimentação feita.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O usuário clica na imagem e arrasta, para cima, ou para baixo ou para os lados;</li> <li>2. O sistema renderiza a imagem na nova posição utilizando o caso de uso "Renderizar os dados";</li> <li>3. O sistema mostra a imagem na nova posição.</li> </ol>
<b>Cenário Alternativo:</b>	Nenhum

4. Realizar inversão das cores do dado:

<b>Descrição</b>	
<b>Ator(es):</b>	O usuário.
<b>Descrição Sucinta:</b>	Inverte o contraste da imagem.
<b>Pré-condições:</b>	Ter realizado o caso de uso "Selecionar o dado para".
<b>Pós-condições:</b>	Mostrar uma imagem renderizada na tela.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O usuário clica no botão com o ícone de referente a inversão;</li> <li>2. O sistema renderiza a imagem utilizando o caso de uso "Renderizar os dados";</li> </ol>
<b>Cenário Alternativo:</b>	Nenhum.

5. Selecionar o mapa de cores:

<b>Descrição</b>	
<b>Ator(es):</b>	O usuário.
<b>Descrição Sucinta:</b>	Permite que o usuário troque o mapa de cores da imagem.
<b>Pré-condições:</b>	Ter realizado o caso de uso "Selecionar o dado para".
<b>Pós-condições:</b>	Mostrar uma imagem renderizada na tela.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O usuário clica no botão com o ícone de referente ao mapa de cores;</li> <li>2. O usuário escolhe o novo mapa de cor;</li> <li>3. O sistema renderiza a imagem utilizando o caso de uso "Renderizar os dados";</li> </ol>
<b>Cenário Alternativo:</b>	Nenhum.

6. Selecionar a escala de visualização do dado:

<b>Descrição</b>	
<b>Ator(es):</b>	O usuário.
<b>Descrição Sucinta:</b>	Permite que o usuário defina o valor da escala da imagem.
<b>Pré-condições:</b>	Ter realizado o caso de uso "Selecionar o dado para".
<b>Pós-condições:</b>	Mostrar uma imagem renderizada na tela.
<b>Cenário Principal:</b>	<ol style="list-style-type: none"> <li>1. O usuário clica no botão com o ícone de referente ao tipo de escala;</li> <li>2. O usuário define o valor da escala;</li> </ol>
<b>Cenário Alternativo:</b>	Nenhum.

# 4 Desenvolvimento

Esta seção contém informações sobre a fase de desenvolvimento da aplicação.

## 4.1 Plataformas e Tecnologias

O programa foi desenvolvido em sua maior parte na linguagem Javascript juntamente com a API WebGL. A Webgl foi utilizada para renderizar os dados em imagens 2D na tela. Foi usado o interpretador anaconda para instalação de várias bibliotecas e execução de algoritmos em python. A linguagem Python foi utilizada para o o serviço de cliente-servidor. Para o desenvolvimento da Interface Gráfica de Usuário (GUI), utilizou-se Html 5 com o uso de css para agregar o estilo as páginas do sistema. Para realizar algumas funcionalidades da GUI foram criadas algumas funções em Javascript. Sendo alguns exemplos de utilização, a parte de visualização, interação e execução, no caso dos dados científicos, por parte do sistema. A tabela 1 resume as plataformas e tecnologias utilizadas no desenvolvimento do programa.

Tabela 1 – Tabela 1 – Resumo das plataformas e tecnologias utilizadas.

<b>Linguagem de Programação</b>	Python 3.6 e Javascript
<b>API</b>	WebGL
<b>Bibliotecas</b>	flask, numpy, os, errno, segyio, simplejson
<b>Plataformas</b>	Windows, Linux, Mac OS X

## 4.2 Modelagem e Arquitetura

A seguir, é apresentada a modelagem e arquitetura feita para o desenvolvimento do programa.

A aplicação utiliza o modelo cliente-servidor para obtenção de dados localmente e o processamento dos dados. Outro modelo utilizado é o padrão Model-view-controller (MVC). Nesse padrão o projeto do software é separado em três camadas independentes: Model, View e controller. A camada Model (modelo) manipula a lógica de dados, fornecendo meios para acessá-los. A View (visão) é a interface do usuário, conhece os modelos, mas não o contrário. O controller (controlador) é onde está o fluxo de aplicação. Esta separação facilita a manutenção do código, que pode ser reutilizado em outros projetos. A Figura 6 ilustra o funcionamento do padrão de arquitetura MVC.

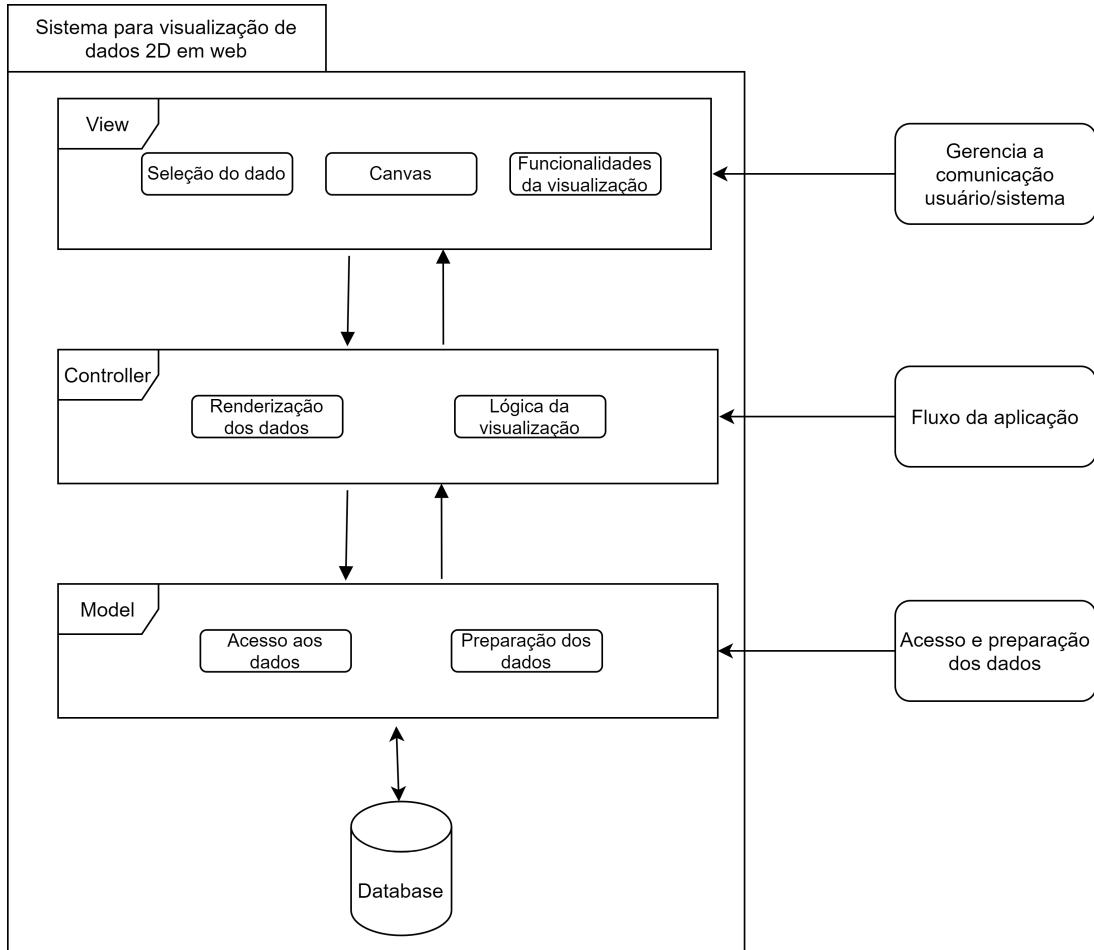


Figura 6 – Funcionamento do padrão de arquitetura Model-view-controller.

A seguir, será apresentado os diagramas de classes que foram criados para a aplicação. A Figura 7 mostra o diagrama da camada de controller. Na classe de CanvasComponent compreende toda a lógica da camada de controller, onde controla a visualização do usuário e o gerenciamento dos eventos pelo usuário. Algumas outras classes estão interligadas a ela, a Camera e o Object2D.

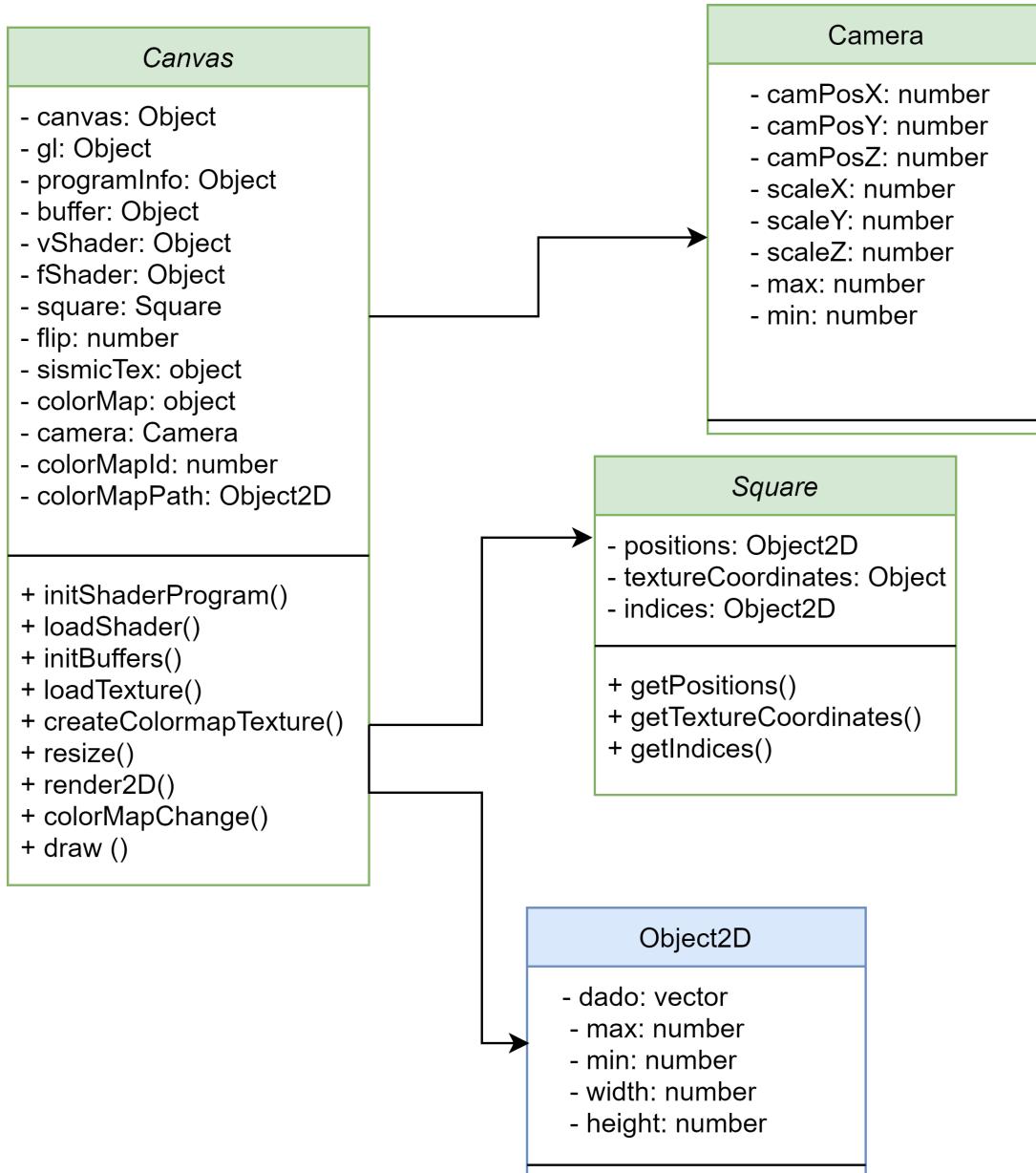


Figura 7 – Diagrama de classes.

### 4.3 Testes

Para verificar o funcionamento correto do programa, foram realizados testes manuais do sistema. Segue abaixo um roteiro para cada um dos casos de uso da aplicação.

1. Selecionar o dado, permite que o usuário selecione o arquivo que deseja visualizar (Figura 8).

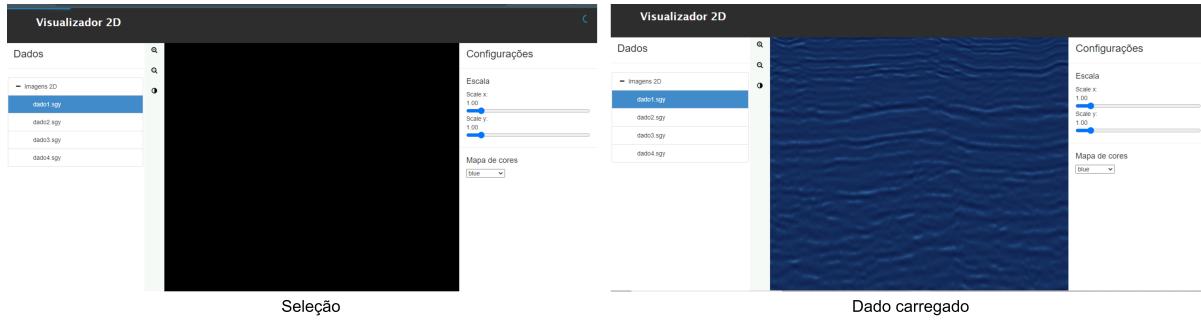


Figura 8 – Seleção e visualização do dado.

- Realizar interações com o mouse, permite que o usuário movimente a imagem no canvas com a utilização do mouse (Figura 9).

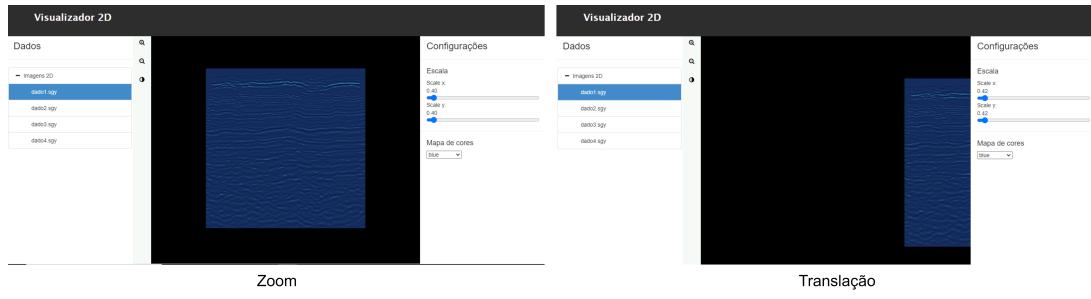


Figura 9 – Interações realizadas com o mouse, zoom e movimentação da imagem.

- Realizar inversão das cores do dado, inverte o contraste da imagem (Figura 10).

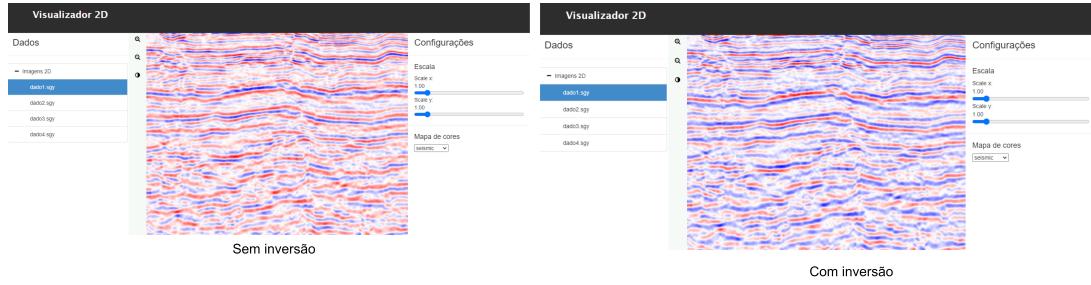


Figura 10 – Inversão das cores dos dados.

- Selecionar o mapa de cores, permite que o usuário troque o mapa de cores da imagem (Figura 11).

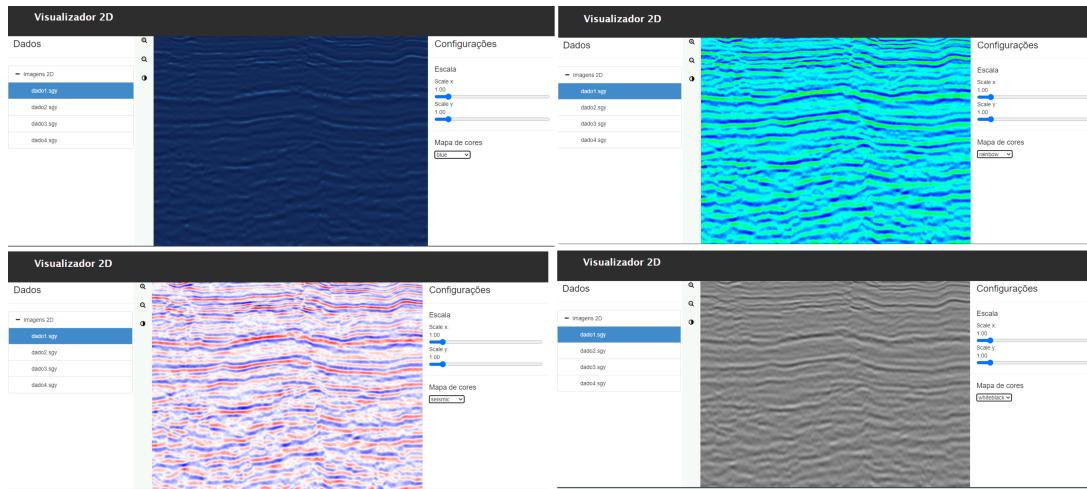


Figura 11 – Inversão das cores dos dados.

5. Selecionar a escala de visualização do dado, Permite que o usuário defina o valor da escala da imagem (Figura 12).



Figura 12 – Alteração de escala.

# 5 Manual do Usuário

Esta seção apresenta o manual do usuário. Nela, é demonstrado como utilizar o Visualizador 2D.

## 5.1 Requisitos Mínimos

Para o funcionamento correto da aplicação é necessário um computador que tenha acesso a internet, e instalação das bibliotecas necessárias da linguagem python.

## 5.2 Interface Gráfica de Usuário

A figura abaixo mostra a GUI associados a página inicial do programa. Os componentes da interface foram especificados na própria imagem para facilitar o entendimento.

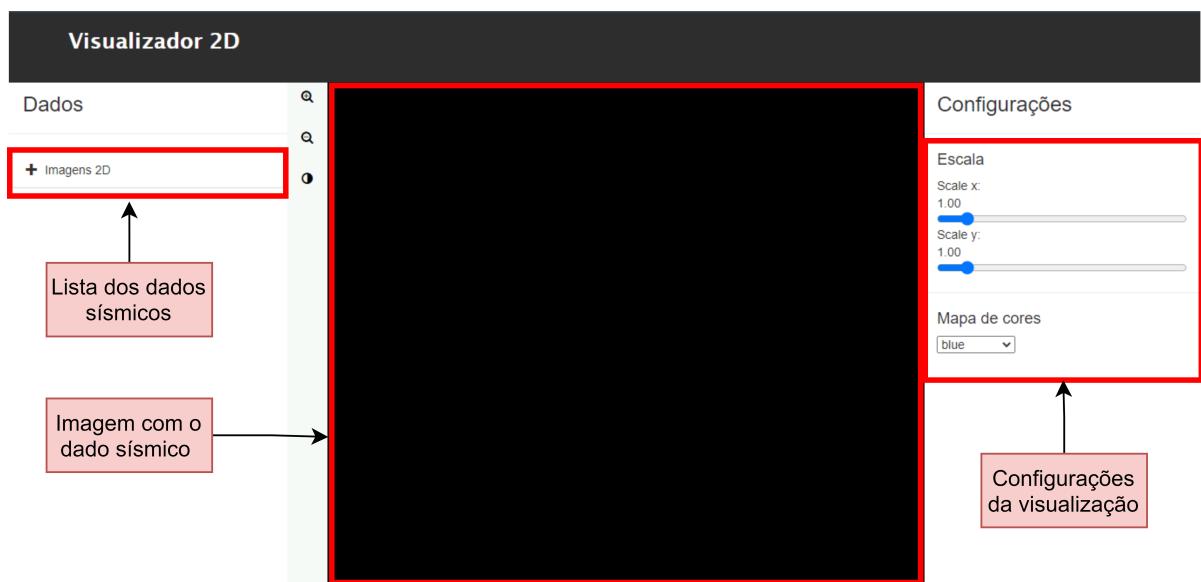


Figura 13 – Interface inicial do sistema.

## 5.3 Principais Funções do Programa

Para o usuário abrir um dado sísmico, primeiramente ele deve selecionar a opção "Imagens 2D" na interface, então a lista de dados disponíveis será exibida, e o usuário seleciona o dado que deseja visualizar, tal como na Figura 14

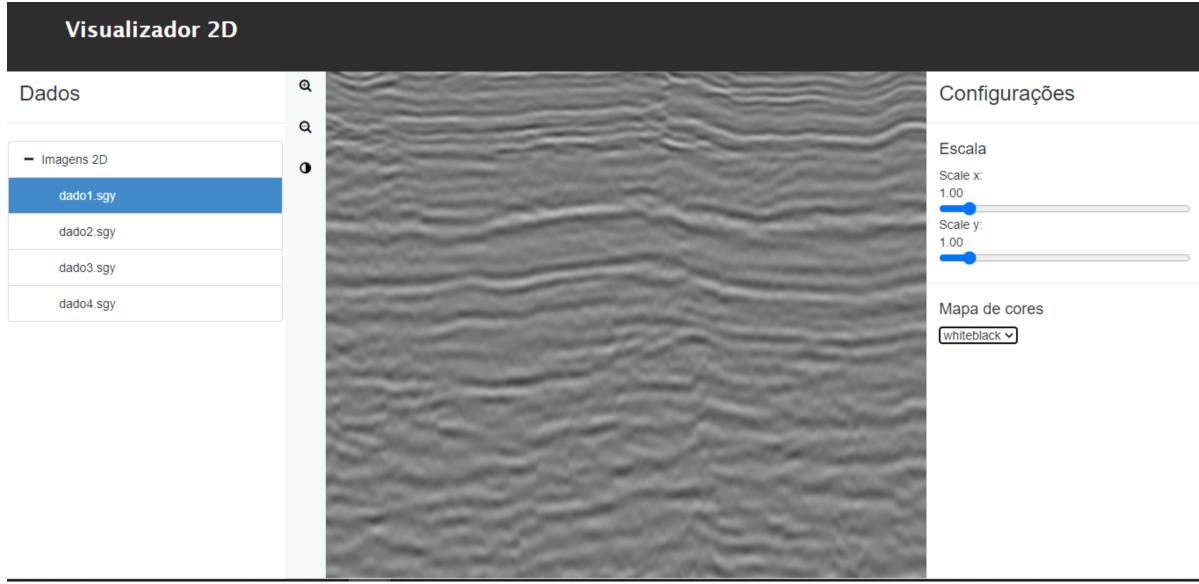


Figura 14 – Seleção de um arquivo.

Uma vez escolhido o arquivo, o dado é renderizado e mostrado na tela. Então, é possível alterar a visualização, aumentando ou diminuindo. A imagem também pode ser movimentada utilizando o mouse, o scroll tem como funcionalidade de zoom e o botão esquerdo do mouse tem a função de movimentar a imagem na tela, basta clicar na imagem e segurar o botão para movimentar a imagem. A Figura 15 mostra as funcionalidades.

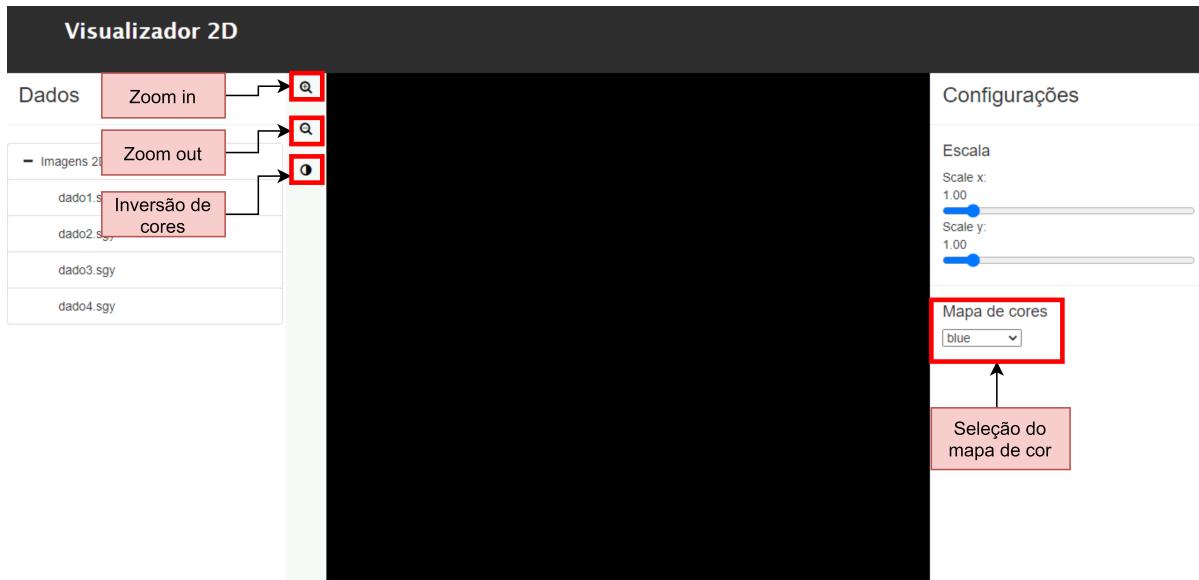


Figura 15 – Funcionalidades disponíveis na interface.

Quando a imagem é gerada na tela, algumas configurações são padrões, mas podem ser alteradas, uma delas é o mapa de cor (Figura 16) e a escala, na Figura 16 a escala do eixo x foi alterada de 1 para 0.32.

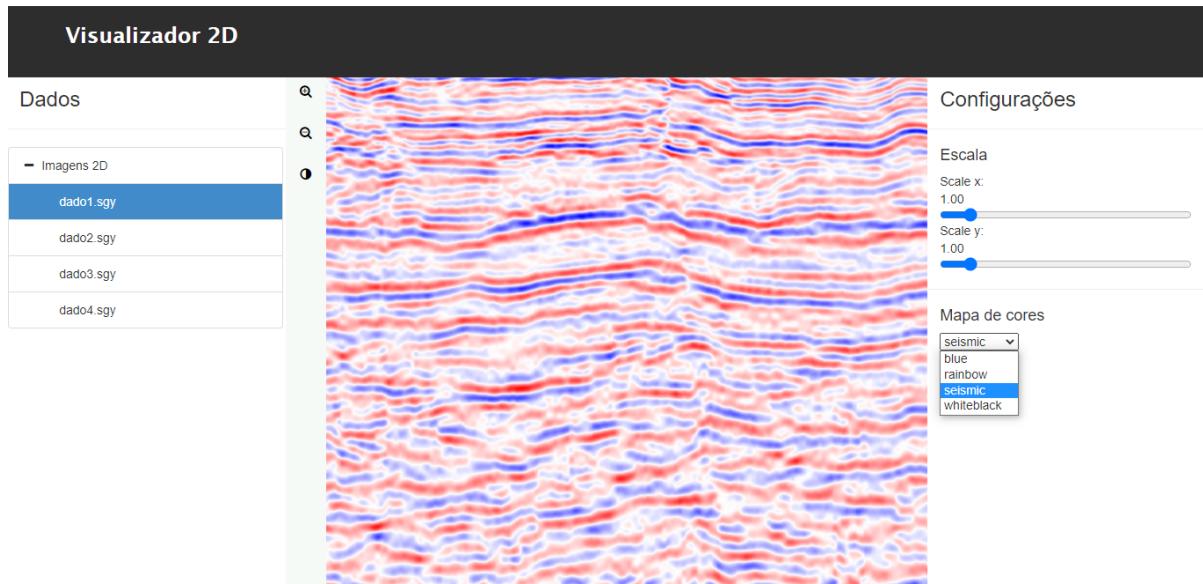


Figura 16 – Exemplo da seleção de um mapa de cor.

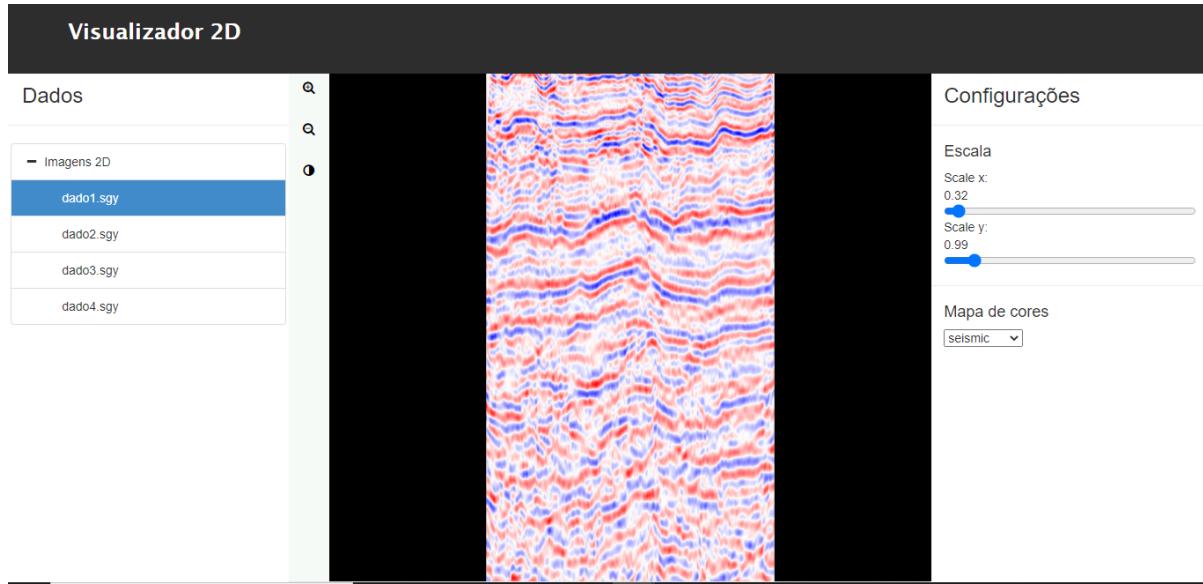


Figura 17 – Exemplo da alteração da escala.

## Referências

- ANYURU, A. *Professional WebGL programming: developing 3D graphics for the Web.* [S.l.]: John Wiley & Sons, 2012. Citado 4 vezes nas páginas [5](#), [6](#), [7](#) e [8](#).
- BARBEDO, J. G. A.; KOENIGKAN, L. V.; SANTOS, T. T. Identifying multiple plant diseases using digital image processing. *Biosystems Engineering*, Elsevier, v. 147, p. 104–116, 2016. Citado na página [6](#).
- JUNIOR, A. A. d. O. Visualizador 3d para dados de radar meteorológico usando webgl. 2013. Citado 4 vezes nas páginas [4](#), [6](#), [7](#) e [8](#).
- PARISI, T. *WebGL: up and running.* [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página [5](#).
- PARISI, T. *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages.* [S.l.]: "O'Reilly Media, Inc.", 2014. Citado 2 vezes nas páginas [4](#) e [7](#).
- PAZA, A. H. Aplicações web 3d com webgl: Visualizador de malhas. 2014. Citado na página [7](#).
- RÖSSLER, L. *Rendering interactive maps on mobile devices using graphics hardware.* Tese (Doutorado), 2012. Citado na página [4](#).
- SILVA, M. S. *HTML5: a linguagem de marcação que revolucionou a web.* [S.l.]: Novatec Editora, 2019. Citado na página [2](#).