# Pré-renderização de fórmulas matemáticas como alternativa ao uso de bibliotecas *Javascript*

**Vitor Hugo da Costa Luz**[1], **Flavio B. Gonzaga**[1]

[1]Instituto de Ciências Exatas – Universidade Federal de Alfenas (UNIFAL-MG)

***Abstract.*** *In all areas of knowledge, it is common to use mathematical formulas to define rules and observed behaviors. Based on the demand to search for this kind of content, search tools like SearchOnMath and Approach0 have been developed over the last few years. However, the search for mathematical content also comes up against another challenge: Properly rendering mathematical formulas in Web browsers. This happens because the Web environment was not originally developed to provide mathematical content. As a result, sites that need to render mathematical formulas (including search engines) tend to use Javascript libraries such as MathJax and Katex. Despite the good functioning of such libraries, there is still a gap between the set of mathematical commands that are supported by the TEX/LATEX compiler when compared to Javascript libraries. Thus, this work verifies the possibility of pre-rendering mathematical formulas using the compiler TeX Live, in order to reduce the dependence on Javascript libraries such as MathJax and Katex. The description described was only presented in the possible 40GB approach of pages followed by 12.5%, however the execution time for these pages was necessary for 15 days, thus a greater optimization of the process.*

***Resumo.*** *Em todas as áreas do conhecimento é comum a utilização de fórmulas matemáticas para a definição de regras e comportamentos observados. A partir da necessidade de buscar por esse tipo de conteúdo, ferramentas como a SearchOnMath e a Approach0 foram desenvolvidas ao longo dos últimos anos. A busca por conteúdo matemático no entanto esbarra também em um outro desafio, a renderização de fórmulas matemáticas em navegadores Web. Este problema ocorre em função do ambiente Web não ter sido originalmente desenvolvido para representar de forma fácil e prática textos matemáticos. Com isso, sites que precisam exibir fórmulas matemáticas (incluindo ferramentas de busca), utilizam bibliotecas Javascript como MathJax e Katex, que são capazes de renderizar fórmulas em formato TEX e LATEX. Apesar do bom funcionamento de tais bibliotecas, ainda existe uma distância entre o que é renderizável pelo compilador TEX e LATEX e o que é suportável pelas mesmas. Assim, o presente trabalho avalia a pré-visualização de fórmulas matemáticas usando o compilador TeX Live, de modo a se reduzir a dependência de bibliotecas Javascript como MathJax e Katex. A abordagem descrita se mostrou possível no quesito de armazenamento utilizando apenas 40GB para armazenar as fórmulas presente em 12,5% páginas, entretanto o tempo de execução para essas páginas foi de 15 dias tornando assim necessário uma maior otimização do processo.*

## 1. Introdução

Dentre todas as ciências uma das mais antigas é a matemática. O pensamento matemático se apresenta anterior aos gregos e se perpetua como essencial até hoje. Os conceitos e conhecimentos matemáticos costumam estar representados em fórmulas, por exemplo o famoso teorema de Pitágoras $a^2 = b^2 + c^2$, que foi postulado e escrito muitos anos antes da invenção do papel como conhecemos. Desde o início do pensamento matemático na humanidade, a maneira de escrever e armazenar esta informação sofreu várias alterações.

Atualmente os principais meios de armazenar e divulgar informação é através de livros e revistas impressos ou produtos digitais como *Ebooks*, páginas Web, documentos digitais entre outros. Com a consolidação dos meios digitais como o principal mecanismo de pesquisa e aprendizado com ênfase, no ambiente Web, faz-se necessário adaptar a representação dos textos matemáticos para esse segmento.

No ambiente Web nenhuma de suas ferramentas base, *HTML*, *CSS* e *JavaScript*, apresentam nativamente uma forma padrão de escrever e exibir fórmulas matemáticas. No ambiente *offline*, por outro lado, existe o padrão TEX, [Knuth 1984], que define um conjunto inicial de macros, e o LATEX [Lamport 1985] que, de certa forma, expande o padrão TEX definindo um conjunto adicional de macros. Na história recente da ciência o LATEX tornou-se um ambiente bastante popular na escrita de textos científicos, incluindo fórmulas matemáticas.

Tomando como base o padrão LATEX, foram desenvolvidas bibliotecas *JavaScript* que possibilitam a escrita e posterior renderização de fórmulas matemáticas presentes em código *HTML*. Dentre as bibliotecas desenvolvidas duas se destacam: o *MathJax* [MathJax a] e o *Katex* [Katex ].

Embora sejam bibliotecas *JavaScript* que renderizam conteúdo LATEX dentro de navegadores, é importante destacar que tais bibliotecas precisariam reimplementar o conjunto de símbolos do universo LATEX, incluindo os inúmeros pacotes disponíveis. Essa tarefa é praticamente inviável, considerando a quantidade de pacotes existentes (e que podem ser incluídos) em um código LATEX a ser compilado pelo compilador *offline*. O resultado é que nem toda fórmula compilável será também suportada por tais bibliotecas no navegador. Assim, erros de renderização de fórmulas podem ocorrer nos navegadores, e estão ligados geralmente a: (i) fórmula que faz uso de algum pacote disponível no compilador LATEX, mas ainda não implementado pela biblioteca online; ou (ii) erro de digitação cometido pelo usuário. Quando há erros de renderização, a biblioteca *MathJax*, por exemplo, exibirá o comando destacado em vermelho, ao invés da sua correta renderização.

Há de se considerar também a questão de desempenho, pois as bibliotecas citadas executam do lado cliente de uma aplicação, podendo causar uma lentidão para carregar página Web e um excesso de processamento para exibir as fórmulas formatadas.

O problema da incompatibilidade parcial entre as bibliotecas para navegadores e o compilador LATEX (executado *offline*) é também percebido no ambiente de ferramentas de busca para o universo matemático. Recentemente a ferramenta de busca SearchOnMath passou a indexar a base de pré-publicações científicas *arXiv* [ARXIV ], mantida pela universidade de Cornell. Neste caso, artigos científicos presentes na *arXiv*, e que portanto foram previamente compilados, precisarão ter seu conteúdo devidamente apresentado na

página de resultados da SearchOnMath, sendo renderizado por alguma das bibliotecas para navegadores.

O processo de renderização no caso de ferramentas Web consiste em manipular o *Document Object Model (DOM)* da página de forma criar marcações que facilitem a estilização das fórmulas em padrão LaTeX para um formato mais visual, por exemplo, a fórmula $\frac{x}{y}$ em notação LaTeX é escrita `\frac{x}{y}`.

Como os problemas de renderização também estão ligados a erros de digitação cometidos por parte dos usuários, é bastante comum encontrar esse tipo de problema em fóruns de conteúdo matemático, como o *StackExchange - Mathematics*. Assim sendo, este trabalho visa avaliar a pré-renderização de todas as fórmulas do fórum *StackExchange - Mathematics*. A escolha por este fórum se deve ao fato dele ser o maior do *StackExchange*. Com a proposta se mostrando viável, ferramentas de busca poderiam utilizar deste recurso (pré-renderizar as fórmulas) para mapear páginas contendo erros de renderização, e possivelmente tomar alguma ação nesse sentido, como por exemplo, não retornar a página.

Esta abordagem traz três vantagens: A primeira é que, intuitivamente, o compilador tende a ter menos erros de renderização que as bibliotecas *MathJax* e *Katex*, por suportar mais pacotes. A segunda vantagem é que, uma vez pré-processadas as fórmulas, poderá se conhecer previamente àquelas que resultam em erro de renderização, de modo a removê-las na fase de indexação das informações pelo buscador. A terceira vantagem é que, uma vez que as fórmulas já terão sido renderizadas, não será mais necessário utilizar bibliotecas como o *MathJax* e o *Katex* na exibição dos resultados da busca, tornando o carregamento da página mais leve. Isto é algo bastante desejável, em especial para dispositivos móveis que tendem a ter recursos mais limitados.

Assim, os objetivos do presente trabalho são: desenvolver um programa paralelo em Java que seja capaz de analisar as postagens existentes no *Mathematics*, extrair as fórmulas, e compilá-las usando o compilador LaTeX *TeX Live*. As chamadas ao compilador serão realizadas através da execução de *scripts* em *Shell* no *Linux*. É importante ressaltar que a base de dados do *Mathematics* já se encontra devidamente importada para o *SGBD MariaDB* no laboratório, assim sendo, todas as 5.575.265 páginas mapeadas já estão inseridas no *SGBD* e as fórmulas contidas nessas páginas identificadas em *tags*.

A abordagem escolhida obteve alguns resultados relevantes quanto ao espaço de armazenamento necessário e tempo utilizado para execução. O armazenamento necessário para um resultado parcial de 12,5% totalizou 40GB, demonstrando ser possível armazenar todas as imagens que representam as fórmulas da base de dados. Entretanto o tempo de execução ficou acima do esperado totalizando 15 dias para esses 12,5%, tornando assim necessária uma otimização para que este processo possa ser utilizado em ambiente de produção.

O artigo segue organizado da seguinte forma: a seção 2 aborda o Referencial Teórico. a seção 3 detalha a Metodologia. Os resultados são apresentados na seção 4. O trabalho se encerra com as Conclusões, proposta de Trabalhos Futuros e as Referências Bibliográficas.

## 2. Referencial Teórico

### 2.1. LaTeX

Como descrito na página oficial do projeto, "LaTeX é um sistema de composição de alta qualidade; inclui funcionalidades destinadas à produção de documentação técnica e científica. LaTeX é o padrão para a comunicação e publicação de documentos científicos e está disponível como Software livre".

O LaTeX foi desenvolvido na década de 80, por Leslie Lamport. Desde o princípio sua ideia era ser um maneira fácil de se escrever sem a preocupação com os elementos visuais complexos. Com o passar do tempo foram sendo criadas distribuições que implementavam mais ferramentas como *TeX Live* e *MiKTeX*. A *MiKTeX* é uma distribuição TeX/LaTeX para diversos sistemas operacionais desenvolvida por Christian Schenk, ela é uma implementação do sistema TeX com um conjunto de programas relacionados. Esta distribuição possui aproximadamente $5.325$ pacotes, criados para facilitar produção de textos ao melhorar comandos e criar composições novas mais expressivas que as originais.

Diferente de outras ferramentas concorrentes utilizadas para redação, como o *Microsoft Word* e o *LibreOffice Writer*, o documento LaTeX não possui uma formatação específica. Sendo assim, fórmulas e demais elementos são demarcados por comandos ou *tags* específicos que são interpretados pelos compiladores. Esta estrutura de comandos permite que o escritor construa um texto sem muita preocupação com a parte visual destinando seu foco para produção textual enquanto o compilador se encarrega de gerar a formatação com base na estrutura construída no texto.

### 2.2. *MathJax*

*MathJax* é um projeto Web que sucede uma biblioteca de formatação matemática *JavaScript* chamada de *jsMath*. O Projeto *MathJax* é gerenciado pela *American Mathematical Society*. Como descrito na pagina oficial do projeto, "*MathJax* é um mecanismo de exibição *JavaScript* para matemática que funciona em todos os navegadores" [MathJax a]. Esta ferramenta está presente em diversos sites como arXiv, MathSciNet, MathOverflow, Coursera, entre outros.

O *MathJax* é incorporado à página Web como uma biblioteca *JavaScript*. Uma vez carregado, ele faz uma varredura em busca de delimitadores matemáticos previamente definidos. Para a renderização dos elementos matemáticos existem diferentes abordagens. No caso mais básico, ocorre uma manipulação do *HTML* e *CSS* da página de forma a exibir as fórmulas no padrão LaTeX. Outra possibilidade é a renderização em formato MathML [MDN ], suportado por alguns navegadores. Outra parte fundamental do processo é compatibilidade do sistema com a fonte *Stixfonts* [Sti ] que é utilizada para gerar caracteres comuns em fórmulas como $\alpha$, $\beta$, $\gamma$, dentre outros. A presença desta fonte no computador local melhora velocidade de composição.

A partir da versão $2.0$ do *MathJax* foram incorporadas algumas melhorias como suporte parcial para *MathML* $2.0$ e algumas construções para *MathML* $3.0$. Foram também acrescentados novos comandos voltados para estilização de cores, bem como o suporte a mais bibliotecas LaTeX.

### 2.3. *Katex*

*Katex* é uma biblioteca que exibe notação matemática em navegadores Web. Ela se baseia na notação TEX de Donald Knuth. Seus principais focos são em performance e facilidade de uso. Esta biblioteca foi desenvolvida pelo *Khan Academy* e é o principal concorrente da *MathJax*. Em relação a sua concorrente, a biblioteca *Katex* lida com um conjunto mais limitado de comandos LATEX.

Outras características relevantes sobre a ferramenta são: (i) ela é autocontida, ou seja, não depende de outras ferramentas ou bibliotecas que estejam no pacote básico do *JavaScript*, (ii) *KaTeX* produz a mesma saída independentemente do navegador ou ambiente, sendo possível pré-renderizar expressões usando *Node.js* e enviá-las como *HTML* simples.

### 2.4. Ferramentas de Busca

Um Software de busca ou motor de busca, em inglês *search engine*, é um Software cujo objetivo é buscar documentos e dados armazenados que tenham relação com uma palavra-chave. No contexto da Internet essas palavras-chaves são buscadas em arquivos presentes na web, por exemplo o texto de páginas online.

Os motores de busca surgiram logo após a Internet com o intuito de solucionar o problema de busca de informações. Os resultados obtidos nas pesquisas costumam aparecer em páginas de resultados, em inglês conhecida por *Search Engine Results Pages* (SERP). A SERP costuma apresentar um conjunto limitado de resultados ordenados segundo critérios de relevância e atualmente é muito comum também o destaque por meio de anúncios para atingir um maior número de acessos.

Os primeiros motores de busca foram desenvolvidos baseando-se na indexação de páginas através da categorização, posteriormente foram incorporados outros mecanismos como procura por palavras-chave em páginas e o uso de referências externas espalhadas pela Web. Mais recentemente ferramentas avançadas como a do Google também incorporam processos de tradução que viabilizam a procura de termos em páginas de diferentes idiomas.

Ferramentas de busca podem ser separadas em várias categorias. Dentre as principais duas se destacam por comumente estarem presentes no dia a dia da população:

- Buscadores globais: pesquisam todos os documentos na rede, e a apresentação do resultado dependendo do ranking de acesso aos sites. Representadas por buscadores como os do Google, Yahoo e Bing;
- Buscadores verticais: realizam pesquisas “especializadas” em bases de dados próprias de acordo com suas propensões. Buscadores como SearchOnMath e a Approach0 são categorizados como verticais por serem especializados no setor matemático.

Mesmo entre esses diferentes motores alguns processos são comuns a todos eles, como o uso de *Web Crawler* [AbuKausar et al. 2013] na obtenção dos dados a partir da Internet; a indexação onde são construídos os índices e definidas as estruturas que facilitarão a busca e recuperação de informação; e a consulta web onde usuário final insere um termo que gostaria de encontrar.

Em buscadores globais é comum existir as chamadas bolhas de filtros (*Filter bubble*). As bolhas de filtros ocorrem quando algoritmos internos tentam adivinhar quais informações o usuário gostaria de acessar segundos históricos de pesquisas e outras informações armazenadas. Esses algoritmos influenciam as pesquisas de forma que dois usuários com a mesma palavra-chave obtenham resultados totalmente diferentes [Bozdag 2013]. Este tipo de filtro é menos comum em buscadores verticais devido a sua especificidade.

## 3. Metodologia

O trabalho desenvolvimento utilizou uma base de dados em modelo relacional para verificação de duas possíveis abordagens, a primeira utilizando ferramenta Selenium para manipulação do navegador Web e análise da renderização das fórmulas através do DOM da página, enquanto que, a segunda utilizou o compilador *Tex Live* e um conversor de PDF para imagem. A principal abordagem trabalhada é a de compilação.

### 3.1. Base de Dados

A base de dados que foi trabalhada se apresentava em um modelo Relacional com uma tabela de $5.575.265$ elementos, cada um desses elementos possuindo 6 atributos descritos na Tabela 1. Dentre eles co_src é o mais relevante, esta coluna é de onde foram extraídas as fórmulas a serem mapeadas. A maior parte das colunas desta base de dados estão em *mediumblob* para maximizar a economia de dados de forma a gastar apenas a quantidade de bits necessária para armazenar a informação desejada.

**Tabela 1. Colunas presentes no banco de dados**

| coluna | tipo | descrição |
|---|---|---|
| co_id | inteiro | chave primária |
| co_url | *mediumblob* | *url* da página indexada |
| co_title | *mediumblob* | título da página web |
| co_src | *mediumblob* | texto da página web |
| co_abstract | *mediumblob* | resumo da página web |
| co_math | inteiro | |

Na co_src não só existem fórmulas matemáticas como também texto comum, então foi necessário extrair as fórmulas presentes. A base cedida possui uma marcação em cada uma das fórmulas, que consiste em uma *tag* $< somX >$fórmula$< /somX >$ onde $X$ é número sequencial da fórmula naquela página. Através dessa *tag* é possível extrair todas as fórmulas já pré-identificadas.

### 3.2. Verificação de fórmulas válidas pelo *MathJax*

Como dito anteriormente, o *MathJax* é uma solução amplamente utilizada mas que não consegue tratar todos os casos e as fórmulas que não atendem aos padrões são destacadas como erros na página, o que visualmente pode ser um incômodo e causar má impressão aos leitores.

Entretanto esses erros são marcados através de *tags* específicas que podem ser utilizadas para identificar problemas na renderização. Assim sendo, é possível através de

um processo de pré-processamento definir quais fórmulas serão exibidas de forma correta. Na documentação oficial do *MathJax* [MathJax b] há algumas ferramentas descritas para lidar com erros mas nenhuma delas impede que os erros apareçam na interface.

Considerando então as *tags* presentes, a principal para demarcar erros é a "mjx-merror". Essa *tag* é exibida na estrutura *HTML* da página quando houve um erro na renderização e não foi possível interpretar a fórmula descrita. A Figura 1 mostra um exemplo de erro ocasionado por uma formula com um } a menos impossibilitando assim a interpretação da fórmula. É possível notar que o *MathJax* traz o possível motivo do erro mas ele oculta a fórmula impedindo o usuário de visualizar qual fórmula escrita que gerou o erro.

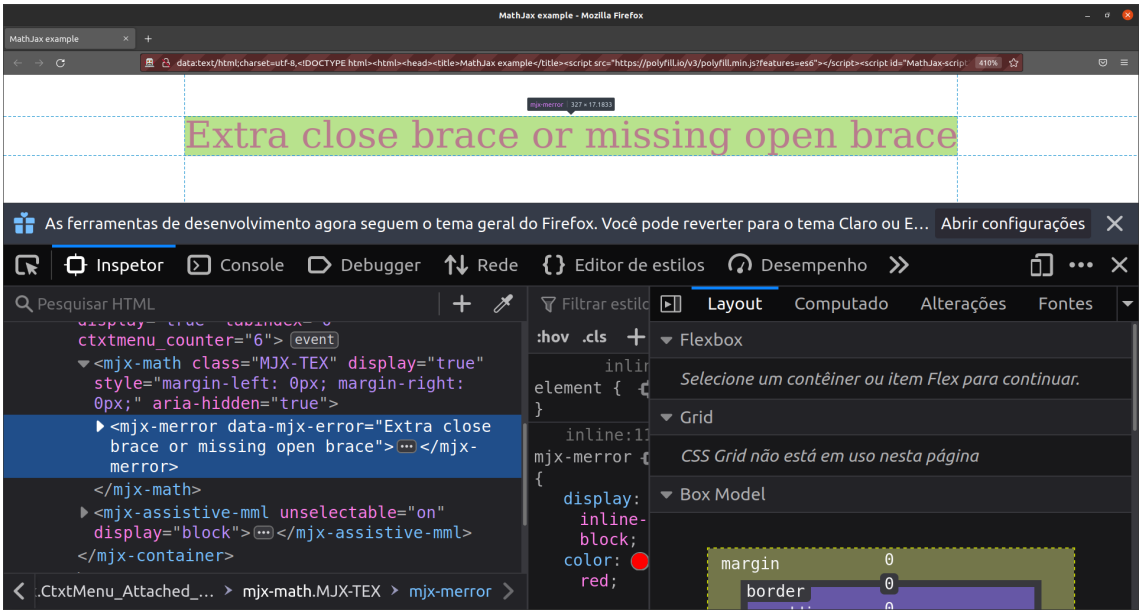


**Figura 1. Exemplo para mjx-merror**

Outra maneira de identificar um erro é menos explícita. Ela ocorre quando um caractere não foi identificado por alguma razão mas não gerou uma falha que impedisse de interpretar o resto da fórmula. Estes elementos não identificados são marcados por "mjx-mtext" com uma estilização em vermelho. A Figura 2 demonstra o caso explicado.

Em versões anteriores do *MathJax*, elementos estilizados também apareciam com a estrutura de "mjx-mtext" com uma estilização em vermelho, na versão mais recente da ferramenta (3.2.0) existe a *tag* "mstyle" que indica elementos estilizados de forma proposital pelo escritor. A nova *tag* facilita a distinção entre elementos estilizados e comandos não identificados, a Figura 3 mostra um caso onde um elemento "a" da formula foi marcado em vermelho.

Através dos casos mostrados anteriormente é possível notar que existe a possibilidade de usar *MathJax* em um processo de pré-processamento para identificar quais fórmulas de uma base de dados são válidas, entretanto o suporte a novos comandos e novas bibliotecas que podem vir a surgir fica refém das atualizações do Software impedindo assim um processo mais assertivo e independente. Além dos aspectos de dependência de novas versões, essa abordagem torna obrigatória a utilização de ferramentas que emu-

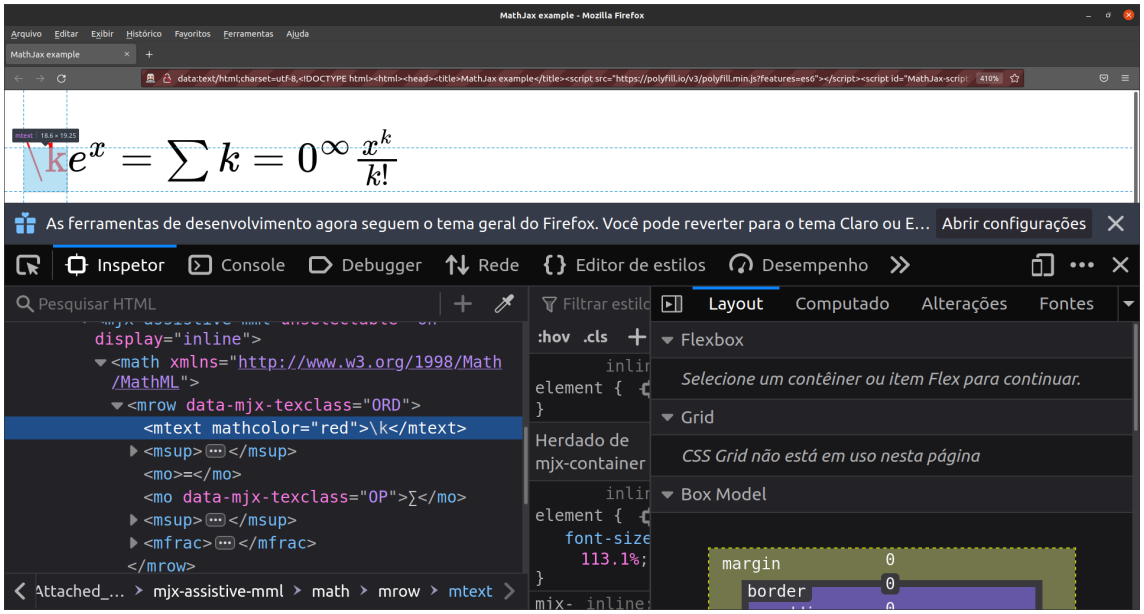


**Figura 2. Exemplo mjx-mtext com estilo em vermelho**

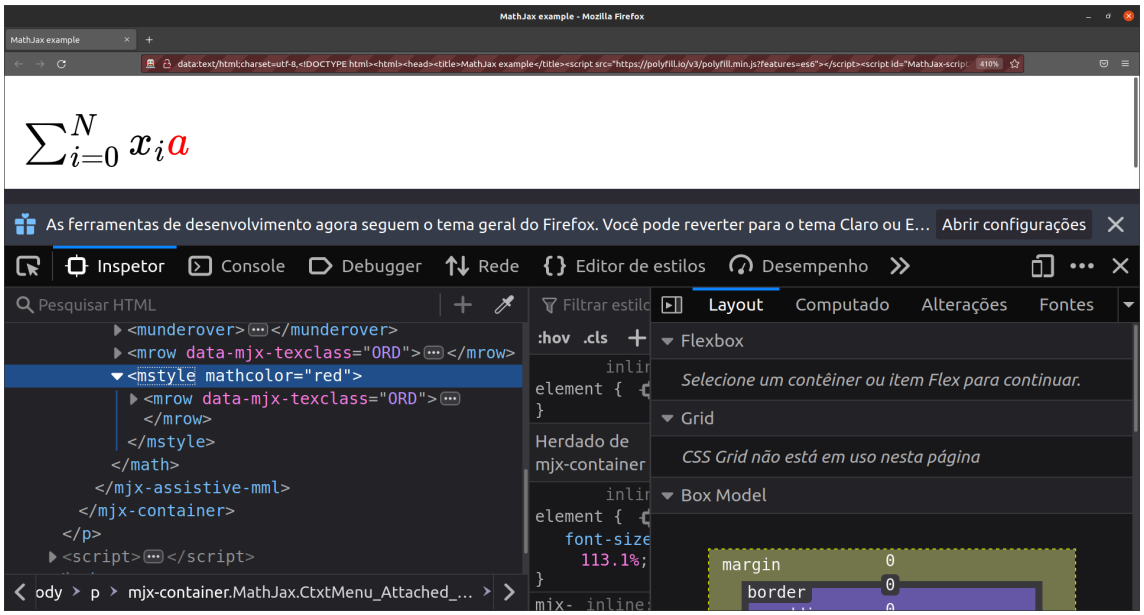


**Figura 3. Exemplo de “mstyle” para estilização**

lam ou mesmo executem navegadores Web, como o *Selenium* [Selenium ] que permite a manipulação de um navegador Web através de uma API.

### 3.3. Verificação de fórmulas e geração das imagens com o compilador

Outra abordagem para verificar fórmulas é a implementação de um processo de compilação de fórmulas LaTeX. Nesse processo, é possível identificar quais expressões utilizam pacotes ainda não configurados ou mesmo se uma expressão matemática possui algum erro de digitação que impeça a compilação. Essa abordagem permite ainda a criação de uma base de dados contendo as imagens de todas as fórmulas LaTeX compiláveis contidas no banco de dados. Como já mencionado anteriormente, isso eliminaria a necessidade do

uso de bibliotecas como o *MathJax* em casos como a geração da páginas de resultados em buscadores.

Para a obtenção das imagens, foi criado um *script* em *Shell* que executa os seguintes passos:

1. compilação do arquivo TEX resultando em um documento PDF cujo conteúdo é uma página contendo apenas a fórmula renderizada.
2. corte do documento PDF gerado no passo anterior, de modo a se gerar um novo arquivo PDF contendo somente a fórmula (e não mais uma página).
3. geração da imagem da fórmula em formato SVG a partir do PDF.
4. otimização do arquivo SVG através da ferramenta SVGO [SVG ].

O Formato SVG foi escolhido especificamente por ser leve e vetorial facilitando assim sua utilização em aplicações que necessitam de responsividade. A otimização do arquivo SVG utilizando-se da ferramenta SVGO foi feita com a finalidade de se reduzir o tamanho dos arquivos de imagens, uma vez que a SVGO elimina redundâncias e metadados não essenciais para o formato de arquivo.

Para a geração das imagens das fórmulas, foi utilizado um arquivo LATEX como base, apresentado na Figura 4. Assim, cada fórmula extraída da base de dados foi inserida no corpo desse arquivo, com o processo de compilação sendo iniciado em seguida.

```
\documentclass{article}
\usepackage{amssymb}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{xcolor}
\thispagestyle{empty}
\newcommand{\\gt}{>}
\newcommand{\\lt}{<}
\newcommand{\\Space}{\\}
\begin{document}
%Região para inserir fórmula
\end{document}
```

**Figura 4. Arquivo base compilação**

Uma vez que as imagens tenham sido criadas, um outro problema a ser resolvido é que geralmente, ferramentas de busca matemáticas terão em seu índice somente as fórmulas em formato TEX. Ou seja, as imagens não estarão armazenadas no índice, mas em um repositório "externo" à arquitetura convencional do buscador. O buscador precisa portanto de uma forma de determinar qual o nome do arquivo de imagem que é referente à fórmula em formato TEX que ele obteve de seu índice textual. Exemplificando, o buscador pode ter obtido de seu índice a fórmula:`f(x)=\sum_{r=0}^{\infty}f^r(c)/r!(x-c)^r`. No repositório de imagens, existe um arquivo cujo conteúdo é a imagem da fórmula já previamente renderizada, ou seja: $f(x) = \sum_{r=0}^{\infty} f^r(c)/r!(x-c)^r$. A pergunta é, como determinar o nome desse arquivo a partir do seu código TEX, que é a única informação que o buscador possui em seu índice.

A forma encontrada de resolver esse problema foi utilizar uma função HASH, especificamente a SHA2-512 [U.S. Department of Commerce and Technology 2011], que

converte um texto de entrada em um hash de saída. A saída da função foi ajustada de modo a se gerar somente sequências contendo 128 caracteres. Isso é importante pois é um limite que não extrapola o limite para nomes de arquivo nos sistemas operacionais modernos.

Assim, no processo de pré-processamento das fórmulas, uma fórmula ao ter seu arquivo de imagem SVG gerado, o nome do arquivo SVG será o resultado do código TEX da fórmula submetido à função HASH SHA2-512. Posteriormente, o buscador será capaz de determinar o nome do arquivo no repositório de imagens da mesma forma. Ou seja, submetendo o código TEX da fórmula obtida a partir do índice textual à mesma função HASH.

Mesmo funções HASH tendo por objetivo gerar resultados únicos de criptografia, existe a possibilidade de colisão. Tal evento ocorre quando duas entradas distintas $A$ e $B$ resultam a mesma saída. Para identificar colisões, no momento da geração das imagens a partir das fórmulas em formato TEX, para cada fórmula, dois arquivos foram armazenados. Um contendo a imagem (fórmula renderizada com extensão SVG), e outro contendo a fórmula em si (arquivo textual com extensão TEX).

Para melhor compreensão, considere uma fórmula simples, como um $x$. Essa fórmula ocorrerá inúmeras vezes dentro de fóruns matemáticos como o *StackExchange - Mathematics*. Assim, a primeira vez que o algoritmo detectar essa fórmula, seu arquivo de imagem (SVG) e seu arquivo textual (TEX) serão gerados. Prosseguindo na geração das imagens, sempre que a fórmula $x$ aparecer, o algoritmo verificará que o arquivo já existe, e consultará o conteúdo do arquivo com extensão TEX. Caso o conteúdo desse arquivo seja $x$, indica que é a mesma fórmula, e que por já ter sido renderizada, não precisará ser renderizada novamente. No entanto, se uma outra fórmula, diferente de $x$, gerar o mesmo nome de arquivo, isso indica uma colisão de HASH. Esse evento será gravado em log pelo algoritmo.

Os passos executados pelo algoritmo são apresentados no fluxograma da Figura 5. Na próxima seção são apresentados os detalhes de execução do algoritmo proposto bem como os resultados obtidos.

## 4. Resultados

O sistema descrito na seção anterior foi implementado de modo a executar de acordo com a arquitetura exibida na Figura 6. Nela pode-se observar que as páginas com as postagens de conteúdo matemático obtidas a partir do fórum *StackExchange - Mathematics* estão centralizadas em um servidor de Banco de Dados que executa o *SGBD MariaDB*. Os demais computadores da arquitetura se comportam como clientes, obtendo um conjunto de páginas por vez, processando e gravando os arquivos de imagem (gerados após a compilação) localmente.

A base de dados completa do *StackExchange - Mathematics* utilizada no trabalho foi obtida em setembro de $2021$, e é composta por $5.783.148$ posts diferentes. O sistema foi executado utilizando-se 8 máquinas do Laboratório de Redes de Computadores e Sistemas distribuídos (LaReS) da Universidade Federal de Alfenas (UNIFAL-MG). As máquinas utilizadas possuem processador Intel i7-7700 com 8 núcleos, 16GB de memória RAM e 1TB de disco rígido de tecnologia IDE.

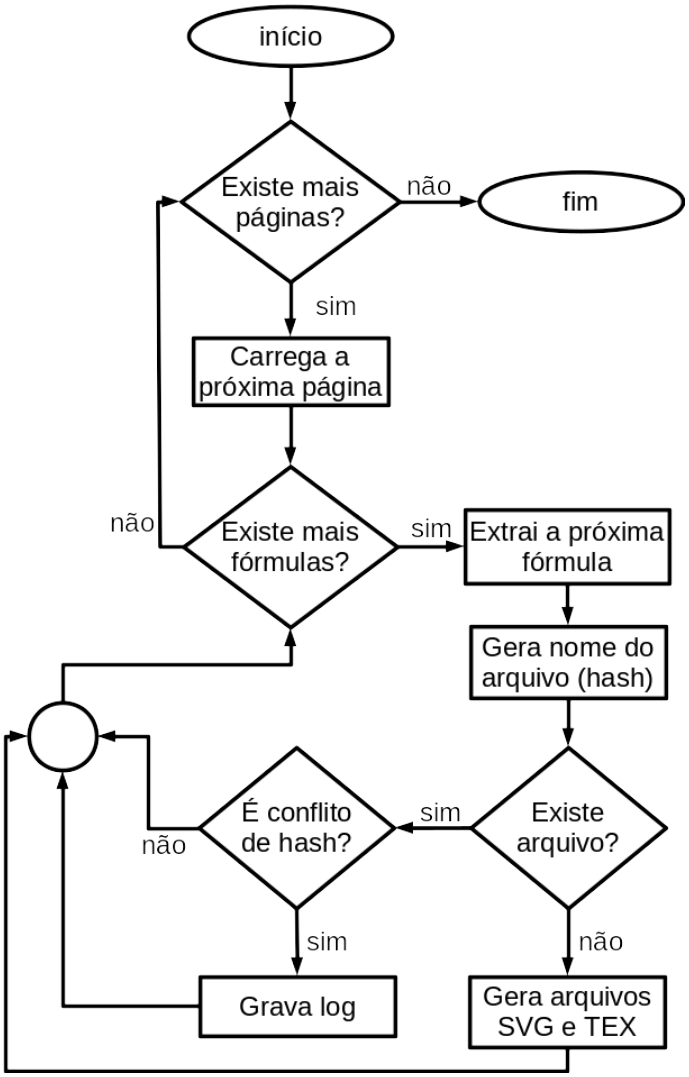


**Figura 5. Fluxograma utilizando compilador Latex**

Inicialmente fez-se uma tentativa de execução considerando-se toda a base de dados, mas constatou-se que o tempo de execução seria demasiadamente longo. Diante deste fato, optou-se por considerar apenas uma parte da base de dados inicial ($722.893$, aproximadamente $12,5\%$). O tempo de execução do algoritmo para esta amostragem da base de dados inicial foi de $15$ dias. Este traz inicialmente duas considerações, que poderão fundamentar a continuidade deste trabalho futuramente: i) por ser um sistema que realiza muitas operações de *I/O*, qual seria o seu desempenho caso as máquinas tivessem discos rígidos de alto desempenho, com tecnologia *SSD*? ii) haveria ainda espaço para otimização do código fonte da aplicação?

Apesar do alto tempo de execução, que é algo a ser considerado e melhor estudado, o trabalho apresentou resultados relevantes, detalhados a seguir.

A quantidade de disco ocupado em cada máquina pela aplicação foi de 5GB em média, considerando-se as fórmulas renderizadas em formato SVG e os arquivos TEX contendo o texto das fórmulas. O que significa que, para a base de dados completa do *StackExchange - Mathematics*, cerca de 320GB de espaço devem ser suficientes para o

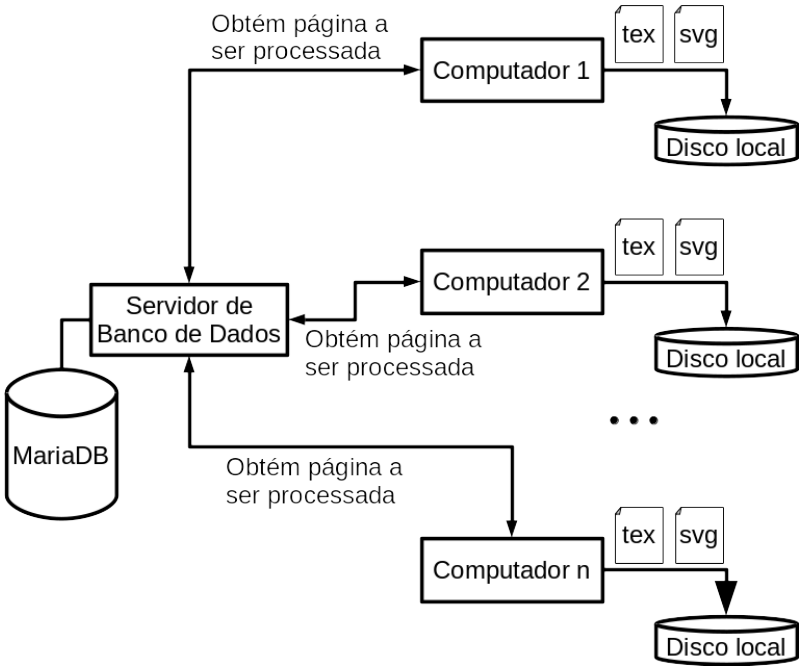


**Figura 6. Arquitetura do sistema**

armazenamento de todas as imagens (em formato SVG) e arquivos TEX (contendo o texto das fórmulas). É importante lembrar que, para fins de ambiente de produção, apenas os arquivos SVG são necessários. Então o espaço para o armazenamento seria ainda menor. Outro detalhe relevante é que nenhuma colisão foi detectada utilizando a função HASH como geradora dos nomes dos arquivos.

Por fim, outro resultado interessante é a relação entre a quantidade de fórmulas corretamente renderizadas e aquelas que resultaram em erro de compilação. Ao término da execução da aplicação, $3.158.280$ de fórmulas foram renderizadas (considerando-se a soma da quantidade de arquivos em cada uma das $8$ máquinas). A quantidade de fórmulas que resultaram em algum erro de compilação, também somando-se os registros de log de cada uma das máquinas, foi de $34.488$ fórmulas, (aproximadamente $1\%$ do total de fórmulas renderizadas). Este valor (de formulas com erro) pode ainda ser reduzido em uma continuação do trabalho. Uma abordagem inicial seria verificar no log quais fórmulas fazem o uso de comandos que estejam presentes em pacotes que não tenham sido inseridos no arquivo LaTeX originalmente usado pela aplicação (exibido na Figura 4). Esta ação certamente resultará em uma redução de fórmulas com erro, uma vez que uma quantidade maior de comandos passará a ser suportada.

## 5. Conclusões e Trabalhos futuros

O trabalho proposto demonstrou uma alternativa ao uso de bibliotecas *Javascript* para a renderização de conteúdo LaTeX em navegadores Web, em especial quando já se possui previamente a coleção de documentos a serem exibidos (como no caso de ferramentas de busca).

A execução do trabalho mostrou que o resultado é viável do ponto de vista de espaço de disco, e a abordagem de se utilizar uma função HASH como geradora do nome

dos arquivos também se mostrou bastante eficaz. O percentual de fórmulas com erro de compilação também foi relativamente baixo (cerca de $1\%$) em comparação com o conjunto de fórmulas renderizadas com sucesso.

Em comparação a abordagem utilizando *JavaScript* o processo utilizando compilador torna o pré-processamento das fórmulas independe de atualizações da ferramenta de renderização, além de criar uma base de imagens a ser utilizada em aplicações Web e *Mobile*.

Os pontos de continuação do trabalho proposto se concentram principalmente em duas frentes. A primeira é a busca pela redução do tempo de execução, ainda bastante elevado considerando-se a base completa. O uso de máquinas com disco rígido *SSD*, bem como possíveis otimizações no algoritmo, são aspectos a serem considerados. A segunda frente consiste em buscar a redução do índice de fórmulas que resultaram em erro de compilação. Esta tarefa consistiria inicialmente em se acrescentar mais pacotes ao arquivo LaTeX original. Uma abordagem um pouco mais sofisticada seria montar um arquivo LaTeX personalizado para cada fórmula, incluindo-se somente os pacotes necessários. Esta abordagem causa um aumento no tempo de análise da fórmula (para se descobrir quais pacotes são necessários), mas uma redução no tempo gasto no processo de compilação.

Por fim, entende-se que o trabalho proposto pode ser, com alguns ajustes, uma alternativa ao uso de bibliotecas *Javascript* para a renderização de conteúdo em navegadores.

## Referências


Stixfonts. `https://www.stixfonts.org`. Acesso em 05 de março de 2022.

AbuKausar, M., S. Dhaka, V., and Kumar Singh, S. (2013). Web crawler: A review. *International Journal of Computer Applications*, 63(2):31–36.

ARXIV. Svgo repository. `https://arxiv.org`. Acesso em 09 de abril de 2022.

Bozdag, E. (2013). Bias in algorithmic filtering and personalization. *Ethics and Information Technology*, 15(3):209–227.

Katex. Katex. `https://katex.org`. Acesso em 02 de março de 2022.

Knuth, D. E. (1984). *The TeX Book*. Addison-Wesley, $1^{st} edition$.

Lamport, L. (1985). *Latex: A Document Preparation System*. Addison-Wesley, $1^{st} edition$.

MathJax. Mathjax. `https://www.mathjax.org`. Acesso em 11 março de 2022.

MathJax. Mathjax documentation. `http://docs.mathjax.org/en/latest/`. Acesso em 05 de março de 2022.

MDN, M. Mathml. `https://developer.mozilla.org/pt-BR/docs/Web/MathML`. Acesso em 05 de março de 2022.

Selenium. Selenium documentation. `https://www.selenium.dev`. Acesso em 09 de abril de 2022.

SVG. Svgo repository. `https://github.com/svg/svgo`. Acesso em 09 de abril de 2022.

U.S. Department of Commerce, N. I. o. S. and Technology (Fevereiro de 2011). Secure hash standard (shs). Technical report, National Institute of Standards and Technology Gaithersburg.