Redimensionamento de Imagem: Implementação do Seam Carving

Henrique Hott e Guilherme Bittencourt

Maio 2022

1 Introdução

O redimensionamento de imagem é uma funcionalidade utilizada constantemente no ramo digital. Atualmente com a diversidade de dispositivos, é necessário redimensionar as imagens para melhor adaptação das telas. Porém, torna-se necessário que ao reduzir ou aumentar o tamanho, não se perca muitos dados da imagem, sem diminuir o nível de detalhamento. Pode-se exemplificar uma de suas utilizações no mundo dos jogo. A surpreendente dificuldade de redimensionar spritessheets demonstra a necessidade de resoluções eficientes para o problema.

Neste trabalho, iremos utilizar a técnica de Seam Carving, desenvolvida por Shai Avidan, da Mitsubishi Electric Research Labs, e Ariel Shamir, do The Interdisciplinary Center MERL. Será utilizado dois métodos distintos para solucionar o problema: A programação dinâmica utilizando representação por matriz e o algoritmo de Djikstra utilizando representação por grafos.

O Seam Carving é um procedimento que utiliza a energia de cada pixel para determinar sua relevância na imagem. Assim, pixels com baixa energia não demonstram um papel importante na composição dos detalhes, podendo ser retirados. Por tanto, busca-se o caminho entre as bordas de menor energia para ser removido da imagem.

A motivação para o trabalho é devido a importância do tema para a tecnologia. Atualmente torna-se cada vez mais necessário o redimensionalmento de imagens. Por isso há uma grande busca por soluções mais eficientes. O objetivo é utilizar os conhecimentos adquiridos em sala com informações extracurriculares e implementar uma solução que seja eficiente em tempo e espaço. Analisar os resultados obtidos e avaliar pontos importantes acerca da problemática em questão.

2 Entrada e saída de dados

A imagem que será reduzida será do tipo PPM (Portable Pixmap Map), um tipo de imagem que possibilita sua leitura em formato de texto. O arquivo de entrada conterá uma string característica "P3", a largura e altura da imagem respectivamente e a sequência de cores de cada pixel. O pixel é formado por um vetor tridimensional RGB, cujos componentes são os valores de vermelho (Red), verde (Green) e azul (Blue).

No arquivo de entrada poderá conter hashtags (#) no começo das linhas, o que significa que são comentários, e espaços em branco arbitrários.

O arquivo de saída deverá conter o tipo da imagem PPM, a largura e altura atual e a sequência de pixels não removidos.

3 Caminho de dados

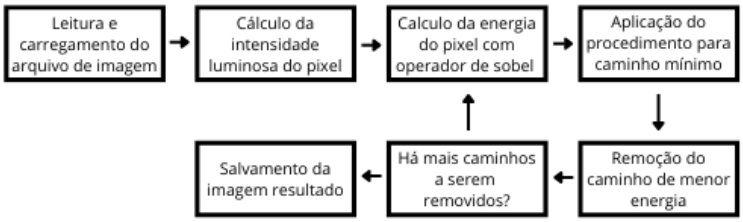

Figura 1: Caminho de execução do programa

3.1 Calculo de energia

(1) - No primeiro passo, é feito a leitura do arquivo de imagem passado por parâmetro.

(2) - Em seguida, é calculado a intensidade luminosa de cada pixel. A intensidade luminosa se dá por uma equação de pesos, onde cada peso representa a sensibilidade do olho humano para cada cor RGB:

$$\forall pixelnamatriz \rightarrow IL(R,G,B) = 0,30R + 0,59G + 0,11B \quad (1)$$

(3) - Posteriormente é aplicado o operado de Sobel que multiplica uma matriz de pesos no pixel analisado e todos os oito adjacentes, calculando o vetor gradiente que representa o valor de menor variação do claro ao escuro em relação aos seus vizinhos. O objetivo é detectar pontos onde não há bordas de objetos na imagem, evitando anomalias na estrutura:

Figura 2: Aplicação do operador de Sobel

Para determinar o vetor gradiente, o operador aplica uma matriz 3x3 nas intensidades luminosas do pixel central e seus oito vizinhos e calcula as derivadas parciais na direção vertical e horizontal:

$$\mathbf{P} = \begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \\ \mathbf{d} & \mathbf{e} & \mathbf{f} \\ \mathbf{g} & \mathbf{h} & \mathbf{i} \end{bmatrix} \quad G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figura 3: Matrizes de peso do operador de Sobel

$$Gx = a*(1) + b*(0) + c*(-1) + d*(2) + e*(0) + f*(-2) + g*(1) + h*(0) + i*(-1)$$
$$Gy = a*(1) + b*(2) + c*(1) + d*(0) + e*(0) + f*(0) + g*(-1) + h*(-2) + i*(-1)$$
$$e(p) = \sqrt{(Gx)^2 + (Gy)^2}$$

Para os pixels localizados nas bordas da imgem, onde não há adjacentes para alguns de seus lados, foi implementado uma lógica que utiliza o valor da intensidade do próprio pixel e de seus adjacentes para que seja efetuada a operação, como mostra na figura 4.

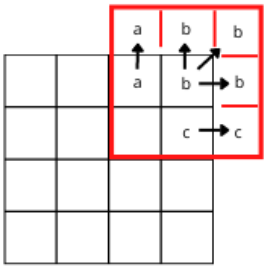

Figura 4: Calculo de pixels nas bordas

(4) - Em seguida, é executado o procedimento requerido (Programação dinâmica ou Dijkstra) e é calculado o caminho mínimo.

(5) - No quinto passo, ocorre a remoção do caminho calculado no passo anterior. Posteriormente, o próximo passo definirá se será necessário remover mais colunas ou terminará o programa salvando a imagem.

4 Programação Dinânimca

4.1 Solução

A programação dinâmica é um paradigma de programação baseada no princípio da otimalidade e pode ser aplicado em problemas que possibilitam a obtenção de subestruturas ótimas. As soluções dos subproblemas são calculados e memorizados para serem usados nos próximos passos.

4.2 Estrutura de Dados

Iniciamos o algoritmo recebendo os dados do imput e colocando dentro de uma matriz de pixels de dimensões w x h, onde w é a largura e h a altura da imagem. A matriz foi implementada utilizando aritmética de ponteiros. Cada par de coordenadas (x, y) representa um pixel da imagem. Na estrutura é armazenado as dimensões iniciais da imagem e as dimensões atuais.

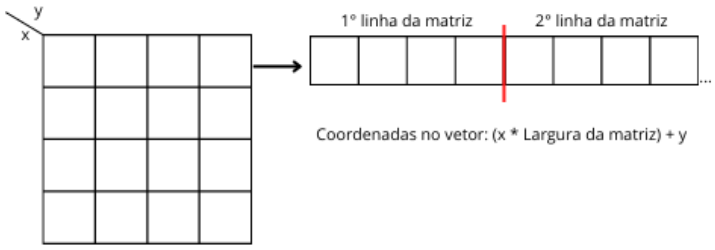

Figura 5: Estrutura da matriz de pixels

4.3 Análise de Complexidade

```
Carregar imagem () {
    Aloca matriz e espaço
    para os pixels;

    Armazena w e h da imagem ();

    Para cada pixel, armazena
    valor RBG ();
}
```

```
Calcula caminhos () {
    Se pixel já checado,
    retorna;

    Se ultima linha {
        Caminho = Energia(pixel);
    Se não {
        Para cada adjacente {
            Calcula caminhos ();
            Analisa caminho;
            Escolhe adjacente de
            menor caminho;
        }
    }
}
```

```
Remove colunas () {
    Para cada coluna a ser
    retirada {
        Calcula energia dos
        pixels ();
        Calcula caminhos ();
        Obtém caminho mínimo ();
        Remove caminho ();
        Reseta estado dos pixels
        restantes ();
    }
}
```

```
Remove linhas () {
    Transpõe matriz ();
    Remove colunas ();
    Transpõe matriz ();
}
```

```
Remove linhas e colunas () {
    Para cada coluna ou linha
    a ser retirada {
        Obtém caminho mínimo ();
        Transpõe matriz ();
        Obtém caminho mínimo ();
        Se caminho mínimo == linha {
            Remove coluna ();
        Se não {
            Transpõe matriz ();
            Remove coluna ();
        }
    }
}
```

Figura 6: Pseudocódigo de matrizes

4.3.1 Carregar Imagem

A função de carregar a imagem é responsável por alocar o vetor matriz da estrutura, armazenar a largura e altura da imagem e processar para todos os pixels seus respectivos valores de RGB. Pelo fato de percorrer todos os pixels da imagem, sua complexidade se da por w x h operações, o que no pior caso leva a complexidade quadrática.

$Complexidadedetempo : O(n^2)$

4.3.2 Calcular Caminhos

1 - Dentro da função de calculo de caminhos, o caso de término da recursão é definido: Cada pixel recebe uma variável de status, a qual fornece ao algoritmo a instrução se aquele elemento já foi checado ou não. Assim, quando se encontra um elemento que já foi checado, o valor do caminho até ele é retornado para o anterior.

2 - No caso base da recursão significa que o programa chegou na última linha da matriz. A variável de status recebe uma instrução para o procedimento de remoção de colunas e o caminho para chegar até o referente pixel é sua própria energia, já que não possui adjacentes abaixo dele.

3 - Caso não estiver na última linha da matriz, para cada adjacente do pixel, é calculado o possível caminho até ele. Ao final da operação, escolhe-se o mínimo entre os vizinhos abaixo e é determinado o caminho a ser seguido posteriormente.

Dado a explicação do procedimento, a análise de tempo é dado da seguinte forma: No primeiro instante, é verificado se o pixel já foi checado. Caso afirmativo, retorna a função. Caso não, inicia-se chamadas recursivas a fim de chegar até os pixels da última linha da matriz. O custo para percorrer uma árvore de recursão é o logaritmo da entrada, que significa sua altura. Posteriormente, calcula-se o melhor caminho para cada pixel, retornando a valor do caminho percorrido para o pixel antecessor. Pelo fato da chamada recursiva formar uma árvore de recursividade onde a cada chamada a entrada é dividida por três, tem-se como função de recursividade: $T(n) = T(n/3) + c$.

$Complexidadedetempo : O(log(n))$

4.3.3 Remove Colunas

Dado o número de colunas a serem retiradas, para cada uma é necessário calcular as energias de cada pixel, o que leva complexidade quadrática por percorrer toda a matriz. Calcula-se os caminhos mínimos com complexidade logaritmica. Para cada caminho encontrado, verifica-se o menor caminho entre eles, complexidade linear. Remove o caminho escolhido e coloca todos os pixels restantes em estado de não checado para a próxima iteração. Dado o pior caso, onde todas as colunas serão removidas, a complexidade da função é dada pelo máximo entre as funções no laço vezes a quantidade de vezes que o laço é executado. No pior caso, serão removidas todas os pixels da imagem.

$Complexidadedetempo : O(n^3)$

4.3.4 Remover Linhas

A função para remover linhas se dá pela complexidade da função de transpor, dada por complexidade quadrática, por percorrer toda a estrutura da matriz, a qual é executada duas vezes, e a complexidade da remoção de colunas, a qual é cúbida. Dado o máximo entre as funções.

$Complexidadedetempo : O(n^3)$

4.3.5 Remove linhas e colunas

Como ação extra do programa, foi implementado uma função que remove linhas e colunas alternadamente dado o melhor caminho adquirido entre elas. Pelo fato de utilizar funções, as quais a maior complexidade é dada cúbida, sua análise se torna semelhante.

$Complexidadedetempo : O(n^3)$

4.4 Paradigma de Programação Dinâmica

A programação dinâmica é um método eficiente para solucionar o problema em questão. O princípio da otimalidade garante que os resultados sejam ótimos, já que garante que os subresultados também sejam ótimos.

Provaremos por indução matemática que a programação dinâmica pode obter um resultado ótimo para o redimensionamento de imagem: Seja uma matriz onde cada célula possui um valor. É preciso calcular o menor caminho do topo até o fundo da matriz e removê-lo da estrutua. Caso base: Calcula-se o menor caminho do primeiro pixel até ele mesmo, sendo assim, sua propria energia.

$$Min(a, a) = energy(a) \tag{2}$$

Para a segunda coluna, calcula-se o melhor caminho de cada pixel somando sua propria energia com o custo gasto para chegar até ela. Assim, todos os pixels armazenam o pixel anterior a ele que possui o menor caminho. Passo indutivo: Para k + 1 passos, executa a mesma lógica do algoritmo, sempre obtendo o menor caminho para cada pixel.

$$Min(a, k + 1) = minEnergy(a, k) + energy(k + 1) \tag{3}$$

Logo, ao final da execução, cada pixel guarda o seu anterior de menor caminho. Portanto, garante-se a solução ótima por subsoluções ótimas.

5 Grafos

5.1 Solução

Na implementação de grafos, criamos uma matriz de vértices, onde cada um representa um pixel e possui uma lista de adjacencia para cada um de seus vizinhos. Para se obter o caminho de menor energia, aplica-se o algoritmo de Dijkstra.

5.2 Estrutura de Dados

O grafo foi construindo utilizando uma matriz de pixels, onde cada pixel guarda uma lista de adjacência para cada um de seus vizinhos. É armazenado no pixel a sua energia assim como a energia necessária para chegar até ele. Na utilização

do Dijkstra, todos os caminhos serão setados como infinito. É colocado uma referência para guardar o seu antecessor e as referências para todos os seus adjacentes, como indica na figura 8.

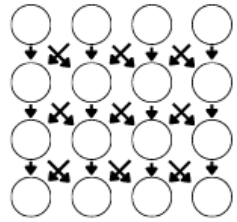

Figura 7: Estrutura da matriz baseada em grafos

5.3 Análise de Complexidade

```
Carregar imagem () {
    Aloca grafo e espaço
    para os pixels;

    Armazena w e h da imagem ();

    Para cada pixel, armazena
    valor RBG ();

    Preenche lista de adjacência de
    cada pixel ();

    Ponto de partida para Dijkstra ();
}
```

```
Organiza Heap () {
    Para cada item da heap {
        Se pai for maior que filho {
            Troca ();
        }
    }
}
```

```
Dijkstra () {
    Para cada pixel, caminho
infinito;

    Constrói heap ();
    Para cada item na heap {
        Para cada adjacente do item {
            Analisa adjacente;
            Escolhe menor caminho
            para adjacente;
        }
    Organiza heap ();
    }
}
```

```
Remove coluna () {
    Para cada coluna a ser
    retirada {
        Calcula energia dos pixels ();
        Dijkstra ();
        Remove caminho ();
        Refaz bordas ();
}
```

```
Remove linha () {
    Transpõe matriz ();
    Remove colunas ();
    Transpõe matriz ();
}
```

```
Remove linhas e colunas () {
    Para cada coluna ou linha
    a ser retirada {
        Obtém caminho mínimo ();
        Transpõe matriz ();
        Obtém caminho mínimo ();
        Se caminho mínimo == linha {
            Remove coluna ();
        Se não {
            Transpõe matriz ();
            Remove coluna ();
        }
    }
}
```

Figura 8: Pseudocódigo de grafos

5.3.1 Carregar Imagem

A função de carregar a imagem é responsável por alocar o vetor de vértices da estrutura, armazenar a largura e altura da imagem e processar para todos os pixels seus respectivos valores de RGB. Ela preenche a lista de adjacência dos pixels e determina o ponto inicial da operação do Dijkstra. Pelo fato de percorrer todos os pixels da imagem, sua complexidade se dá pela área varrida. Logo:

$Complexidadedetempo : O(n^2)$

5.3.2 Organiza Heap

A função para organizar a heap faz uma comparação entre os pais e filhos da árvore gerada na construção, trocando caso o pai seja maior que algum de seus filhos. Logo, sua complexidade se dá pela altura correspondente da árvore gerada.

$Complexidadedetempo : O(log(n))$

5.3.3 Dijkstra

1 - Primeiramente, é atribuído infinito para os caminhos e o anterior como nulo para todos os vértices do grafo.

2 - Cria-se uma heap mínima que guardará os possíveis caminho que podem ser tomados pelo Dijkstra. Assim, sabe-se que o primeiro item a ser retirado da heap é o que apresenta o menor caminho calculado. Inicialmente, apenas os valores

da primeira linha são adicionados np heap. Ao decorrer da execução, os adjacentes são adicionados com o valor do caminho calculado.

3 - Ao se retirar o valor inicial da heap, calcula-se o valor do caminho até cada um de seus adjacentes, adicionando-os na heap. Repete o processo até que se encontre o menor caminho possível. Ao final, a heap é reordenada.

O algoritmo de Dijkstra puro tem complexidade do número de arestas percorridas vezes a altura da árvore gerada pelo caminho, porém, dentro da função existe um laço aninhado que coloca os valores do caminho de cada pixel como infinito. Devido a isso, sua complexidade se diferencia para quadrática.

$Complexidadedetempo : O(n^2)$

5.3.4 Remover colunas

A função para remover colunas executa seu laço para cada coluna a ser retirada. No interior do loop, chama-se a função de calcular energia de cada pixel do grafo, que possui complexidade quadrática. Em seguida, executa-se o Dijkstra para se encontrar o caminho mínimo, remove o caminho encontrado e refaz as referências dos pixels das bordas da imagem reduzida. Sua complexidade é cúbida devido a complexidade do Dijkstra vezes o número de colunas retiradas no pior caso.

$Complexidadedetempo : O(n^3)$

5.3.5 Remove linha

Analogamente, a função para remover linhas se dá transpondo a imagem, a qual tem complexidade quadrática. Remove a coluna calculada com complexidade cúbida e transpõe novamente o grafo. Vale ressaltar que a transposição do grafo não é semelhante ao apresentado na literatura. Portanto, segue o mesmo padrão da matriz, onde os dados são invertidos e a diagonal principal é mantida.

$Complexidadedetempo : O(n^3)$

5.3.6 Remove linhas e colunas

Por fim, como ação extra do programa, foi implementado uma função que remove linhas e colunas alternadamente dado o melhor caminho adquirido entre elas. Pelo fato de utilizar funções, as quais a maior complexidade é dada cúbida, sua análise se torna semelhante:

$Complexidadedetempo : O(n^3)$

5.4 Algoritmo Guloso de Dijkstra

Grafos é uma ótima modelagem para representação de elementos que possuem fatores em comum, que podem ser representados como ligações entre eles. O algoritmo de Dijkstra é um procedimento capaz de encontrar o caminho mínimo de um vértice para todos os vértices do grafo, o que é a solução ideal para o problema analisado, sendo efetivo e eficaz na solução.

6 Análise de Complexidade Geral de Tempo e Espaço das Rotinas

Nesta seção, faremos a análise de complexidade geral de tempo e espaço das rotinas internas do programa, dando ênfase aos principais procedimentos executados.

6.1 Matriz

Devido ao grande número de operações executadas no procedimento por matrizes, a complexidade geral do programa ao se escolher este módulo é dado com complexidade cúbida, pois além de percorrer a matriz w x h, que no seu pior caso é uma matriz quadrada, ela executa funções que percorrem a estrutura mais de uma vez a fim de encontrar caminhos mínimos. Dado isso, conclui-se a sua complexidade de tempo cúbica. Para a complexidade de espaço, ocorre-se dois momentos: No primeiro momento, removerá apenas colunas, alocando apenas uma matriz. No segundo momento, se remove linhas, alocando uma segunda matriz como auxiliar na transposição. Consideramos o pior caso sendo aquele removendo linhas.

$Complexidadedetempo : O(n^3) - Complexidadedeespaço : O(n)$

6.2 Grafos

Analogamente, os grafos também utilizam uma estrutura de matrizes para armazenamento de seus vértices. Além desse quesito quadrático, é necessário passar pela estrutura diversas vezes para checar estados dos pixels, calculos de energia e armazenamento na heap. Simplificações foram feitas a fim de melhorar a complexidade de tempo do procedimento, o qual é cúbico. Já na complexidade de espaço, o algoritmo tem comportamento quadrático, devido a quantidade de alocações feitas da matriz, dos vértices e dos ponteiros para adjacentes.

$Complexidadedetempo : O(n^3) - Complexidadedeespaço : O(n^2)$

7 Análise de Resultados

7.1 Análise Qualitativa

Essa seção será dedicada a uma análise qualitativa das imagens processadas. A análise em questão vale para ambos os resultados, já que possuem semelhanças na conclusão.

7.1.1 Imagens com bons resultados

(A) - Thrill (B) - Praça Sete (C) - Death Valley

Figura 9: Imagens que geraram resultados bons

(A) - Remoção de 300 colunas

(B) - Remoção de 300 colunas

(B) - Remoção de 400 linhas

Figura 10: Resultado do Redimensionamento

Na imagem (A) - Thrill, há um grande espaço de mesma cor ao redor do homem, o que facilita os cálculos do algoritmo. Sendo assim, ao se remover 300 colunas, o resultado obtido é ótimo e sem nenhuma deformação. Na imagem (B) - praça sete, o elemento central se dispõe no meio do céu escuro, deixando um grande contraste. O chão, apesar de cores semelhantes, possui semelhança com os prédios atrás, o que também impede que o obelisco seja deformado. Na figura (C) - Death Valley, é possível perceber que o chão tem cores bastante homogêneas, assim como o céu acima. Ao se remover 400 linhas, o resultado não perde tantos detalhes, mesmo removendo um número alto de caminhos.

(A) - Anders

(B) - CircusPerformer

(C) - Papa

Figura 11: Imagens que geraram resultados ruins

(A) - Remoção de 300 linhas

(B) - Remoção de 300 linhas

(C) - Remoção de 200 colunas

Figura 12: Resultados

7.1.2 Imagens com maus resultados

Já nas imagens da figura 12, houveram muitas distorções em relação a original devido a alguns fatores que o algoritmo falha em cobrir. Na imagem (A) - Anders, cuja está em preto e branco, o cantor fica completamente deformado devido a similiaridade do tom de sua pele com o fundo da imagem. Na figura (B) - CircusPerformer, há uma distorção na parte inferior. Ao se remover 300 linhas, em dado momento da remoção, não existe caminhos possível os quais não alterarão mais o detalhamento da imagem. Já na imagem (C) - Papa, ao contrário da figura (B), há vários caminhos a serem seguidos pelo

algoritmo. Porém, em dado momento, o elemento principal da imagem se asselha ao fundo, causando uma deformação em sua forma.

7.2 Conclusão da Análise Qualitativa

É possível perceber que o algoritmo é muito eficiente para alguns casos em questão, principalmente quando há um plano de fundo sem muito detalhamento. A performance do procedimento é rápida e eficaz, principalmente com programação dinâmica, dando um resultado ótimo na maior parte das vezes. Porém, para alguns casos como demonstrado anteriormente, o algorimo falha devido a similiaridades de energias ou impossibilidade de tirar caminhos sem distorções de detalhamento.

8 Análise Quantitativa

Nesta seção, será abordado uma análise dos resultados de tempo e espaço obtidos empiricamente, dado uma diversidade de entradas. Todas as imagens testes e resultados estão no arquivo compactado do programa.

8.1 Imagens que produziram bons resultados

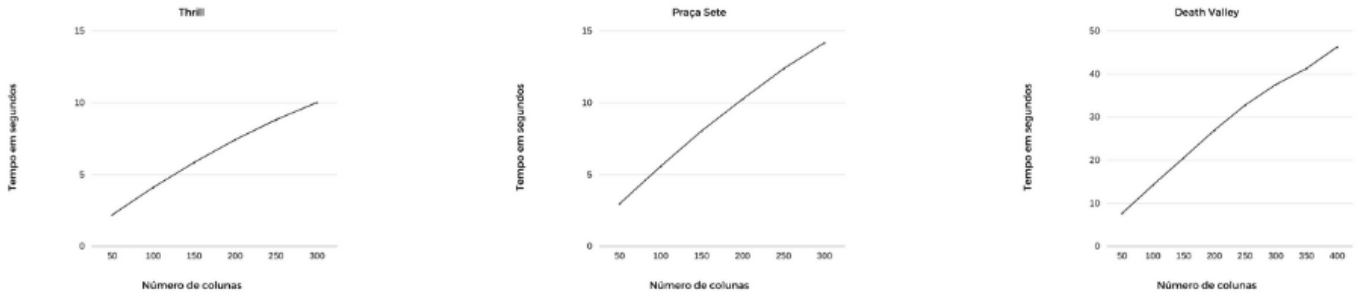

Figura 13: Comportamento de tempo para imagens que produziram melhores resultados

Nos gráficos acima é possível notar o comportamento linear para as imagens que produziram os melhores resultados. Devido a detalhes importantes nas figuras que facilitavam o processamento do algoritmo, foi possível obter resultados em tempos não necessariamente grandes e com uma taxa de crescimento linear.

8.2 Imagens que produziram maus resultados

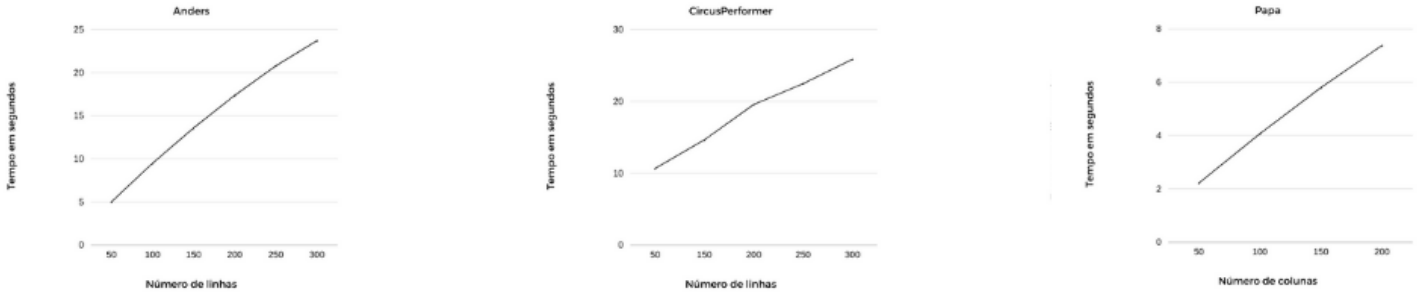

Figura 14: Comportamento de tempo para imagens que produziram piores resultados

Nas imagens que produziram resultados ruins, é possível observar que o tempo de execução se mantém semelhante ao das imagens com resultados bons. Isso indica que a qualidade do resultado não impacta no tempo de execução do algoritmo, o que demonstra que o principal erro ocorrido é no calculo do operador, que não leva em consideração alguns casos particulares das imagens.

8.3 Comportamento Geral

8.3.1 Análise geral de tempo

Como informado na análise de complexidade, o programa possui complexidade assintótica cúbica, o que indica que a medida que a entrada cresce, seu comportamento se torna computacionalmente demorado. Nos gráficos a seguir é demonstrado para os dois diferentes modos de execução como o tempo em segundos aumenta de acordo com o aumento da entrada. Para tal análise, foi considerado imagens com a mesma largura e altura com variação constante a cada teste.

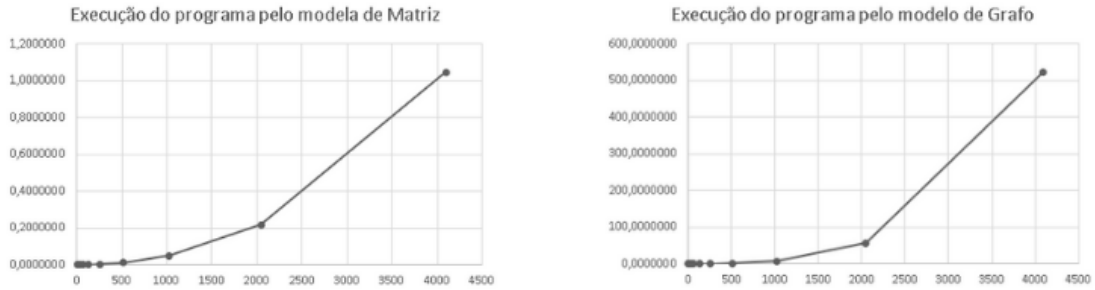

Figura 15: Comportamento de tempo geral do algoritmo

8.4 Análise geral de espaço

Já para o comportamento assintótico do espaço utilizado na execução, os modos se diferenciam drasticamente. Enquanto no modelo de grafos o espaço utilizado aumenta linearmente em relação ao tamanho da entrada, devido a quantidade de adjacêntes alocados, na matriz o comportamento é praticamente constante, já que seu manejamento é feito com coordenadas e acontece apenas uma alocação de um vetor para armazenar a imagem e duas alocações para transpor no caso de remoção de linhas.

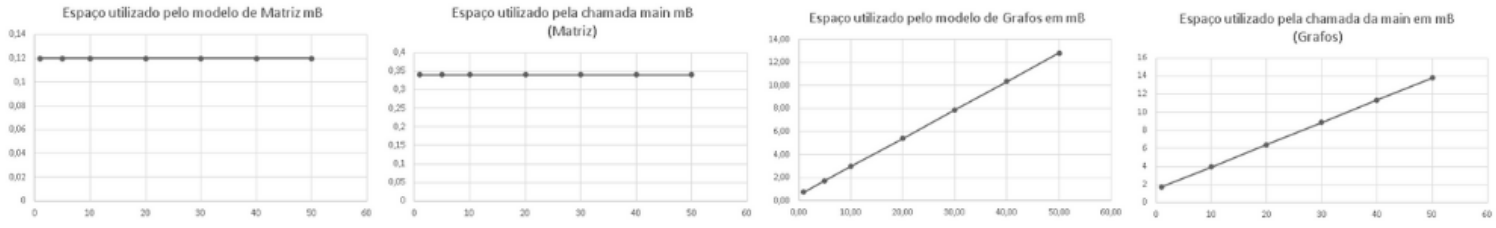

Figura 16: Comportamento de espaço geral do algoritmo

9 Extra

Duas implementações extras foram realizadas no trabalho: Como descrito anteriormente e calculado sua complexidade, foi implementado uma função a qual remove linhas e colunas alternadamente, dando dinamismo ao procedimento. E também é possível alterar o operador no momento da chamada do programa. Existem diversos operadores que podem ser utilizados para o redimensionamento de imagem, todos com qualidades e defeitos distintos. O operador extra utilizado foi o de Scharr.

$$G_x = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \quad G_y = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

Figura 17: Operador de Scharr

10 Conclusão

A conclusão desse trabalho prático permitiu um maior conhecimento sobre algoritmos de redimensionamento de imagem, computação gráfica e uma visão abrangente do futuro mercado de dispositivos, os quais têm se tornado cada vez mais compactos. Além disso, exerceu a compreenção acerca de paradigmas de programação e suas aplicações, as quais podem gerar resultados ótimos dependendo de sua modelagem. Abrangiu o conhecimento sobre grafos e algoritmos conhecidos da literatura capazes de resolverem problemas computacionalmente complexos. Em meio a isso, foi possível concluir o trabalho com êxito e eficiência, trazendo conceitos extraclasse para a composição de soluções.

11 Referências

[1] SILVA, Oscar Paesi da. "Redimensionamento de imagens preservando a proporção dos objetos". Universidade Federal do Rio Grande do Sul. 2009.

Avidan, Shai. Shamir, Ariel. "Seam Carving for Content-aware Image Resizing. Mitsubishi Eletric Research Labs". 2012.

Danahay, Ethan E. "Algorithms for the resizing of binary and grayscale images using a logical transform". Tufh University. 2014.

"The surprising Difficult of Resizing Images on Spritesheets". Construct Online. 2020.

Rocha, Leonardo Chaves Dutra da. "Redimensionamento de Imagens Baseado no Conteúdo". Universidade Federal de São João del Rei. 2020.