MAC 417 – Visão e Processamento de Imagens
EP2

Affonso Amendola – No USP 9301753

Problema 1:

Para cada uma das duas imagens (menino.png e princesa.png) o mesmo processo foi aplicado, somente mudando alguns parametros experimentalmente até conseguir o melhor resultado.

Cada imagem foi aberta usando a função cv2.imread, as imagems foram separadas em canais Azul, Verde e Vermelho usando cv2.split e normalizadas usando uma função própria chamada normalize, que consiste de dividir pixel a pixel pelo maior valor encontrado na imagem.

A imagem normalizada então passou pela função de FFT do OpenCV, a imagem complexa resultante foi separada em partes real e imaginária (O que em retrospecto talvez não seja nescessário, talvez seja possível simplesmente operar em complex128) segue a imagem da transformada do canal Azul da imagem menino.png (Com gamma = 0.15 para melhor visualização)

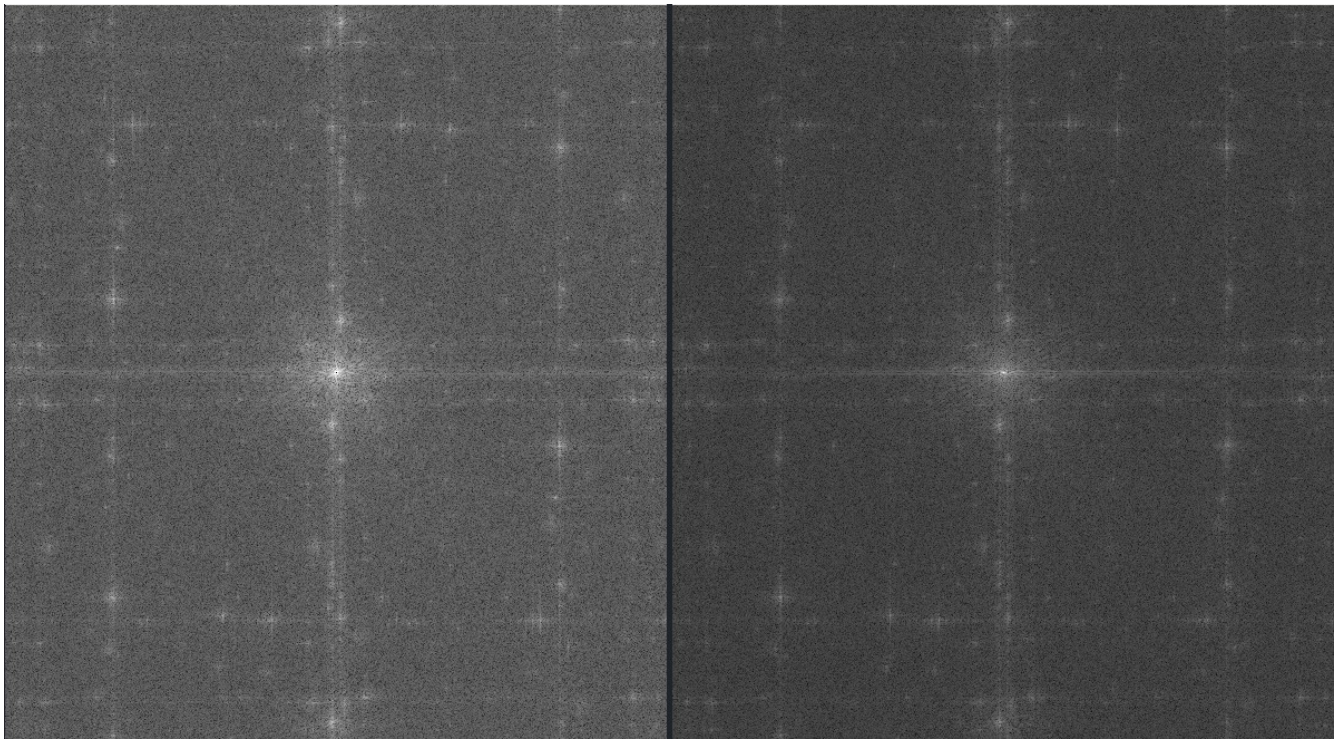


Imagem 1: Esquerda: Parte Imaginária da FFT do canal Azul, Direita: Parte Real da FFT do canal Azul

É possível notar os picos por volta da imagem, esses picos são os responsáveis pelo ruído periódico, então remove-los é crucial para melhorar a qualidade da imagem.

As duas imagems real e imaginaria foram combinadas em uma imagem de espectro (Magnitude) para os proximos passos, mas no final, a máscara de bits será aplicada nas transformadas originais, e não na imagem de espectro.

A imagem de espectro foi normalizada, convertida para uint8 e passada para a função cv2.adaptiveThreshold para se obter as regiões de máximo da imagem, usando as opções ADAPTIVE_THRESH_GAUSSIAN e THRESH_BINARY, com tamanho de bloco igual a 15

Essa função retorna uma mascara de bits nos pontos onde o valor excede um certo threshold, essa mascara de bits for então passada para a função cv2.dilate com elemento estrutural sendo uma MORPH_ELLIPSE de tamanho 7x7, ou seja um circulo de 3.5 pixels de raio, a máscara resultante foi invertida usando a função cv2.bitwise_not (para regiões de máximo ficarem com valor 0 e outras regiões ficarem com valor 255) e um circulo branco (255) de raio 25 foi desenhado no meio da imagem da máscara usando a função cv2.circle, para ignorar as regiões de baixa frequencia

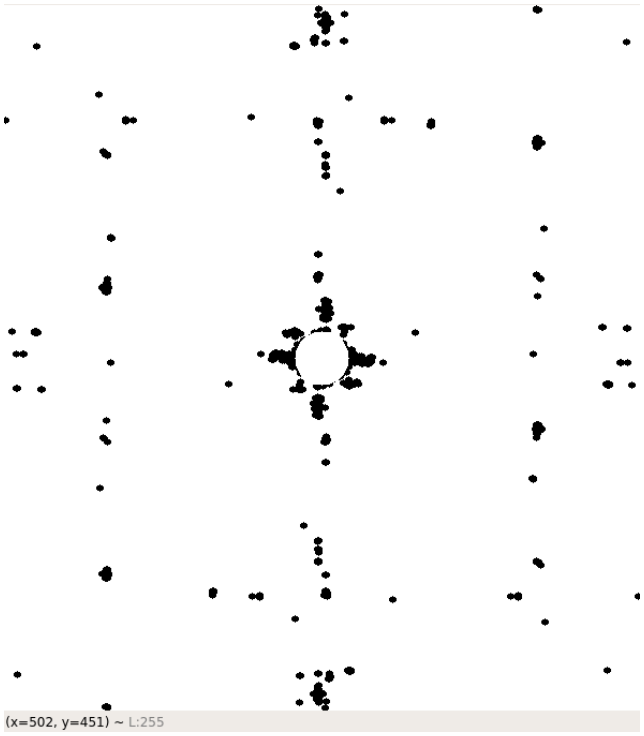


Imagem 2: Máscara de bits para o canal Azul após inversão e adição de circulo central

As Imagens de cada canal, reais e imaginarias foram mascaradas com suas respectivas máscaras usando a função própria alpha_mask_image, que normaliza a imagem da mascara e multiplica pixel a pixel pelo valor da mascara normalizada, ou seja, onde o valor da máscara é 255 o valor da imagem sendo mascarada é mantido, e onde for 0, é substituido por 0.

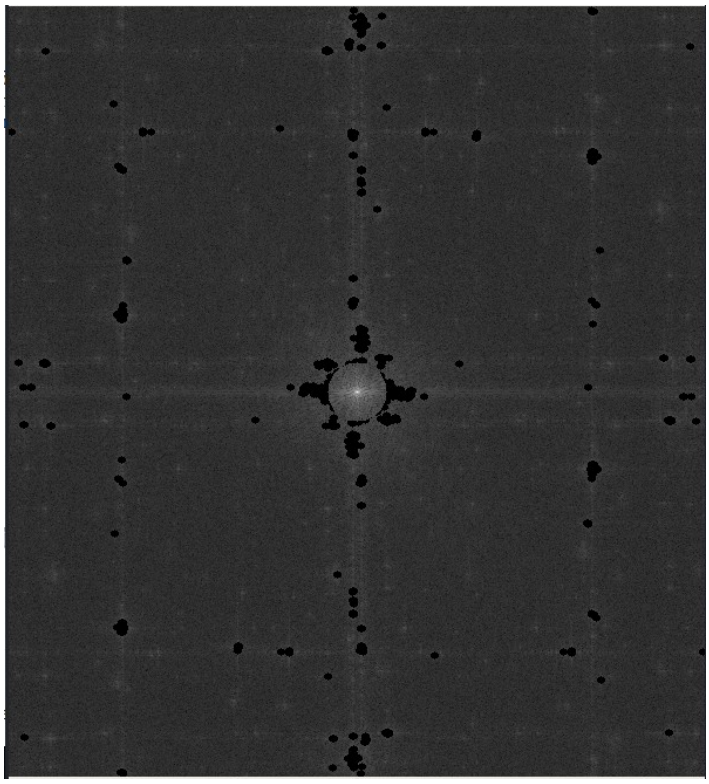


Imagem 3: FFT do canal Azul mascarado com a Bitmask previamente obtida

As imagens de transformadas de todos os canais, agora mascaradas, foram passadas para a função de transformada de Fourier inversa (ifft) e recombinadas para obter a imagem final, agora com muito menos ruído periódico.

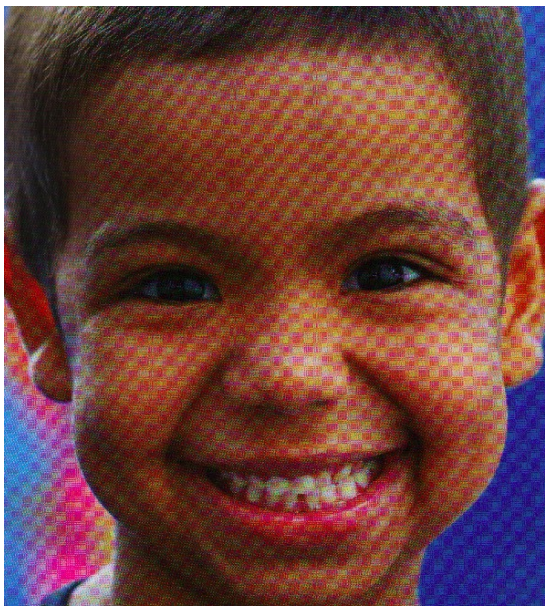


Imagem 5: Menino.png pré-filtro

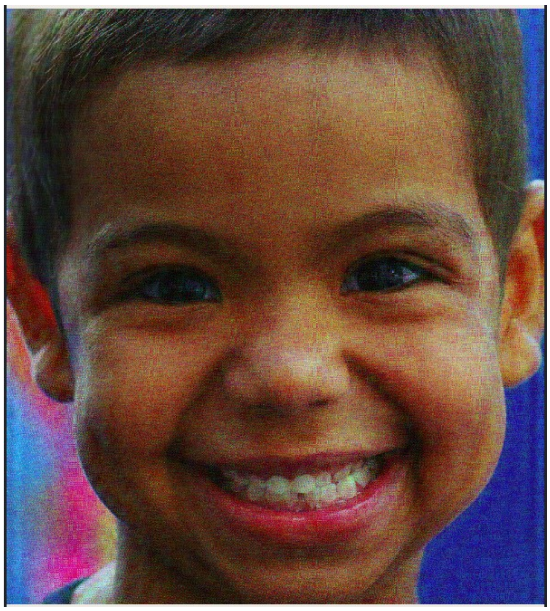


Imagem 4: Menino.png pós-filtro

Exatamente o mesmo procedimento foi empregado para Princesa.png.

Imagem 6: Princesa.png pré-Filtro

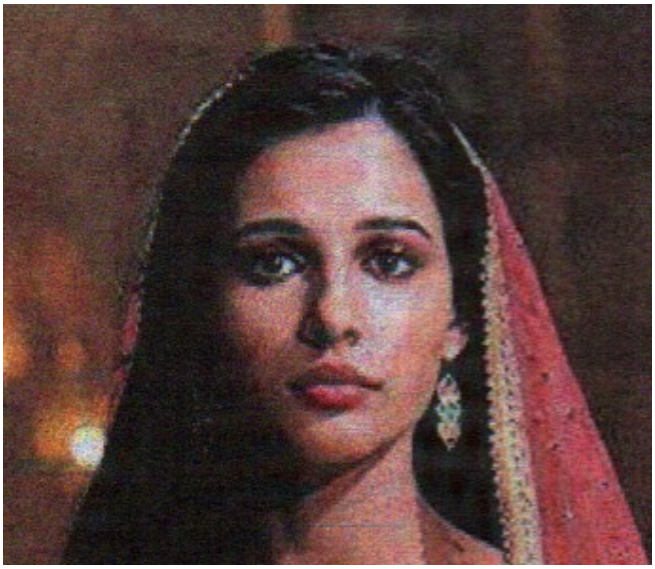


Imagem 7: Princesa.png pós-filtro

Claramente a imagem sem o filtro é melhor, apesar do ruído periodico ter sido removido, os efeitos colaterais da filtragem são fortes demais para justíficar o seu uso nessa imagem, talvez com alguma configuração mais especifica (Diminuir o tamanho da dilatação por exemplo) seja possível obter um resultado melhor.

Problema 2:

Para reparar a imagem da maçã corrompida foi usada a biblioteca siamxt como instruido.

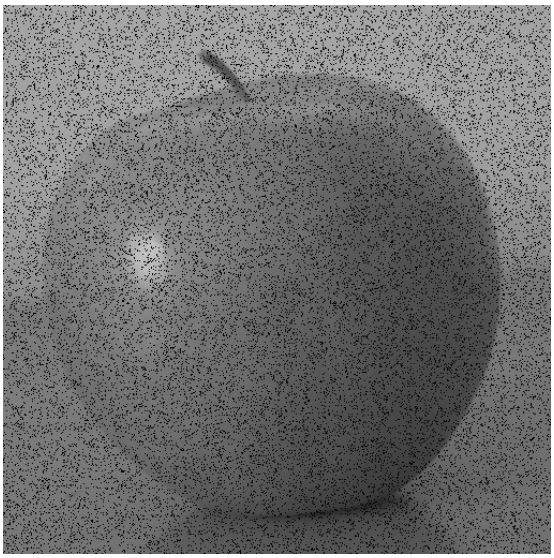


Imagem 8: Maçã original, corrompida.

A imagem foi aberta no código usando OpenCV, cv2.imread (Com cv2.IMREAD_GRAYSCALE pois aparentemente o siamxt não lida muito bem com imagens em outros formatos, causando um crash relacionado a double free de memória)

A imagem carregada foi imediatamente invertida, para considerar os pontos pretos como maximos na árvore,

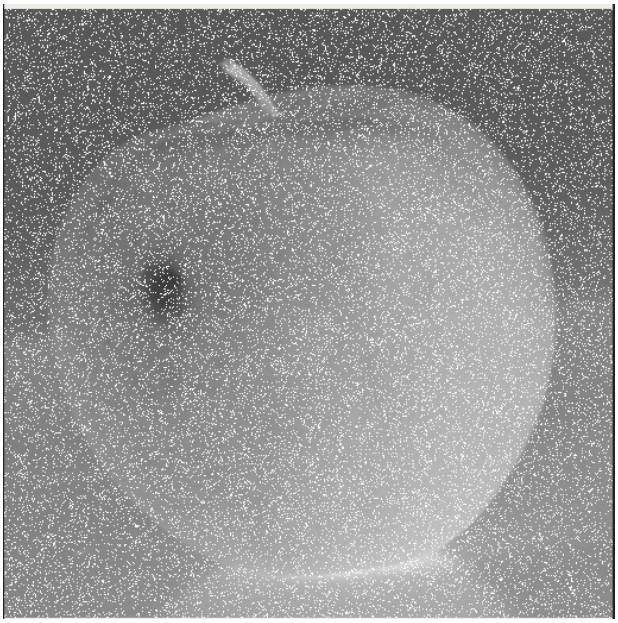


Imagem 9: Maçã invertida.

O elemento estrutural para a construção da árvore foi escolhido como vizinhança-8, e portanto foi inicializado como um array bidimensional de 3x3 cheio de 1s.

A árvore foi então construida usando a função siamxt.MaxTreeAlpha, e logo a seguir o filtro de area open foi aplicado, usando o método areaOpen com parametro 25, já que as areas corrompidas pareciam todas menores que esse valor.

A imagem foi reconstruida usando o método getImage e invertida para obter a imagem final.

Imagem 10: Maçã reconstruida

O sucesso do filtro fala por sí mesmo, os únicos pontos com imperfeições são nas bordas da maçã que pode ser notado um efeito pixelado, mas com a resolução atual, é imperceptivel.

Problema 3:

A imagem foi aberta novamente usando a função imread do OpenCV, com o parametro IMREAD_GRAYSCALE já que a imagem está em tons de cinza, e normalizada usando a função própria normalize, que foi basicamente divide cada pixel da imagem pelo maior valor da imagem.

A função de FFT do numpy foi usada para fazer a transformada de fourier da imagem normalizada.

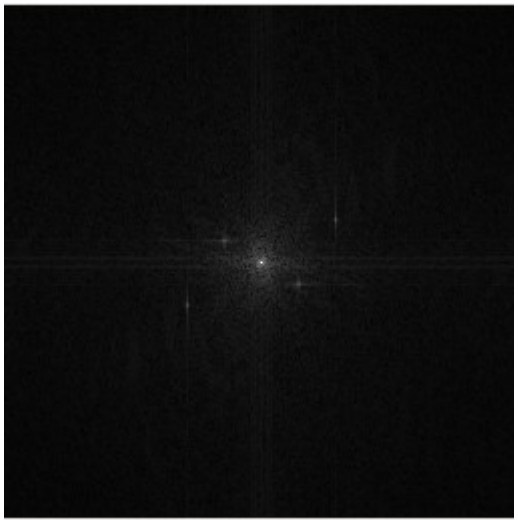


Imagem 11: Transformada de fourier da imagem original.

Usando essa imagem foi construida uma árvore de componentes com vizinhança-8, o filtro de extinção foi configurado usando a função computeExtinctionValues, e aplicado, deixando 3 leaves preservadas.

O filtro de extinção retornou uma imagem sem os picos que causam o ruido periodo, como pode ser visto a seguir:

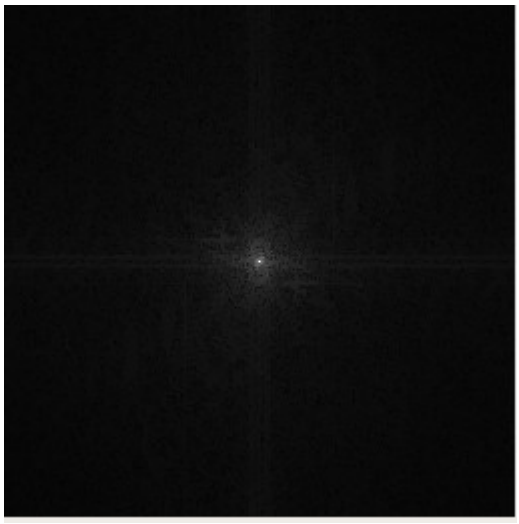


Imagem 12: Filtro de extinção aplicado.

Aplicar a transformada inversa nessa imagem nos retornaria um resultado quase perfeito, infelizmente devido ao tipo de dado nescessário para aplicar o filtro de extinção (Uma imagem Real guardada com uint8s) reconstruir a imagem a partir dela nos daria péssimos resultados, já que muita informação é perdida na conversão de float64 para uint8, então a seguinte solução foi utilizada; A imagem 12 foi subtraida da imagem 11, nos dando uma imagem que contém somente os picos do ruido periodico que queremos remover.

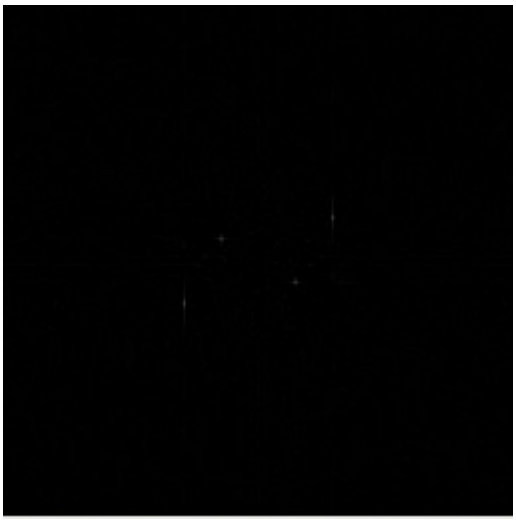


Imagem 13: Picos do ruído periódico

A função adaptiveThreshold foi usada para detectar os pontos de maximo da imagem subtraida, nos dando uma máscara de bits, que foi então dilatada e invertida, exatamente do mesmo metodo que no problema 1, mas sem a remoção da região de baixa frequencia, já que esse problema é resolvido com o uso da arvore de componentes.

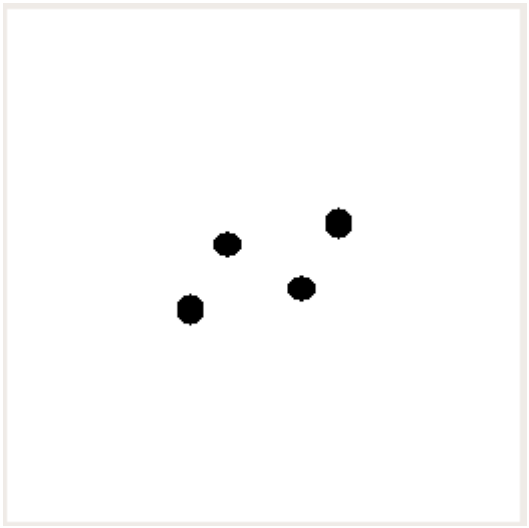


Imagem 14: Máscara de bits

A máscara foi então aplicada na transformada original, usando a função própria alpha_mask_image.

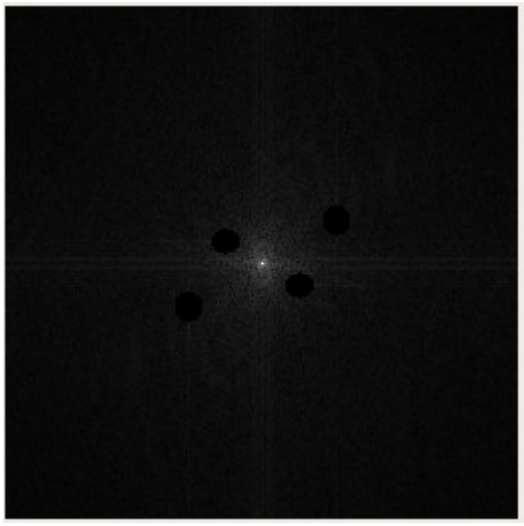


Imagem 15: Transformada mascarada

A transformada, agora mascarada foi então passada para a função de transformada de fourier inversa para finalmente obter o resultado final.

Imagem 17: Palhaco.jpg Original

Imagem 16: Palhaco.jpg Filtrado

Problema 4:

Para a imagem a seguir, onde os numeros representam o valor em escala de cinza de cada pixel:

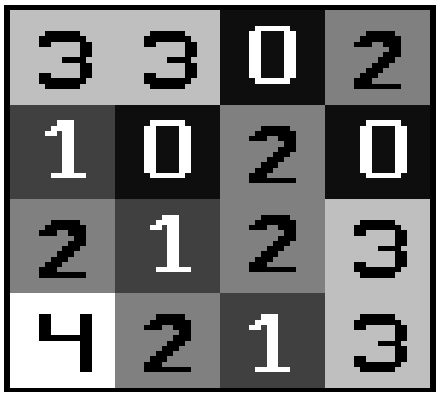
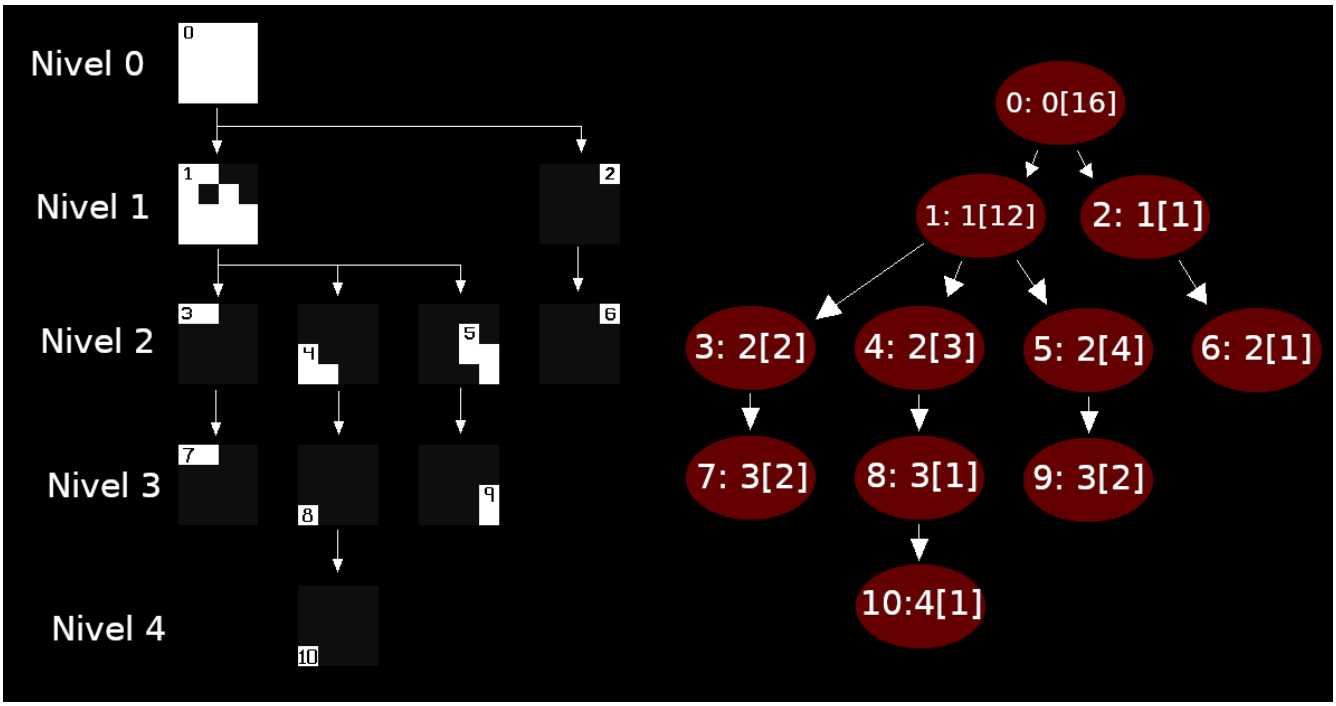
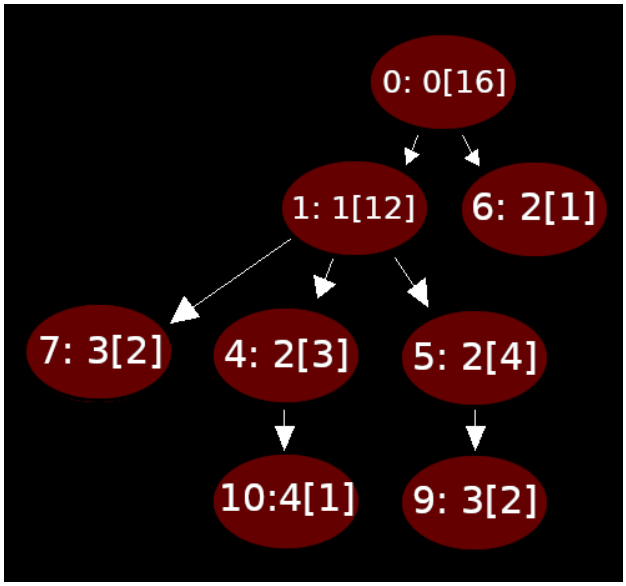


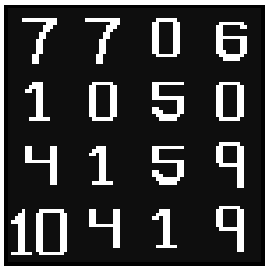
Imagem 18: Imagem original, valores numéricos representam o valor de branco em cada pixel

Usando as definições de árvore de componentes e Max-Tree, foram manualmente construidas as seguintes arvores, levando em consideração vizinhança-4, cada componente foi separado.

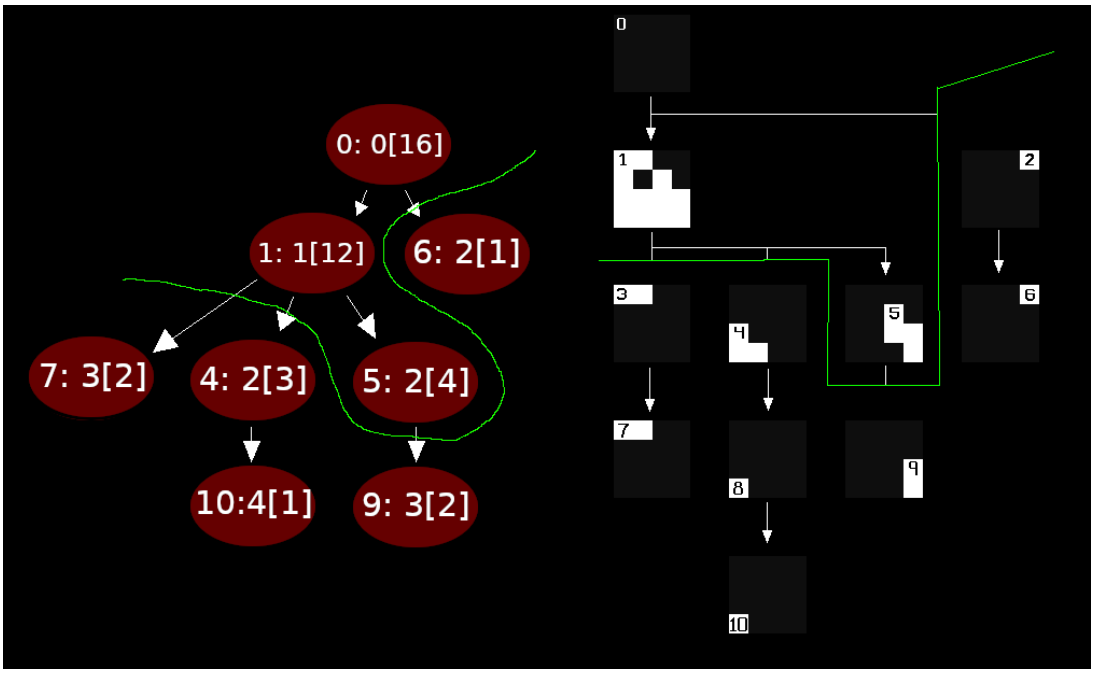


Na esquerda temos uma arvore com todos os componentes e seus formatos, com relações de parentesco representadas pelas setas, na direita temos uma árvore de componentes simplificada, sem a informação de formatos de cada componente, a nomenclatura de cada nó é : IdNode: Nivel[Area], uma Max-Tree pode ser facilmente construida a partir da arvore de componentes simplificada, removendo os nós que representam as mesmas areas, fazendo isso, obtemos a seguinte max-tree, junto com o seu array de Indices de Nós associados, para não perder as informações de formato da imagem:





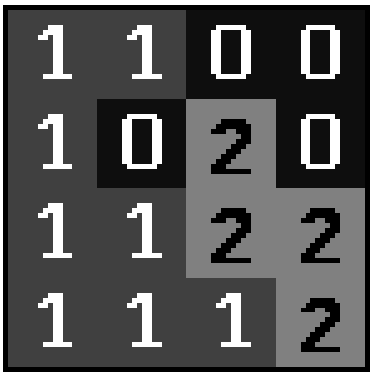
Usando as arvores de componentes, é possivel aplicar um filtro Area Opening removendo leaves da árvore onde a area é menor que um certo valor, para fazer isso (com area menor ou igual a 3) nós só precisamos checar quais leaves da arvore tem areas menores que 3 e remove-las.



Todos os nós abaixo da linha verde tem área menor ou igual a 3, e portanto são removidos, o processo é o mesmo em max-trees e arvores de componentes, a unica diferença surge na hora de reconstruir a imagem, na arvore de componentes da direita só é nescessario combinar os nós levando

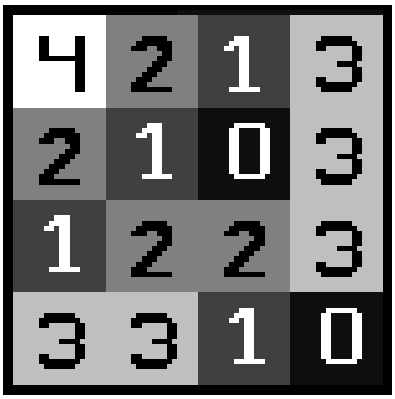
em conta o nivel do patamar em que ele se encontra, enquanto na max-tree é preciso levar em conta um Node Index array, como o da figura 4., para cada pixel no Node Index array é feita uma checagem, se o valor de IDNode corresponde a um nó que se encontra acima da linha verde, é só substituir o valor de nivel daquele nó no pixel correspondente na imagem final, (Ex. O pixel 6 tem valor 5, o nó 5 se encontra acima da linha verde então o valor 2 é colocado no pixel 6 da imagem final) se o IDNode corresponde a um nó abaixo da linha verde (Ou seja, um nó com area <= 3 nesse caso), o pixel correspondente na imagem final ganha o valor de nivel do parente mais proximo acima da linha verde, (Ex. No caso de ID Node 10, o pixel ganha o valor de 1, que é o valor de nivel do nó 1, que é o parente de 10 acima da linha verde.

Seguindo esse algorítmo, a imagem reconstruida após a aplicação do filtro foi a seguinte:

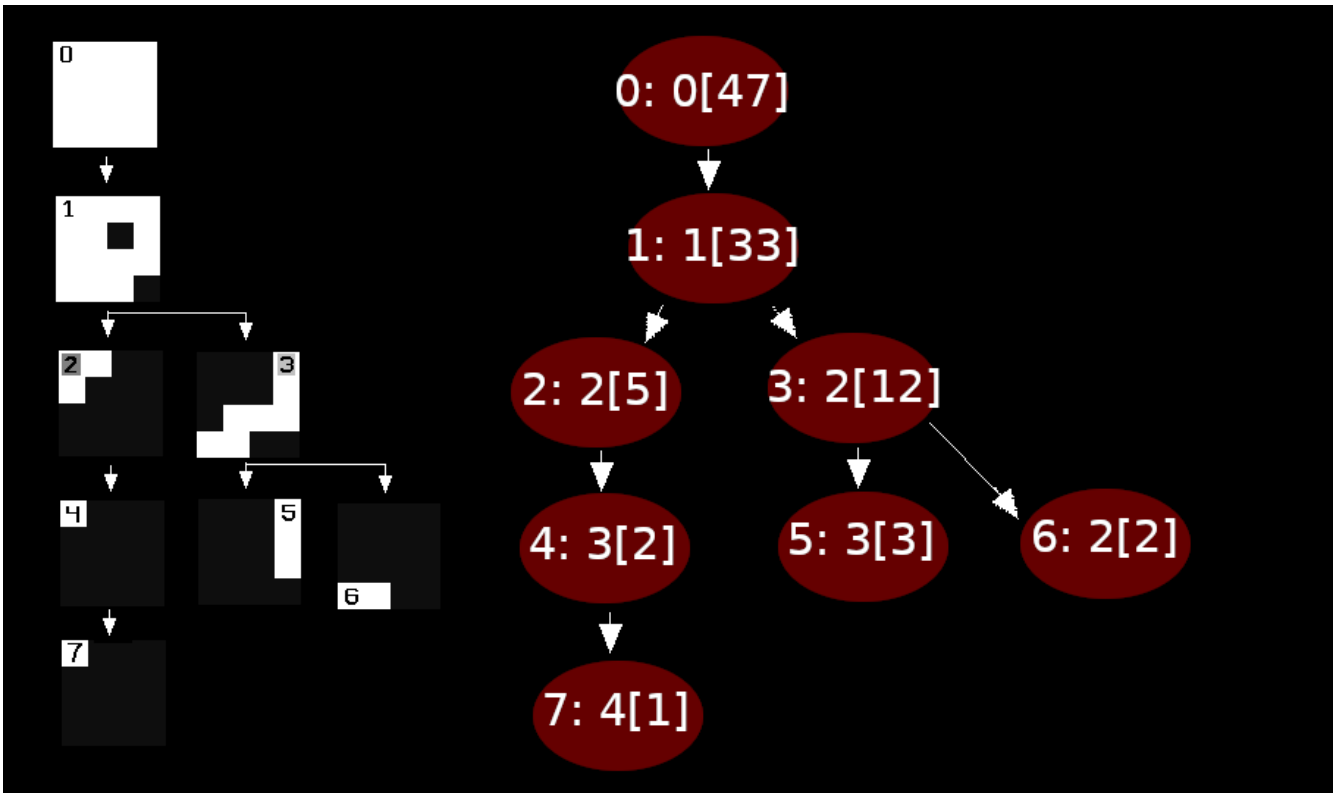


Problema 5:

O mesmo processo de criação de árvores de componentes e Max-tree do problema 4. foi usado, mas dessa vez para esta imagem

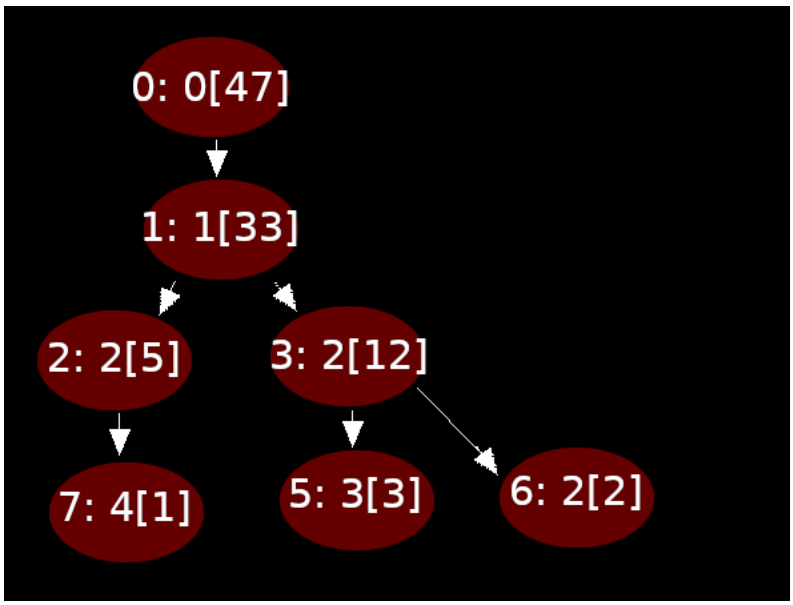


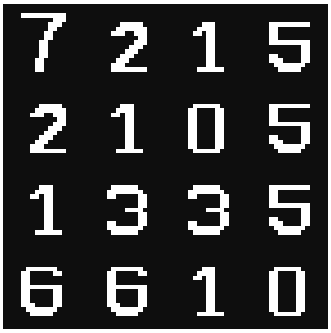
Levando em consideração vizinhança-4 as seguintes árvores foram construidas:

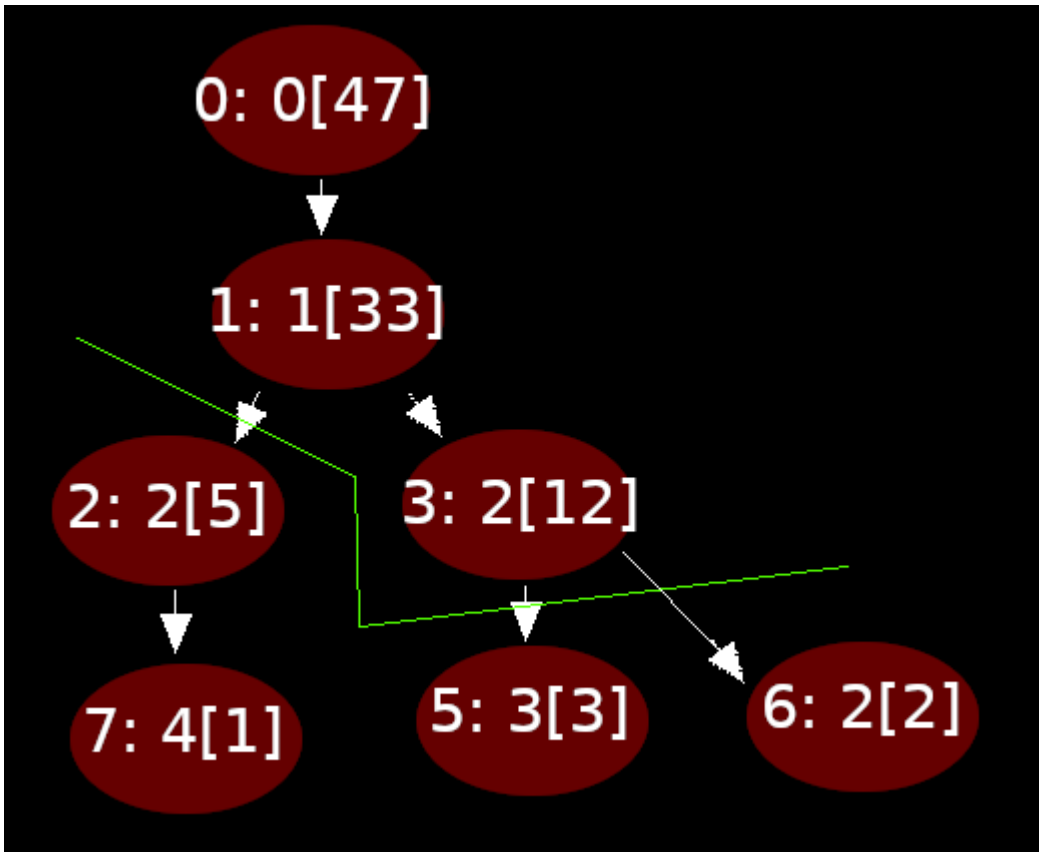


Na esquerda temos uma arvore com todos os componentes e seus formatos, com relações de parentesco representadas pelas setas, na direita temos uma árvore de componentes simplificada, sem a informação de formatos de cada componente, a nomenclatura de cada nó é : IdNode: Nivel[Volume],

A max-tree foi construida removendo os nós que representam a mesma area na imagem, que nesse caso eram só o nó 7 e o nó 4, o nó maior foi deixado, e criando um Node Index Array que relaciona cada pixel da imagem a um nó, para podermos reconstruir a imagem mais tarde.







Remover os domos com volume menor ou igual a 8 é equivalente a remover os nós abaixo da linha verde na imagem acima. Para fazer isso, ao reconstruir a imagem usando o Node Index Array, se encontrarmos um valor de NodeID que está abaixo da linha, viajamos a arvore para cima até encontrar um que não esteja, e o valor de branco daquele pixel vira o valor do nó encontrado, ao fazer isso obtemos a seguinte imagem