
UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
CURSO DE ENGENHARIA DE COMPUTAÇÃO

GUSTAVO GINO SCOTTON

# SEGMENTAÇÃO DE IMAGENS COM CORES

Araranguá
2018

GUSTAVO GINO SCOTTON

# SEGMENTAÇÃO DE IMAGENS COM CORES

Trabalho realizado para a disciplina de Tópicos especiais III da Universidade Federal de Santa Catarina, no curso de Engenharia de Computação para obtenção de uma das notas referentes aos trabalhos.

Professor: Dr. Antônio Carlos Sobieranski

Araranguá

2018

SUMÁRIO

## 1. INTRODUÇÃO

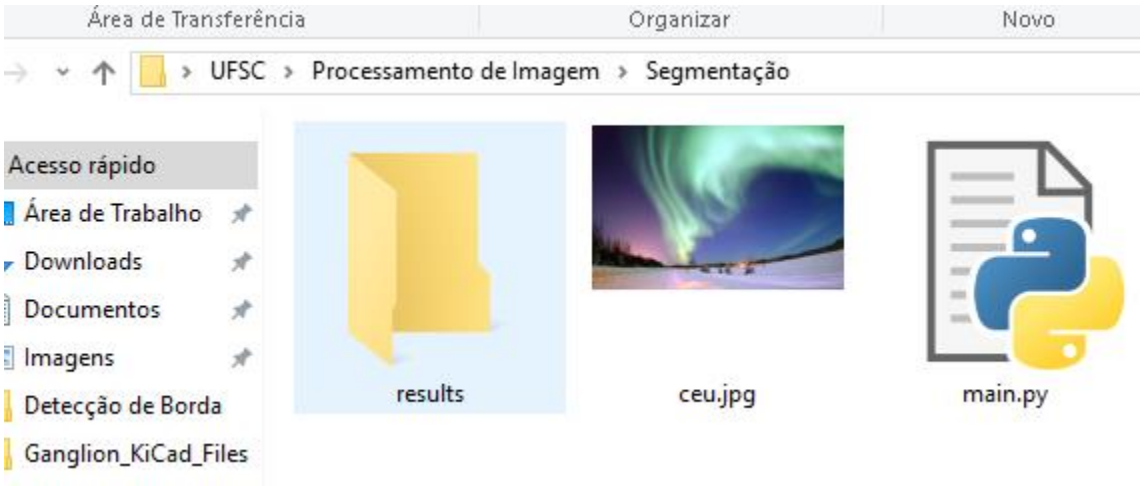
Na visão computacional, a segmentação de imagens é o processo de particionar uma imagem digital em múltiplos segmentos (conjuntos de pixels, também conhecidos como super-pixels). O objetivo da segmentação é simplificar e / ou alterar a representação de uma imagem em algo mais significativo e mais fácil de analisar. A segmentação de imagens é normalmente usada para localizar objetos e limites (linhas, curvas, etc.) em imagens. Mais precisamente, a segmentação de imagens é o processo de atribuir um rótulo a cada pixel de uma imagem, de modo que os pixels com o mesmo rótulo compartilhem determinadas características.

O método que utilizarei para realização deste trabalho será o k-means clustering, embora exista outros métodos como por exemplo, limiar, métodos de crescimento da região (demonstrado em aula), métodos baseados em equações diferenciais parciais, dentre outros.

Antes de começarmos, uma breve introdução sobre o método por mim escolhido para esta realização. k-Means Clustering é um método de particionamento que particiona dados em k-clusters mutuamente exclusivos e retorna o índice do cluster ao qual ele atribuiu cada observação. Diferentemente do clustering hierárquico, o k-means clustering opera em observações reais (em vez do conjunto maior de medidas de desigualdades) e cria um único nível de clusters. As distinções significam que o clustering k-means é geralmente mais adequado do que o agrupamento hierárquico para grandes quantidades de dados, neste caso, grande quantidade de pixels, já que se trata de uma imagem.

## 2. INICIALIZANDO O PROGRAMA

Inicialmente, temos em nossa pasta de arquivos os itens “main.py” e “ceu.jpg”, a pasta “results”, armazenará as imagens de saída, caso seja optado por salva-las.




Executando então o arquivo main.py no prompt de comando, temos o seguinte menu:

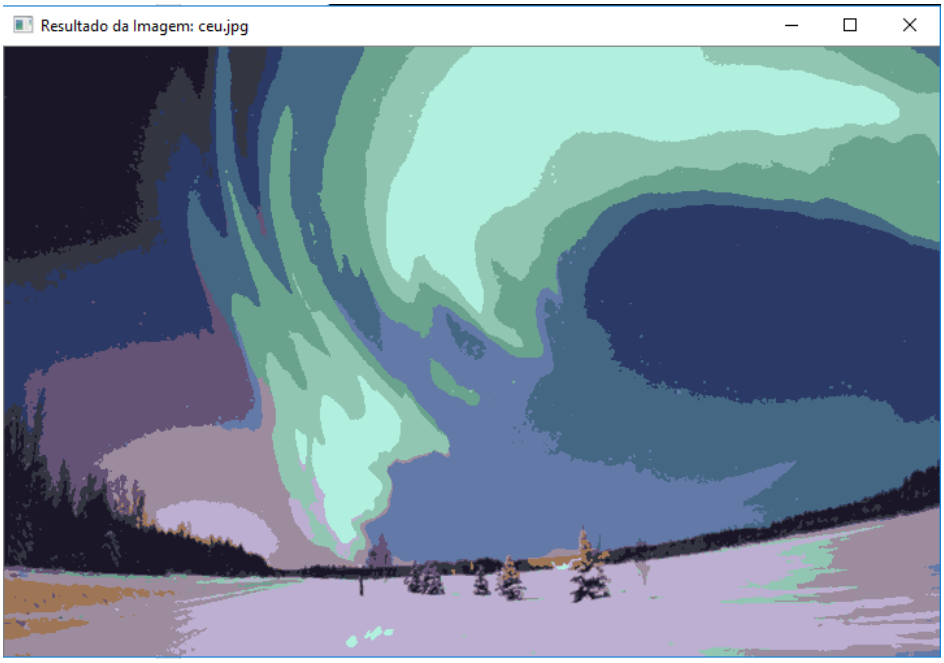

Aqui, digitamos o nome da imagem que gostaríamos de realizar o filtro, portanto, “ceu.jpg”.

```
Digite o nome e a extensão da imagem que deseja abrir para editar.
Exemplo: imagem.jpg

Nome da imagem: ceu.jpg

Quantidade de cores que deseja como saída:
```

Com isso, vamos a opção de selecionar a quantidade de cores que desejamos como saída. Neste primeiro exemplo, usarei 12 cores. Posteriormente, demonstrarei a diferença entre cores com diversos testes variados. Com o valor digitado, devemos aguardar o processamento dos dados, ao finalizar teremos o seguinte:

```
Digite o nome e a extensão da imagem que deseja abrir para editar.
Exemplo: imagem.jpg

Nome da imagem: ceu.jpg

Quantidade de cores que deseja como saída: 12

> Abrindo imagem...
> Aplicando filtro...
> Convertento para numérico...
> Agrupando cores...
> Convertendo para imagem...
> Resultado finalizado!

Deseja salvar a imagem? [Y] [N] : y
Como você gostaria de chamar esta imagem?
Exemplo: Editada.jpg

Nome da imagem: ceu-new.jpg
Imagem salva com sucesso na pasta 'results'!


Deseja visualizar a imagem agora? [Y] [N] :
```

Se optarmos por salvar a imagem, como mencionado, devemos digitar o nome que desejamos salvar esta nova imagem gerada, sua saída irá para a pasta “results”, caso optarmos por não salvar, teremos a opção de visualizar a imagem, e o salvamento é ignorado. A saída será:



## 3. ALGORITMO

Aqui podemos observar o funcionamento do algoritmo, utilizando a função kmeans() do OpenCV, que será explicada logo a baixo.

```
26  read= input("       Nome da imagem: ")
27  img = cv2.imread(read)
28  print("")
29  K= int(input("      Quantidade de cores que deseja como saída: "))
30  print("")
31  print(" > Abrindo imagem...")
32  time.sleep(1.5)
33
34  #Aplica filtro gaussiano pra suavizar a imagem e detectar melhor
35  img_gaussian = cv2.GaussianBlur(img,(3,3),0)
36  img_float = img_gaussian.reshape((-1,3))
37  print("     > Aplicando filtro...")
38  time.sleep(1.5)
39
40  # converte pra numpy float32 para realizar as operações
41  img_float = np.float32(img_float)
42  print("         > Convertento para numérico...")
43  time.sleep(1.5)
44
45  # define critérios e aplica a função cv2.kmeans()
46  criterio = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
47  ret,omega,img_centro = cv2.kmeans(img_float, K, None, criterio, 10, cv2.KMEANS_RANDOM_CENTERS)
48  print("             > Agrupando cores...")
49  time.sleep(1.5)
50
51  # Agora converte de volta para uint8, transformando em formato imagem
52  img_centro = np.uint8(img_centro)
53  print("                 > Convertendo para imagem...")
54  time.sleep(1.5)
55  resultado = img_centro[omega.flatten()]
56  resultado = resultado.reshape((img.shape))
57
58  print("                     > Resultado finalizado!")
59  time.sleep(1)
60  print("")
61
62  #Oferece opção para salvar a imagem na pasta "results"
63  resposta=input("        Deseja salvar a imagem? [Y] [N] : ")
64  if (resposta == "Y" or resposta=="y"):
65      print("     Como você gostaria de chamar esta imagem?")
66      print("     Exemplo: Editada.jpg")
67      print("")
68      write = input("Nome da imagem: ")
69      cv2.imwrite("results/"+write, resultado)
```

A função kmeans implementa um algoritmo k-means que localiza os centros de clusters cluster_count e agrupa as amostras de entrada em torno dos clusters. Como saída, os ‘omegas’ contém um índice de cluster baseado em 0 para as amostras armazenadas no índice da matriz de amostras.

**PARÂMETROS DE ENTRADA:**

**Imagem:** Deve ser do tipo de dados np.float32 , e cada recurso deve ser colocado em uma única coluna.

**nclusters (K):** Número de clusters desejados no final (cores)

**critérios:** é o critério de terminação de iteração. Quando esse critério é satisfeito, a iteração do algoritmo é interrompida.

**cv2.TERM_CRITERIA_EPS:** interrompe a iteração do algoritmo se a precisão especificada, épsilon (Exigência necessária), for atingida.

**cv2.TERM_CRITERIA_MAX_ITER**: interrompe o algoritmo após o número especificado de iterações, max_iter (número máximo de iterações).

**Tentativas:** Especifica o número de vezes que o algoritmo é executado usando diferentes rótulos iniciais. O algoritmo retorna os rótulos que produzem a melhor compactação. Essa compactação é retornada como saída.

**Flags:** flag é usado para especificar como os centros iniciais são obtidos. Normalmente, dois sinalizadores são usados para isso: cv2.KMEANS_PP_CENTERS e cv2.KMEANS_RANDOM_CENTERS.

**PARÂMETROS DE SAÍDA:**

**Compacidade:** É a soma da distância ao quadrado de cada ponto para seus centros correspondentes.

**Omega:** Este é o array de rótulos.

**Centros:** Esta é a matriz de centros de clusters.

**FUNÇÃO KMEANS:**

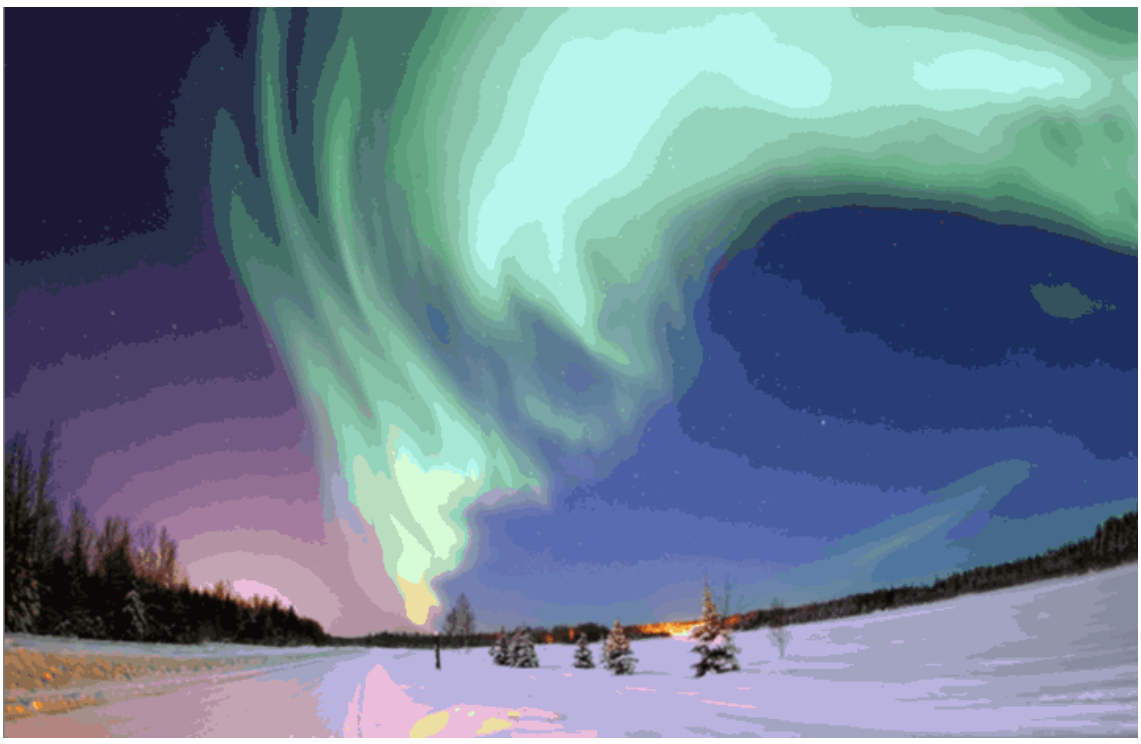
*cv.KMeans2(samples, nclusters, labels, termcrit, attempts=1, flags=0, centers=None)*

## 4. RESULTADOS

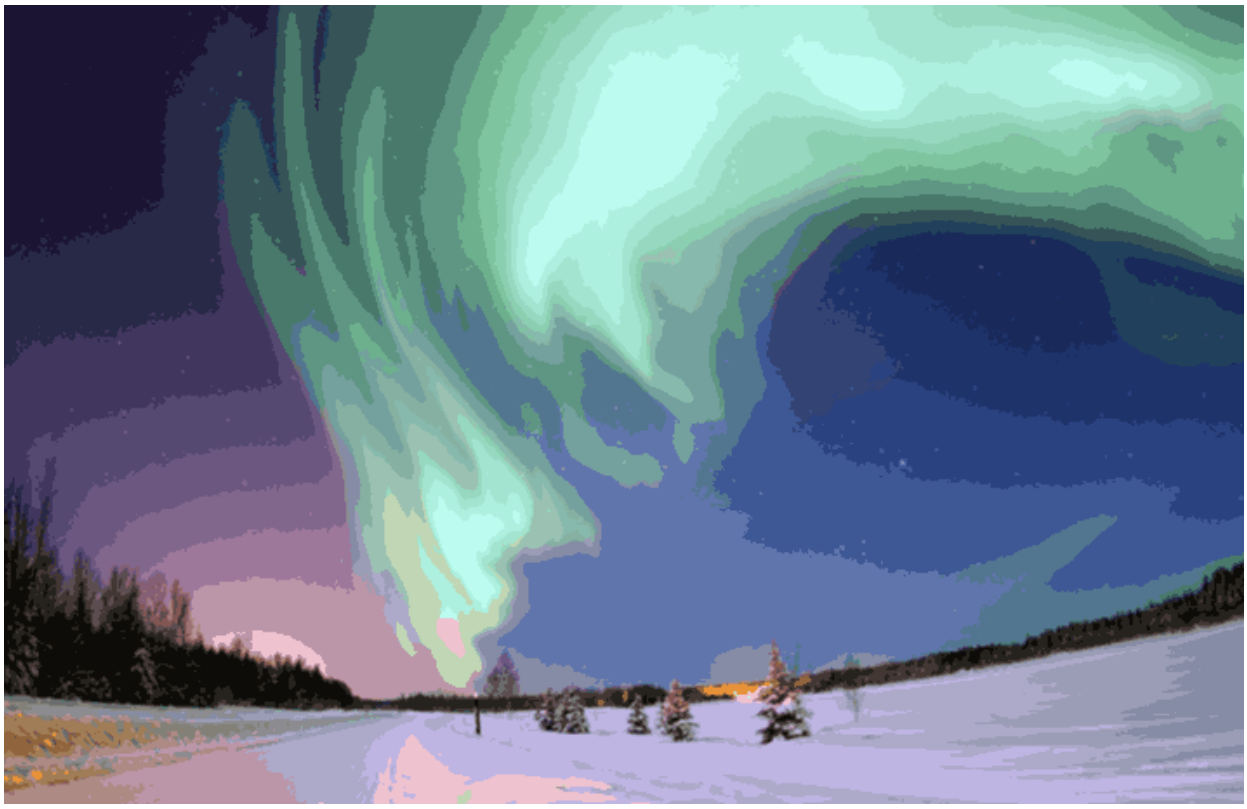
Agora, vejamos os resultados da mesma imagem, porém mudando a quantidade de clusters.

Imagem original

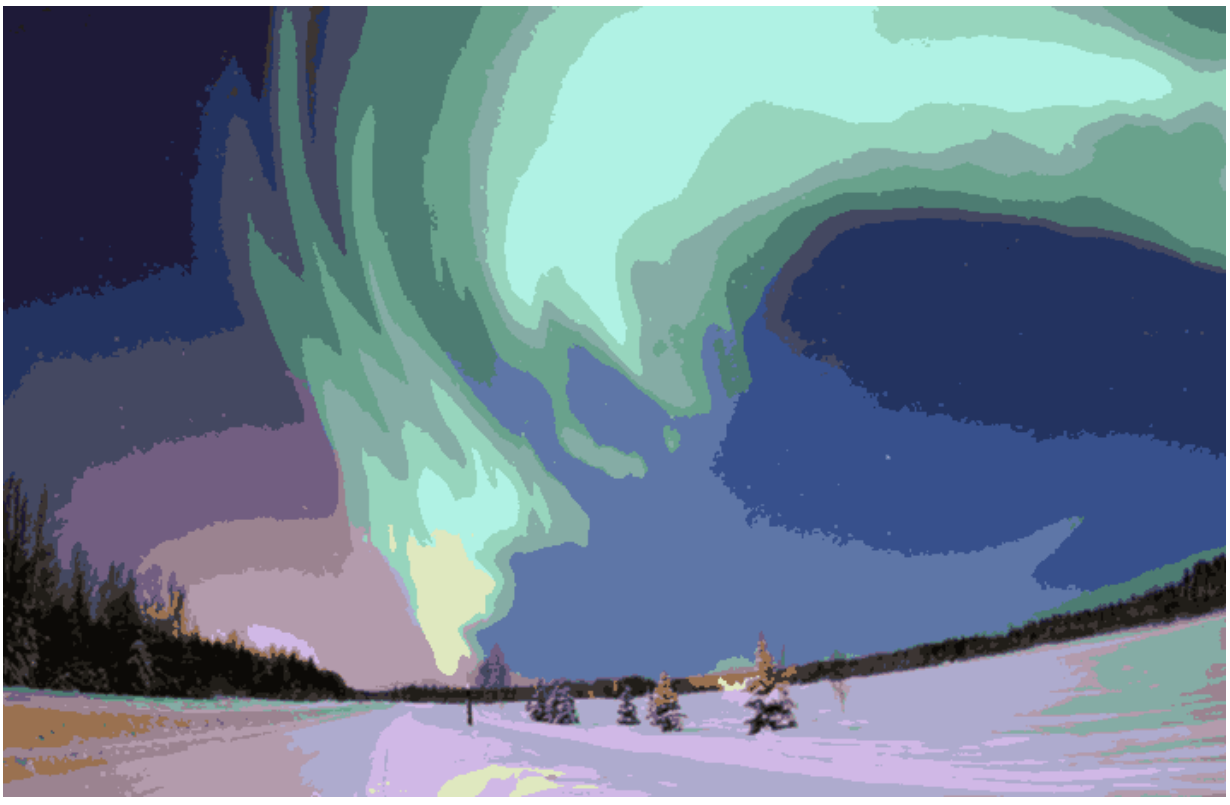
Clusters = 100



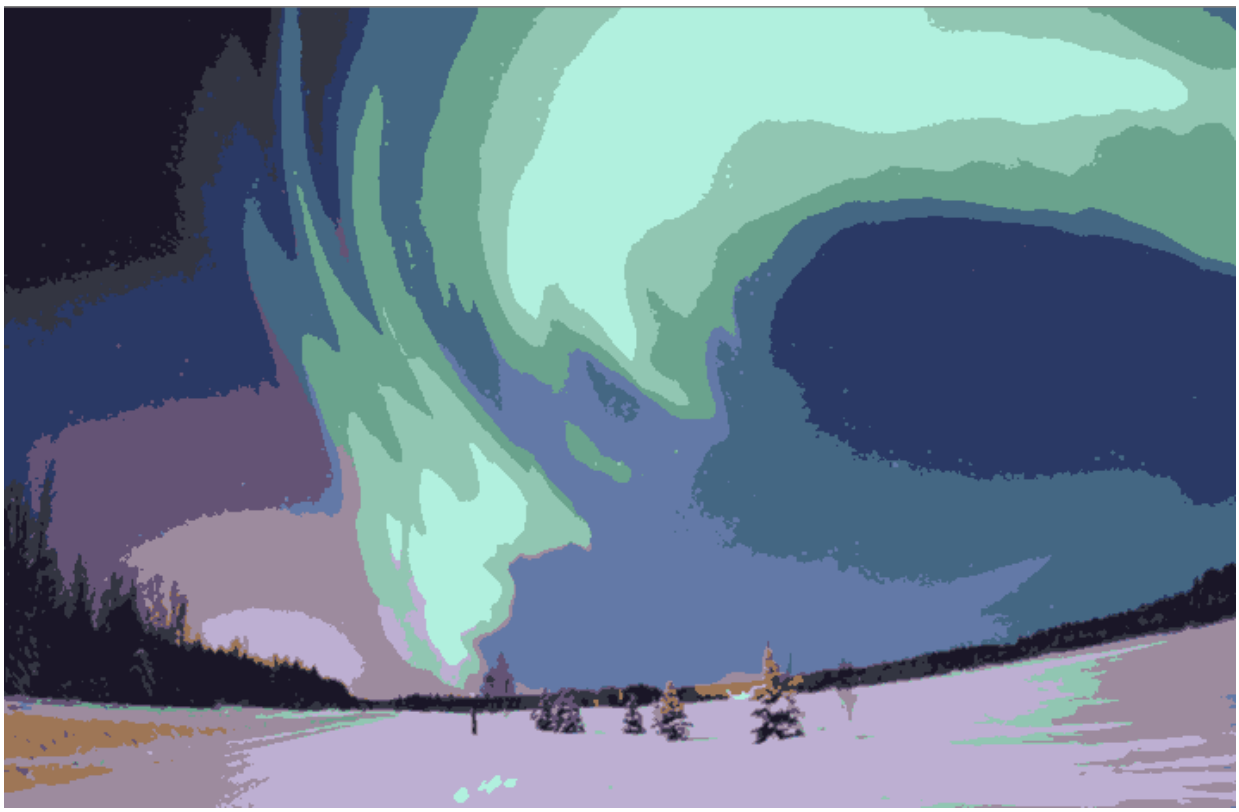
Clusters = 50



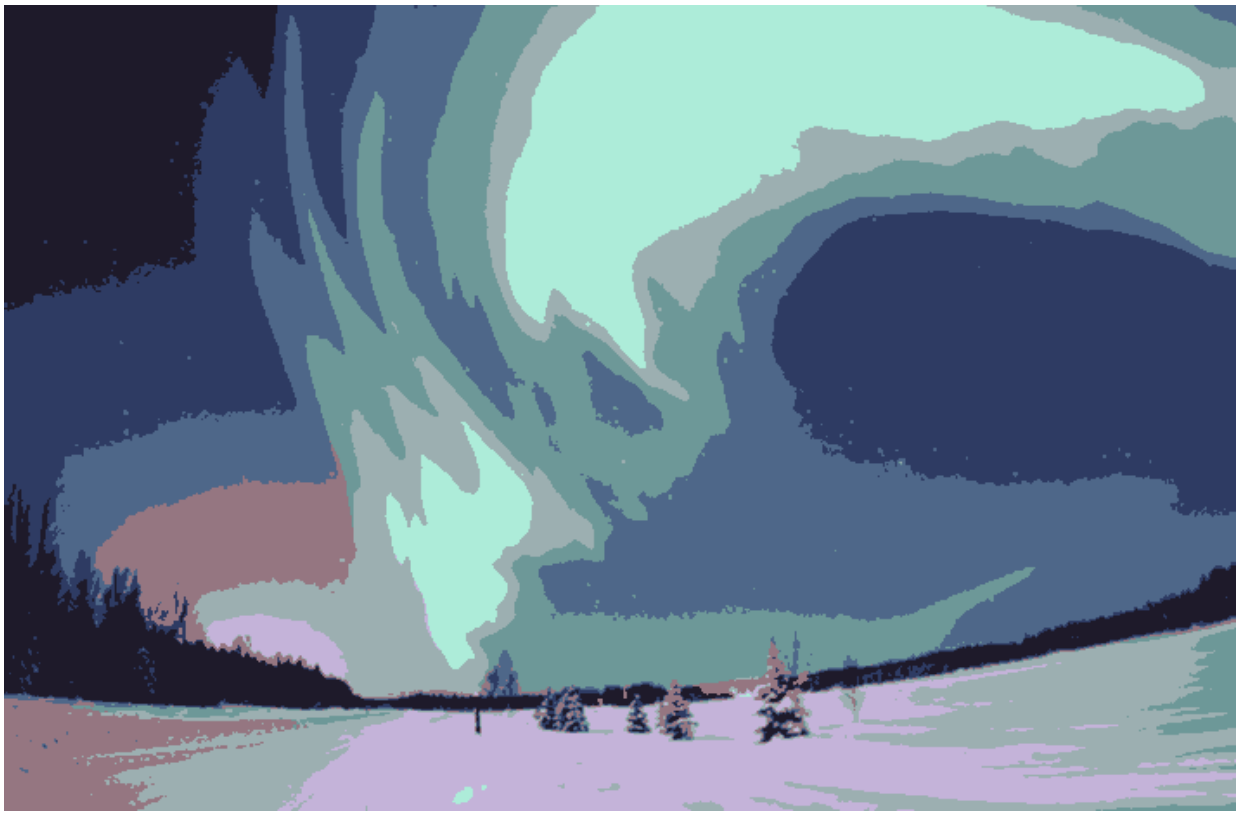
Clusters = 20



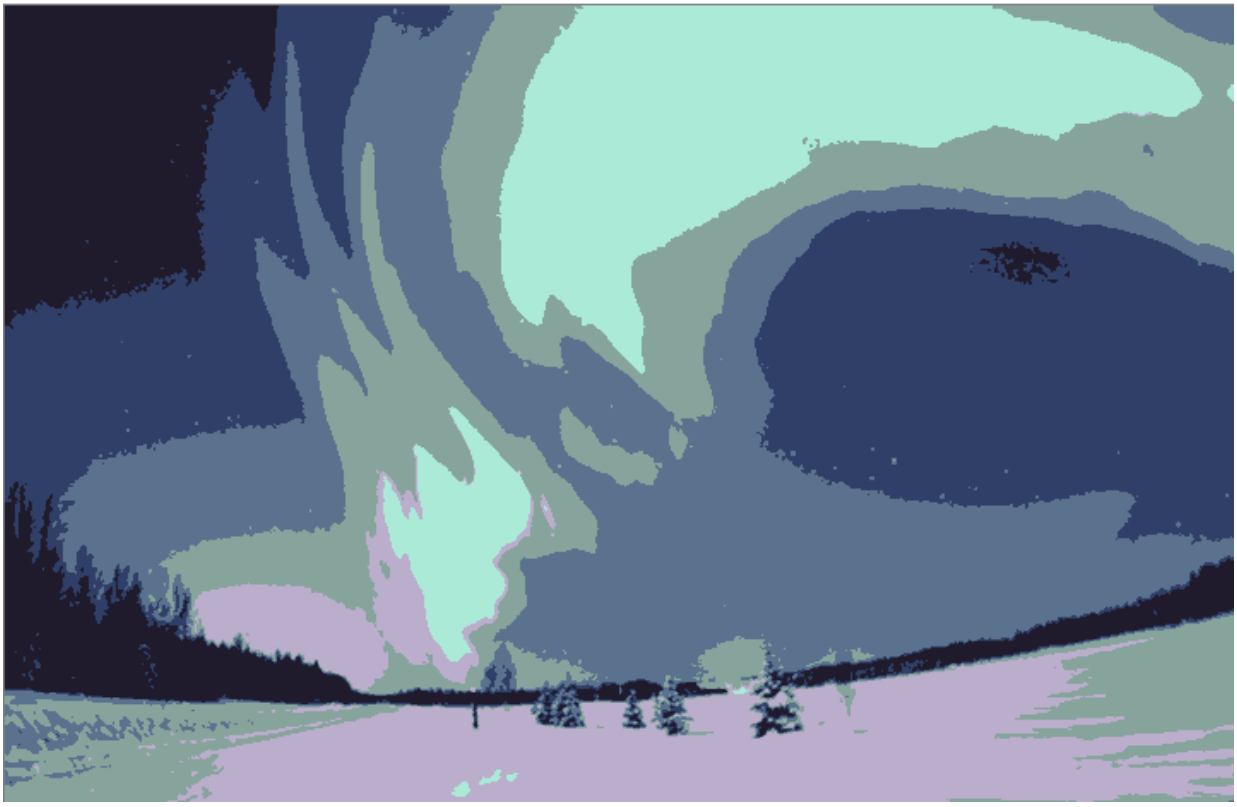
Clusters = 12



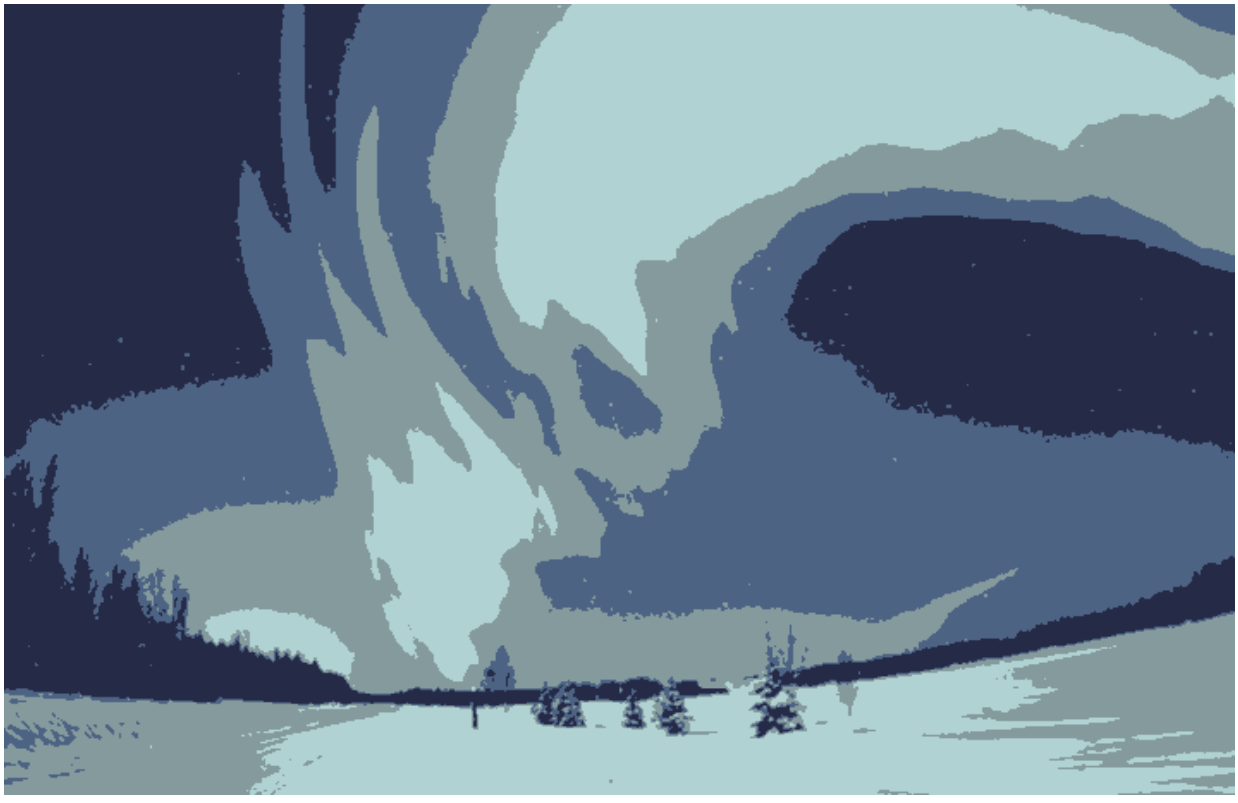
Clusters = 8



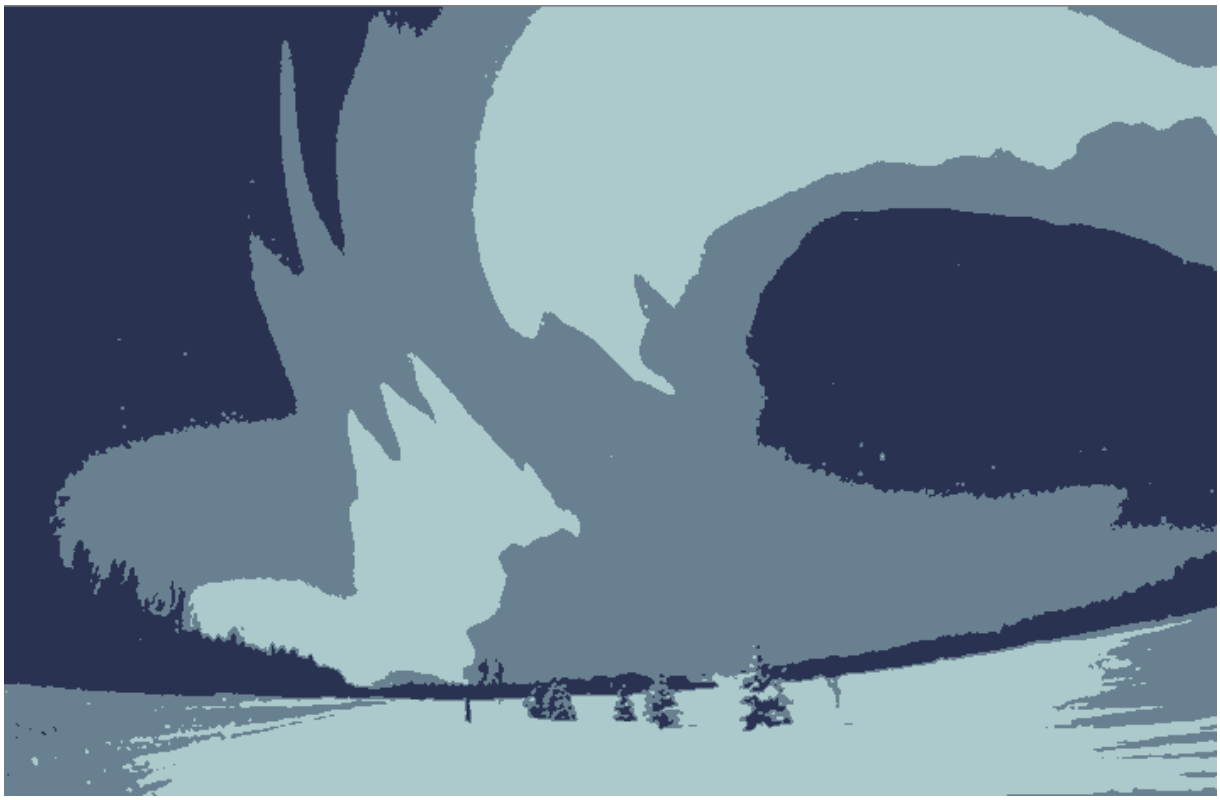
Clusters = 6



Clusters = 4



Clusters = 3



Clusters = 2

## 5. CONCLUSÃO

Podemos observar que quanto maior o cluster mais próximo a imagem original fica, isso se dá pelo fato de abranger uma gama de cores maior, ou seja, deixando a imagem com mais separações de objetos baseado na média de cor. Já quando reduzimos o cluster, vemos os agrupamentos de cores mais semelhantes, até que se torne algo único, como por exemplo se utilizarmos o cluster = 1, teríamos a seguinte imagem:

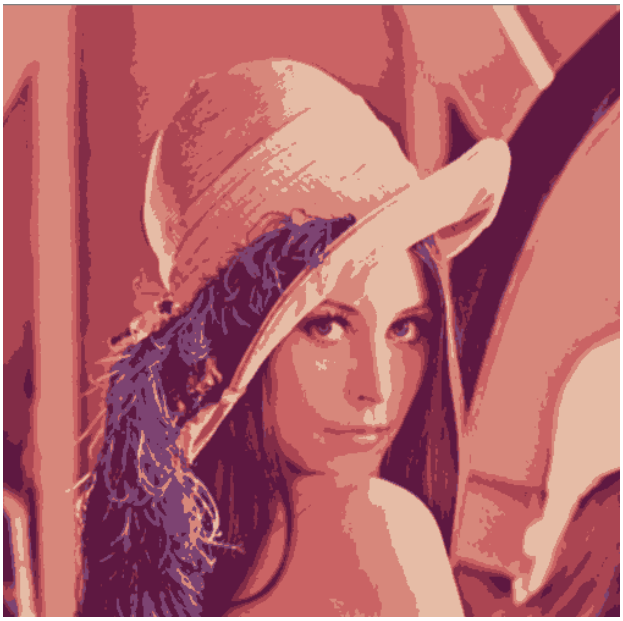
Afinal, estamos dizendo que queremos apenas uma cor na saída, portanto é de se esperar que seja apenas um fundo de cor sólida.

Contrapartida, utilizando o Clusters = 5000. Vale lembrar que quanto maior o número de clusters, maior será o tempo de processamento (demorou aproximadamente 5 minutos). Portanto, temos a imagem mais próxima a original, pode ser conferida abaixo. (A imagem possui 5mil objetos/cores diferentes)

## 6. APLICAÇÃO EM OUTRAS IMAGENS

| ORIGINAL | SEGMENTADA |
|---|---|

*Imagem Original*



*Segmentação com K = 7*

*Imagem Original*

*Segmentação com K = 8*

*Imagem Original*

*Segmentação com K = 6*

*Imagem Original*

*Segmentação com K = 12*

*Imagem Original*

*Segmentação com K = 14*

*Imagem Original*

*Segmentação com K = 20*

*Imagem Original*

*Segmentação com K = 10*

*Imagem Original*

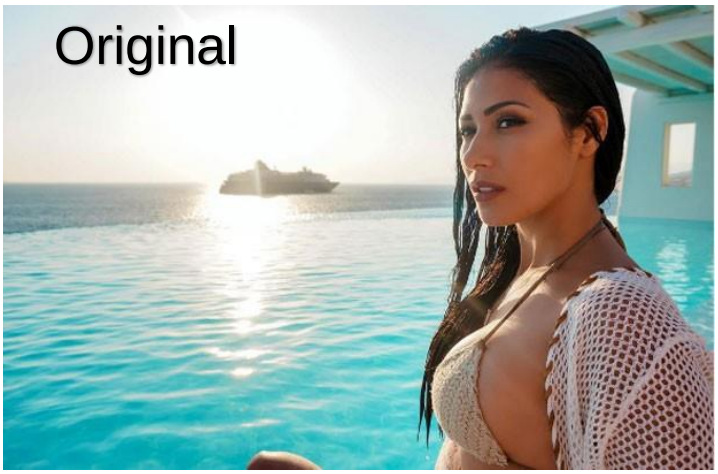
*Segmentação com K = 4*


*Imagem Original*


*Segmentação com K = 6*

*Imagem Original*

*Segmentação com K = 3*


Original


K=30


K=20

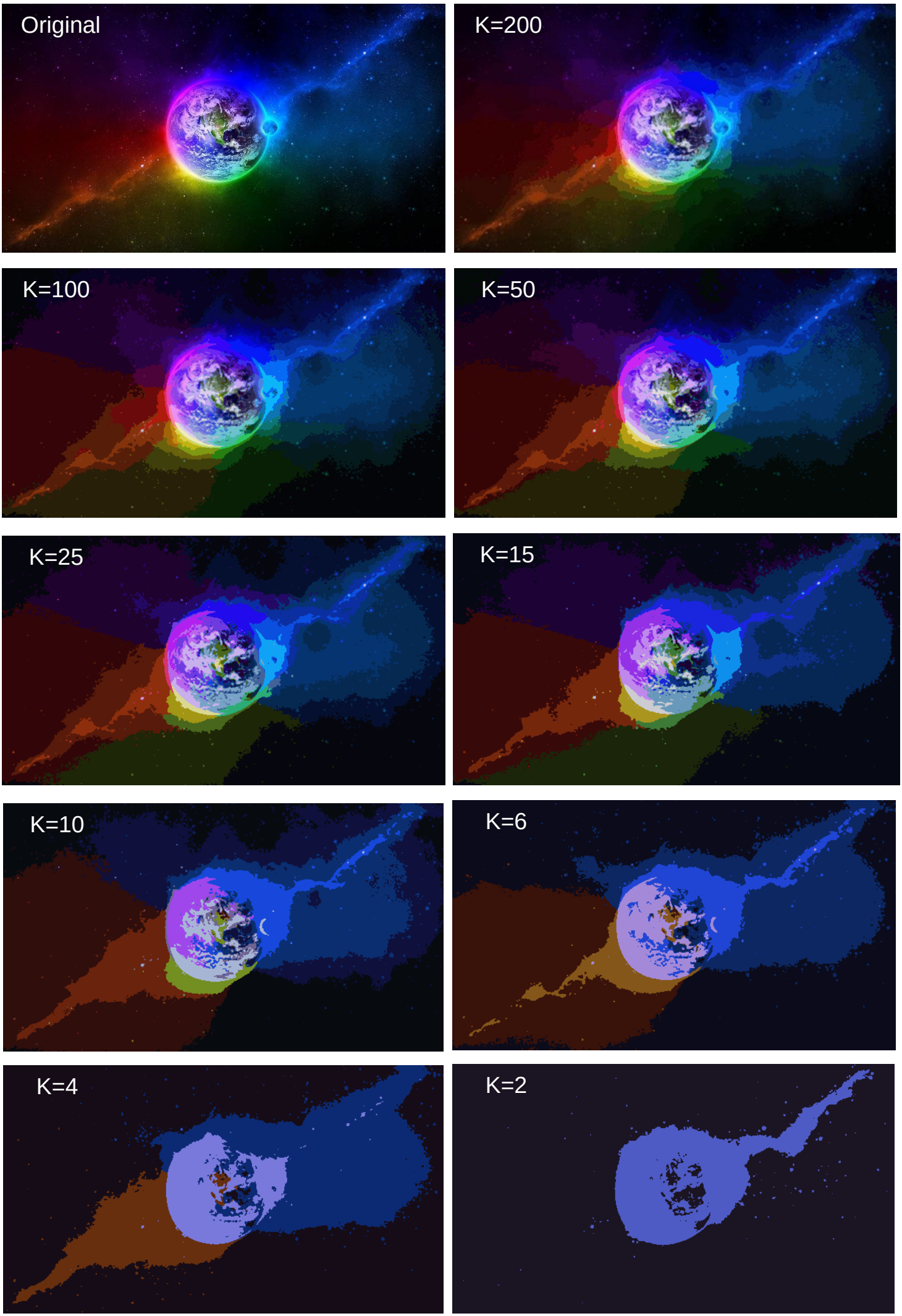

K=10

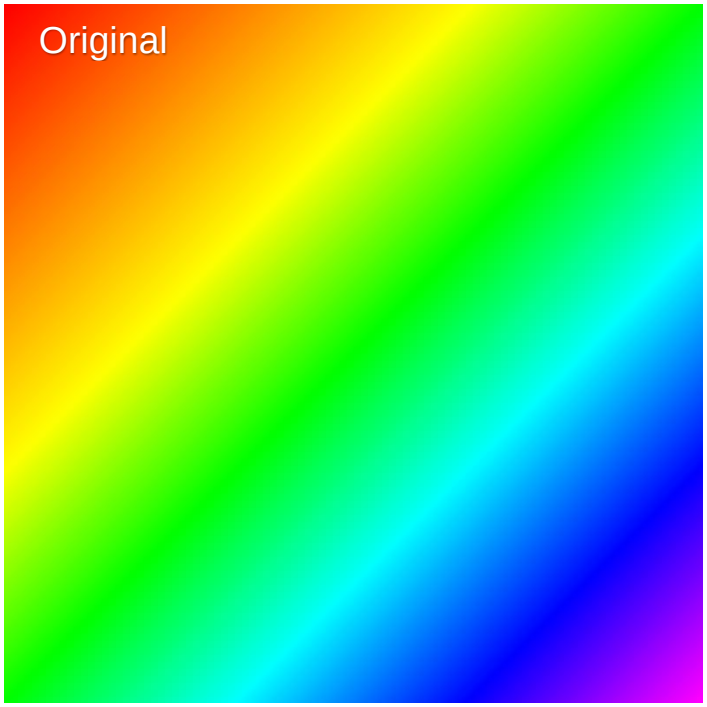

K=5

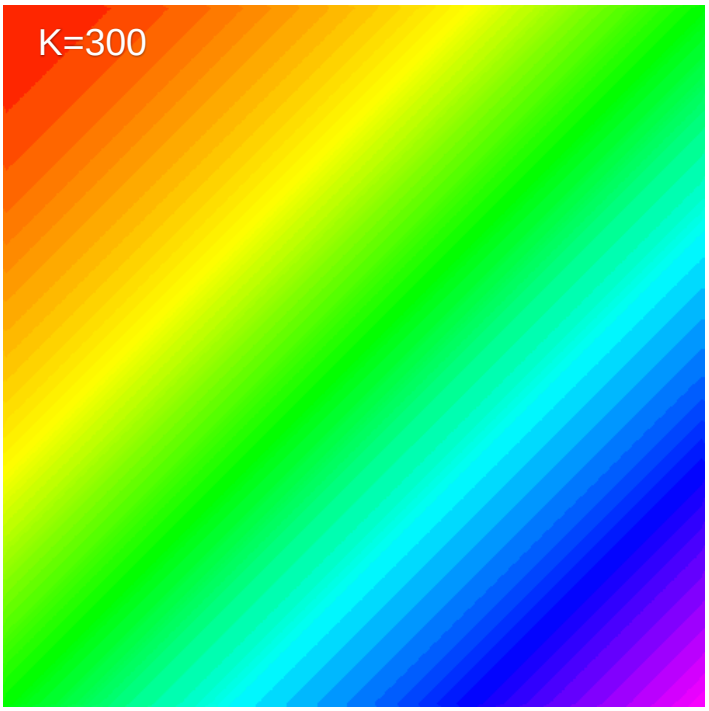

K=4

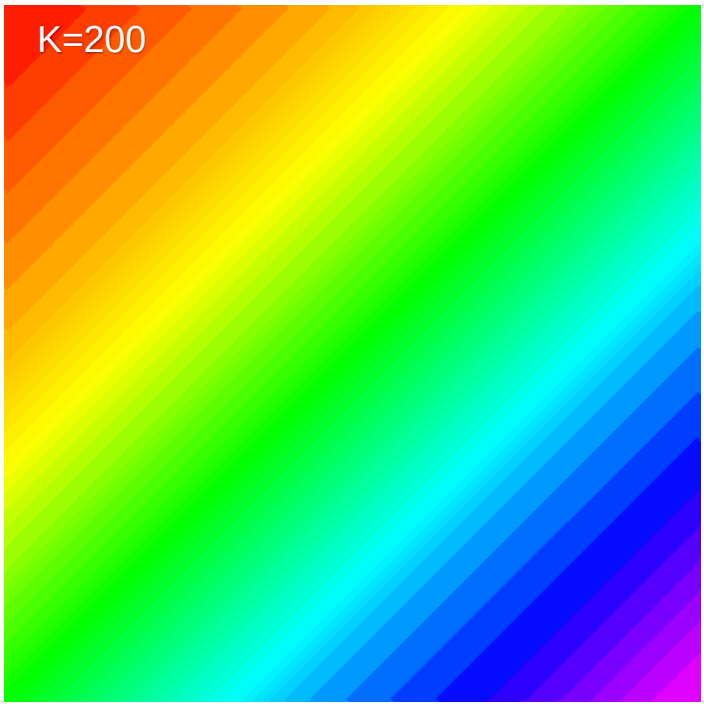

K=3

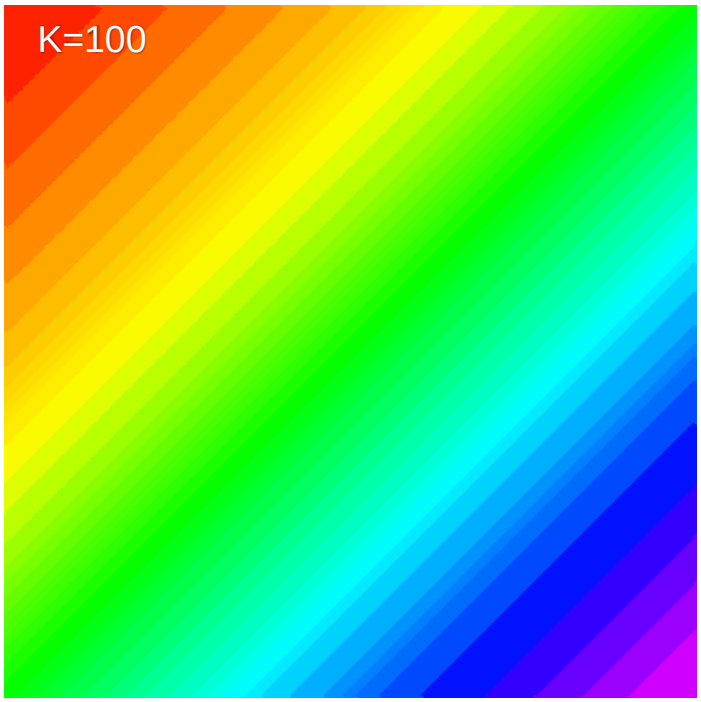

K=2

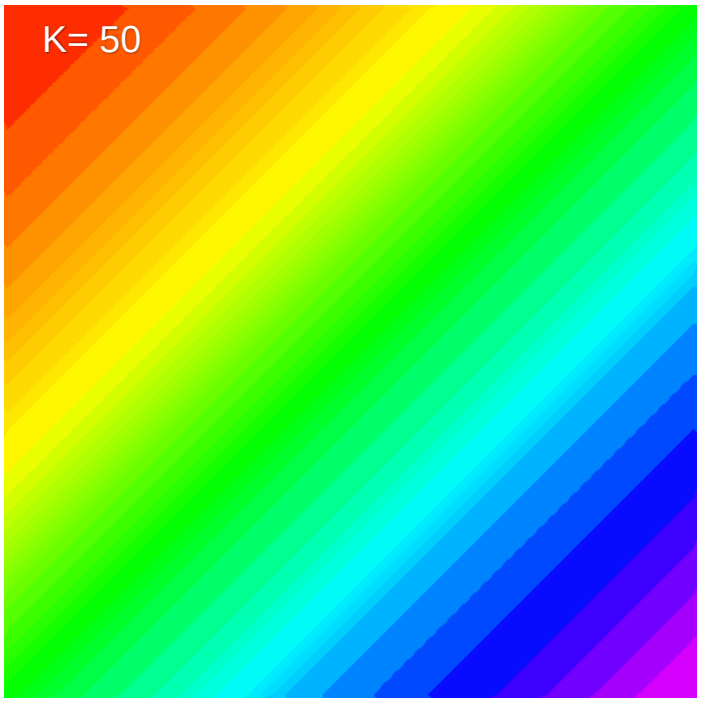

Original
K=200
K=100
K=50
K=25
K=15
K=10
K=6
K=4
K=2

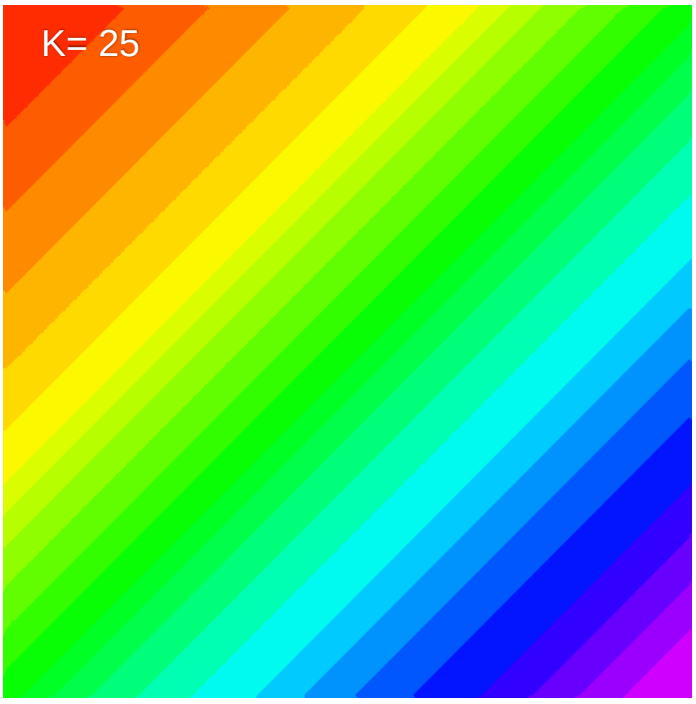

Original

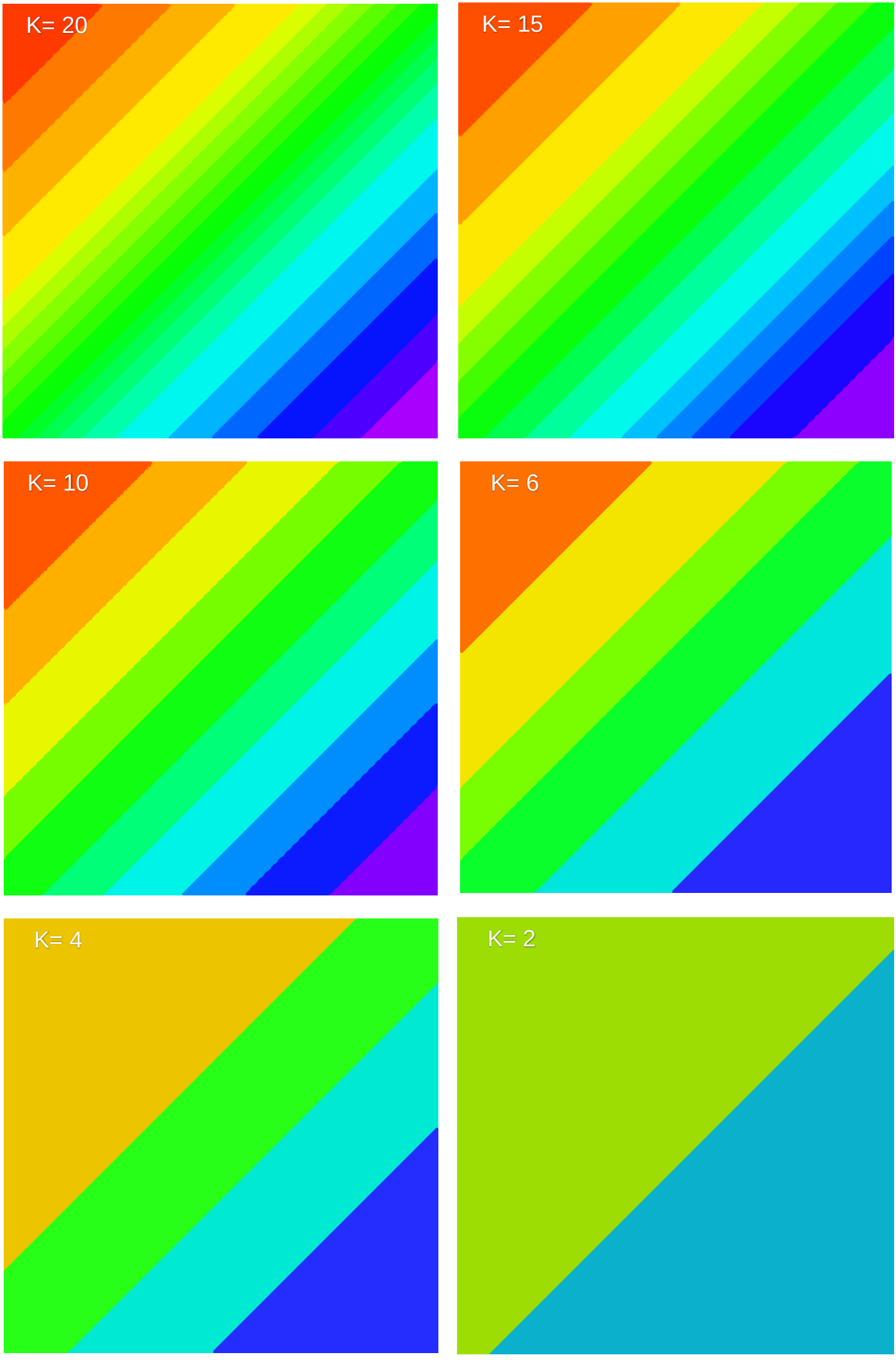

K=300


K=200


K=100


K= 50


K= 25


K= 20
K= 15
K= 10
K= 6
K= 4
K= 2