DCA – CT – UFRN

COMPUTAÇÃO GRÁFICA

Lista de Exercícios 03

https://www.dca.ufrn.br/~lmarcos/courses/compgraf/exercicios/ex3.html

Questão 1: Explique o que é luz e cor, fisicamente (comprimento de onda, faixa visível do espectro, etc). Cite alguns exemplos de cores com seus comprimentos de ondas aproximados.

R: A luz visível é uma parte do espectro eletromagnético com frequências entre aproximadamente $4.3 \cdot 10^{14}$ Hz até $7.5 \cdot 10^{14}$ Hz , o que corresponde a comprimentos de onda entre 400 nm e 700 nm. Isso significa que a luz visível abrange as cores do arco-íris, do vermelho ao violeta.

Cada cor corresponde a uma frequência específica dentro desse espectro. Por exemplo, o vermelho possui uma frequência mais baixa, enquanto o violeta possui uma frequência mais alta. Além disso, as cores podem ser combinadas através da sobreposição de ondas eletromagnéticas, conforme a equação $E_1 + E_2 = E$ onde E_1 e E_2 são ondas eletromagnéticas que se combinam para criar uma nova onda.

Questão 2: Nós enxergamos completamente o que fisicamente existe? Discorra um pouco sobre isso.

R: Nós não enxergamos completamente tudo o que fisicamente existe. O espectro eletromagnético é vasto e abrange muito mais do que a luz visível, que é apenas uma pequena faixa. Existem frequências abaixo do vermelho, como o infravermelho, utilizado em tecnologias como comunicação por controle remoto de televisores. Além disso, há frequências acima do violeta, como o ultravioleta. Portanto, nossa percepção visual é limitada a apenas uma pequena parte do espectro eletromagnético.

Questão 3: Descreva o processo de captação da luz e cor em nossos olhos (retina, fóvea, cones, bastões, pigmentos visuais, etc).

R: Quando a luz de um objeto externo está adequadamente focada pelo olho, ela forma uma imagem na retina. Nossa retina contém dois tipos principais de receptores: cones e bastonetes. Cada olho possui cerca de 7.6 à 7 milhões de cones. Os cones são sensíveis à cor e são responsáveis pela visão detalhada em condições de luz adequada, um fenômeno conhecido como *visão fotópica*. Eles estão concentrados principalmente na região central da retina, chamada fóvea. Os bastonetes, por outro lado, são muito mais numerosos (cerca de 75 a 150 milhões) e estão distribuídos pela retina, exceto na *fóvea*. Eles são responsáveis pela visão periférica e são mais sensíveis à luz fraca, proporcionando uma visão geral do campo visual, chamada de visão *escotópica*. É importante mencionar também os pigmentos visuais presentes nos cones e bastonetes, que são fundamentais para a percepção da luz e da cor.

Questão 4: O que são espaços de cores? Para que servem? Quais são os mais comuns usados?

R: Espaços de cores, também conhecidos como modelos de cores ou sistemas de cores, são utilizados para padronizar a especificação das cores de forma mais fácil e compreensível. Basicamente, um modelo de cores é um sistema de coordenadas onde cada cor é representada por um ponto.

Os espaços de cores mais comuns são adaptados de acordo com o equipamento onde serão aplicados, como monitores e impressoras. Para monitores, o modelo mais comum é o RGB (Red, Green e Blue), enquanto para impressoras mais antigas, era utilizado o modelo CMY (Cyan, Magenta, Yellow). No entanto, a necessidade de adicionar o preto ao conjunto de cores surgiu, pois a combinação dessas três cores não era suficiente para criar um preto puro. Assim, atualmente, as impressoras utilizam o modelo CMYK (Cyan, Magenta, Yellow e blacK).

O modelo RGB pode ser expandido em alguns casos, como em animações, onde é adicionado um canal alfa (α) para representar a transparência da cor, resultando em RGBα.

Questão 5: Qual a diferença entre um espaço aditivo e um subtrativo? Explique a aplicação de ambos (onde são utilizados).

R: Um espaço de cores subtrativo é aquele onde as cores são produzidas pela remoção de certas frequências de luz. Isso é comum em impressão, onde as cores são formadas pela sobreposição de pigmentos. Por exemplo, no modelo CMYK (Cyan, Magenta, Yellow, Black), o ciano remove a frequência vermelha, o magenta remove a frequência verde, o amarelo remove a frequência azul, e o preto é utilizado para aprimorar a qualidade de impressão.

Por outro lado, um espaço de cores aditivo é aquele onde as cores são formadas pela adição de luz de diferentes cores. Este método é comum em dispositivos de exibição, como monitores e telas. No modelo RGB (Red, Green, Blue), diferentes intensidades de luz vermelha, verde e azul são combinadas para criar uma ampla gama de cores.

Portanto, modelos que adicionam pigmentos para remover determinadas frequências de luz, como no caso do CMYK, são chamados de subtrativos. Já os modelos que adicionam luz para criar cores, como o RGB, são chamados de aditivos.

Questão 6: O que é uma imagem? Como representar imagens (explique os formatos vetorial e raster)? Por que os displays vetoriais morreram?

R: Uma imagem digital é uma representação de uma cena utilizando números binários. Existem dois tipos primordiais de representação: a matricial e a vetorial.

- **Representação matricial (*raster*)**: Neste formato, a imagem é representada como uma grade de pixels, onde cada pixel possui uma cor específica. Os formatos de arquivo mais comuns para

imagens raster incluem JPEG, PNG e GIF.

- **Representação vetorial**: Aqui, a imagem é representada por meio de formas geométricas, como linhas e curvas, definindo pontos e suas relações espaciais. Os formatos vetoriais mais comuns incluem SVG (*Scalable Vector Graphics*) e AI (*Adobe Illustrator*).

A popularidade dos displays vetoriais diminuiu principalmente devido à evolução tecnológica e às demandas do mercado. Embora os gráficos vetoriais possam oferecer escalabilidade sem perda de qualidade, a complexidade de implementação e a preferência por imagens de alta resolução em aplicações como jogos, filmes e web levaram à predominância de displays rasterizados.

Questão 7: O que é um Display Full-color (ou true-color)? Explique por que alguns frame-buffers (true-collor ou high-end) possuem até 96 bits de profundidade (ou mais).

R: Um Display Full-color, também conhecido como true-color, é um tipo de exibição que é capaz de reproduzir uma ampla gama de cores visíveis pelo olho humano. Geralmente, isso é alcançado utilizando uma representação matricial de imagem, onde cada pixel é composto por três canais de cor: vermelho, verde e azul (RGB). Cada canal é representado por um certo número de bits, sendo comum o uso de 8 bits para cada canal, resultando em um total de 24 bits por pixel. No entanto, alguns frame-buffers (ou buffers de imagem) de alta qualidade podem ter profundidades de cor de até 96 bits ou mais. A razão para o uso de uma profundidade de cor tão alta está relacionada com a precisão na reprodução de cores e detalhes, especialmente em aplicações que demandam alta fidelidade visual, como edição de imagens e vídeos. Com uma profundidade de cor maior, há mais nuances de cores disponíveis, o que permite uma representação mais precisa de tons e sutilezas na imagem.

Questão 8: O que é uma tabela de cores (color-map)? Tente entender e faça um algoritmo que mapeia M cores (RGB) em uma colormap com N entradas formando uma rampa de tons de cinza?

R: Uma tabela de cores, também conhecida como color-map, é uma estrutura de dados que associa valores de intensidade ou valores de canal (como vermelho, verde e azul) a cores específicas. É frequentemente utilizada em visualização de imagens e gráficos para mapear valores de dados para cores visuais.

Um algoritmo em C++ para mapear M cores RGB em uma color-map com N entradas, formando uma rampa de tons de cinza, pode ser implementado da seguinte maneira:

```
#include <iostream>
#include <vector>

/**
 * @brief Creates a color map with N entries, mapping M RGB
colors to gray tones.
 *
 * @param M The number of RGB colors.
 * @param N The number of entries in the color map.
 * @return A 2D vector representing the color map.
 */
std::vector<std::vector<int>> createColorMap(int M, int N)
{
  std::vector<std::vector<int>> colorMap(N, std::vector<int>(3));
  int step = M / N;
  int grayTone;

  for (int i = 0; i < N; ++i)
  {
    grayTone = i * step;
    colorMap[i][0] = grayTone;
    colorMap[i][1] = grayTone;
    colorMap[i][2] = grayTone;
  }

  return colorMap;
}
```

```
/**
 * @brief Main function.
 */
int main()
{
  int M = 256;
  int N = 100;

  std::vector<std::vector<int>> colorMap = createColorMap(M, N);

  std::cout << std::endl
            << "Color-map result:" << std::endl
            << std::endl;
  for (int i = 0; i < N; ++i)
  {
    std::cout << "Entry " << i << ": (" << colorMap[i][0] << ", "
<< colorMap[i][1] << ", " << colorMap[i][2] << ");" << std::endl;
  }

  std::cout << std::endl;

  return 0;
}
```

Para N = 10 e M = 256 teremos a seguinte saida:

```
Entry 0: (0, 0, 0);
Entry 1: (25, 25, 25);
Entry 2: (50, 50, 50);
Entry 3: (75, 75, 75);
Entry 4: (100, 100, 100);
Entry 5: (125, 125, 125);
Entry 6: (150, 150, 150);
Entry 7: (175, 175, 175);
Entry 8: (200, 200, 200);
```

```
Entry 9: (225, 225, 225);
```

Como podemos notar, isso se trata de uma rampa partindo do 0 até o 255 em escala de cinza visto que a intensidade do Vermelho, verde e azul são as mesmas para cada um dos pontos.

Questão 9: Cite alguns dos formatos mais usados para armazenamento e manipulação de imagens (descreva suscintamente).

R:

- PNG (Portable Network Graphics): Um formato de imagem sem perdas, adequado para gráficos com áreas de cores sólidas e transparência.
- JPEG (Joint Photographic Experts Group): Um formato com perdas comumente usado para fotografias e imagens digitais. Ele utiliza um algoritmo de compressão que pode reduzir o tamanho do arquivo, mas com alguma perda de qualidade.
- WEBP: Desenvolvido pelo Google, é um formato de imagem moderno que oferece compressão eficiente e suporte para animações e transparência.
- SVG (Scalable Vector Graphics): Um formato baseado em XML que representa imagens vetoriais, ideal para gráficos escaláveis sem perda de qualidade.

Questão 10: Para que serve a operação de convolução? Coloque a sua forma matemática, contínua e discreta, bidimensional.

R: A operação de convolução é amplamente utilizada em processamento de sinais e imagens para diversos fins, como filtragem, detecção de bordas, suavização, entre outros. Ela consiste em combinar duas funções para produzir uma terceira função que representa a forma como uma delas é "misturada" com a outra.

$$f(x,\ y) * g(x,\ y) \ = \ \iint_{-\infty}^{\infty} f(\alpha,\ \beta) g(x-\alpha,\ y-\beta)\, d\alpha d\beta$$

$$f[x,\ y] * g[x,\ y] \ = \ \frac{1}{mn} \sum_{m=0}^{m-1} \sum_{n=0}^{n-1} f[m, n] g[x-m,\ y-n]$$

Questão 11: Descreva o que é filtragem e para que serve.

R: Filtragem é um processo utilizado para modificar ou extrair informações específicas de um sinal ou conjunto de dados. Em imagens digitais, a filtragem é frequentemente usada para remover ruídos ou informações indesejadas, realçar características de interesse ou suavizar a imagem.

Questão 12: Descreva a aplicação (para que serve), coloque a forma matemática e apresente exemplos de máscaras 3x3 para:

R:

- **Filtro suavizante (blurring):** Serve para reduzir o ruído e suavizar a imagem, resultando em

uma versão menos granulada ou menos detalhada da mesma.

$$g(x,y) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * f(x,y)$$

- **Filtro para realce de arestas (detector de arestas):** Utilizado para realçar bordas e características de alta frequência em uma imagem.

$$g(x,y) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} * f(x,y)$$

- **Filtro Gaussiano:** Aplica uma operação de suavização à imagem, onde o valor de cada pixel é substituído pela média ponderada dos valores dos pixels vizinhos.

$$g(x,y) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * f(x,y)$$

- **Filtro Gradiente do Gaussiano:** Utilizado para realçar bordas e características de uma imagem, combinando os efeitos do filtro Gaussiano com a detecção de gradientes de intensidade.

$$g(x,y) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f(x,y)$$

- **Filtro Laplaciano do Gaussiano:** Serve para realçar bordas e características de uma imagem, enfatizando regiões de alta frequência.

$$g(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * f(x,y)$$

Questão 13: Coloque as imagens resultantes das convoluções do filtro gradiente de Sobel 3 x 3, nas duas direções X e Y, com a imagem abaixo. Coloque a imagem que representa a magnitude dos filtros aplicados.

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 0 0
0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

R:

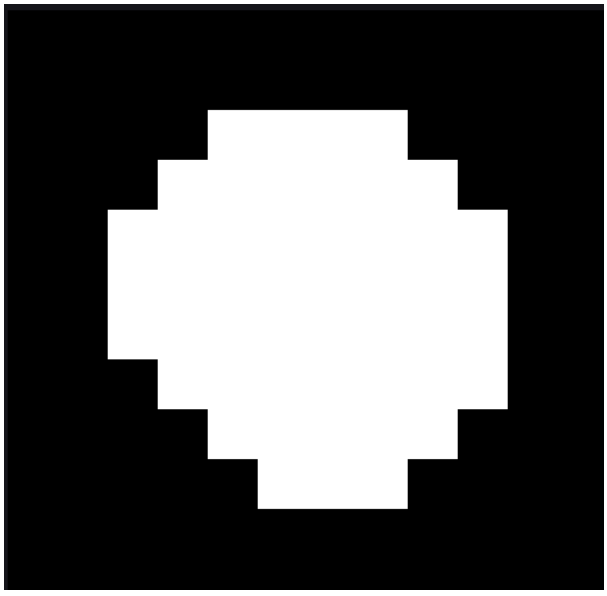

Figure 1: Original - 12x12

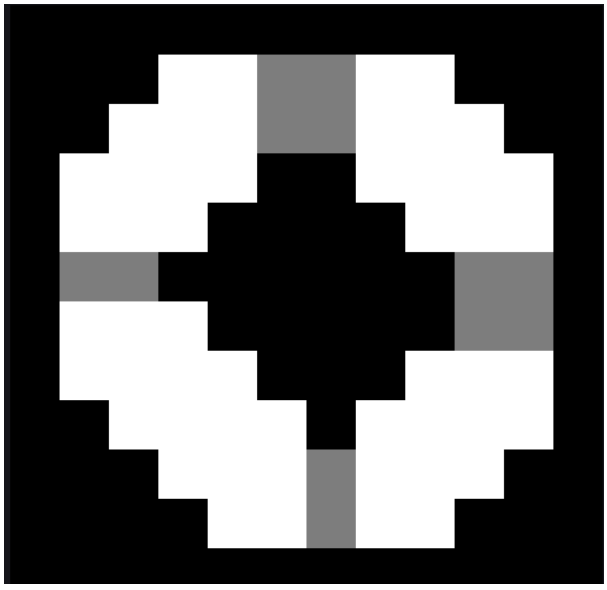

Figure 2: Gradiente de Sobel 3x3 completo - 12x12

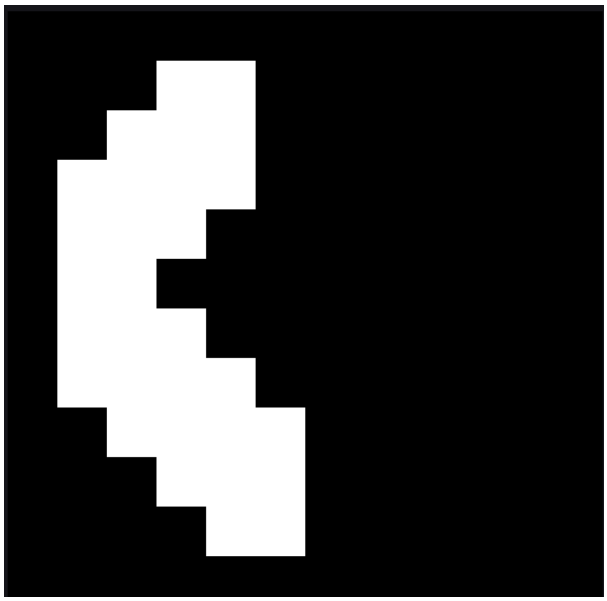

Figure 3: Gradiente de sobel 3x3 X - 12x12

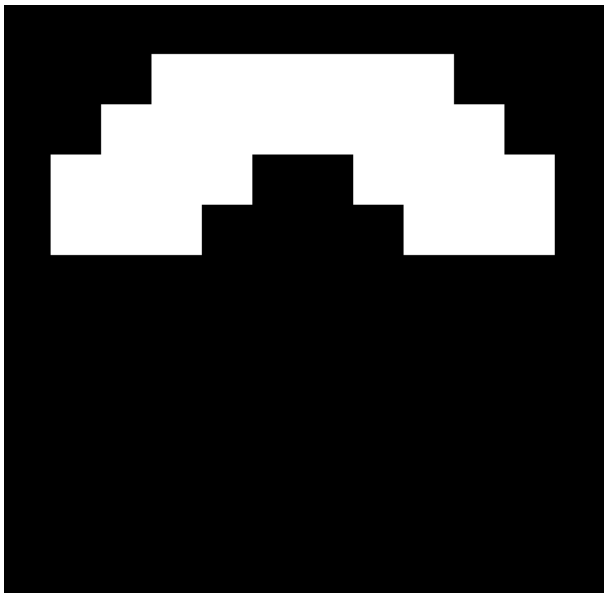

Figure 4: Gradiente de sobel 3x3 Y - 12x12

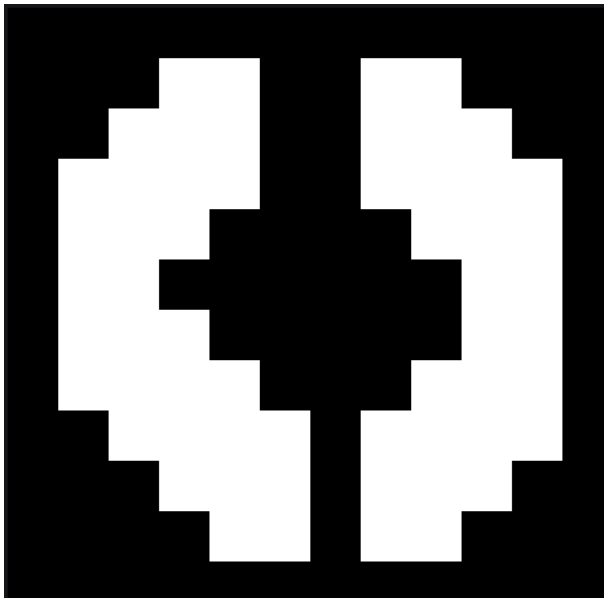

Figure 5: Gradiente de sobel 3x3 X (ABS) - 12x12

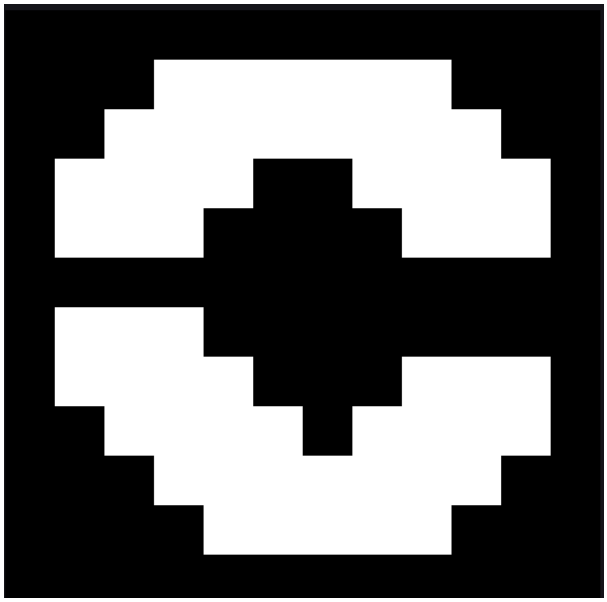

Figure 6: Gradiente de sobel 3x3 Y (ABS) - 12x12

Para mais pontos, fica mais interessante de se observar as imagens. Por exemplo uma matriz 400x400:

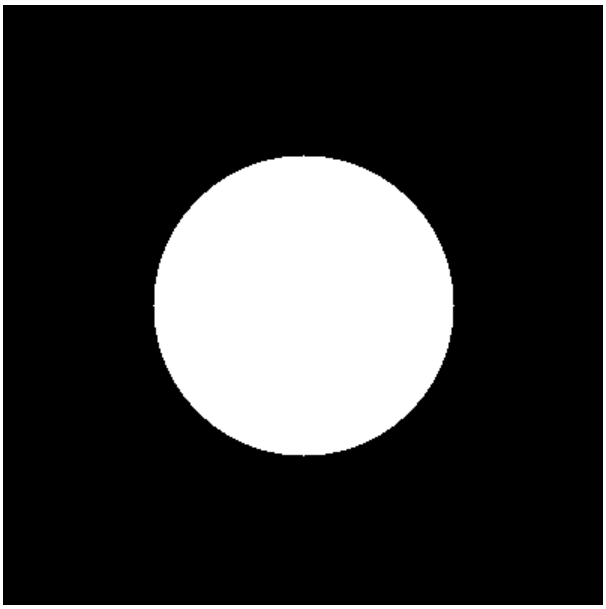

Figure 7: Original - 400x400

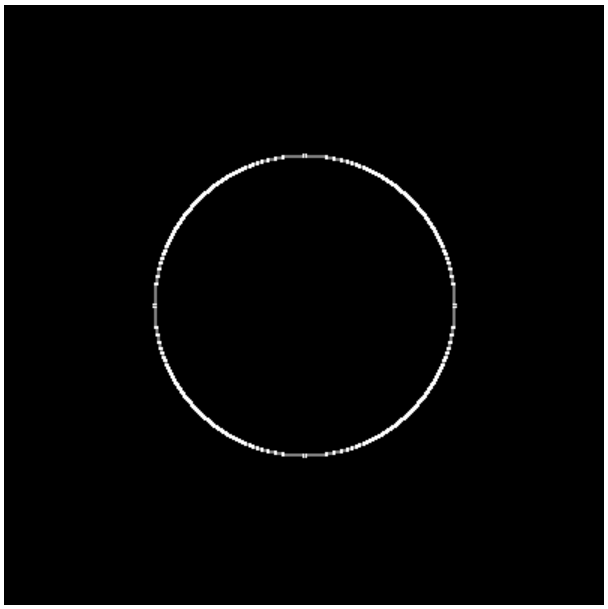

Figure 8: Gradiente de Sobel 3x3 completo - 400x400

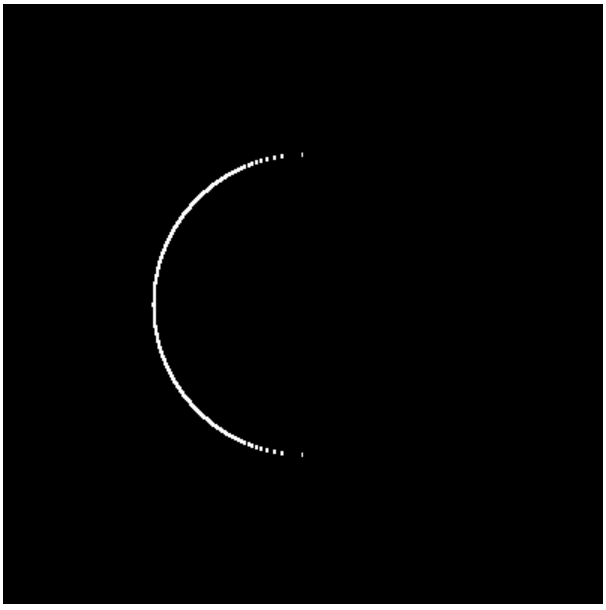

Figure 9: Gradiente de sobel 3x3 X - 400x400

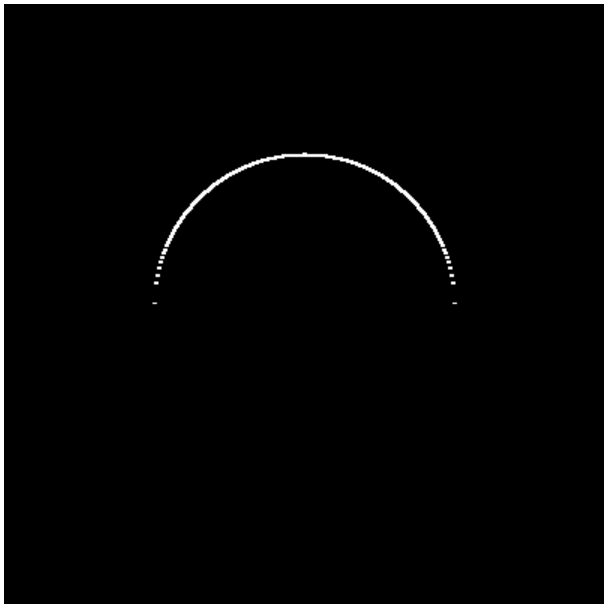

Figure 10: Gradiente de sobel 3x3 Y - 400x400

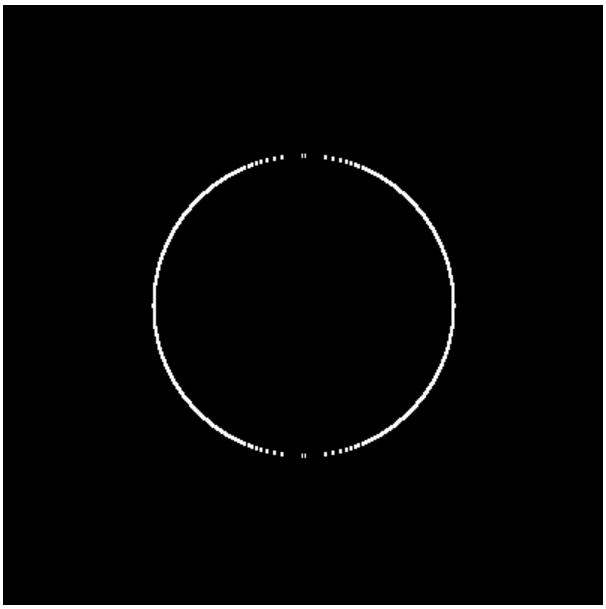

Figure 11: Gradiente de sobel 3x3 X (ABS) - 400x400

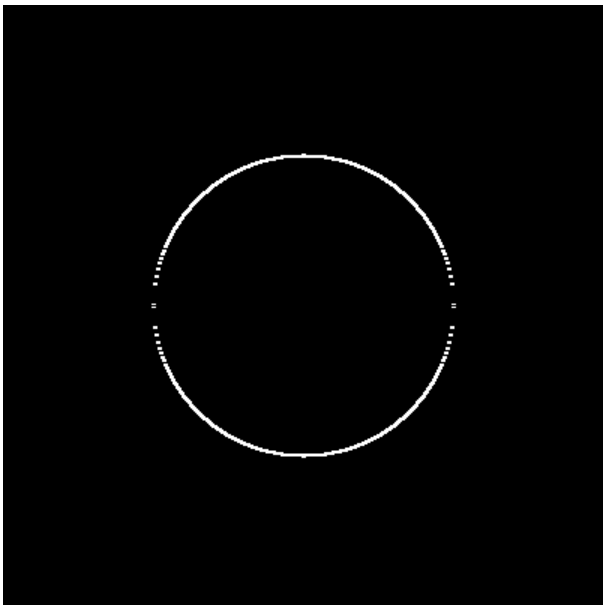

Figure 12: Gradiente de sobel 3x3 Y (ABS) - 400x400

Código utilizado para desenvolvimento:

```
#include <iostream>
#include <opencv2/opencv.hpp>

/**
 * @brief Default image data with values equal to 1 converted to
255.
 */
unsigned char defaultImageData[12][12] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 255, 255, 255, 255, 0, 0, 0, 0},
    {0, 0, 0, 255, 255, 255, 255, 255, 255, 0, 0, 0},
    {0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0},
    {0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0},
    {0, 0, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0},
    {0, 0, 0, 255, 255, 255, 255, 255, 255, 255, 0, 0},
    {0, 0, 0, 0, 255, 255, 255, 255, 255, 0, 0, 0},
    {0, 0, 0, 0, 0, 255, 255, 255, 0, 0, 0, 0},
```

```
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

/**
 * @brief Prints the matrix representation of the image.
 *
 * @param image The image matrix.
 * @param N Size of the image matrix.
 */
void printImageMatrix(cv::Mat image, int N = 12)
{
  for (int i = 0; i < N; i++)
  {
    for (int j = 0; j < N; j++)
    {
      int pixel_value = static_cast<int>(image.at<uchar>(i, j));
      std::cout << pixel_value << " ";
    }
    std::cout << std::endl;
  }
}

/**
 * @brief Generates a circular image with given size.
 *
 * @param N Size of the image.
 * @return Generated image matrix.
 */
cv::Mat generateImage(int N)
{
  cv::Mat image(N, N, CV_8UC1, cv::Scalar(0));

  int center_x = N / 2;
  int center_y = N / 2;
  int radius = N / 4;

  for (int i = 0; i < N; i++)
  {
```

```
    for (int j = 0; j < N; j++)
    {
      int distance = (i - center_x) * (i - center_x) + (j -
center_y) * (j - center_y);
      if (distance <= radius * radius)
      {
        image.at<uchar>(i, j) = 255;
      }
    }
  }

  return image;
}

int main()
{
  const int N = 12;
  // cv::Mat image = generateImage(400);
  cv::Mat image(N, N, CV_8UC1, defaultImageData);

  std::cout << "image" << std::endl;
  printImageMatrix(image, N);
  cv::imwrite("image.png", image);

  int scale = 1;
  int delta = 0;
  cv::Mat grad;
  cv::Mat grad_x, grad_y;
  cv::Mat abs_grad_x, abs_grad_y;

  cv::Sobel(image, grad_x, CV_16S, 1, 0, 3, scale, delta,
cv::BORDER_DEFAULT);
  cv::imwrite("grad_x.png", grad_x);
  std::cout << "grad_x" << std::endl;
  printImageMatrix(grad_x, N);

  cv::Sobel(image, grad_y, CV_16S, 0, 1, 3, scale, delta,
cv::BORDER_DEFAULT);
```

```
  cv::imwrite("grad_y.png", grad_y);
  std::cout << "grad_y" << std::endl;
  printImageMatrix(grad_y, N);

  cv::convertScaleAbs(grad_x, abs_grad_x);
  cv::imwrite("abs_grad_x.png", abs_grad_x);
  std::cout << "abs_grad_x" << std::endl;
  printImageMatrix(abs_grad_x, N);

  cv::convertScaleAbs(grad_y, abs_grad_y);
  cv::imwrite("abs_grad_y.png", abs_grad_y);
  std::cout << "abs_grad_y" << std::endl;
  printImageMatrix(abs_grad_y, N);

  cv::addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);
  cv::imwrite("grad.png", grad);
  std::cout << "grad" << std::endl;
  printImageMatrix(grad, N);

  return 0;
}
```

É possível conferir mais informações sobre acessando: https://github.com/ErnaneJ/cg.