Università
della
Svizzera
italiana

Faculty
of Informatics

Computational Fabrication (Spring 2024)

Assignment 3: Halftoning

Submitted by: Krunal Rathod | krunal.rathod@usi.ch

Submission Deadline: 03 May 2024

1 Thresholding

(a) Test Image 1 (b) Test Image 2

Figure 1: Thresholding

The thresholding method is a simple and efficient way to convert a grayscale image into a binary image. This method compares each pixel value in the input image to a threshold value (0.5 in this implementation). If the pixel value is greater than the threshold, the corresponding pixel in the output image is set to 1.0 (white). Otherwise, it is set to 0.0 (black). The thresholding method is implemented in the `convert_using_thresholding` function. The function uses two nested for-loops to iterate over each pixel in the input image. For each pixel, the function checks if the pixel value in the `inputImage` array is greater than 0.5. If the pixel value is greater than 0.5, the corresponding pixel in the `outputImage` array is set to 1.0. Otherwise, the pixel is set to 0.0.

The thresholding method is fast and easy to implement, but it can produce a binary image with visible artifacts and loss of detail. This is because the method does not take into account the spatial relationship between neighboring pixels. As a result, the binary image may have a "blocky" appearance as visible in the above images, with large areas of white or black pixels.

2 Dithering

The code implements the dithering technique. This method quantize the intensity of each pixel in the input image to either 0 or 1, based on a threshold determined by a dithering matrix. The `convert_using_dithering` function takes an input image and produces an output image using the dithering. The function uses a simple 3x3 dithering matrix, which is a fixed matrix of values used to determine the threshold for each pixel. The threshold for each pixel is determined by the value of the dithering matrix at the corresponding position in the matrix. The function iterates through the each pixel in the input image and quantizes its intensity to either 0 or 1 based on the threshold determined by the dithering matrix. If the intensity of the pixel is greater than the threshold, the pixel is quantized to 1, otherwise it is quantized to 0.

(a) Test Image 1 (b) Test Image 2

Figure 2: Dithering

The error (difference between the original intensity and the quantized value) is then distributed to neighboring pixels in a specific pattern, with weights determined by the dithering matrix. The pattern is designed to minimize the overall error in the output image. The resulting image is a binary image, where each pixel is either 0 or 1. Overall, the dithering algorithm is a simple halftoning technique that produces high-quality binary images with smooth shading and minimal artifacts.

3 Error Diffusion

(a) Test Image 1 (b) Test Image 2

Figure 3: Error Diffusion

Error diffusion helps to preserve the overall appearance of the grayscale image in the black and white image. The code implements a basic error diffusion algorithm that is used to convert grayscale images to black and white images. The `convert_using_error_diffusion` function takes an input image and produces an output image using the error diffusion. The algorithm works by iterating over each pixel in the grayscale image. For each pixel, a new pixel value is calculated based on a threshold. In this case, the threshold is 0.5. If the grayscale value of the pixel is greater than 0.5, then the new pixel value is set to 1.0 (white). Otherwise, the new pixel value is set to 0.0 (black).

The difference between the grayscale value of the pixel and the new pixel value is called the error. This error is then diffused to neighboring pixels that have not yet been processed. The weights used to diffuse the error. In the code, the error is diffused to the right, bottom left, bottom center, and bottom right neighbors of the current pixel.

(a) Original Image

(b) Thresholding

(c) Dithering

(d) Error diffusion

Figure 4: Input and output images

In summary, the C++ code implements three halftoning methods for converting a grayscale image into a binary image: thresholding, dithering, and error diffusion. The thresholding method is a simple and efficient way to convert a grayscale image into a binary image by comparing each pixel value in the input image to a threshold value. The dithering method is a more sophisticated way to convert a grayscale image into a binary image by using a dithering matrix to determine the threshold value for each pixel. The error diffusion method is a more advanced way to convert a grayscale image into a binary image by distributing the error between the original pixel value and the binary pixel value to neighboring pixels. The thresholding method is fast and easy to implement, but it can produce a binary image with visible artifacts and loss of detail. The dithering method is more computationally intensive than thresholding, but it can produce a binary image with more detail and fewer artifacts. The error diffusion method is more computationally intensive than the other two methods, but it can produce a binary image with the highest quality.