Project Report

ON

"ORDERED DITHERING"

By

Omar Rawashdeh
Harshal Raut
Utsav Shah

Under the Guidance of
Prof. William Wee

Department of Electrical Engineering and Computer Science
College of Engineering and Applied Sciences
Cincinnati, OH 45220
2016-2017

University of Cincinnati

Contents

Chapter 1

Introduction

1.1 Overview

Halftoning or analog halftoning is a process that simulates shades of gray by varying the size of tiny black dots arranged in a regular pattern. This technique is used in printers, as well as the publishing industry. If you inspect a photograph in a newspaper, you will notice that the picture is composed of black dots even though it appears to be composed of grays. This is possible because of the spatial integration performed by our eyes. Our eyes blend fine details and record the overall intensity[1].

1.2 Digital Halftoning

Digital halftoning is similar to halftoning in which an image is decomposed into a grid of halftone cells. Elements (or dots that halftoning uses in simulates shades of grays) of an image are simulated by filling the appropriate halftone cells. The more number of black dots in a halftone cell, the darker the cell appears. For example, in Figure 4, a tiny dot located at the center is simulated in digital halftoning by filling the center halftone cell; likewise, a medium size dot located at the top-left corner is simulated by filling the four cells at the top-left corner. The large dot covering most of the area in the third image is simulated by filling all halftone cells.

1.3 Methods

Many image rendering technologies only have binary output. For example, printers can either fire a dot or not. Halftoning is a method for creating the illusion of continuous tone output with a binary device. Effective digital halftoning can substantially improve the quality of rendered images at minimal cost[2].

1.4 Types

- Dithering
- Patterning
- Error Diffusion

1.4.1 Dithering

Dithering is a color reproduction technique in which dots or pixels are arranged in such a way that allows us to perceive more colors than are actually used. This method of ”creating” a large color palette with a limited set of colors is often used in computer images, television and the printing industry. The graphic illustrations below show how a full color image can been approximated by using only a few colors. As magnification is increased, see how each color seen is broken down to a combination of the primary colors used in the dither pattern.

The technique used for generating digital halftoning images is dithering. Unlike patterning, dithering creates an output image with the same number of dots as the number of pixels in the source image. Dithering can be thought of as thresholding the source image with a dither matrix. The matrix is laid repeatedly over the source image. Wherever the pixel value of the image is greater than the value in the matrix, a dot on the output image is filled. A well-known problem of dithering is that it produces artifacts of patterns introduced by fixed thresholding matrices[3].

1.5 Ordered Dithering

Ordered dithering and cluster dithering are the fastest methods. They are effective for reducing the number of colors to 256 colors or less. Ordered dithering is the default dithering method when painting to a display device that is 256 colors or less.

These methods take advantage of the fact that the colors in most palettes are ordered so that similar shades are next to each other in the palette. Pixels used in dithering are taken from these adjacent colors to achieve the nearest shade. They place pixels from shades near to the original color to achieve a smoothing effect. Ordered dithering avoids blotches of color by adding to or subtracting from the nearest-color value of each pixel to ensure that adjacent pixels do not have exactly the same color. However, If the colors in the palette are not ordered, the results may be poor[4].

Chapter 2

Project Description

2.1 MATLAB Implementation

Step 1:
The original image is fed into MATLAB to be processed upon.

Step 2:
The original image is cut into smaller blocks of 8x8 pixels.

Step 3:
The average value of graylevel of each block of 8x8 pixels is calculated.

Step 4:
A new block that holds number of pixels representing the average value of original block. This is the first part of the new image.
8x8 pixel block is represented by one value of graylevel.

Step 5:
Placed the dots according to designed dither matrix.

Step 6:
The desired result is obtained from this Ordered Dithering Technique.
newline Note: Some information of the original image is lost as the average value of 8x8 pixel blocks is considered.

Chapter 3

Implementation

The following is the code implementing the Ordered Dithering Algorithm in MATLAB:

```
%########################################################################
%########################################################################
%##                     Digital Image Processing HW1                  ##
%##                   Implementing the Find_S algorithm                ##
%##                         Matlab R2016a was used                     ##
%##                             v9.0.0.341360                          ##
%########################################################################
%########################################################################


% [filename,pathname] = uigetfile({'*.png', '*.bmp', '*.jpg'; '*.*'}, 'File
    Selector');
[FileName,PathName,FilterIndex] = uigetfile('*.jpg','Select the Image file');

%if the user did not select a file set the file to be "1.jpg" in the folder
%"Images"
if isequal(FileName,0)
    PathName = 'Images\';
    FileName = '1.jpg';
end

%dimension of the area that we are going to work with
n = 8;
%which Ordered Dithering matrix we want to use
%Order dithering matrices are stored in the properties of the ImgManager
%class
OrderedDitherMatrixNum = 2;
%load the original image to compare if we want to show them side by side,
%the original and the dithered
OriginalImg = imread(strcat(PathName, FileName));
%instatiate a new instance of the ImgManager class
ImgM = ImgManager();
%the full path of the image is the combination of both the path and the
%name of the image file
path = strcat(PathName, FileName);
%again load the image but this time through the ImgManager class which will
%convert it to gray scale image after it loads it, so this copy of the
%image is what we are going to use for processing
img = ImgM.load(path);
%call the function that dither the image
DitheredImg = ImgM.Dither(n, img, (n^2+1), OrderedDitherMatrixNum);

%show the image on the screen
hFig = figure(); %figure window
set(hFig, 'Position', [300 150 1200 800]);  %resize window
imshow(DitheredImg); %show the image
% subplot(1,2,1), imshow(OriginalImg);
```

```
% subplot(1,2,2), imshow(DitheredImg);



%not used code, leaving it here just in case i need to reuse part of it
%==========================
% subplot(1,2,1), imshow(img);
% subplot(1,2,2), imshow(DitheredImg);
% subplot(1,2,2), imshow(imcomplement(NewImage));
% imshow(NewImage);


% Dimentions(1:4) = size(DitherImg);
% NewImageWidth = Dimentions(1)*Dimentions(3);
% NewImageHeight = Dimentions(2)*Dimentions(4);
% NewImage = zeros(NewImageWidth, NewImageHeight);
% %convert from 4D matrix to 2D matrix that we can use to show the image
% for i=1:NewImageHeight
%     for j=1:NewImageWidth
%         NewImage(j,i) = DitherImg(ceil(j/n), ceil(i/n), max(mod(j, n), 1), max(mod(i, n), 1));
%     end
% end
```

Class ImgManager and Function DitheredImg is defined here:

```

classdef ImgManager<handle
    %IMGMANAGER Summary of this class goes here
    %   Detailed explanation goes here

    properties (Access = private)
        OrderedDither;
%       ODNum;
    end

    methods
        %constructor
        function obj = ImgManager()
%           obj.ODNum = 2;
            obj.OrderedDither(1,:,:) = [6, 48, 14, 40, 8, 46, 16, 38; ...
                                        64, 21, 51, 28, 60, 9, 53, 32; ...
                                        13, 35, 1, 42, 10, 33, 3, 45; ...
                                        56, 27, 58, 18, 50, 26, 59, 24; ...
                                        7, 44, 9, 34, 2, 41, 12, 39; ...
                                        62, 20, 49, 25, 57, 17, 52, 30; ...
                                        15, 37, 4, 43, 11, 36, 5, 47; ...
                                        54, 31, 61, 23, 55, 29, 63, 22];
            %matrix 2
            obj.OrderedDither(2,:,:) = [15, 62, 19, 34, 1, 50, 31, 46; ...
                                        18, 35, 16, 63, 32, 47, 2, 49; ...
                                        61, 14, 33, 20, 51, 4, 45, 30; ...
                                        36, 17, 60, 13, 64, 29, 48, 3; ...
                                        11, 58, 21, 40, 5, 52, 27, 44; ...
                                        22, 37, 12, 59, 28, 41, 6, 53; ...
                                        57, 10, 39, 24, 55, 8, 43, 26; ...
                                        38, 23, 56, 9, 42, 25, 54, 7];
        end

        %convert image to grayscale then load it and return the loaded
        %image
        function handle = load(~, path)
            if nargin < 2
```

```
                path = 'Images/1.jpg';
            end
            handle = rgb2gray(imread(path));
        end

        %this function halftone an image using an ordered dither algorithm
        function DitheredImg = Dither(obj, areaDimension, ReadyImg,
            Brightnesslevels, Order)
            %check the number of arguments and fill the missing parameters
            %with default values
            if nargin < 2
                areaDimension = [1, 1];
            end
            if nargin < 3
                ReadyImg = obj.load('Images/1.jpg');
            end
            if nargin < 4
                Brightnesslevels = 65;
            end
            if nargin < 5
                Order = 1;
            end

%               ReadyImg = obj.load(Img);
            %get width and height
            [width, height] = size(ReadyImg);
            FullSegments = [fix(width/areaDimension) fix(height/areaDimension)];
            LeftOver = [mod(width, areaDimension), mod(height, areaDimension)];

            %determine the number of segments of the image, if it is an image
            %with dimensions which are multiplication of 8 then there would
            %be no leftovers at the edges
            if(LeftOver(1) ~= 0)
                w = FullSegments(1)+1;
            else
                w = FullSegments(1);
            end
            if(LeftOver(2) ~= 0)
                h = FullSegments(2)+1;
            else
                h = FullSegments(2);
            end
            %initialize the matrix to store the dithered image values
            ImageSegmentsValues = zeros(w, h);
            ZerOneMatrix = zeros(w, h, areaDimension, areaDimension);

            %for each column and each row
            for i=1:FullSegments(2)
                for j=1:FullSegments(1)
                    AllValuesInCurrentImageSegment = obj.readImgPartValues(
                        ReadyImg, ((j-1)*areaDimension)+1, ((i-1)*areaDimension)
                        +1, areaDimension);
                    %fill in number of dots that we want to show per segment.
                    %example we want 65 level of brightness, and the mean of
                        values was 255 (completely white)
                    %(65-1) - round((255+1) / (256/(65-1)) =
                    %64 - round(256 / (256/64)) =
                    %64 - round(256 / 4) =
                    %64 - round(64) =
                    %64 - 64 = 0 (so we are going to fill no dots at all)
                    ImageSegmentsValues(j, i) = (Brightnesslevels -1) - round((
                        mean(mean(AllValuesInCurrentImageSegment))+1) / (256/(
                        Brightnesslevels-1)));
                    ZerOneMatrix(j,i,:,:) = obj.getzero1matrix(
```

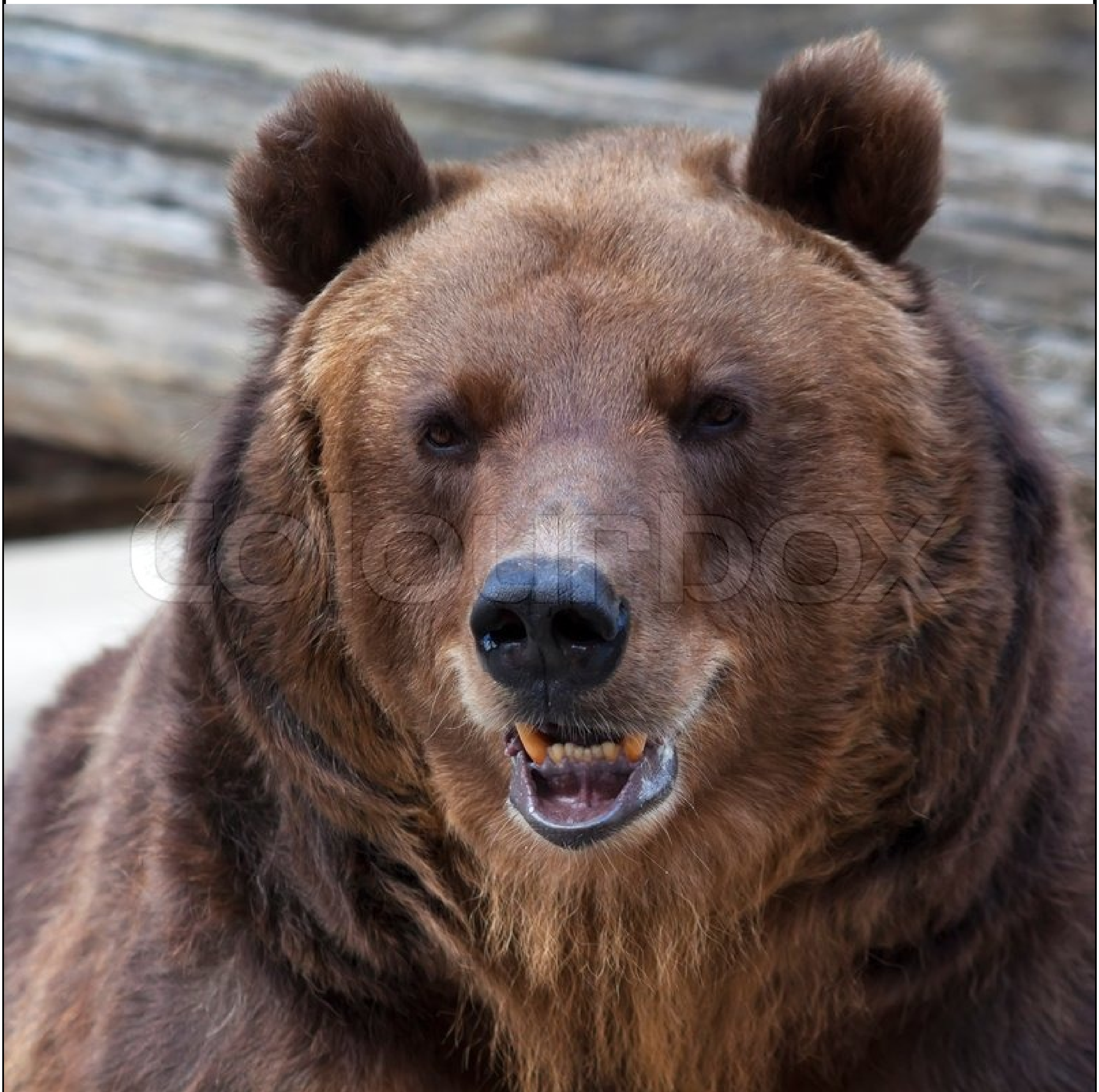
```
                    ImageSegmentsValues(j, i), obj.OrderedDither(Order,:,:), areaDimension, areaDimension);
            end
            %deal with the leftover on the most right column if
            %exist
            if(LeftOver(1) ~= 0)
                AllValuesInCurrentImageSegmentLeftOver = obj.readImgPartValues(ReadyImg, (FullSegments(1)*areaDimension)+1, ((i-1)*areaDimension)+1, LeftOver(1), areaDimension);
                ImageSegmentsValues(FullSegments(1)+1, i) = (Brightnesslevels -1) - round((mean(mean(AllValuesInCurrentImageSegmentLeftOver))+1) / (256/(Brightnesslevels-1)));
                ZerOneMatrix(FullSegments(1)+1,i,:,:) = obj.getzero1matrix(ImageSegmentsValues(FullSegments(1)+1, i), obj.OrderedDither(Order,:,:));
            end
        end
        %deal with the leftover on the bottom row
        if(LeftOver(2) ~= 0)
            for k=1:FullSegments(1)
                AllValuesInCurrentImageSegmentLeftOver = obj.readImgPartValues(ReadyImg, ((k-1)*areaDimension)+1, FullSegments(2)+1, areaDimension, LeftOver(2));
                ImageSegmentsValues(FullSegments(1)+1, i) = (Brightnesslevels -1) - round((mean(mean(AllValuesInCurrentImageSegmentLeftOver))+1) / (256/(Brightnesslevels-1)));
                ZerOneMatrix(k,FullSegments(2)+1,:,:) = obj.getzero1matrix(ImageSegmentsValues(k, FullSegments(2)+1), obj.OrderedDither(Order,:,:));
            end
        end

        %deal with the bottom right corner
        if(LeftOver(1)>0 && LeftOver(2)>0)

        end
        TempMatrix = obj.Get2Dfrom4D(ZerOneMatrix, areaDimension);
        DitheredImg = TempMatrix(1:width, 1:height)*255;
    end

    %this function gets return a matrix from the mean values of a part
    %of the image, the matrix only have 0s and 1s, which then can be
    %used to generate each part of the new dithered image
    function DotZero1Matrix = getzero1matrix(obj, BrigthnessValue, matrix, n, m)
        %deal with missing arguments
        if nargin < 2
            BrigthnessValue = 65;
        end
        if nargin < 3
            matrix = obj.OrderedDither(1,:,:);
        end
        if nargin < 4
            n = 8;
        end
        if nargin < 5
            m = n;
        end

        DotZero1MatrixTemp = ones(size(matrix));
        for i=1:(BrigthnessValue-1)
```

```
                DotZero1MatrixTemp(find(matrix == i, 1)) = 0;
            end
            DotZero1Matrix = DotZero1MatrixTemp(1, 1:n, 1:m);
        end

        %this function return a part of an image
        function values = readImgPartValues(obj, img, x, y, n, m)
            if nargin < 3
               x = 1;
            end
            if nargin < 4
               y = x;
            end
            if nargin < 5
               n = 1;
            end
            if nargin < 6
               m = n;
            end

            [width, height] = size(img);
            NumDotsX = n;
            NumDotsY = m;

            %is the starting point inside the image
            if x<=width && y<=height
                %does the area that we want extend beyond the edges of
                %the images if yes then only read from x,y to the edges
                if (x+n > width)
                    NumDotsX = width - x;
                end
                if (y+n > height)
                    NumDotsY = height - y;
                end
                %return the values of all the pixels that we want to read
                values = img(x:(x+NumDotsX), y:(y+NumDotsY));
            else
                'error: x or y is outside of image'
                values = 0;
            end
        end

        %convert 4d matrix to 2D
        function Matrix2D = Get2Dfrom4D(obj, Matrix4D, n)
            if nargin < 3
                n = 8;
            end
            Dimensions(1:4) = size(Matrix4D);
            width = Dimensions(1)*Dimensions(3);
            height = Dimensions(2)*Dimensions(4);
            Matrix2D = zeros(width, height);
            for i=1:height
               for j=1:width
                   Matrix2D(j,i) = Matrix4D(ceil(j/n), ceil(i/n), max(mod(j, n),
                        1), max(mod(i, n), 1));
               end
            end
        end
    end
end


%                 ZerOneMatrix = zeros(FullSegments(1), FullSegments(2), area,
    area);
```

```
%                   for i=1:FullSegments(2)
%                       for j=1:FullSegments(1)
%                           AllValuesInCurrentImageSegment = obj.readImgPartValues(
    ReadyImg, ((j-1)*area)+1, ((i-1)*area)+1, area);
%
%                           %fill in number of dots that we want to show per
    segment.
%                           %example we want 65 level of brightness, and the mean
    of values was 255 (completely white)
%                           %(65-1) - round((255+1) / (256/(65-1)) =
%                           %64 - round(256 / (256/64)) =
%                           %64 - round(256 / 4) =
%                           %64 - round(64) =
%                           %64 - 64 = 0 (so we are going to fill no dots at all)
%                           ImageSegmentsValues(j, i) = (Brightnesslevels -1) -
    round((mean(mean(AllValuesInCurrentImageSegment))+1) / (256/(
    Brightnesslevels -1)));
% %                           ZerOneMatrix(j,i,:,:) = obj.getzero1matrix(
    ImageSegmentsValues(j, i));
%                           ZerOneMatrix(j,i,:,:) = obj.getzero1matrix(
    ImageSegmentsValues(j, i), obj.OrderedDither(2,:,:));
%                       end
%                   end
%                   DitheredImg = 255*ZerOneMatrix;

ans =

  ImgManager with no properties.




Published with MATLAB R2016a
```

Chapter 4

Results



References

[1] Digital Halftoning Techniques. *University of British Columbia*. http://www.ece.ubc.ca/~irenek/techpaps/introip/manual04.html

[2] C. A. Bouman. *Digital Image Processing,* January 12, 2015. https://engineering.purdue.edu/~bouman/ece637/notes/pdf/Halftoning.pdf

[3] Image Maker. *Dithering.* University of Minnesota, Minneapolis, MN 55455, April 2001. http://www.colorcube.com/illusions/dither.htm

[4] Variations in Dithering Methods. *Leadtools,* 2016. https://www.leadtools.com/help/leadtools/v19/dh/to/leadtools.topics~leadtools.topics.introductioncolorresolutionanddithering.html