Multimedia Software Systems
CS4551

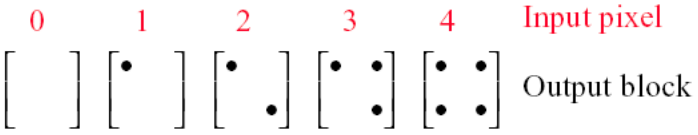
Image Quantization
Dithering and Error Diffusion

Issues

- Suppose that an image uses N intensity levels (or uses N colors) to represent its content.
- Suppose that the rendering device on which this is to be displayed can use only n colors, and $n<N$
- How can we reproduce the image in a satisfactory way?
- Solution
  - Quantization from N to n levels, but as n gets smaller, quantization starts showing artifacts
  - What if $n=2$: use Dithering and Error Diffusion

Dithering

- Assume the original image has only k^2+1 intensity levels.
- Each pixel of the original image is mapped into a k x k block of the dithered image.
- Example: $k=2$



- Problem: we reduce image resolution by $1/k$ (because each pixel is "zoomed" into a k x k block)
  - It's no problem if the display array is larger than the image array

Dithering Matrix

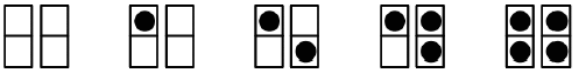
- Dithering is controlled by the dither matrix D.
- For example, with k=3:

$$D = \begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}$$

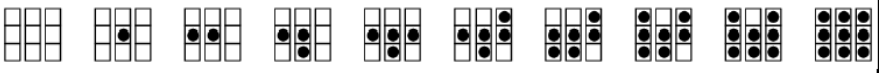
- To display an intensity I (with $0 \le I \le 9$), just "turn on" all pixels in the 3x3 block whose values are less than I
- Must be carefully designed to avoid artifacts
  - What happens with D $\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 2 \\ 5 & 3 & 7 \end{bmatrix}$ and constant I=3

Dithering Matrices

Example: k=2 $\Rightarrow$ D = $\begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$



Example: k=3 $\Rightarrow$ D = $\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}$



Dithering Algorithm

- It is possible to keep the image resolution the same as the original image by ensuring that each image point *(x,y)* should control only the *(x,y)* display pixel
- A possible technique:
  - Compute i=*x* mod *k*, j=*y* mod *k*
  - Intensify the pixel at point *(x,y)* only if *I(x,y)>D(i,j)*, where *D(i,j)* is the entry of D in the i^{th} row and j^{th} column

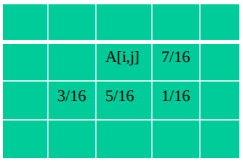
Example - Dithering

Error Diffusion

- When trying to display an image having colors more in number than the display device (or printer device), a selection has to be made to approximate the value of the display color, which is done using
  - Precomputed Lookup Tables (LUT)
  - Dynamic Color Quantization
- Either way the difference in the selector color value and the original value results in an error
- Large color errors degrade the quality of the displayed image. If the error is diffused in the neighborhood, such effects are minimized.
- Error Diffusion Algorithms do this by distributing the errors among all the pixels such that the total error for the entire image is zero (or very close to zero).

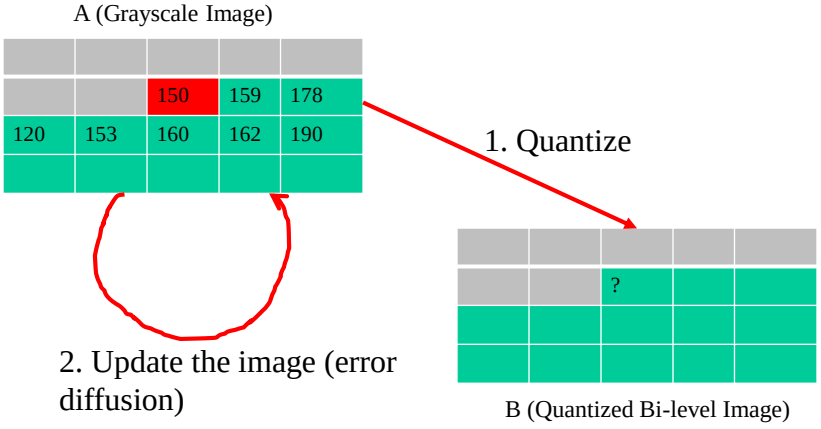
Error Diffusion Algorithm

- ChoosLet A be the original image and B be the final quantized image
- For each pixel A[i,j],
  - **Quantization**
    - e the value Q (0 or 255 if B is a bi-level image) that is nearest to the original pixel's value A[i,j].
    - Assign Q to B[i,j] (**Quantization step**)
  - **Error diffusion**
    - Calculate the quantization error ***e*=A[i,j]-B[i,j]** for the pixel A[i,j]. Error *e* can be negative.
    - Distribute this error *e* to **four** of A[i,j]'s nearest neighbors that haven't been scanned yet (the one on the right and the three ones centered below) according to a filter weight (eg –Floyd-Steinberg)

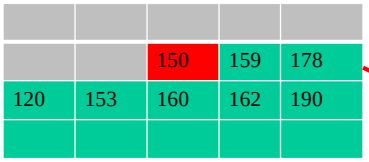


Eg –Floyd-Steinberg Weights
the neighbor A[i+1,j] will be updated as A[i+1,j] = A[i+1,j] + *e**7/16

** Updated neighbor pixel values can be negative.*

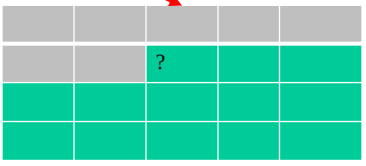


A (Grayscale Image)



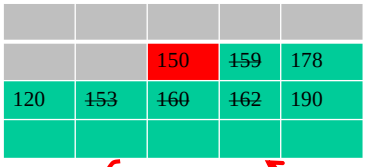
1. Quantize

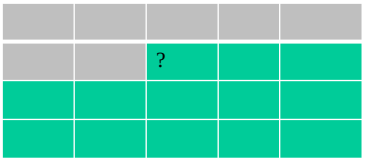
150 > 256/2 then 255 else 0



B (Quantized Bi-level Image)

A (Grayscale Image)





B (Quantized Bi-level Image)

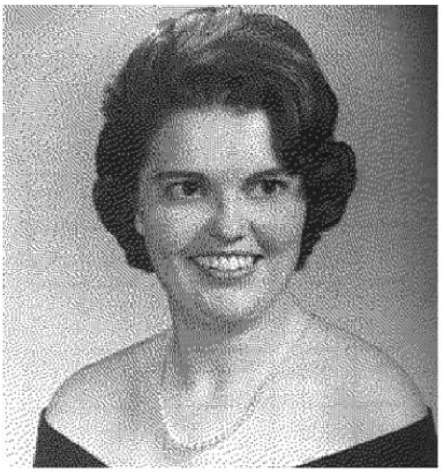
2. Update the image (error diffusion):

Calculate e=A[i,j]-B[i,j]

and distribute e to 4 neighbors

using Floyd-Steinberg weights.

Example – Error Diffusion



Example – Error Diffusion