CS 465 Homework 1a

out: Wednesday 3 September 2003
due: Monday 8 September 2003

Problem 1: Image formats

Assume we have stored in memory a representation of a grayscale image I as an 1280x1024 8-bit raster image I_8. In your answers, round memory figures to the nearest tenth of a megabyte (where 1MB $= 1024^2$ bytes).

1. How much memory is occupied by I_8?
2. Suppose we convert the image to a 4-bit image I_4 as follows:

   $$I_4[i,j] = I_8[i,j]/16$$

   (using integer division, of course). How does I_4 look different from I_8? How much memory does it occupy? What technique might we use to improve the appearance of I_4?
3. Suppose we convert the image to a 16-bit image I_{16} as follows:

   $$I_{16}[i,j] = 256 * I_8[i,j].$$

   How much memory does I_{16} occupy? What is the maximum pixel value that will result from this conversion? Give an alternate expression (still using integer arithmetic) for converting the pixels that will map full white to full white.

Assume that for images of any precision the maximum representable pixel value always maps to the brightest available white on the display. For instance, in a 4 bit image, pixel value 15 is full white, whereas in a 16 bit image, pixel value 65535 is full white.

Problem 2: Gamma correction

In class we discussed *gamma correction*, the practice of encoding an image using nonuniform quantization that compensates for the nonuniform response of the display on which it will be shown. A reasonable model is that when you hand the display a pixel value $a \in [0, 1]$ it will produce a light intensity

$$L = L_M a^\gamma$$

where L_M is the maximum available intensity and γ is a number that depends on the monitor and display electronics. A typical system these days has γ around 2, so let's assume that nice round number. Speaking antropomorphically, this means that the display takes our value and squares it to determine the light intensity it will actually produce.

Assume that we have an image $I(x, y)$ that we want to show on the screen, which we know exactly. In order to find the appropriate image $a(x, y)$ to give to the display, we need to compensate for the squaring that the monitor does by taking the square root of each pixel value in the image. This is known as *gamma correction*.

This question is about when one should apply gamma correction. Between when wee compute or measure the exact image $I(x, y)$ and when it is sent to the display, assume we will store it in an 8-bit grayscale image. This means we are *quantizing* the pixel values: rounding them to the nearest representable gray value. Let's compare two approaches:

(a) Quantize, then gamma correct. In this case we will store an 8-bit approximation to $I(x, y)$, and then gamma correct it when we later send the image to the display. We will store the integer value

$$n_1(x, y) = round(255 * I(x, y))$$

and then hand to the display the value[1]

$$a(x, y) = \sqrt{n_1(x, y)/255}.$$

(b) Gamma correct, then quantize. In this case we will store an 8-bit approximation to $\sqrt{I(x, y)}$, and do nothing special when we send the image to the display. We will store the integer value

$$n_2(x, y) = round(255 * \sqrt{I(x, y)})$$

and then hand to the display the value

$$a(x, y) = n_2(x, y)/255.$$

[1] Do not assume any more quantization happens—this is the exact value that goes to the display.

Each of these schemes has the right overall idea: the display gets an approximation of the square root of the original image, so that when it emits the square of that much light intensity we get the right answer. The difference is in how the gray levels are quantized: each scheme has 256 different gray levels, but they are not the same intensities.

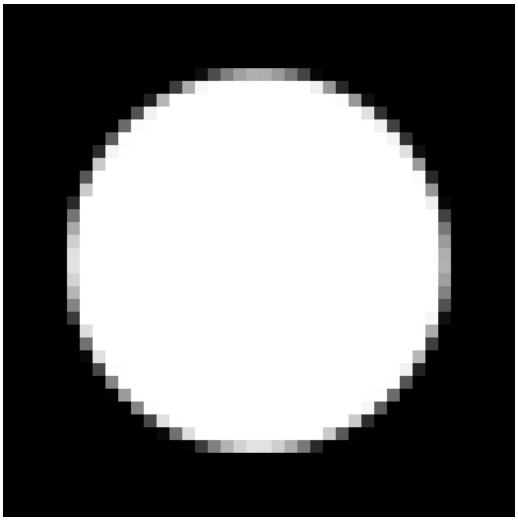
1. What are the first five and last five representable pixel intensities in each scheme? Express them as percentages of L_M with 2 digits of precision.

2. In each scheme, what pixel values straddle the intensities 1% and 99%? Which scheme can distinguish smaller variations in dark shadows? In bright highlights? In which case can the human visual system make use of more precision and why?

3. Assume we use 8-bit pixels in scheme (b). How many bits of precision in scheme (a) are required to match the performance of scheme (b) for pixels at 1% intensity?

Problem 3: Compositing

We have an image A with an 8-bit alpha channel that consists of a circle with soft edges, as shown here:



The image is solid white—so with premultiplied alpha the color channels are equal to the alpha channel.

Suppose we composite the image over and over, starting with a solid black image. That is, we start with the image I populated with zeroes in the color channels and ones in the alpha channel and repeatedly set $I = A$ **over** I.

1. What will be the pixel values after 1, 2, 3, 4, and 5 compositing operations for pixels where the alpha channel has value 0, 15, 128, 240, and 255? (Your answer is a 5 by 5 grid of numbers.)

2. After a large number of compositing operations, how will the image look different than it does after one?

3. If the alpha channel shown above is meant to approximate a sharp-edged opaque circle, is this behavior appropriate or is it an artifact?