1996-11-25 Bo Nordin (bosse@cb.uu.se)

Lecture notes, computer graphics (F12) (Foley 11)

Issues

What is light ??
How do we represent colour in the computer ??
How do we display it on screens and hardcopy devices ??

Achromatic light (11.1.1)

Achromatic light (literally absence of colour):
intensity, luminance - physics, sense of energy
brightness - phychological, perceived energy.

Selection of intensities for display
we might distribute the intensities evenly
the eye is sensitive to ratios of intensity levels, rather than to absolute values
i.e. on a scale from 0-1, we perceive the diff 0.11-0.10 as equal to 0.77-0.70
Intensity levels should be spaced logarithmically rather than linearly.
$I_0 = I_0$, $I_1 = rI_0$, $I_{255}=r^{255}I_0 = 1.0$ and $r= (1/I_0)^{1/255}$, $I_j = r^jI_0 = (1/I_0)^{j/255}I_0 = I_0^{(255-j)/255}$

$r = (1/I_0)^{1/n}$, $I_j = I_0^{(n-j)/n}$
The minimum attainable intensity for a CRT is from 1/200 to 1/40 of the maximum.
So I_0 is between 0.005 and 0.025
The ratio is called the *dynamic range*
r = 1.01546 if $I_0 = 0.02$ and (# of levels-1) is 255
gamma correction:a lookup-table (because the light output produced by the screen
phosphor isn't linearly proportional to the pixel's given intensity: brightness = kV^{χ}
$V_i = (I_i / k)/^{1/\chi}$
How many intensities are enough?? (to make the reproduction appear continuous)
ok if the ratio r <= 1.01
if r <= 1.01 ok, i.e. $n = \log_{1.01}(1/I_0)$ (table pg 397)
In practise, look at pg 398-399. (32 levels OK)
I tested this morning on a ramp image: 64 levels ok

Halftoning (11.1.2)

How do we represent a grey-scale image on a binary device
Representation of grey-scale on bi-level devices.
Same technique can be used to represent many colours by a few
uses the *spatial integration* that our eye performs.
Simple: threshold (2 levels) if pix[x,y] < T then black; else white
Thresholding may be dynamic (from a histogram, DRAW!) or fixed
halftoning: each small resolution unit is imprinted with a circle of black ink whose area is proportinal to the blackness (draw) (fig. 11.3, pg 399)
How do we achieve this on a raster? approximate the dot size!! (DRAW!)

We can approximate this, for instance a 2x2 pixel area of a bilevel device can be used to produce five different intensity levels at the cost of halving the spatial resolution.along each axis. An nxn group of bilevel pixels can provide n^2+1 levels. There is tradeoff between spatial resolution and intensity resolution.

Can be generalized to any device with fewer levels than we want to display.

(ex. 8 grey levels 2*2*(8-1)+1)
General: #pixels in the matrix * (#greylevels-1) + 1

Problem with pixel-grid patterns: artificical patterns in areas with constant grey scale. - this is especially bad because our brain is sensitive to edges. (fig. 11.5, pg 400)

Thus: how do we select which dots to set?
Placement of new dots: maximum distance between the dot added last and the new dot matrices may be recursively defined. Avoid symmetry

But what if the image array and the display device array are equally large??
Dithering
clustered-dot ordered dithering (matrices constructed as before)
Dithering matrix for 4x4

```
 1  9  3 11
13  5 15  7
 4 12  2 10
16  8 14  6
```

$i = (x \bmod n) + 1$
$j = (y \bmod n) + 1$
if $(i > D_n(i,j)$ turn the pixel on

Dithering can be described as adding random noise to an image in order to get small variations and then thresholding the image.

error diffusion algorithm:
The error (the difference between the exact value and the value displayed) is added to the values for the 4 image-array pixels to the right and below the pixel in question

```
        O   7/16
3/16 5/16 1/16
```

Algorithm: allocate two pixel vectors err1 (*p1) and err2 (*p2) (set both to zero initially)
Length of err1 and err2 is xsize (+2 to accomodate last and first pixel on line)

```
for (j=0; j < ysize; j++) {
  err = 0;
  swap(err1,err2); p1 = err1+1; p2 = err2+1;
  clear( err1); /* next line */
  for (i=0; i < xsize; i++)  {
     if ((v =  image[j][i] + (err + p2[i])/16) > 128 {setpix[j][i]; err = v -255; }
     else err = v;;
     p1[i-1] += 3*err: p1[i] = 5*err; p1[i+1]=err: err = 7*err;
  }
}
```

Better result if alternate scan left->right and right->left
Colour theory (11.2)

Colour important for realism!
Physical, Physiological, Psychological, Artistic, Technical
The colour is not only the colour of the object, but also of the light source illuminating the objects, the colour of the surrounding area, and on our visual system. And certain objects reflect light, whereas others also transmit light.

Hue - red, green, etc.
Saturation - distance to a gray of equal intensity
Lightness - perceived intensity (brightness, perceived intensity of self-luminous objects)

Questions: How are colours described accurately in numerical terms?
How do these descriptions relate to everyday ways of describing colours?
How does one compare colours?
What range of colour can a CRT display?
How do we display on a CRT. How do we load colour tables?

We want an objective way of defining colour: colorimetry
dominant wavelength: wavelength of the colour we see (cf. hue)
luminance(L): amount or intensity of light (a pedestal of white light + red) (area!!)

excitation purity: saturation (completely pure colour is 100-percent saturated no white light)
100 * (D-A)B/L
Pastel colours contain a large amount of white and are unsaturated
We can distinguish about 128 different hues, 20 different saturation levels
Chromaticity describes dominant frequency and purity

Light is electromagnetic energy in the visible spectrum:
400 (violet)- 700 (red) nm wavelength.
Pure colours: violet at 400, blue 480, green 540, yellow 570, orange 590, red 620
Perceived light:
white -- all wavelengths , black -- no wavelengths

tristimulus theory: the retina has 3 kinds of sensors: peak sensitivity to red, green, and blue.
the eye's response to blue light is less strong than its response to red or green

DRAW figure, page 404

This gives some credibility to the idea that colours may be specified by a weighted sum
of red, green, and blue.
C = rR + gG + bB (true of any choice of primaries as long as thay are linearly independent)
(fig 11.11 shows that we need negative values to obtain som of 'em
All physically realizable primaries require using negative coefficients for some visible cols

CIE: X, Y, and Z primary (fig 11.12, pg 406) The point is to remove negative weights
useful to define colour gamuts (i.e. the realizable colours on a device)
Complementary colours: can be mixed to produce white light

RGB: additive primaries add from black
A cube

CMY:
cyan, magenta, yellow (subtractive primaries) subtract from white
often used by hardcopy devices

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} c \\ m \\ y \end{bmatrix}$$

CMYK: black added K = min(C,M,Y) C = C - K, M = M - K, Y = Y - K
use in ink jet printers, colour printers
yellow absorbs (subtracts) blue, cyan absorbs red, and magenta absorbs green
i.e. adding yellow and blue -> white

YIQ: US TV

HSV: not hardware-oriented (cylindrical coordinate system)

Loosely corresponds to the tint, shade (V), and tone of the artist.
S = (max(r,g,b) - min(r,g,b)) / max(r,g,b) r==g==b==0: s = 0 greyscale one 0, one1 S=1
V = max(r,g,b)

Hue is an angle (draw) red zero, green 120, blue 240 (diff = max-min
if (r == max) hue = (g-b)/diff
if (g == max) hue = 2.0 + (b-r)/diff
if (b == max) hue = 4.0 + (r-g)/diff
hue *= 60

RGB to HSV and HSV to RGB (pg. 415)

A straight line in RGB does not map into a straight line in HSV (generally), this means that when interpolating colours the choice of colour system is essential.

The use of colour in computer graphics (previously discussed).

Colour quantization and mapping
We want to show an image on a screen:
grey-level iamge -> greylevel (or pseudo colouring)
colour image -> colours (or gray by mixing 0.35, 0.58, 0.07)

24-bit colour screen: just send r,g,b data to the screen
8-bit colour screen: 8 bit image memory (can address 256 of 2^{24} simultaneous colours)
The problem with pseudo-display workstations
Draw a LUT

Colour quantization
Uniform: 3-bit for red, 3-bit for green, and 2-bit for blue. - contouring
(draw and explain!!)
quick but dirty, used to preview images

a better strategy is to analyse the image and choose colours from that (slow, but for natural images subsampling is possible):
1/ scan and analyze data 2/ build LUT, map from 3-tuple to LUT 3/ map data 4/ write

popularity: find the 256 most popular colours (often after requantization to fewer levels)
calculate a histogram, other colours are allocated to one of the 256 selected (nearest)
a problem is that the histogram becomes large: 256*256*256
also hard to find nearest representatives
will "remove" small objects

median cut:subdivide the RGB cube to ensure that each LUT entry represemts the same # of pixels always done by planes parallel to the axes (not optimal!)
start by cutting the R axis in two, then the G axis
Optimally, the first cut should be normal to the axis where the colours are spread the most
All pixels in a volume get the value of the mean pixel
Mapping is then done by comparing each of the pixel primary components with the bounding planes of the box.

diversity (used in xv):
works for gif images, etc. with 256 colours

0/ make a histogran
1/ pick the most used colour (throw away zero)
2/ find the one with the greatest distance from the 1st (use Manhattan distance)
3/ for each remaining colour find and selects the furthest until we run out of colour cells.
too much emphasis on finding different colours, too little on finding correct. pick every 10th as the most popular one

Loading the LUT

How do we load the LUT?
Assume first that there are 256 levels per colour
Assume first we use 3 (8 values), 3(8 values), and 2(4 values) bits for r,g,b:

```
int LUT[256], red[8], green[8], blue[4], r, g, b;
for (r = 0; r < 8; r++) red[r] = (256/8)*r;
for (g =0; g < 8; g++) green[g] = (256/8)*g;
for (b=0; b < 4; b++) blue[b] = (256/4)*b;
for (r=0; r < 8; r++)
  for (g=0; g < 8; g++)
    for (b=0; b < 4; b++)
      LUT[32*r + 4*g + b] = 65536 * (red[r] + 16) + 256 * (green[g] + 16) + blue[g] + 32
```

Mapping (i.e. finding in index in the LUT):
index = R/(256/8)*32 + G/(256/8)*4 + B/(256/4)
This can be done using a lookup table

Colour Dithering

Hard, no general theory exists. research area,