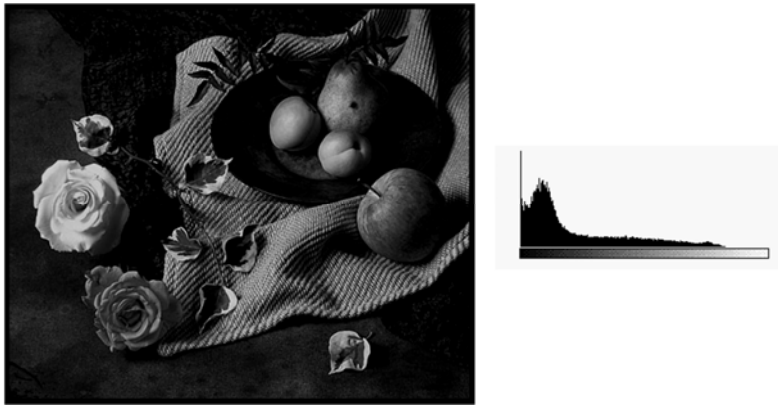
Histograms

- A histogram is a graph representing number of pixels at each intensity level



Histograms

- We can make non-uniformly brighter using gamma correction, or uniformly brighter by adding constant intensity to each pixel

Histogram Code (Example)

```
int[][] histogram;

histogram=new int[256][3];

for (j=0; j<h; j++)
   for (i=0; i<w; i++)
      for (c=0; c<3; c++)
         histogram[data[c+3*i+3*j*w]&255][c]++;

for (i=0; i<256; i++)
    System.out.println(i+" r="+histogram[i][2]+
                       " g="+histogram[i][1]+
                       " b="+histogram[i][0]);
```

pixel value

Histograms

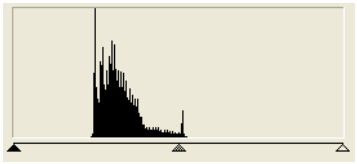
Typical Output

233 r=790 g=491 b=566
234 r=783 g=672 b=1214
235 r=704 g=796 b=1308
236 r=577 g=970 b=227
237 r=542 g=982 b=1278
238 r=614 g=1045 b=190
239 r=614 g=1093 b=1134
240 r=572 g=1065 b=240
241 r=700 g=844 b=828

- e.g. 790 pixels have a red value of 233, 491 pixels have a green value of 233, and 566 pixels have a blue value of 233

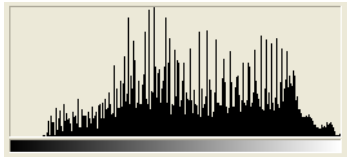
Histograms

- Reduced contrast in Photoshop
- Displayed histogram (combined RGB into grey level image)



Histograms

- Used Histogram Equalization
- Note how the histogram is more spread out



Histogram Equalization

- Contrast can be improved by spreading distribution of intensity evenly over the pixels
- Total number of pixels in image is height*width (h*w)
- Number of grey levels is g_levels
- Ideal histogram is flat
- Ideal number of pixels at each level:
- p_num=(h*w)/g_levels

Histogram Equalization (code)

For grey level images only:

```
int[] histogram, mapping;
int red, green, blue, grey, t_i=0;

int g_levels=256;

histogram=new int[g_levels];
mapping=new int[g_levels];
```

Histogram Equalization

- Mapping F is defined as:
- *If t(i) is the actual number of pixels at all old grey levels up to and including grey level i, then*

F(i)=max{0,round((g_levels*t(i))/(h*w))-1}

t(i) is known as the cumulative distribution function

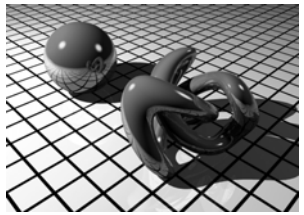
Histogram Equalization

- In code (after making image grey):

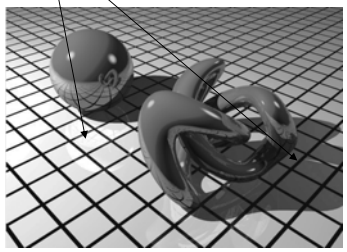
```
for (i=0; i<g_levels; i++) {
  t_i+=histogram[i];
  mapping[i]=
max(0,Math.round((g_levels*t_i)/(h*w))-1);
}
```

Histogram Equalization

110 115
111 116
112 116
113 117
114 117
115 118
116 119
117 119
118 120
119 121
120 121
121 122
122 122
123 123
124 124
125 124

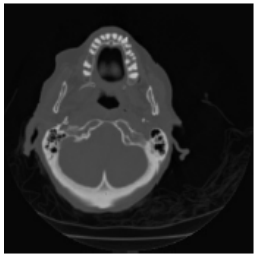


- e.g. old grey pixels of value 112 map to new grey pixel of value 116
- see contrast in shadows and reflections



Histogram Equalization

- Lowering the dynamic range in images
- e.g. 12 bit CT scanner – display on 8 bits?
- naïve – remove lowest 4 bits



- Better – use histogram equalization with g_levels=max_intensity-min_intensity=3365 (for this CT head)

Colour Histogram Equalization

- Convert RGB to HSV
- Build histogram on V (brightness)
- Equalize V
- Convert HSV back to RGB
- Display

Histogram Equalization (on the CThead data set)

Definitions of terms:
Data set – this is the set of values input to the process. It could be the input image, or the input CThead data set.
Element – this is a single element from the set. It could be a pixel in an image or a voxel in the CThead data set.
Index – this is a position in an array. Array positions (index) can only be of type integer and must be positive.
Size – this is the count of the input elements. e.g. a 256x256x113 data set has a size of 7405568.

Stage 1 Create the histogram
Find the minimum value in the source data set and assign this to min.
Find the maximum value in the source data set and assign this to max.
There will be max-min+1 values in the histogram (and the other arrays we will create in the next stages).
e.g. for an input image, usually the min will be 0 and the max 255 giving 256 values in the histogram. For the CThead, min=-1117 and max=2248, so there will be 3366 values in the histogram.

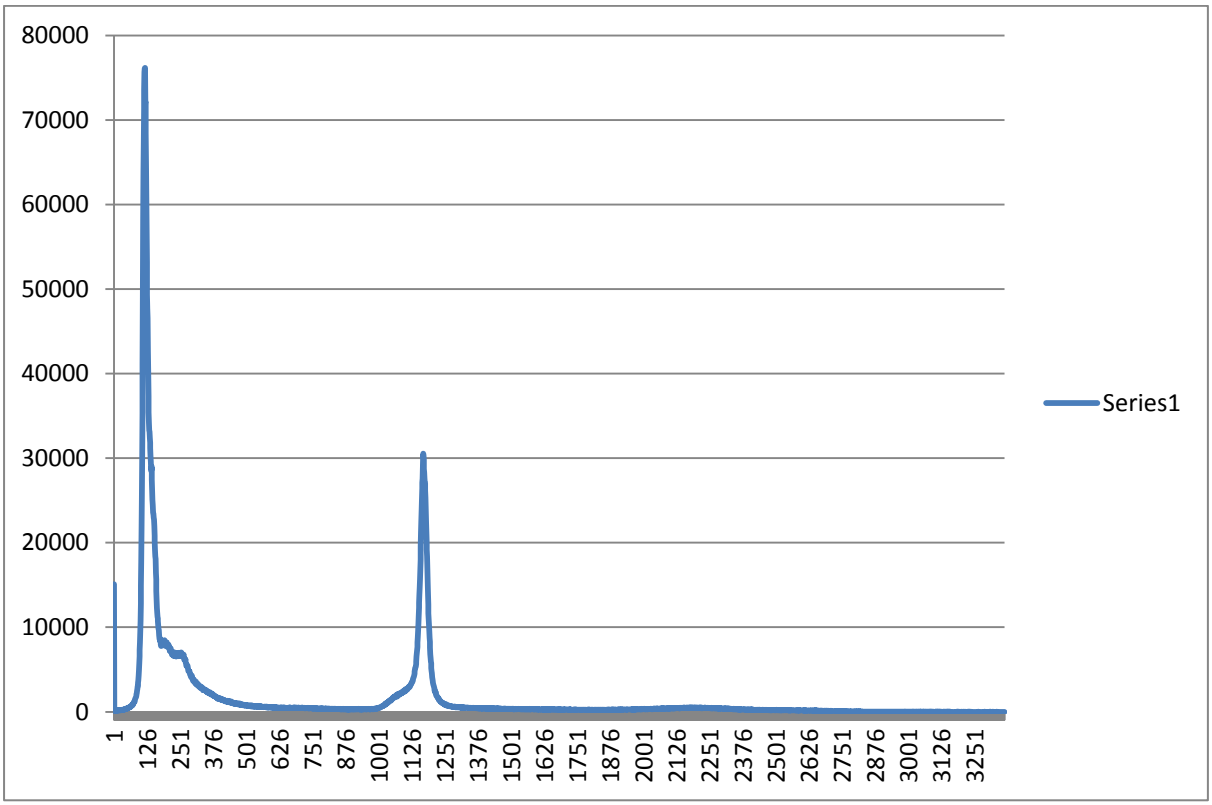
The histogram is an array that counts how many elements of the data set have that value.
So histogram[p]=n indicates n elements have the value p, where p takes values from min to max.
Because you can't access an array with a negative index, for this process, when accessing the data set use
index=cthead[z][y][x]-min;

This will give index values between 0 and 3365.

To create a histogram, loop through every element of the data set, obtaining index, and doing
histogram[index]++;
(make sure you have previously initialised histogram[index] to zero for all index.)

Here is an example output. This is the histogram from the cthead (with one extremely large value probably due to the ctscanner surround reduced to 72k so the rest of the graph is easier to see).

We see two peaks – one is the soft tissue, the other is the denser bone.



Stage 2 Create the cumulative distribution function t

t[i] is the number of elements in the data set that have values up to and including i.

So

t[0]=histogram[0]

t[1]=histogram[0]+histogram[1]=t[0]+histogram[1]

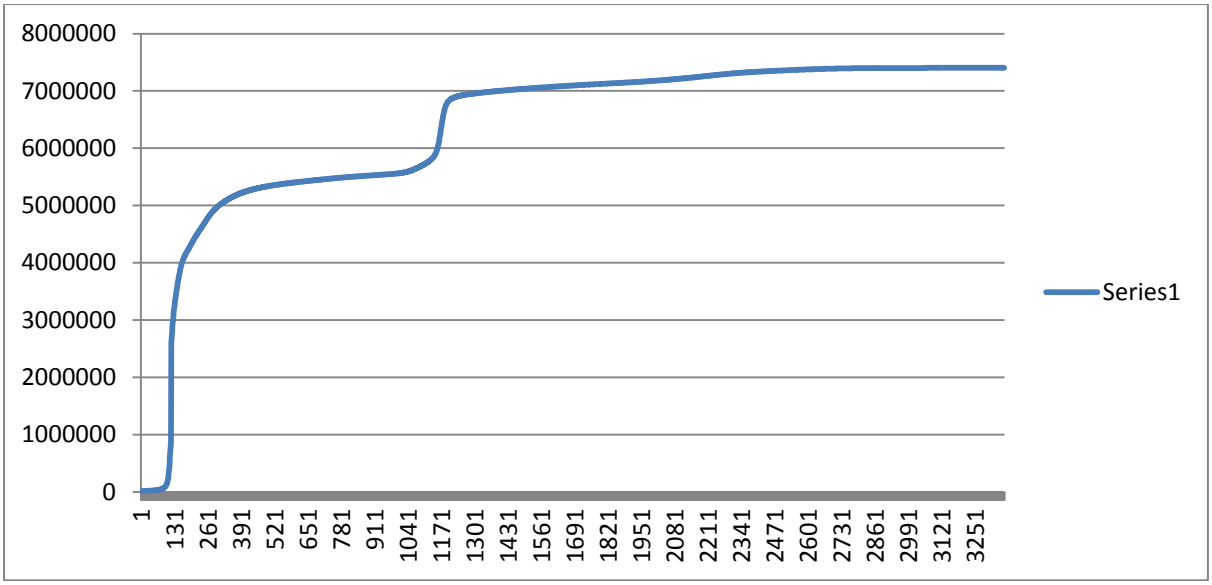
t[2]=histogram[0]+histogram[1]+histogram[2]=t[1]+histogram[2]

In general

t[n]=t[n-1]+histogram[n]

i.e. t can be calculated using a single loop.

The next graph is a plot of the t[i] function from the CThead. You can see that as we encounter a peak, the cumulative function jumps up. As we encounter lots of lower values, the function gets steadily larger. For those who have done AS/A maths, this function at i is the integral of the histogram from 0 to i.



Here are the last few values printed to the screen:
7403073
7403075
7403077
7403079
7403079
7403080
7403081
7403082
7403084
7403084
7405568

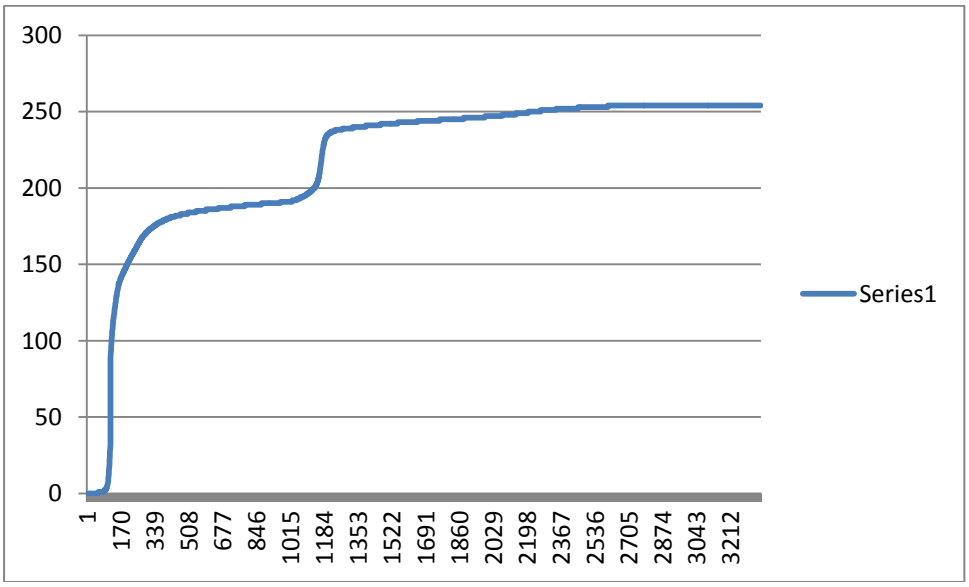
Note that 256x256x113=7405568. The last number should be the total number of elements in the data set as all elements in the data set have values up to and including the maximum value.

Stage 3 Create the mapping
We need to map the elements of the data set to a new range $[i_1, i_2]$:
mapping[i]=$((i_2-i_1)*(t[i]/Size))+i_1$
Where we want to map **to** an image with $i_1=0$ and $i_2=255$, this becomes:
mapping[i]=255.0*(t[i]/Size)

Stage 4 Create the image
Loop through the image pixels as usual. Get the value from the cthead data
datum=cthead[k][j][i]
and find its mapping to colour:

col=mapping[datum-min]
(the –min because we did the same in stage 1).
And assign col to the rgb values of the pixel as usual.

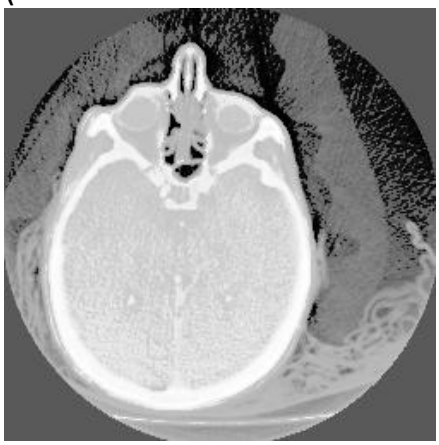
Here is the plot of the mapping function. You can see that given an input data value, i, mapping[i] gives a colour intensity between 0 and 255. So instead of using the normalisation code:
col=(255.0f*((float)datum-(float)min)/((float)(max-min)));
we use the line above:
col=mapping[datum-min];
Advanced GUI Hint: You can hold several mapping functions in memory at the same time, and switch between them.



Stages 2 and 3 combined
Since mapping[i] only requires t[i] **and** t[i] only requires t[i-1] and histogram[i], it is possible to combine stages 2 and 3 as:

```
for (i=0; i<max-min+1; i++) {
        t_i=t_i+histogram[i];
        mapping[i]=255.0f*(t[i]/Size);
}
```

(this is modified from the slides – see definitions on first page for Size)



Here is an example image.