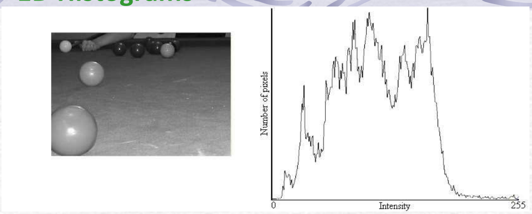
Histograms

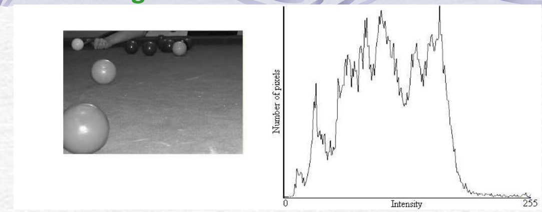
- ➢1D Histograms
- ➢3D Histograms
- ➢Equalisation
- ➢Histogram Comparison
- ➢Back Projection

1D Histograms



- Determine the frequency of brightness values
  - Initialise:
    - h(z) = 0 for all values of z
  - Compute:
    - For all pixels (i,j): h(f(i,j))++

1D Histograms



- Is this useful?
  - Global information
  - Useful for classification ?
  - Not unique

1D Histograms

```
Mat  display_image;
MatND  gray_histogram;
const int* channel_numbers = { 0 };
float channel_range[] = { 0.0, 255.0 };
const float* channel_ranges = channel_range;
int number_bins = 64;
calcHist( &gray_image, 1, channel_numbers, Mat(),
        gray_histogram, 1, &number_bins, &channel_ranges );
OneDHistogram::Draw1DHistogram( &gray_histogram, 1,
                                                display_image );
imshow("Greyscale histogram", display_image);
```

1D Histograms – Smoothing

- Local minima and maxima are useful.
- But how can we deal with noise though?

10	20	30	25	20	40	50	40	20	20
?	20	25	25	28	37	43	37	27	?

- Smooth the histogram…
  - For all values v: $h_{new}(v) = (h(v-1) + h(v) + h(v+1)) / 3$
- What do we do at the ends?
  - Do not compute OR Wraparound OR Duplicate values OR Reflect values OR Constant values ??

1D Histograms – Smoothing

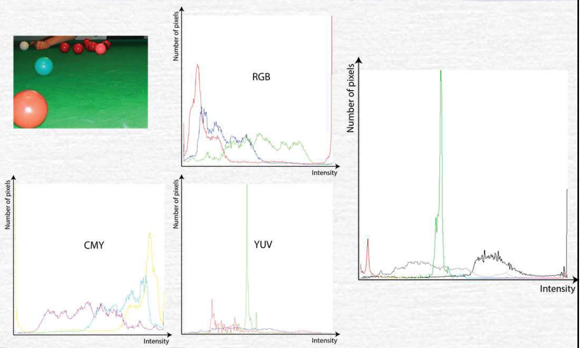
10	20	30	25	20	40	50	40	20	20
?	20	25	25	28	37	43	37	27	?

```
MatND smoothed_histogram = histogram[channel].clone();
for(int i = 1; i < histogram[channel].rows - 1; ++i)
  smoothed_histogram[channel].at<float>(i) =
    (histogram.at<float>(i-1) + histogram.at<float>(i) +
               histogram.at<float>(i+1)) / 3;
```


1D Histograms – Colour Histograms

- Determine histograms for each channel independently…
- Choice of colour space…

1D Histograms – Colour Histograms

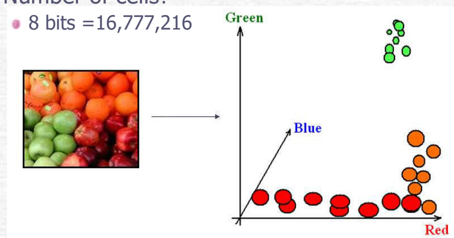


1D Histograms – Colour Histograms

```
MatND* histogram = new MatND[image.channels() ];
vector<Mat> channels(image.channels() );
split( image, channels );
const int* channel_numbers = { 0 };
float channel_range[] = { 0.0, 255.0 };
const float* channel_ranges = channel_range;
int number_bins = 64;
for (int chan=0; chan < image.channels(); chan++)
        calcHist( &(channels[chan]), 1, channel_numbers, Mat(),
           histogram[chan], 1, &number_bins, &channel_ranges );
OneDHistogram::Draw1DHistogram(histogram,
                                        image.channels(), display_image );
```

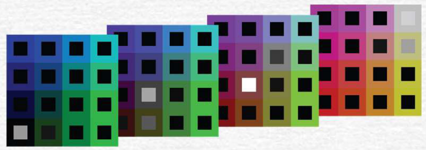

3D Histograms

- Channels are not independent
- Better discrimination comes from considering all channels simultaneously
- Number of cells?
  - 8 bits =16,777,216



3D Histograms

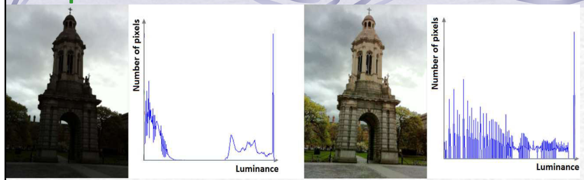
- Reduce quantisation
  - 6 bits = 262,144
  - 4 bits = 4,096
  - 2 bits = 64



3D Histograms

```
MatND histogram;
int channel_numbers[] = { 0, 1, 2 };
int* number_bins = new int[image.channels()];
for (ch=0; ch < image.channels(); ch++)
        number_bins[ch] = 64;
float ch_range[] = { 0.0, 255.0 };
const float* channel_ranges[] = {ch_range, ch_range, ch_range};
calcHist( &image, 1, channel_numbers, Mat(), histogram,
            image.channels(), a_number_bins, channel_ranges );
```

Equalisation

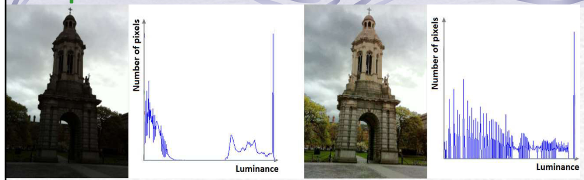


- If an image has insufficient contrast
- Human can distinguish 700-900 greyscales
- Evenly distribute the greyscales…
  - Result has missing greyscales
- Normally equalise only the greyscales / luminance

Equalisation

```
// Create a lookup table to map the luminances
// h[x] = histogram of luminance values image f(i,j).
pixels_so_far = 0
num_pixels = image.rows * image.cols
output = 0
for  input = 0 to 255
  pixels_so_far = pixels_so_far + h[ input ]
  new_output = (pixels_so_far*256) / (num_pixels+1)
  LUT[ input ] = (output+1+new_output) / 2
  output = new_output
// Apply the lookup table LUT(x) to the image:
for  every pixel f(i,j)
  f'(i,j) = LUT[ f(i,j) ]
```


Equalisation




```
vector<Mat> channels( hls_image.channels() );
split(hls_image, channels);
equalizeHist( channels[1], channels[1] );
merge( channels, hls_image );
```


Histogram Comparison

- To find similar images…
  - Use metadata
  - Compare images
- Compare the colour distributions
- Need a metric for comparisons…

Histogram Comparison

Histogram Comparison – Metrics

- $D_{Correlation}(h_1, h_2) = \frac{\sum_i (h_1(i) - \overline{h_1})(h_2(i) - \overline{h_2})}{\sqrt{\sum_i (h_1(i) - \overline{h_1})^2 \sum_i (h_2(i) - \overline{h_2})^2}}$
- $D_{Chi-Square}(h_1, h_2) = \sum_i \frac{(h_1(i) - h_2(i))^2}{(h_1(i) + h_2(i))}$
- $D_{Intersection}(h_1, h_2) = \sum_i min(h_1(i), h_2(i))$
- $D_{Bhattacharyya}(h_1, h_2) = \sqrt{1 - \frac{1}{\sqrt{\overline{h_1}.\overline{h_2}.N^2}} \sum_i \sqrt{h_1(i).h_2(i)}}$
- where
  - N is the number of bins in the histograms,
  - $\overline{h_k} = {\sum_i (h_k(i))}/{N}$

Histogram Comparison – Earth Mover

- An alternative to the metrics is to compute the Earth Mover's Distance…
  - Minimum cost for turning a distribution into another distribution
  - Compare images
- 1D solution:
  - EMD(-1) = 0
  - EMD(i) = h_1(i) + EMD(i-1) - h_2(i)
  - Earth Mover's Distance = Σ_i | EMD(i) |
- Colour EMD is harder to compute…

Histogram Comparison

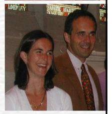
```
normalize( histogram1, histogram1, 1.0);
normalize( histogram2, histogram2, 1.0);
double matching_score = compareHist( histogram1,
                histogram2, CV_COMP_CORREL);
```

We can also use Chi-Square (CV_COMP_CHISQR), Intersection (CV_COMP_INTERSECT) or Bhattacharyya (CV_COMP_BHATTACHARYYA) metrics or alternatively can use the Earth Mover's Distance function (EMD())

Histogram Back Projection

- A better approach to selecting colours (based on samples):

1. Obtain a representative sample set of the colours.
2. Histogram those samples.
3. Normalize that histogram so that the maximum value is 1.0.
4. Back project the normalized histogram onto any image f(i,j).
5. This will provide a 'probability' image p(i,j) which indicates the similarity between f(i,j) and the sample set.



Histogram Back Projection

```
calcHist( &hls_samples_image, 1, channel_numbers, Mat(),
  histogram,image.channels(),number_bins,channel_ranges);
normalize( histogram, histogram, 1.0);
Mat probabilities = histogram.BackProject( hls_image );
```