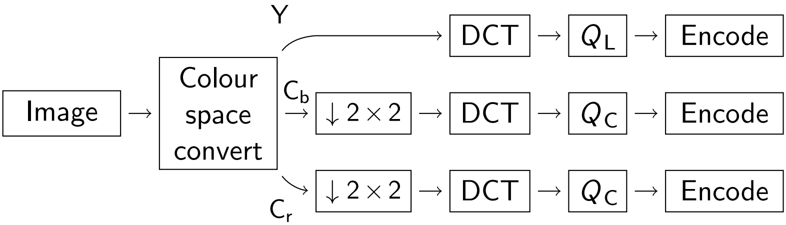
JPEG tutorial

Andrew B. Lewis

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

The JPEG algorithm



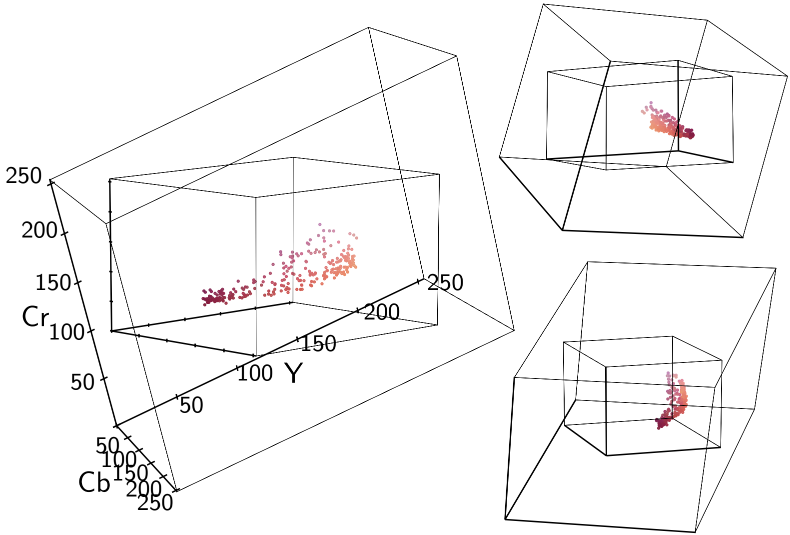
Colour space conversion

A YC_bC_r representation $\mathbf{v}$ of an RGB image $\mathbf{u}$ ($w \times h$ rows, 3 columns) is given by the per-pixel calculation

$$v_i^{\mathsf{T}} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} u_i^{\mathsf{T}} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}.$$

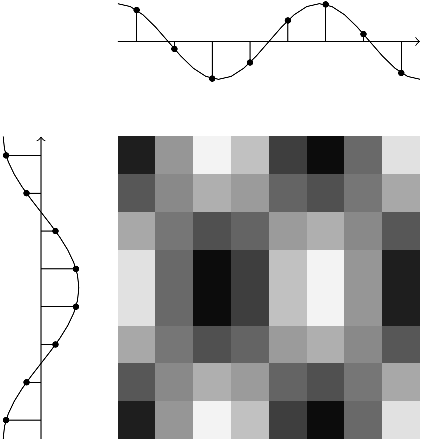
RGB to YC_bC_r conversion as a coordinate transform

Samples are taken from a 16×16 neighbourhood in the ‘lena’ image.



Discrete cosine transform

The 2-D DCT is a linear, separable transform which represents a block of sample values as the weighting factors of sampled cosine functions at various frequencies.



Discrete cosine transform

The forward transform of a block $\mathbf{x}_b$ is given by

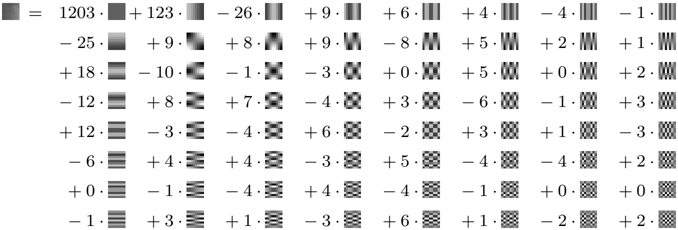
$$(\mathbf{X}_b)_{u,v} = \frac{C(u)}{\sqrt{N/2}} \frac{C(v)}{\sqrt{N/2}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\mathbf{x}_b)_{i,j} \cos \frac{(2i+1)u\pi}{2N} \cos \frac{(2j+1)v\pi}{2N},$$

where $0 \leq u, v < 8$ and

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u > 0 \end{cases}.$$

Discrete cosine transform

The transform represents an 8 × 8 matrix of samples as a weighted sum of the DCT basis vectors:



Matlab code to simulate a JPEG compression cycle (1)

```
function jpeg_result = jpeg_compression_cycle(original)
  % Transform matrices
  dct_matrix = dctmtx(8);
  dct = @(block_struct) dct_matrix * block_struct.data * dct_matrix';
  idct = @(block_struct) dct_matrix' * block_struct.data * dct_matrix;

  % Quantization tables
  q_max = 255;
  q_y = ...
      [16 11 10 16 124 140 151 161;
       12 12 14 19 126 158 160 155;
       14 13 16 24 140 157 169 156;
       14 17 22 29 151 187 180 162;
       18 22 37 56 168 109 103 177;
       24 35 55 64 181 104 113 192;
       49 64 78 87 103 121 120 101;
       72 92 95 98 112 100 103 199];
  q_c = ...
      [17 18 24 47 99 99 99 99;
       18 21 26 66 99 99 99 99;
       24 26 56 99 99 99 99 99;
       47 66 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99;
       99 99 99 99 99 99 99 99];
```

Matlab code to simulate a JPEG compression cycle (2)

```
% Scale quantization matrices based on quality factor
qf = 75;
if qf < 50
  q_scale = floor(5000 / qf);
else
  q_scale = 200 - 2 * qf;
end
q_y = round(q_y * q_scale / 100);
q_c = round(q_c * q_scale / 100);

% RGB to YCbCr
ycc = rgb2ycbcr(im2double(original));

% Down-sample and decimate chroma
cb = conv2(ycc(:, :, 2), [1 1; 1 1]) ./ 4.0;
cr = conv2(ycc(:, :, 3), [1 1; 1 1]) ./ 4.0;
cb = cb(2 : 2 : size(cb, 1), 2 : 2 : size(cb, 2));
cr = cr(2 : 2 : size(cr, 1), 2 : 2 : size(cr, 2));
y  = ycc(:, :, 1);

% Discrete cosine transform, with scaling before quantization
y  = blockproc( y, [8 8], dct) .* q_max;
cb = blockproc(cb, [8 8], dct) .* q_max;
cr = blockproc(cr, [8 8], dct) .* q_max;

% Quantize DCT coefficients
y  = blockproc( y, [8 8], @(block_struct) round(round(block_struct.data) ./ q_y));
cb = blockproc(cb, [8 8], @(block_struct) round(round(block_struct.data) ./ q_c));
cr = blockproc(cr, [8 8], @(block_struct) round(round(block_struct.data) ./ q_c));
```

Matlab code to simulate a JPEG compression cycle (3)

```
  % Dequantize DCT coefficients
  y  = blockproc( y, [8 8], @(block_struct) block_struct.data .* q_y);
  cb = blockproc(cb, [8 8], @(block_struct) block_struct.data .* q_c);
  cr = blockproc(cr, [8 8], @(block_struct) block_struct.data .* q_c);

  % Inverse DCT
  y  = blockproc( y ./ q_max, [8 8], idct);
  cb = blockproc(cb ./ q_max, [8 8], idct);
  cr = blockproc(cr ./ q_max, [8 8], idct);

  % Up-sample chroma
  upsample_filter_1d = [1 3 3 1] / 4;
  upsample_filter = upsample_filter_1d' * upsample_filter_1d;
  cb = conv2(upsample_filter,
             upsample(upsample(padarray(cb, [1 1], 'replicate'), 2)', 2)');
  cb = cb(4 : size(cb, 1) - 4, 4 : size(cb, 2) - 4);
  cr = conv2(upsample_filter,
             upsample(upsample(padarray(cr, [1 1], 'replicate'), 2)', 2)');
  cr = cr(4 : size(cr, 1) - 4, 4 : size(cr, 2) - 4);

  % Concatenate the channels
  jpeg_result = ycbcr2rgb(cat(3, y, cb, cr));
end
```