◊ VIII.5

Contrast Limited Adaptive Histogram Equalization

Karel Zuiderveld
Computer Vision Research Group
Utrecht University
Utrecht, The Netherlands
karel@cv.ruu.nl

This Gem describes a contrast enhancement technique called *adaptive histogram equalization*, AHE for short, and an improved version of AHE, named *contrast limited adaptive histogram equalization*, CLAHE, that both overcome the limitations of standard histogram equalization. CLAHE was originally developed for medical imaging and has proven to be successful for enhancement of low-contrast images such as portal films (Rosenman *et al.* 1993).

◊ Introduction ◊

Probably the most used image processing function is contrast enhancement with a lookup table, a 1-to-1 pixel transform as described in (Jain 1989). When an image has poor contrast, the use of an appropriate mapping function (usually a linear ramp) often results in an improved image.

The mapping function can also be non-linear; a well-known example is gamma correction. Another non-linear technique is *histogram equalization*; it is based on the assumption that a good gray-level assignment scheme should depend on the frequency distribution (histogram) of image gray levels. As the number of pixels in a certain class of gray levels increases, one likes to assign a larger part of the available output gray ranges to the corresponding pixels. This condition is met when cumulative histograms are used as a gray-level transform as is shown in Figure 1.

The histogram of the resulting image is approximately flat, which suggests an optimal distribution of the gray values. However, Figure 1 shows that histogram equalization in its basic form can give a result that is worse than the original image. Large peaks in the histogram can also be caused by uninteresting areas (especially background noise); in this case, histogram equalization mainly leads to an improved visibility of image noise. The technique does also not adapt to local contrast requirements; minor contrast differences can be entirely missed when the number of pixels falling in a particular gray range is small.

IBM ISBN 0-12-336155-9
Macintosh ISBN 0-12-336156-7

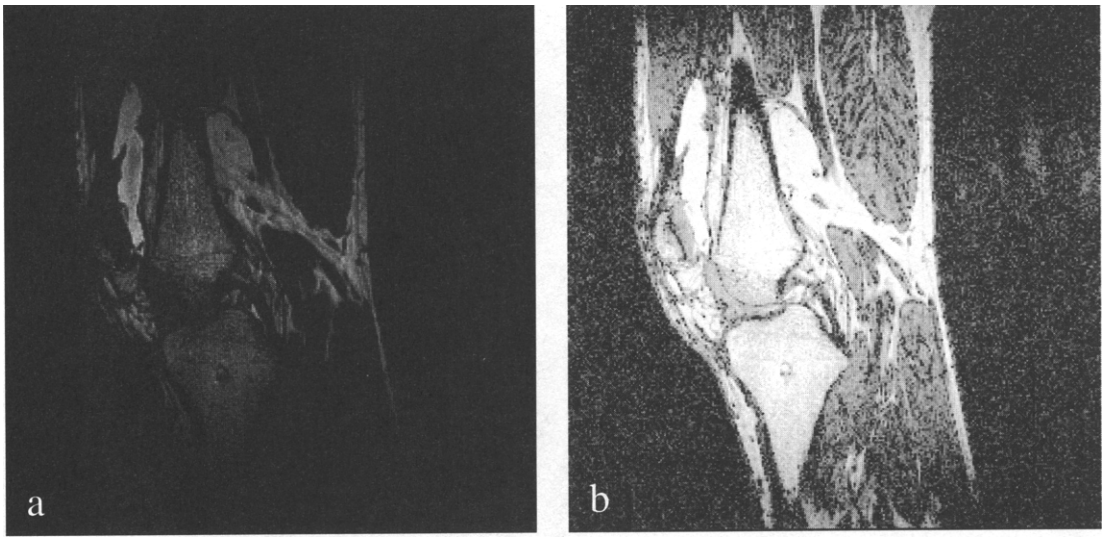

Figure 1. Example of contrast enhancement using histogram equalization. (a) The original image, an image of a human knee obtained with Magnetic Resonance Imaging. (b) Result of histogram equalization.

♢ Adaptive Histogram Equalization (AHE) ♢

Since our eyes adapt to the local context of images to evaluate their contents, it makes sense to optimize local image contrast (Pizer *et al.* 1987). To accomplish this, the image is divided in a grid of rectangular *contextual regions* in which the optimal contrast must be calculated. The optimal number of contextual regions depends on the type of input image, and its determination requires some experimentation. Division of the image into 8×8 contextual regions usually gives good results; this implies 64 contextual regions of size 64×64 when AHE is performed on a 512×512 image.

For each of these contextual regions, the histogram of the contained pixels is calculated. Calculation of the corresponding cumulative histograms results in a gray-level assignment table that optimizes contrast in each of the contextual regions, essentially a histogram equalization based on local image data.

To avoid visibility of region boundaries, a bilinear interpolation scheme is used (see Figure 2).

Applying adaptive histogram equalization on the image in Figure 1a results in the image that can be found in Figure 2b. Although the contrast of the relevant structures in the knee is largely improved, the most striking feature of the image is the background noise that has become visible. Although one can argue that AHE does what it is supposed to do — optimal presentation of information present in the image — noise present in AHE images turns out to be a major drawback of the method.

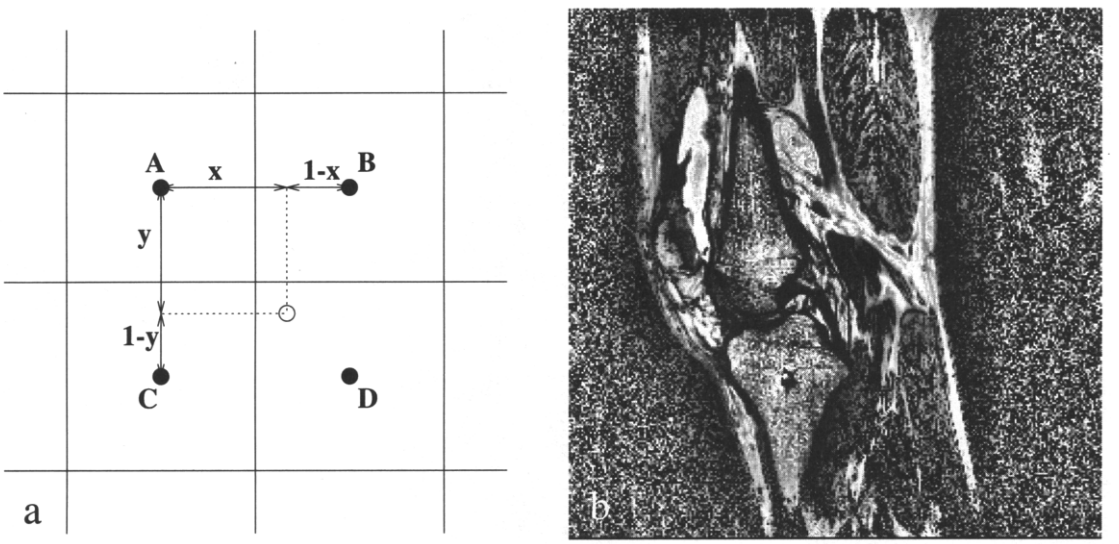

Figure 2. Subdivision and interpolation scheme used with adaptive histogram equalization and a typical result of AHE. (a) The gray-level assignment at the sample position, indicated by a white dot, is derived from the gray-value distributions in the surrounding contextual regions. The points A, B, C, and D form the center of the relevant contextual regions; region-specific gray-level mappings ($g_A(s)$, $g_B(s)$, $g_C(s)$ and $g_D(s)$) are based on the histogram of the pixels contained. Assuming that the original pixel intensity at the sample point is s, its new gray value is calculated by bilinear interpolation of the gray-level mappings that were calculated for each of the surrounding contextual regions: $s' = (1-y)((1-x)g_A(s) + xg_B(s)) + y((1-x)g_C(s) + xg_D(s))$ where x and y are normalized distances with respect to the point A. At edges and corners, a slightly different interpolation scheme is used. (b) Result of AHE using 8×8 contextual regions applied on the image in Figure 1a. Although structures in the knee can be better distinguished, the overall appearance of the image suffers due to noise enhancement.

◊ Contrast Limited Adaptive Histogram Equalization (CLAHE) ◊

The noise problem associated with AHE can be reduced by limiting contrast enhancement specifically in homogeneous areas. These areas can be characterized by a high peak in the histogram associated with the contextual regions since many pixels fall inside the same gray range. With CLAHE, the slope associated with the gray-level assignment scheme is limited; this can be accomplished by allowing only a maximum number of pixels in each of the bins associated with local histograms. After clipping the histogram, the pixels that were clipped are equally redistributed over the whole histogram to keep the total histogram count identical (see Figure 3).

The clip limit (or contrast factor) is defined as a multiple of the average histogram contents. With a low factor, the maximum slope of local histograms will be low and

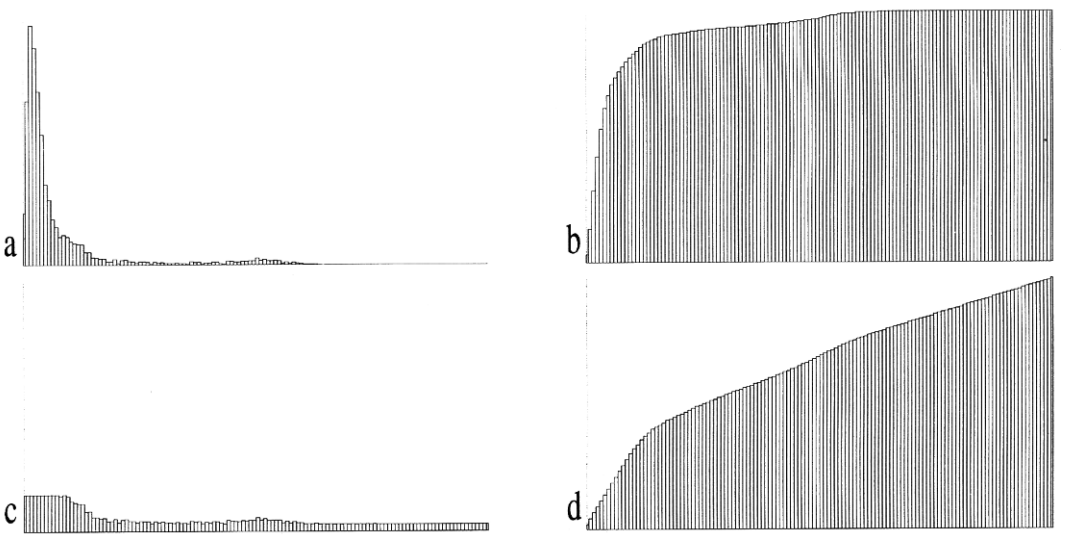

Figure 3. Principle of contrast limiting as used with CLAHE. (a) Histogram of a contextual region containing many background pixels. (b) Calculated cumulative histogram; when used as a gray-level mapping, many bins are wasted for visualization of background noise. (c) Clipped histogram obtained using a clip limit of three. Excess pixels are redistributed through the histogram. (d) Cumulative clipped histogram; its maximum slope (equal to the contrast enhancement obtained) is equal to the clip limit.

therefore result in limited contrast enhancement. A factor of one prohibits contrast enhancement (giving the original image); redistribution of histogram bin values can be avoided by using a very high clip limit (one thousand or higher), which is equivalent to the AHE technique.

Figure 4 shows two examples of contrast enhancement using CLAHE; although the image at the right was CLAHE processed using a high clip limit, image noise is still acceptable.

The main advantages of the CLAHE transform as presented in this Gem are the modest computational requirements, its ease of use (requiring only one parameter: the clip limit), and its excellent results on most images.

CLAHE does have disadvantages. Since the method is aimed at optimizing contrast, there is no 1 to 1 relationship between the gray values of the original image and the CLAHE processed result; consequently, CLAHE images are not suited for quantitative measurements that rely on a physical meaning of image intensity. A more serious problem are artifacts that sometimes occur when high-intensity gradients are present; see (Cromartie and Pizer 1991) for an explanation of these artifacts and a possible (but computationally expensive) solution. A detailed overview of AHE and other histogram equalization methods can be found in (Gauch 1992).

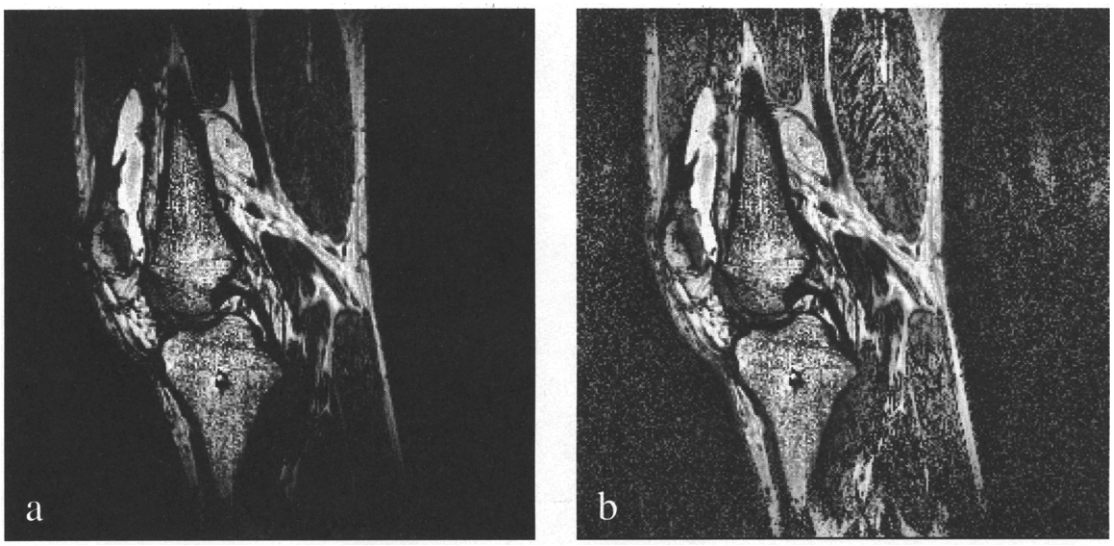

Figure 4. Result of CLAHE applied on the image in Figure 1a. (a) CLAHE with clip limit 3. (b) CLAHE with clip limit 10. Both images were obtained using 8 x 8 contextual regions.

◊ Implementation ◊

Since CLAHE has its roots in medical imaging, the earlier CLAHE implementations assumed 16-bit image pixels, since medical scanners often generate 12-bit images. This implementation is a rewrite of a K&R C version written more than five years ago; it is now Ansi-C as well as C++ compliant and can also process 8-bit images.

For a 512×512 image, this implementation of CLAHE requires less than a second on an HP 9000/720 workstation when 8×8 contextual regions are used.

```
/*
 *  These functions implement contrast limited adaptive histogram equalization.
 *  The main routine (CLAHE) expects an input image that is stored contiguously in
 *  memory;  the CLAHE output image overwrites the original input image and has the
 *  same minimum and maximum values (which must be provided by the user).
 *  This implementation assumes that the X- and Y image resolutions are an integer
 *  multiple of the X- and Y sizes of the contextual regions. A check on various other
 *  error conditions is performed.
 *
 *  #define the symbol BYTE_IMAGE to make this implementation suitable for
 *  8-bit images. The maximum number of contextual regions can be redefined
 *  by changing uiMAX_REG_X and/or uiMAX_REG_Y; the use of more than 256
 *  contextual regions is not recommended.
 *
 *  The code is ANSI-C and is also C++ compliant.
 *
 *  Author: Karel Zuiderveld, Computer Vision Research Group,
 *            Utrecht, The Netherlands (karel@cv.ruu.nl)
 */

#ifdef BYTE_IMAGE
typedef unsigned char kz_pixel_t;         /* for 8 bit-per-pixel images */
#define uiNR_OF_GREY (256)
#else
typedef unsigned short kz_pixel_t;        /* for 12 bit-per-pixel images (default) */
# define uiNR_OF_GREY (4096)
#endif

/******** Prototype of CLAHE function. Put this in a separate include file. *****/
int CLAHE(kz_pixel_t* pImage, unsigned int uiXRes, unsigned int uiYRes, kz_pixel_t Min,
          kz_pixel_t Max, unsigned int uiNrX, unsigned int uiNrY,
          unsigned int uiNrBins, float fCliplimit);

/********************** Local prototypes ************************/
static void ClipHistogram (unsigned long*, unsigned int, unsigned long);
static void MakeHistogram (kz_pixel_t*, unsigned int, unsigned int, unsigned int,
               unsigned long*, unsigned int, kz_pixel_t*);
static void MapHistogram (unsigned long*, kz_pixel_t, kz_pixel_t,
              unsigned int, unsigned long);
static void MakeLut (kz_pixel_t*, kz_pixel_t, kz_pixel_t, unsigned int);
static void Interpolate (kz_pixel_t*, int, unsigned long*, unsigned long*,
        unsigned long*, unsigned long*, unsigned int, unsigned int, kz_pixel_t*);

/**************  Start of actual code **************/
#include <stdlib.h>                       /* To get prototypes of malloc() and free() */

const unsigned int uiMAX_REG_X = 16;      /* max. # contextual regions in x-direction */
const unsigned int uiMAX_REG_Y = 16;      /* max. # contextual regions in y-direction */
```

```
/************************* main function CLAHE *****************/
int CLAHE (kz_pixel_t* pImage, unsigned int uiXRes, unsigned int uiYRes,
         kz_pixel_t Min, kz_pixel_t Max, unsigned int uiNrX, unsigned int uiNrY,
              unsigned int uiNrBins, float fCliplimit)
/*   pImage - Pointer to the input/output image
 *   uiXRes - Image resolution in the X direction
 *   uiYRes - Image resolution in the Y direction
 *   Min - Minimum gray-value of input image (also becomes minimum of output image)
 *   Max - Maximum gray-value of input image (also becomes maximum of output image)
 *   uiNrX - Number of contextial regions in the X direction (min 2, max uiMAX_REG_X)
 *   uiNrY - Number of contextial regions in the Y direction (min 2, max uiMAX_REG_Y)
 *   uiNrBins - Number of gray bins for histogram ("dynamic range")
 *   float fCliplimit - Normalized clip limit (higher values give more contrast)
 * The number of "effective" gray levels in the output image is set by uiNrBins; selecting
 * a small value (eg. 128) speeds up processing and still produce an output image of
 * good quality. The output image will have the same minimum and maximum value as the input
 * image. A clip limit smaller than 1 results in standard (non-contrast-limited) AHE.
 */
{
    unsigned int uiX, uiY;                  /* counters */
    unsigned int uiXSize, uiYSize, uiSubX, uiSubY; /* size of context. reg. and subimages */
    unsigned int uiXL, uiXR, uiYU, uiYB;  /* auxiliary variables interpolation routine */
    unsigned long ulClipLimit, ulNrPixels;/* clip limit and region pixel count */
    kz_pixel_t* pImPointer;                 /* pointer to image */
    kz_pixel_t aLUT[uiNR_OF_GREY];          /* lookup table used for scaling of input image */
    unsigned long* pulHist, *pulMapArray; /* pointer to histogram and mappings*/
    unsigned long* pulLU, *pulLB, *pulRU, *pulRB; /* auxiliary pointers interpolation */

    if (uiNrX > uiMAX_REG_X) return -1;     /* # of regions x-direction too large */
    if (uiNrY > uiMAX_REG_Y) return -2;     /* # of regions y-direction too large */
    if (uiXRes % uiNrX) return -3;          /* x-resolution no multiple of uiNrX */
    if (uiYRes & uiNrY) return -4;          /* y-resolution no multiple of uiNrY */
    if (Max >= uiNR_OF_GREY) return -5;     /* maximum too large */
    if (Min >= Max) return -6;              /* minimum equal or larger than maximum */
    if (uiNrX < 2 || uiNrY < 2) return -7;/* at least 4 contextual regions required */
    if (fCliplimit == 1.0) return 0;        /* is OK, immediately returns original image. */
    if (uiNrBins == 0) uiNrBins = 128;      /* default value when not specified */

    pulMapArray=(unsigned long *)malloc(sizeof(unsigned long)*uiNrX*uiNrY*uiNrBins);
    if (pulMapArray == 0) return -8;        /* Not enough memory! (try reducing uiNrBins) */

    uiXSize = uiXRes/uiNrX; uiYSize = uiYRes/uiNrY;  /* Actual size of contextual regions */
    ulNrPixels = (unsigned long)uiXSize * (unsigned long)uiYSize;

    if(fCliplimit > 0.0) {                  /* Calculate actual cliplimit  */
       ulClipLimit = (unsigned long) (fCliplimit * (uiXSize * uiYSize) / uiNrBins);
       ulClipLimit = (ulClipLimit < 1UL) ? 1UL : ulClipLimit;
    }
    else ulClipLimit = 1UL<<14;             /* Large value, do not clip (AHE) */
```

```
    MakeLut(aLUT, Min, Max, uiNrBins);      /* Make lookup table for mapping of gray values */
    /* Calculate gray-level mappings for each contextual region */
    for (uiY = 0, pImPointer = pImage; uiY < uiNrY; uiY++) {
        for (uiX = 0; uiX < uiNrX; uiX++, pImPointer += uiXSize) {
            pulHist = &pulMapArray[uiNrBins * (uiY * uiNrX + uiX)];
            MakeHistogram(pImPointer,uiXRes,uiXSize,uiYSize,pulHist,uiNrBins,aLUT);
            ClipHistogram(pulHist, uiNrBins, ulClipLimit);
            MapHistogram(pulHist, Min, Max, uiNrBins, ulNrPixels);
        }
        pImPointer += (uiYSize - 1) * uiXRes;              /* skip lines, set pointer */
    }

    /* Interpolate gray-level mappings to get CLAHE image */
    for (pImPointer = pImage, uiY = 0; uiY <= uiNrY; uiY++) {
        if (uiY == 0) {                                    /* special case: top row */
            uiSubY = uiYSize >> 1;  uiYU = 0; uiYB = 0;
        }
        else {
            if (uiY == uiNrY) {                            /* special case: bottom row */
                uiSubY = uiYSize >> 1;  uiYU = uiNrY-1;  uiYB = uiYU;
            }
            else {                                         /* default values */
                uiSubY = uiYSize; uiYU = uiY - 1; uiYB = uiYU + 1;
            }
        }
        for (uiX = 0; uiX <= uiNrX; uiX++) {
            if (uiX == 0) {                                /* special case: left column */
                uiSubX = uiXSize >> 1; uiXL = 0; uiXR = 0;
            }
            else {
                if (uiX == uiNrX) {                        /* special case: right column */
                    uiSubX = uiXSize >> 1;  uiXL = uiNrX - 1; uiXR = uiXL;
                }
                else {                                     /* default values */
                    uiSubX = uiXSize; uiXL = uiX - 1; uiXR = uiXL + 1;
                }
            }

            pulLU = &pulMapArray[uiNrBins * (uiYU * uiNrX + uiXL)];
            pulRU = &pulMapArray[uiNrBins * (uiYU * uiNrX + uiXR)];
            pulLB = &pulMapArray[uiNrBins * (uiYB * uiNrX + uiXL)];
            pulRB = &pulMapArray[uiNrBins * (uiYB * uiNrX + uiXR)];
            Interpolate(pImPointer,uiXRes,pulLU,pulRU,pulLB,pulRB,uiSubX,uiSubY,aLUT);
            pImPointer += uiSubX;                          /* set pointer on next matrix */
        }
        pImPointer += (uiSubY - 1) * uiXRes;
    }
    free(pulMapArray);                                     /* free space for histograms */
    return 0;                                              /* return status OK */
}
```

```
void ClipHistogram (unsigned long* pulHistogram, unsigned int
                    uiNrGreylevels, unsigned long ulClipLimit)
/* This function performs clipping of the histogram and redistribution of bins.
 * The histogram is clipped and the number of excess pixels is counted. Afterwards
 * the excess pixels are equally redistributed across the whole histogram (providing
 * the bin count is smaller than the clip limit).
 */
{
    unsigned long* pulBinPointer, *pulEndPointer, *pulHisto;
    unsigned long ulNrExcess, ulUpper, ulBinIncr, ulStepSize, i;
    long lBinExcess;

    ulNrExcess = 0;  pulBinPointer = pulHistogram;
    for (i = 0; i < uiNrGreylevels; i++) { /* calculate total number of excess pixels */
        lBinExcess = (long) pulBinPointer[i] - (long) ulClipLimit;
        if (lBinExcess > 0) ulNrExcess += lBinExcess;      /* excess in current bin */
    };

    /* Second part: clip histogram and redistribute excess pixels in each bin */
    ulBinIncr = ulNrExcess / uiNrGreylevels;               /* average bin increment */
    ulUpper =  ulClipLimit - ulBinIncr;  /* Bins larger than ulUpper set to clip limit */

    for (i = 0; i < uiNrGreylevels; i++) {
      if (pulHistogram[i] > ulClipLimit) pulHistogram[i] = ulClipLimit; /* clip bin */
      else {
          if (pulHistogram[i] > ulUpper) {               /* high bin count */
              ulNrExcess -= pulHistogram[i] - ulUpper; pulHistogram[i]=ulClipLimit;
          }
          else {                                         /* low bin count */
              ulNrExcess -= ulBinIncr; pulHistogram[i] += ulBinIncr;
          }
       }
    }

    while (ulNrExcess) {   /* Redistribute remaining excess  */
        pulEndPointer = &pulHistogram[uiNrGreylevels]; pulHisto = pulHistogram;

        while (ulNrExcess && pulHisto < pulEndPointer) {
            ulStepSize = uiNrGreylevels / ulNrExcess;
            if (ulStepSize < 1) ulStepSize = 1;            /* stepsize at least 1 */
            for (pulBinPointer=pulHisto; pulBinPointer < pulEndPointer && ulNrExcess;
                 pulBinPointer += ulStepSize) {
                if (*pulBinPointer < ulClipLimit) {
                    (*pulBinPointer)++;  ulNrExcess--;     /* reduce excess */
                }
            }
            pulHisto++;            /* restart redistributing on other bin location */
        }
    }
}
```

```
void MakeHistogram (kz_pixel_t* pImage, unsigned int uiXRes,
                    unsigned int uiSizeX, unsigned int uiSizeY,
                    unsigned long* pulHistogram,
                    unsigned int uiNrGreylevels, kz_pixel_t* pLookupTable)
/* This function classifies the gray-levels present in the array image into
 * a grey-level histogram. The pLookupTable specifies the relationship
 * between the gray-value of the pixel (typically between 0 and 4095) and
 * the corresponding bin in the histogram (usually containing only 128 bins).
 */
{
    kz_pixel_t* pImagePointer;
    unsigned int i;

    for (i = 0; i < uiNrGreylevels; i++) pulHistogram[i] = 0L; /* clear histogram */

    for (i = 0; i < uiSizeY; i++) {
        pImagePointer = &pImage[uiSizeX];
        while (pImage < pImagePointer) pulHistogram[pLookupTable[*pImage++]]++;
        pImagePointer += uiXRes;
        pImage = &pImagePointer[-uiSizeX];
    }
}

void MapHistogram (unsigned long* pulHistogram, kz_pixel_t Min, kz_pixel_t Max,
                   unsigned int uiNrGreylevels, unsigned long ulNrOfPixels)
/* This function calculates the equalized lookup table (mapping) by
 * cumulating the input histogram. Note: lookup table is rescaled in range [Min..Max].
 */
{
    unsigned int i;  unsigned long ulSum = 0;
    const float fScale = ((float)(Max - Min)) / ulNrOfPixels;
    const unsigned long ulMin = (unsigned long) Min;

    for (i = 0; i < uiNrGreylevels; i++) {
        ulSum += pulHistogram[i]; pulHistogram[i]=(unsigned long)(ulMin+ulSum*fScale);
        if (pulHistogram[i] > Max) pulHistogram[i] = Max;
    }
}

void MakeLut (kz_pixel_t * pLUT, kz_pixel_t Min, kz_pixel_t Max, unsigned int uiNrBins)
/* To speed up histogram clipping, the input image [Min,Max] is scaled down to
 * [0,uiNrBins-1]. This function calculates the LUT.
 */
{
    int i;
    const kz_pixel_t BinSize = (kz_pixel_t) (1 + (Max - Min) / uiNrBins);

    for (i = Min; i <= Max; i++)  pLUT[i] = (i - Min) / BinSize;
}
```

```
void Interpolate (kz_pixel_t * pImage, int uiXRes, unsigned long * pulMapLU,
     unsigned long * pulMapRU, unsigned long * pulMapLB,  unsigned long * pulMapRB,
     unsigned int uiXSize, unsigned int uiYSize, kz_pixel_t * pLUT)
/* pImage       - pointer to input/output image
 * uiXRes       - resolution of image in x-direction
 * pulMap*      - mappings of gray-levels from histograms
 * uiXSize      - uiXSize of image submatrix
 * uiYSize      - uiYSize of image submatrix
 * pLUT         - lookup table containing mapping gray values to bins
 * This function calculates the new gray-level assignments of pixels within a submatrix
 * of the image with size uiXSize and uiYSize. This is done by a bilinear interpolation
 * between four different mappings in order to eliminate boundary artifacts.
 * It uses a division; since division is often an expensive operation, I added code to
 * perform a logical shift instead when feasible.
 */
{
    const kz_pixel_t Max = (kz_pixel_t) uiNR_OF_GREY - 1;
    const unsigned int uiIncr = uiXRes-uiXSize; /* Pointer increment after processing row */
    kz_pixel_t GreyValue; unsigned int uiNum = uiXSize*uiYSize; /* Normalization factor */

    unsigned int uiXCoef, uiYCoef, uiXInvCoef, uiYInvCoef, uiShift = 0;

    if (uiNum & (uiNum - 1))    /* If uiNum is not a power of two, use division */
    for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef < uiYSize;
         uiYCoef++, uiYInvCoef--,pImage+=uiIncr) {
        for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef < uiXSize;
             uiXCoef++, uiXInvCoef--) {
            GreyValue = pLUT[*pImage];              /* get histogram bin value */
            *pImage++ = (kz_pixel_t ) ((uiYInvCoef * (uiXInvCoef*pulMapLU[GreyValue]
                                    + uiXCoef * pulMapRU[GreyValue])
                                + uiYCoef * (uiXInvCoef * pulMapLB[GreyValue]
                                    + uiXCoef * pulMapRB[GreyValue])) / uiNum);
        }
    }
    else {                          /* avoid the division and use a right shift instead */
        while (uiNum >>= 1) uiShift++;              /* Calculate 2log of uiNum */
        for (uiYCoef = 0, uiYInvCoef = uiYSize; uiYCoef < uiYSize;
             uiYCoef++, uiYInvCoef--,pImage+=uiIncr) {
             for (uiXCoef = 0, uiXInvCoef = uiXSize; uiXCoef < uiXSize;
               uiXCoef++, uiXInvCoef--) {
               GreyValue = pLUT[*pImage];          /* get histogram bin value */
               *pImage++ = (kz_pixel_t)((uiYInvCoef* (uiXInvCoef * pulMapLU[GreyValue]
                                    + uiXCoef * pulMapRU[GreyValue])
                                + uiYCoef * (uiXInvCoef * pulMapLB[GreyValue]
                                    + uiXCoef * pulMapRB[GreyValue])) >> uiShift);
            }
        }
    }
}
```

◊ Bibliography ◊

(Cromartie and Pizer 1991) R. Cromartie and S.M. Pizer. Edge-affected context for adaptive contrast enhancement. In A. C. S. Colchester and D. J. Hawkes, editors, *Proceedings of the XIIth International Meeting on Information Processing in Medical Imaging: Lecture Notes in Computer Science*, pages 474–485, Springer-Verlag, Berlin, 1991.

(Gauch 1992) J. M. Gauch. Investigations of image contrast space defined by variations on histogram equalization. *CVGIP: Graphical Models and Image Processing*, 54(4):269–280, July 1992.

(Jain 1989) A. K. Jain. *Fundamentals of digital image processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

(Pizer *et al.* 1987) S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39:355–368, 1987.

(Rosenman *et al.* 1993) J. Rosenman, C. A. Roe, R. Cromartie, K. E. Muller, and S. M. Pizer. Portal film enhancement: Technique and clinical utility. *Int. J. Radiat. Oncol. Biol. Physics*, pages 333 338, 1993.