> 本节为opencv数字图像处理（2）：灰度变换与空间滤波的第二小节，直方图处理，主要包括：直方图均衡化与直方图规定化的原理、推导与C++代码实现。

2. 直方图处理

灰度级范围为[0,L-1]的数字图像的直方图是离散函数$h(r_k)=n_k$，其中r_k是第k级灰度值，n_k是图像中灰度为r_k的像素个数。

直方图归一化可以用这个式子表示：$p(r_k)=n_k/MN$，其中k=0,1,...,L-1，p其实就是灰度级r_k在图像中出现的一个概率的估计，归一化后所有分量之和应为1。

2.1 直方图均衡化

考虑连续的灰度值，用变量r表示待处理图像的灰度，r的取值区间为[0,L-1]，考虑变换形式：

$$s=T(r),0\leqslant r\leqslant L-1$$

对输入图像中每个具有r值的像素值产生一个输出灰度值s，假设：

- a. T()在区间[0,L-1]上为严格单调递增函数
- b. r和T()均在区间[0,L-1]上

一幅图像的灰度级可看成区间[0,L-1]内的随机变量，随机变量的基本描绘子是概率密度函数。令$p_r(r)$和$p_s(s)$分别表示概率密度函数，若前者和T已知且T在感兴趣值域上连续可微，则有：

$$p_s(s)=p_r(r)\left|\frac{\mathrm{d}r}{\mathrm{d}s}\right|$$

也即输出灰度变量s的概率密度函数由输入灰度的概率密度函数和所用变换函数决定。

图像处理中一个重要的变换函数有如下形式：

$$s=T(r)=(L-1)\int_0^r p_r(w)\mathrm{d}w$$

公式右边是随机变量的累计分布函数。该变换满足单调递增且定义域值域均为[0,L-1]，又因为：

$$\frac{\mathrm{d}s}{\mathrm{d}r}=\frac{\mathrm{d}T(r)}{\mathrm{d}r}=(L-1)\frac{\mathrm{d}}{\mathrm{d}r}\left[\int_0^r p_r(w)\mathrm{d}w\right]=(L-1)p_r(r)$$

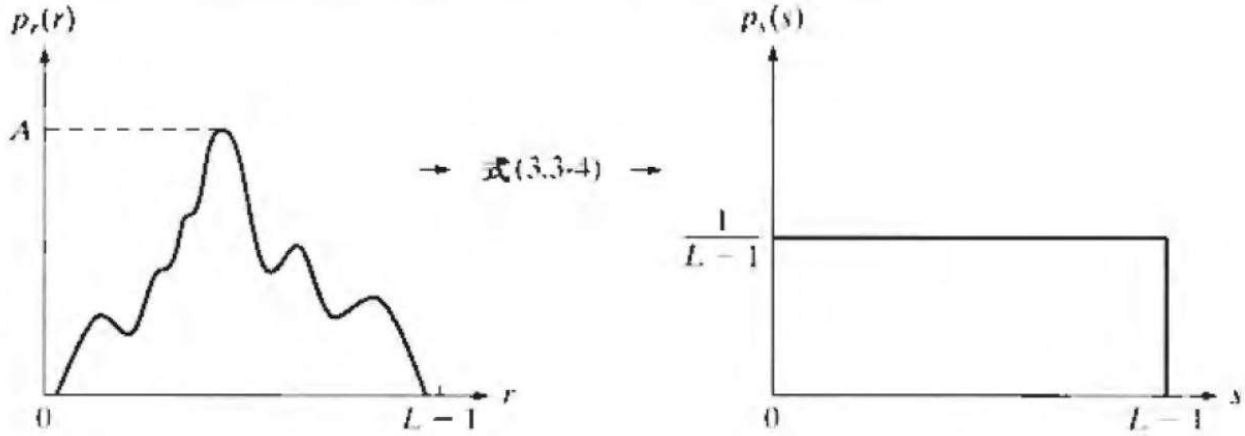
把这个结果代入：

$$p_s(s)=p_r(r)\left|\frac{dr}{ds}\right|$$

则有：

$$p_s(s)=p_r(r)\left|\frac{dr}{ds}\right|=p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right|=\frac{1}{L-1},\quad 0\leqslant s\leqslant L-1$$

可以看出$p_s(s)$始终是一个均匀概率密度函数，与$p_r(s)$无关，如下图所示，作图是一个任意的概率密度函数，变换的结果总是一个均匀的概率密度函数：



现假设输入图像中的灰度值具有如下的概率密度函数：

$$p_r(r)=\begin{cases}\dfrac{2r}{(L-1)^2}, & 0\leqslant r\leqslant L-1\\ 0, & \text{其他}\end{cases}$$

则输出像素为：

$$s=T(r)=(L-1)\int_0^r p_r(w)dw=\frac{2}{L-1}\int_0^r wdw=\frac{r^2}{L-1}$$

上面讨论了连续灰度的情况，若是离散值则用概率（直方图值）与求和来代替处理概率密度函数与积分，之前提到一幅数字图像中灰度级r_k出现的概率近似为：$p_r(r_k)=\frac{n_k}{MN}$，其中MN是图像中像素的总数，n_k是灰度为r_k的像素个数，与r_k对应的$p_r(r_k)$图形通常称为直方图，离散的变换形式如下式：

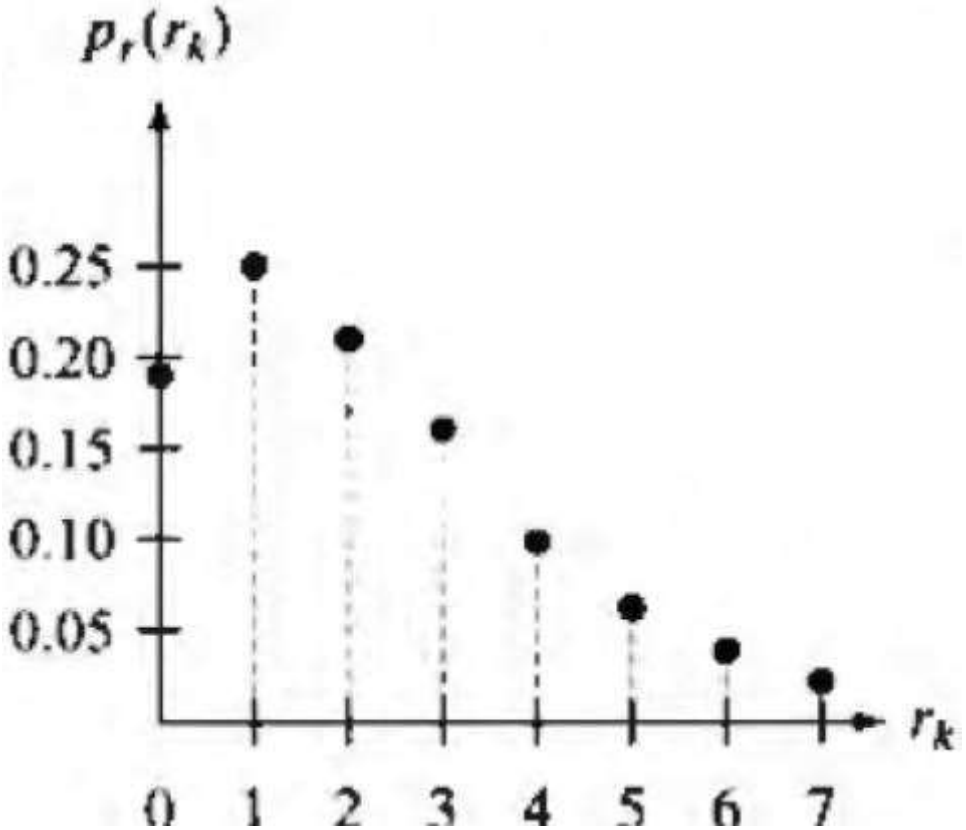
$$s_k=T(r_k)=(L-1)\sum_{j=0}^{k}p_r(r_j)=\frac{(L-1)}{MN}\sum_{j=0}^{k}n_j,\quad k=0,1,2,\cdots,L-1$$

这样，输入图像中灰度级为r_k的各像素映射到输出图像中灰度级为s_k的对应像素，上式中变换T称为直方图均衡或直方图线性变换。

看一个简单的例子，假设一幅64x64像素的3比特图像的灰度分布如下表所示，其中灰度级范围[0,7]

r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0=0$	790	0.19
$r_1=1$	1023	0.25
$r_2=2$	850	0.21
$r_3=3$	656	0.16
$r_4=4$	329	0.08
$r_5=5$	245	0.06
$r_6=6$	122	0.03
$r_7=7$	81	0.02

该图像的直方图如下图所示：

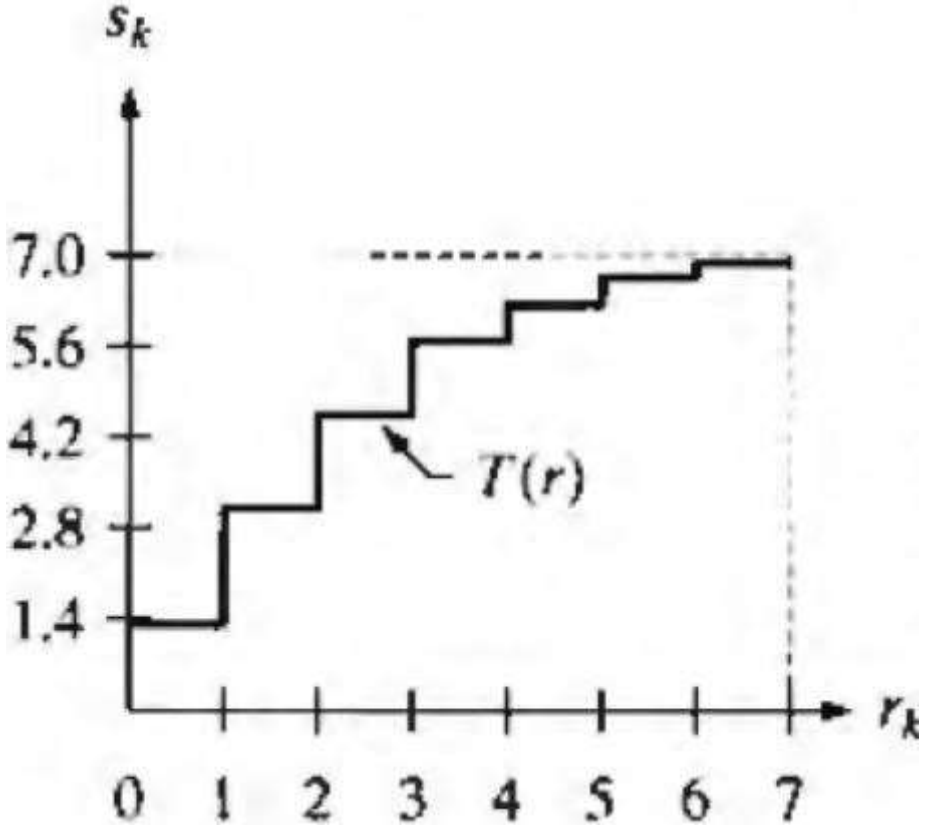


直方图变化函数的值根据上面的式子得到，有：

$$s_0 = T(r_0) = 7\sum_{j=0}^{0} p_r(r_j) = 7\,p_r(r_0) = 1.33$$

$$s_1 = T(r_1) = 7\sum_{j=0}^{1} p_r(r_j) = 7\,p_r(r_0) + 7\,p_r(r_1) = 3.08$$

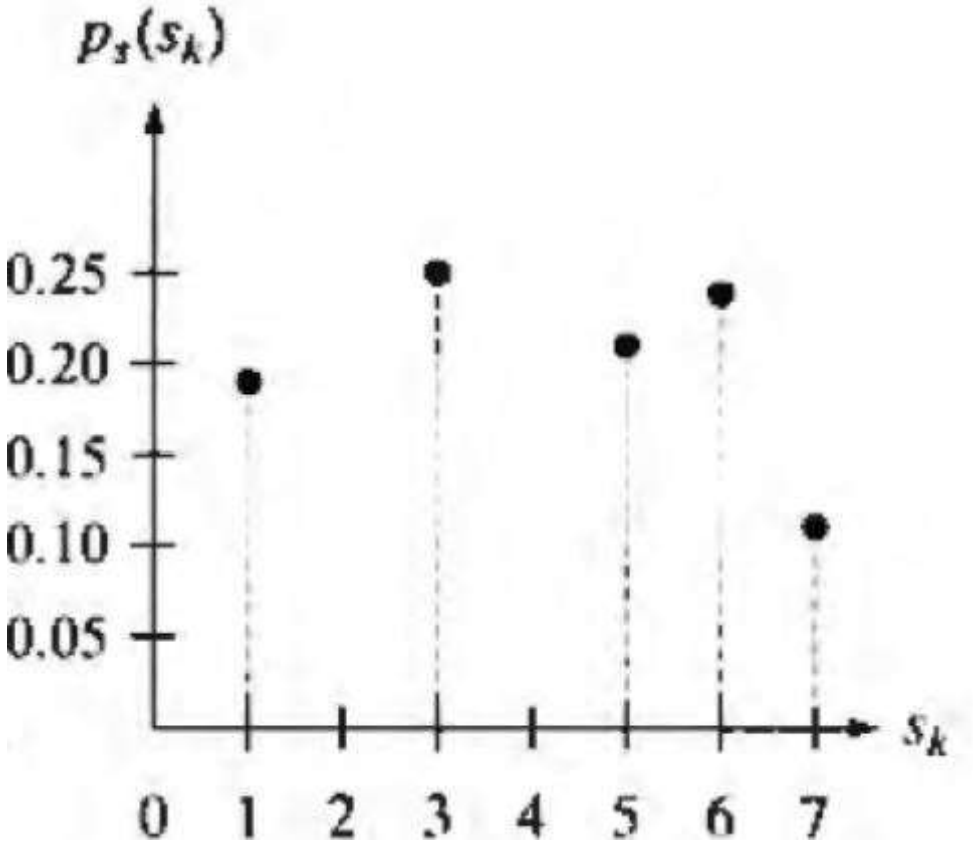
类似的可以得到$s_2 \sim s_7$的值，变换函数的形状为阶梯状，如下图所示：



因为$s_0 \sim s_7$是通过求概率值的和产生的，我们将它们近似为最接近的整数，有：

$$s_0 = 1.33 \to 1 \quad s_4 = 6.23 \to 6$$
$$s_1 = 3.08 \to 3 \quad s_5 = 6.65 \to 7$$
$$s_2 = 4.55. \to 5 \quad s_6 = 6.86 \to 7$$
$$s_3 = 5.67 \to 6 \quad s_7 = 7.00 \to 7$$

可以看出有5个不同的灰度级，统计之后可以得到均衡化之后的直方图：



因为直方图是概率密度函数的近似且不允许造成新的灰度级，所以实际的直方图均衡应用中，很少见到完美平坦的直方图，也就是说离散的情况下通常无法证明直方图均衡会导致均匀的直方图，但是该变换公式确实具有输入图像直方图的趋势，均衡后的图像的灰度级跨越更宽灰度级范围，即增强了对比度。C++代码如下：

```
// 1.调用库函数
// 彩色图像需要分通道均衡化，灰度图像直接调用即可
void equalizeHistOpencv()
{
 Mat srcImage = imread("test3.JPG", 1);
 Mat dstImage;
 if (!srcImage.data)
 {
  cout << "fail to load" << endl;
  return;
 }
 Mat channels[3];
 split(srcImage, channels);
 for (int i = 0; i < 3; i++)
 {
  equalizeHist(channels[i], channels[i]);
 }
 merge(channels, 3, dstImage);
 imshow("ori", srcImage);
 imshow("dst", dstImage);
 waitKey(0);
}
```

上面是调用库函数，也可以自己实现，如下：

```
// 2. 自己实现
Mat equalizeHistMine(Mat srcImage)
{
 int gray[256] = { 0 };
 double gray_prob[256] = { 0 };
 double gray_disSum[256] = { 0 };
 int gray_equal[256] = { 0 };
 Mat dstImage = srcImage.clone();
 int gray_sum = srcImage.cols * srcImage.rows;
 //统计每个灰度下的像素个数
 for (int i = 0; i < srcImage.rows; i++)
 {
```

```
        uchar* p = srcImage.ptr<uchar>(i);
        for (int j = 0; j < srcImage.cols; j++)
        {
         int value = p[j];
         gray[value]++;
        }
       }
       //统计灰度频率
       for (int i = 0; i < 256; i++)
       {
        gray_prob[i] = ((double)gray[i] / gray_sum);
       }
       //计算累计密度
       gray_disSum[0] = gray_prob[0];
       for (int i = 1; i < 256; i++)
       {
        gray_disSum[i] = gray_disSum[i - 1] + gray_prob[i];
       }
       //重新计算均衡化的灰度值，
       for (int i = 0; i < 256; i++)
       {
        gray_equal[i] = (uchar)(255 * gray_disSum[i] + 0.5);
       }
       //更新
       for (int i = 0; i < dstImage.rows; i++)
       {
        uchar* p = dstImage.ptr<uchar>(i);
        for (int j = 0; j < dstImage.cols; j++)
        {
         p[j] = gray_equal[p[j]];
        }
       }
       return dstImage;
      }
```

这里 `Mat equalizeHistMine(Mat srcImage)` 的作用相当于 `void equalizeHistOpencv()` 中的 `equalizeHist(channels[i], channels[i]);` ，只不过用法略有差别，可以改为 `channels[i]=equalizeHistMine(channels[i]);`

综上，直方图均衡化是一种简单有效的图像增强技术，通过改变图像的直方图来改变图像中各像素的灰度，主要用于增强动态范围偏小的图像的对比度。原始图像由于其灰度分布可能集中在较窄的区间，造成图像不够清晰。例如，过曝光图像的灰度级集中在高亮度范围内，而曝光不足将使图像灰度级集中在低亮度范围内。采用直方图均衡化，可以把原始图像的直方图变换为均匀分布（均衡）的形式，这样就增加了像素之间灰度值差别的动态范围，从而达到增强图像整体对比度的效果。换言之，直方图均衡化的基本原理是：对在图像中像

素个数多的灰度值（即对画面起主要作用的灰度值）进行展宽，而对像素个数少的灰度值（即对画面不起主要作用的灰度值）进行归并，从而增大对比度，使图像清晰，达到增强的目的。

但是如果一幅图像整体偏暗或者偏亮，那么直方图均衡化的方法很适用。但直方图均衡化是一种全局处理方式，它对处理的数据不加选择，可能会增加背景干扰信息的对比度并且降低有用信号的对比度（如果图像某些区域对比度很好，而另一些区域对比度不好，那采用直方图均衡化就不一定适用）。此外，均衡化后图像的灰度级减少，某些细节将会消失；某些图像（如直方图有高峰），经过均衡化后对比度不自然的过分增强。针对直方图均衡化的缺点，已经有局部的直方图均衡化方法出现。

2.2 直方图匹配/直方图规定化

直方图均衡能自动地确定变换函数，该函数希望产生有均匀直方图的输出图像，需要自动增强时可以应用，这种方法比较简单，但是有时候希望处理后的图像具有规定的直方图形状，这种产生处理后有特殊直方图的方法称为直方图匹配或直方图规定化。 令r和z分别表示输入和输出图像的灰度级（假设是连续灰度，这样r和z就是连续型随机变量），$p_r(r)$和$p_z(z)$分别表示它们对应的连续概率密度函数。给定输入图像估计$p_r(r)$，$p_z(z)$是我们希望输出图像所具有的指定概率密度函数。

假设一个具有如下特性的连续性随机变量：

$$s=T(r)=(L-1)\int_0^r p_r(w)\mathrm{d}w$$

这也是直方图均衡的连续形式（w是假积分变量），接着定义另一个具有如下特性的随机变量z:

$$G(z)=(L-1)\int_0^z p_z(t)\mathrm{d}t=s$$

其中t为假积分变量，于是有G(z)=T(r)，也就是说z必须满足条件：

$$z=G^{-1}[T(r)]=G^{-1}(s)$$

这样根据输入图像估计出$p_r(r)$，就可得到$T(r)$；同理，$p_z(z)$已知，那么$G(z)$也可得。

通过下面的步骤就可以由一幅给定图像得到一幅灰度级具有指定概率密度函数的图像：

- 由输入图像得$p_r(r)$，求出s；
- 根据指定得$p_z(z)$求出$G(z)$;
- 计算$z=G^{-1}(s)$;
- 即输入图像均衡化得到输出图像，均衡化之后的图像执行反映射得到最后具有指定概率密度函数的输出图像。

这就是直方图规定化。现假设一幅图像的灰度的概率密度函数为$p_r(r)=2r/(L-1)^2$，其中$0\le r\le(L-1)$，对于其他的r，$p_r(r)=0$，寻找一个变换函数，使得产生的图像的灰度的概率密度函数是$p_z(z)=3z^2/(L-1)$，同样，$0\le z\le(L-1)$，对于其他的z，$p_z(z)=0$。

首先，对区间[0,L-1]寻找直方图均衡变换：

$$s=T(r)=(L-1)\int_0^r p_r(w)\mathrm{d}w=\frac{2}{(L-1)}\int_0^r w\mathrm{d}w=\frac{r^2}{(L-1)}$$

对均衡化的图像寻找下一个直方图均衡变换：

$$G(z)=(L-1)\int_0^z p_z(w)\mathrm{d}w=\frac{3}{(L-1)^2}\int_0^z w^2\mathrm{d}w=\frac{z^3}{(L-1)^2}$$

最后要求$G(z)=s$，又$G(z)=z^3/(L-1)^2$，于是有$s=z^3/(L-1)^2$，即$z=[(L-1)^2s]^{t/3}$。　　此时用$(L-1)^2$乘以直方图均衡过的每一个像素，取该乘积的$1/3$次幂，就得到了具有指定灰度概率密度函数的图像。

因为$s=r^2/(L-1)$，所以$z=[(L-1)r^2]^{1/3}$。这样，原图像中每一个像素值的平方与(L-1)相乘，取该积的$1/3$次幂，即输出图像的像素灰度值。

直方图规定化原理简单，困难的是寻找$T(r)$，G^{-1}有意义的表达式，但是处理离散量的时候问题可以大大简化，因为这里仅希望得到一个近似的直方图，离散形式下直方图均衡变换：

$$s_k=T(r_k)=(L-1)\sum_{j=0}^{k}p_r(r_j)=\frac{(L-1)}{MN}\sum_{j=0}^{k}n_j,\quad k=0,1,2,\cdots,L-1$$

同理：

$$G(z_q)=(L-1)\sum_{i=0}^{q}p_z(z_i)$$

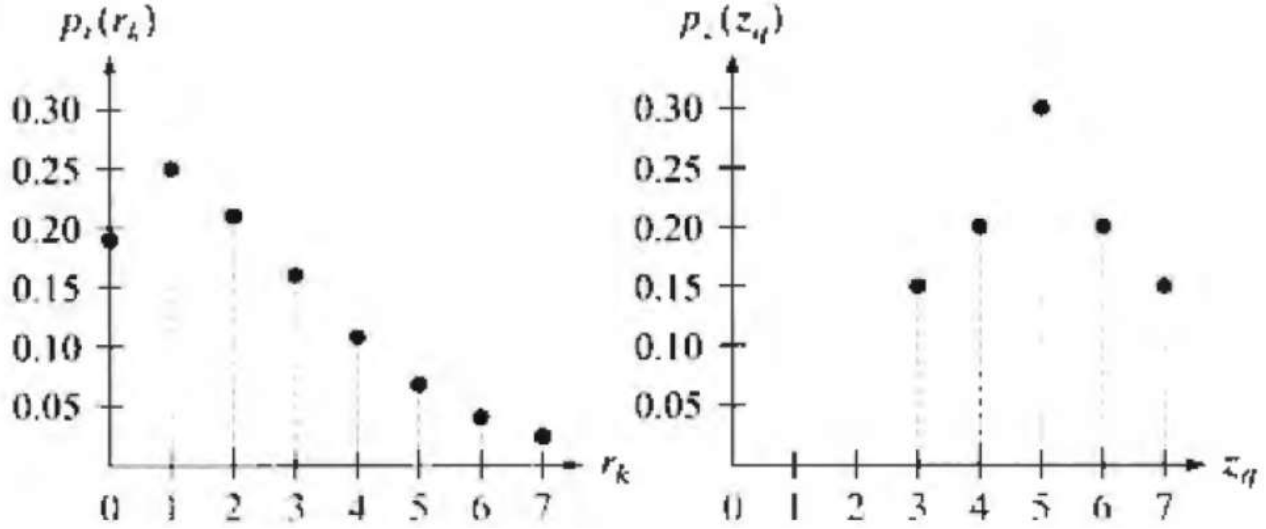
对于一个q值有$G(z_q)=s_k$，其中$p_z(z_i)$是规定的直方图中的第i个值，相似 地，用反变换找到期望的值$z_q=G^{-1}(s_k)$。

实践中，并不需要计算G的反变换，因为一般处理的灰度级为整数，这样事先将所有计算结果存储在一个表中，计算特殊的s_k值后可查询表即可。

举个例子，64x64的3比特图像，有下表：

z_q	规定的 $p_z(z_q)$	实际的 $p_z(z_k)$
$z_0=0$	0.00	0.00
$z_1=1$	0.00	0.00
$z_2=2$	0.00	0.00
$z_3=3$	0.15	0.19
$z_4=4$	0.20	0.25
$z_5=5$	0.30	0.21
$z_6=6$	0.20	0.24
$z_7=7$	0.15	0.11

原始图像的直方图如下左图，规定的输出图像的直方图如下右图：



首先得到标定的直方图均衡后的值：

$$s_0=1 \quad s_2=5 \quad s_4=6 \quad s_6=7$$
$$s_1=3 \quad s_3=6 \quad s_5=7 \quad s_7=7$$

接着计算变换函数G的值：

$$G(z_0)=7\sum_{j=0}^{0}p_z(z_j)=0.00$$

$$G(z_1)=7\sum_{j=0}^{1}p_z(z_j)=7\left[p(z_0)+p(z_1)\right]=0.00$$

$$G(z_2)=0.00, G(z_4)=2.45, G(z_6)=5.95, G(z_3)=1.05, G(z_5)=4.55, G(z_7)=7.00$$

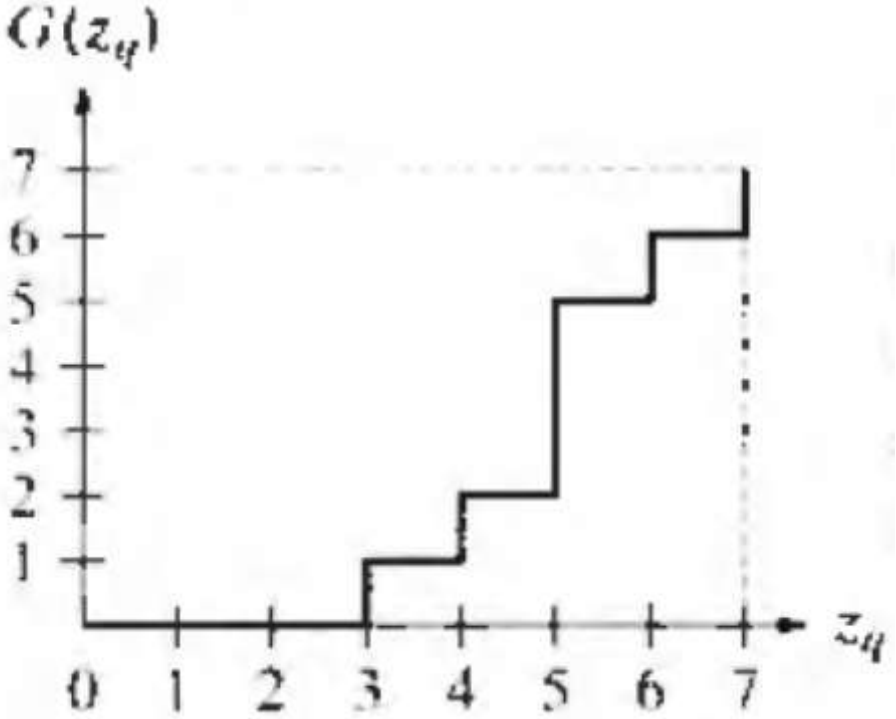
取整：

$$G(z_0)=0.00\rightarrow 0 \quad G(z_4)=2.45\rightarrow 2$$
$$G(z_1)=0.00\rightarrow 0 \quad G(z_5)=4.55\rightarrow 5$$
$$G(z_2)=0.00\rightarrow 0 \quad G(z_6)=5.95\rightarrow 6$$
$$G(z_3)=1.05\rightarrow 1 \quad G(z_7)=7.00\rightarrow 7$$

映射结果如下表：

z_q	$G(z_q)$
$z_0=0$	0
$z_1=1$	0
$z_2=2$	0
$z_3=3$	1
$z_4=4$	2
$z_5=5$	5
$z_6=6$	6
$z_7=7$	7

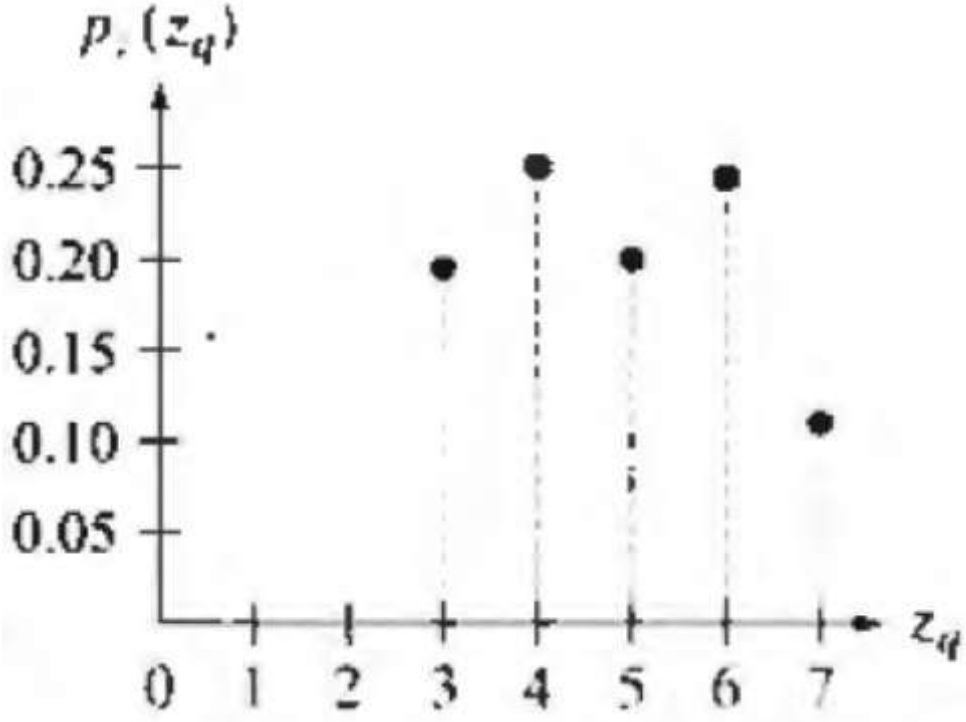
对应的变换函数如下图：



可以看出G并不是严格单调的，令$G(z_q)$尽可能接近s_k，例如，$s_0=2$，$G(z_3)=1$，所以有映射$s_0 \to z_3$，也即此情况下，直方图均衡后的图像中每个值为1的像素映射为直方图规定化后的图像中响应位置值为3的像素，继续，得到下表所示的映射：

s_k	→	z_q
1	→	3
3	→	4
5	→	5
6	→	6
7	→	7

输出图像的直方图如下所示：



灰度图的直方图匹配C++代码如下，彩色图应该就是分通道匹配再合并吧（其中目标直方图分布比较简单，代码中有体现）：

```
void hisMatch()
{
 Mat srcImage = imread("test4.JPG", 0);
 float zHist[256];
 for (int i = 0; i < 256; i++)
 {
  if (i < 128)
   zHist[i] = 1.5 / 256;
  else
   zHist[i] = 0.5 / 256;
```

```
}
Mat dstImage;
srcImage.copyTo(dstImage);
int h = srcImage.rows;
int w = srcImage.cols;
int hist[256] = { 0 };//各级像素数目
int S[256] = { 0 };
map<int, int> S2Z;//S（均衡化灰度）到Z（输出图像灰度）的映射
map<int, int> R2Z;//R（原始图像灰度）到Z（输出图像灰度）的映射
//直方图统计
for (int i = 0; i < h; i++)
{
 uchar* p = srcImage.ptr<uchar>(i);
 for (int j = 0; j < w; j++)
 {
  int value = p[j];
  hist[value]++;
 }
}
//归一化累加直方图
float sumHist[256] = { 0 };
for (int i = 0; i < 256; i++)
{
 int sum = 0;
 for (int j = 0; j <= i; j++)
  sum += hist[j];
 sumHist[i] = sum * 1.0 / (h * w);
}
//根据sumHist建立均衡化后的灰度级数组S
for (int i = 0; i < 256; i++)
 S[i] = 255 * sumHist[i] + 0.5;
//根据zSumHist建立均衡化后灰度级数组G
int G[256] = { 0 };
float zSumHist[256] = { 0.0 };
for (int i = 0; i < 256; i++) {
 float sum = 0;
 for (int j = 0; j <= i; j++)
  sum += zHist[j];
 zSumHist[i] = sum;
}
for (int i = 0; i < 256; i++)
 G[i] = zSumHist[i] * 255 + 0.5;

//令G(Z)=S 建立S->Z的映射表
```

```
for (int i = 0; i < 256; i++) {
 for (int j = 1; j < 256; j++) {
  //G[i]递增，只需满足下面的判断条件，即为最接近
  if (abs(S[i] - G[j - 1]) < abs(S[i] - G[j]))
  {
   S2Z[S[i]] = j - 1;
   break;
  }
 }
}
S2Z[S[255]] = 255;
//建立R->Z的映射
for (int i = 0; i < 256; i++)
 R2Z[i] = S2Z[S[i]];
//重建图像
for (int i = 0; i < h; i++)
{
 uchar* pdata = dstImage.ptr<uchar>(i);
 for (int j = 0; j < w; j++)
  *(pdata + j) = R2Z[*(pdata + j)];
}
imshow("ori", srcImage);
imshow("dst", dstImage);
waitKey(0);
}
```

欢迎扫描二维码关注微信公众号 深度学习与数学 [每天获取免费的大数据、AI等相关的学习资源、经典和最新的深度学习相关的论文研读，算法和其他互联网技能的学习，概率论、线性代数等高等数学知识的回顾]