第3章

数字图像的运算

3.1 点 运 算

点运算(Point Operation)是指对一幅图像中部分像素点的灰度值进行计算。它将输入图像映射为输出图像,输出图像中每个像素点的灰度值仅由对应的输入像素点的灰度值决定,运算结果不会改变图像内像素点之间的空间关系。设输入图像在坐标(x,y)处的灰度值为$f(x,y)$,输出图像在坐标(x,y)处的灰度值为$g(x,y)$,则点运算为

$$g(x,y)=T[f(x,y)] \tag{3-1}$$

其中,T是对$f(x,y)$的一种数学运算,即点运算是一种像素的逐点运算,是灰度到灰度的映射过程,通常称T为灰度变换函数。

点运算又称为“对比度增强”“对比度拉伸”“灰度变换”等。按灰度变换函数T的性质,可将点运算分为以下两类。

(1) 灰度变换增强,又包括线性灰度变换(线性点运算)、分段线性灰度变换(分段线性点运算)、非线性灰度变换(非线性点运算)。

(2) 直方图增强。

灰度变换是图像增强的重要手段之一,用于改善图像显示效果,属于空间域处理方法,它可以使图像的动态范围加大,图像的对比度扩展,图像更加清晰,特征更加明显。灰度变换的实质是按一定的规则修改图像每个像素的灰度,从而改变图像的灰度范围。

下面给出了灰度值增加的C++代码,点运算示例如图3-1所示,其中,显示的示例图为每个像素增加给定的灰度值的效果。

C++示例代码3-1:灰度值增加

```
/*
将图像中的每个像素的蓝、绿、红3个分量的灰度值分别加上blue、green、red,以增强图像的亮度。注意:若灰度值增加后超过255,则直接取255。
*/
void CFgImage::Add(int blue,int green,int red){
    for (int row = 0; row < m_height; row++){
        for (int col = 0; col < m_width; col++){
```

```
            BYTE * p = m_pBmpBuf + (m_height-row-1) * m_widthStep + col * 3;

            BGR bgr;
            bgr.Blue = *p;
            bgr.Green = *(p + 1);
            bgr.Red = *(p + 2);

            if (bgr.Blue + blue>=255)
                *p = 255;
            else
                *p = (bgr.Blue + blue);
            if (bgr.Green + green >= 255)
                *(p + 1) = 255;
            else
                *(p + 1) = (bgr.Green + green);

            if (bgr.Red + red >= 255)
                *(p + 2) = 255;
            else
                *(p + 2) = (bgr.Red + red);
        }
    }
}
```

(a) 原图

(b) 应用程序之后的效果

图 3-1　点运算示例

3.1.1　线性灰度变换

线性灰度变换又称为线性点运算，主要有对比度增强等。假定原图像 $f(x,y)$ 的灰度范围为 $[a,b]$，若希望变换后的图像 $g(x,y)$ 的灰度范围扩展为 $[c,d]$，则其函数表现形式如式(3-2)所示。线性灰度变换如图 3-2 所示。

$$g(x,y)=\frac{d-c}{b-a}[f(x,y)-a]+c \tag{3-2}$$

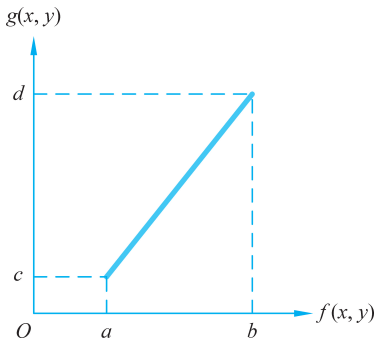

图 3-2　线性灰度变换

下面给出了线性灰度变换的 C++ 代码实现，图 3-3 为线性灰度变换示例。

C++ 示例代码 3-2：线性灰度变换

```
void CFgImage::GrayTrans1(double a, double b, double c, double d) {
    struct TranFunc{
        double a, b, c, d;
        TranFunc(double a, double b, double c, double d) :a(a), b(b), c(c), d(d){

        }
        BYTE func(int gray){
            double val = (d - c) * (gray - a) / (b - a) + c;
            return saturate_cast(val);
        }
        BGR operator()(BYTE Blue, BYTE Green, BYTE Red){
            BGR bgr;
            bgr.Blue = func(Blue);
            bgr.Green = func(Green);
            bgr.Red = func(Red);
            return bgr;
        }
    };

    GrayTrans(TranFunc(a, b, c, d));
}

template <typename Func> void CFgImage::GrayTrans(Func& func) {
    for (int row = 0; row < m_height; row++){
        for (int col = 0; col < m_width; col++){
            BYTE * p = m_pBmpBuf + row * m_widthStep + col * 3;
```

```
                BGR bgr = func( * p, * (p + 1), * (p + 2));
                * p = bgr.Blue;
                * (p + 1) = bgr.Green;
                * (p + 2) = bgr.Red;
            }
        }
    }
```

图 3-3　线性灰度变换示例

3.1.2　分段线性灰度变换

为了突出图像中用户感兴趣的目标或者灰度区间，相对抑制那些用户不感兴趣的灰度区域而不牺牲其他灰度级上的细节，可以采用分段线性灰度变换，它可将需要的图像细节灰度拉伸，增强对比度，而将不需要的细节灰度级压缩。

分段线性灰度变换是将输入图像 $f(x,y)$ 的灰度级区间分成两段乃至多段，然后分别对每一段做线性灰度变换，以获得增强图像 $g(x,y)$。典型的 3 段线性灰度变换如图 3-4 所示，其函数表达形式如下。

$$g(x,y)=\begin{cases}\dfrac{c}{a}f(x,y) & 0<f(x,y)<a\\ \dfrac{d-c}{b-a}[f(x,y)-a]+c & a\leqslant f(x,y)\leqslant b\\ \dfrac{G_{\max}-d}{F_{\max}-b}[f(x,y)-b]+d & b<f(x,y)\leqslant F_{\max}\end{cases}\tag{3-3}$$

其中，参数 a、b、c、d 表示用于确定 3 条线段斜率的常数，取值可根据具体变换设定。

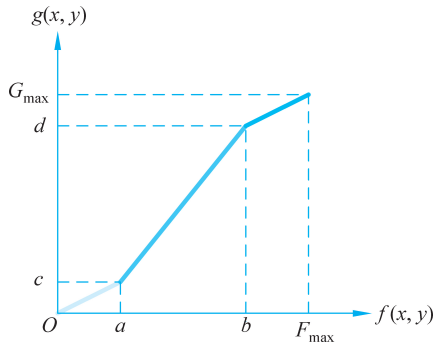

图 3-4　三段线性灰度变换

此外，也存在一些情况，即用户可能仅对某个范围内的灰度感兴趣，这时只需对其进行线性拉伸，以便清晰化。例如：

(1) 在式(3-3)中，当 $0<f(x,y)<a$ 和 $b<f(x,y)<F_{max}$ 时，$g(x,y)=f(x,y)$，这表示对区间$(0,a)$和(b,F_{max})范围内的像素不做变换，而将处于$[a,b]$的原图像的灰度线性地变换成新图像的灰度。

(2) 在式(3-3)中，当 $0<f(x,y)<a$ 时，$g(x,y)=c$；当 $b<f(x,y)<F_{max}$ 时，$g(x,y)=d$，这表示只将处于$[a,b]$的原图像的灰度线性地变换成新图像的灰度，而对于$[a,b]$以外的像素，则将原图像的灰度强行压缩为灰度 c 和 d。

下面给出了分段线性灰度变换的 C++ 代码示例，图 3-5 为分段线性灰度变换示例。

C++ 示例代码 3-3：分段线性灰度变换

```
void CFgImage::GrayTrans2(double a, double b, double c, double d, int G_max,
int F_max) {
    struct TranFunc{
        double a, b, c, d;
        int G_max, F_max;
        TranFunc(double a, double b, double c, double d, int G_max, int F_max) :
                        a(a), b(b), c(c), d(d), G_max(G_max), F_max(F_max){

        }
        BYTE func(int gray){
            double val = 0;
            if (gray < a) val = c * gray / a;
            else if (gray <= b) val = (d - c) * (gray - a) / (b - a) + c;
            else val = (G_max - d) * (gray - b) / (F_max - b) + d;
            return saturate_cast(val);
        }
        BGR operator()(BYTE Blue, BYTE Green, BYTE Red){
            BGR bgr;
```

```
                bgr.Blue = func(Blue);
                bgr.Green = func(Green);
                bgr.Red = func(Red);
                return bgr;
            }
        };
        GrayTrans(TranFunc(a, b, c, d, G_max, F_max));
    }
```

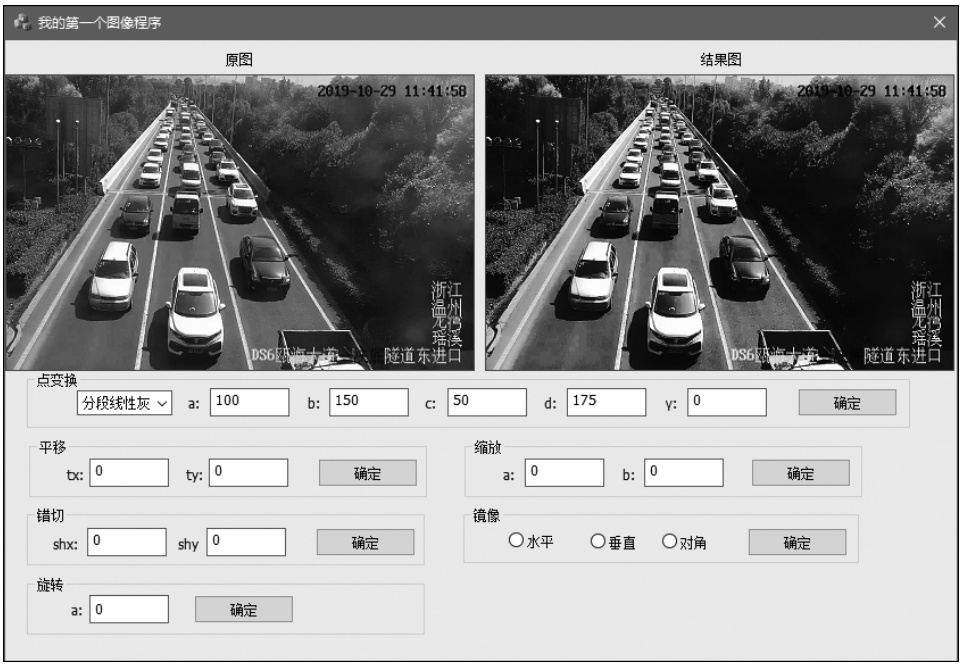

图 3-5　分段线性灰度变换示例

3.1.3　非线性灰度变换

当用某些非线性变化函数作为灰度变换的变换函数时，可实现图像灰度的非线性变换。对数变换、指数变换和幂次变换是常见的非线性变换。

1. 对数变换

对数变换如图 3-6 所示，其函数形式如下所示。

$$g(x,y)=c\cdot\log[f(x,y)+1] \tag{3-4}$$

其中，c 是尺度比例常数，其取值可以结合输入图像的范围来定。$f(x,y)$的取值为$[f(x,y)+1]$是为了避免对 0 求对数，确保 $\log[f(x,y)+1]\geqslant 0$。

当希望对图像的低灰度区做较大拉伸，对高灰度区做压缩时，可采用这种变换，它能使图像的灰度分布与人的视觉特性相匹配。对数变换一般用于处理过暗图像。

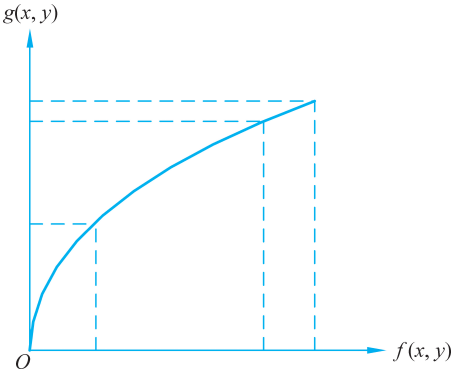

图 3-6　对数变换

下面给出了对数变换的 C++ 代码实现，图 3-7 为对数变换示例。

C++ 示例代码 3-4：对数变换

```
void CFgImage::GrayTrans3(double c) {
    struct TranFunc{
        double c;
        TranFunc(double c) :c(c){

        }
        BYTE func(int gray){
            double val = c * std::log(gray + 1);
            return saturate_cast(val);
        }
        BGR operator()(BYTE Blue, BYTE Green, BYTE Red){
            BGR bgr;
            bgr.Blue = func(Blue);
            bgr.Green = func(Green);
            bgr.Red = func(Red);
            return bgr;
        }
    };
    GrayTrans(TranFunc(c));
}
```

2. 指数变换

指数变换如图 3-8 所示，函数形式如下所示。

$$g(x,y)=b^{c[f(x,y)-a]}-1 \tag{3-5}$$

其中，a 用于决定指数变换函数曲线的初始位置；当 $f(x,y)=a$ 时，$g(x,y)=0$，曲线与 x 轴相交；b 是底数；c 用于决定指数变换曲线的陡度。

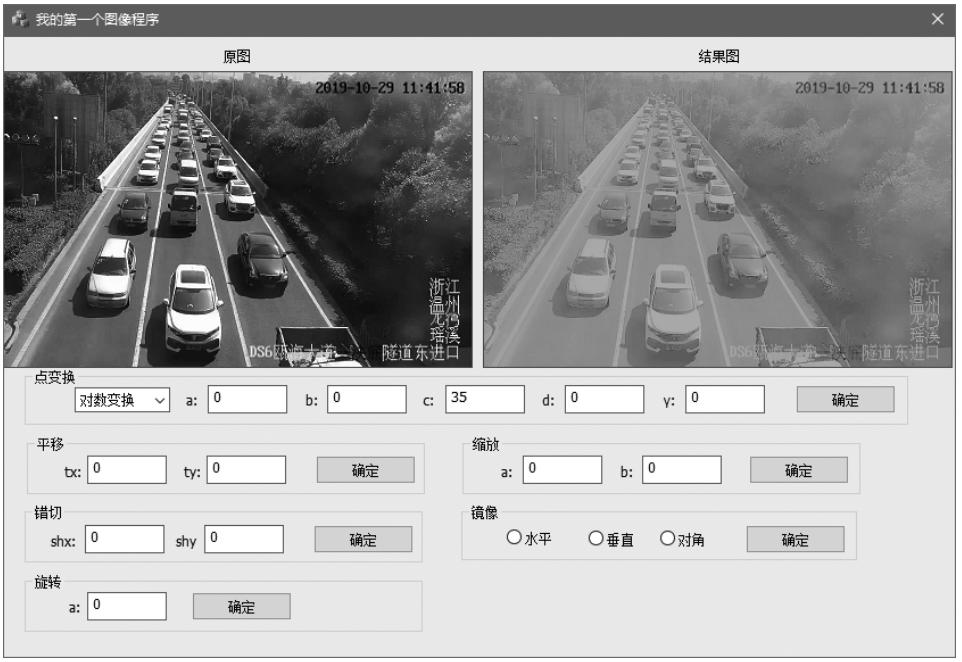

图 3-7　对数变换示例

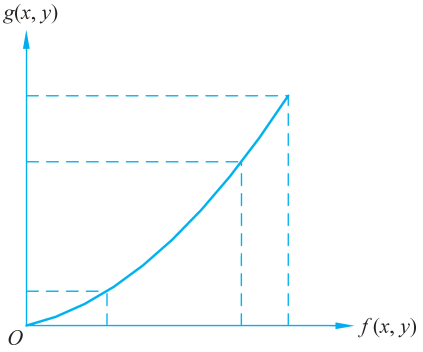

图 3-8　指数变换

当希望对图像的低灰度区做压缩，对高灰度区做较大拉伸时，可采用这种变换。指数变换一般适用于处理过亮图像。对数变换常用来扩展低值灰度，压缩高值灰度，这样可使低值灰度的图像细节更清晰。

下面给出了指数变换的 C++ 代码实现，图 3-9 为指数变换示例。

C++ 示例代码 3-5：指数变换

```
void CFgImage::GrayTrans4(double a, double b, double c) {
    struct TranFunc{
        double a,b,c;
        TranFunc(double a, double b, double c) :a(a), b(b), c(c){
```

```
        }
        BYTE func(int gray){
            double val = std::pow(b, c * (gray - a)) - 1;
            return saturate_cast(val);
        }
        BGR operator()(BYTE Blue, BYTE Green, BYTE Red){
            BGR bgr;
            bgr.Blue = func(Blue);
            bgr.Green = func(Green);
            bgr.Red = func(Red);
            return bgr;
        }
    };
    GrayTrans(TranFunc(a, b, c));
}
```

图 3-9　指数变换示例

3. 幂次变换

幂次变换的函数形式如下所示。

$$g(x,y)=c[f(x,y)]^{\gamma} \tag{3-6}$$

其中，c 和 γ 为正的常数，当 c 取 1、γ 取不同值时，可得到一簇变换曲线，如图 3-10 所示。

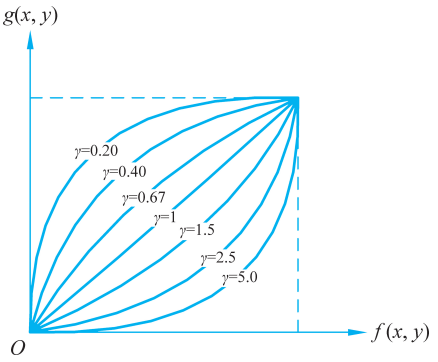

图 3-10　幂次变换

与对数变换的情况类似，幂次变换可以将部分灰度区域映射到更宽的区域中，从而增强图像的对比度。当 $\gamma=1$ 时，幂次变换变成线性正比变换；当 $0<\gamma<1$ 时，幂次变换可以扩展原始图像的中、低灰度级，压缩高灰度级，从而使得图像变亮，增强原始图像中暗区的细节；当 $\gamma>1$ 时，幂次变换可以扩展原始图像的中、高灰度级，压缩低灰度级，从而使得图像变暗，增强原始图像中亮区的细节。

幂次变换常用于图像获取、打印和显示的各种转换设备的伽马校正中，这些设备的光电转换特性都是非线性的，是根据幂次规律产生响应。幂次变换的指数即伽马值，因此幂次变换也称为伽马变换。

下面给出了幂次变换的 C++ 代码实现，图 3-11 为幂次变换示例。

C++ 示例代码 3-6：幂次变换

```
void CFgImage::GrayTrans5(double c, double gamma) {
    struct TranFunc{
        double c, gamma;
        TranFunc(double c, double gamma) :c(c), gamma(gamma){

        }
        BYTE func(int gray){
            double val = c * std::pow(gray, gamma);
            return saturate_cast(val);
        }
        BGR operator()(BYTE Blue, BYTE Green, BYTE Red){
            BGR bgr;
            bgr.Blue = func(Blue);
            bgr.Green = func(Green);
            bgr.Red = func(Red);
            return bgr;
        }
    };
```