C#处理数码相片之马赛克的实现 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/245/2021_2022_C_23_E5_A4_84_E7_90_86_E6_95_c97_245976.htm 很多图片处理的算法从原理上讲其实非常简单，难点往往在如何去写算法实现它，更加难的就是如何去优化实现的算法。虽说我一向认为程序员的效率比程序的效率更重要，但为了等处理一张自己拍摄的数码照片，溜出去买杯奶茶顺便再买张彩票回来发现还没算好，无论如何都是不能忍受的。 马赛克算法很简单，说白了就是把一张图片分割成若干个val * val像素的小区块（可能在边缘有零星的小块，但不影响整体算法），每个小区块的颜色都是相同的。为了方便起见，我们不妨让这个颜色就用该区域最左上角的那个点的颜色。当然还可以有其他方法，比如取区块中间点的颜色，或区块中随机点的颜色作代表等等。 下面的示意图就是取val=2的结果。 原图像素 ABCDEFG HIJKLMN OPQRSTU VWXYZ01 2345678 马赛克处理后 AACCEEG AACCEEG OOQQSSU OOQQSSU 2244668 原理就是那么简单。具体实现就看各人的思维习惯了。我的想法是：当y（当前高度）是val的整数倍时：扫描当前行中的每一点x，如果x也是val的整数倍，记录下当前x,y的颜色值；如果x不是val的整数倍，则沿用最近一次被记录的颜色值。 当y不是val的整数倍：很简单，直接复制上一行。 简单的说就是以线带面，最终实现让大家都看不清楚 下面就是源代码。写算法不是我的强项，不过偶尔勉为其难的写个可以跑跑的不求甚解版还是可以做到的，不指望可以帮到你，只希望没有误导你。 public static Bitmap KiMosaic(Bitmap b, int val) { if

(b.Equals(null)) { return null. } int w = b.Width. int h = b.Height. int stdR, stdG, stdB. stdR = 0. stdG = 0. stdB = 0. BitmapData srcData = b.LockBits(new Rectangle(0, 0, w, h), ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb). unsafe { byte* p = (byte*)srcData.Scan0.ToPointer(). for (int y = 0. y { for (int x = 0. x { if (y % val == 0) { if (x % val == 0) { stdR = p[2]. stdG = p[1]. stdB = p[0]. } else { p[0] = (byte)stdB. p[1] = (byte)stdG. p[2] = (byte)stdR. } } else { // 复制上一行 byte * pTemp = p - srcData.Stride. p[0] = (byte)pTemp[0]. p[1] = (byte)pTemp[1]. p[2] = (byte)pTemp[2]. } p = 3. } // end of x p = srcData.Stride - w * 3. } // end of y b.UnlockBits(srcData). } return b. } 