**使用说明书**

目录

**PPM、PGM和PBM图像处理**

**YUV图像处理**

**RAW图像处理**

**BMP图像处理**

**其它处理**

**高级算子**

**PPM、PGM和PBM图像处理**

|  |  |
| --- | --- |
| void OTSUBinarization(char\* input,char\* output) | OTSU二值化。input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void PPMtoBMP(char\* input,char\* output,int bpp) | PPM图像转为BMP图像。bpp是BMP图像的色深。 |
| void BMPtoPPM(char\* input,char\* output) | BMP图像转PPM图像。 |
| void PPMtoBMP1(char\* input,char\* output,int bpp) | PPM图像转为BMP图像。bpp是BMP图像的色深。 |
| void BMPtoPPM1(char\* input,char\* output) | BMP图像转PPM图像。 |
| void BMPtoPGM(char\* input, char\* output) | BMP转PGM。 |
| void BMPtoPPM2(char\* input, char\* output) | BMP转PPM。 |
| void PPMtoPGM(char\* input, char\* output) | PPM转PGM。 |
| void BlurPPM(char\* input, char\* output) | PPM图像滤波。 |
| void BlurPGM(char\* input, char\* output) | PGM图像滤波。 |
| void SegmentsOTSUBinarization(char\* input,char\* output) | OTSU二值化划分。input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void P3PPMBlur(char\* input,char\* output) | PPM图像模糊，input是输入文件名，output是输出文件名。支持P3格式的PPM图像。 |
| unsigned char\*\* ReadPBM(char\* input) | 读取PBM图像并返回图像数据。input是要读取的PBM图像文件名。支持P4格式的PBM图像。 |
| void WritePBM(unsigned char\*\* Input,char\* output) | 保存PBM图像。Input是输入的图像数据，output是输出文件名。支持P4格式的PBM图像。 |
| void PGMHistogramEqualization(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| PPMImage\* ReadPPM(char\* input) | PPM图像读取，input是要读取的PPM图像文件名。支持P6格式的PPM图像。  需要引入的结构体：  typedef struct {  unsigned char red, green, blue; //像素的颜色由RGB（红/绿/蓝）表示  } PPMPixel;  typedef struct {  unsigned int width, height; // 图像的宽度和高度（以像素为单位）  PPMPixel \*data; // 构成图像的像素  } PPMImage; |
| void WritePPM(char\* output,PPMImage\* img) | PPM图像保存， output是输出的PPM图像文件名，img是输入的图像数据。支持P6格式的PPM图像。  需要引入的结构体：  typedef struct {  unsigned char red, green, blue; //像素的颜色由RGB（红/绿/蓝）表示  } PPMPixel;  typedef struct {  unsigned int width, height; // 图像的宽度和高度（以像素为单位）  PPMPixel \*data; // 构成图像的像素  } PPMImage; |
| void InvertColor(char\* input,char\* output) | 负滤波器，input是输入文件名，output是输出文件名。支持P6格式的PPM图像。 |
| void GrayFilter(char\* input,char\* output) | 灰度过滤器，input是输入文件名，output是输出文件名。支持P6格式的PPM图像。 |
| void SepiaFilter(char\* input,char\* output) | 乌贼墨过滤器，input是输入文件名，output是输出文件名。支持P6格式的PPM图像。 |
| void AdjustSaturation(char\* input,char\* output,double a) | 调整图像饱和度，input是输入文件名，output是输出文件名。a是目标饱和度，如a=30。支持P6格式的PPM图像。 |
| void Resize(char\* input,char\* output,unsigned int NewWidth, unsigned int NewHeight) | 调整图像大小，input是输入文件名，output是输出文件名。NewWidth和NewHeight分别是输出图像的宽和高。支持P6格式的PPM图像。 |
| void AdjustHue(char\* input,char\* output,int a) | 调整图像的色调，input是输入文件名，output是输出文件名。a是目标色调，如a=125。支持P6格式的PPM图像。 |
| void AdjustBrightness(char\* input,char\* output,double a) | 调整图像亮度，input是输入文件名，output是输出文件名。a是目标亮度，如a=60。支持P6格式的PPM图像。 |
| void AdjustContrast(char\* input,char\* output,double a) | 调整图像对比度，input是输入文件名，output是输出文件名。a是目标对比度，如a=60。支持P6格式的PPM图像。 |
| void AdjustBlur(char\* input,char\* output,double a) | 通过sigma因子模糊图像，input是输入文件名，output是输出文件名。a是sigma因子，如a=5。支持P6格式的PPM图像。 |
| void MeanGrayFilter(char\* input,char\* output,double a) | 平均灰度滤波器，input是输入文件名，output是输出文件名。a是平均值系数，如a=3。支持P6格式的PPM图像。 |
| void Pixelate(char\* input,char\* output,unsigned int a) | 像素化，input是输入文件名，output是输出文件名。a是幅度值，如a=8。支持P6格式的PPM图像。 |
| void Rotate(char\* input,char\* output,short a) | 旋转图像，input是输入文件名，output是输出文件名。a是旋转的角度，如a=45。支持P6格式的PPM图像。 |
| void GammaCorrection(char\* input,char\* output,double a) | 伽马校正，input是输入文件名，output是输出文件名。a是gamma数，如a=0.5。支持P6格式的PPM图像。 |
| void GrayAndChannelSeparation(char\* input,char\* Grayoutput,char\* Routput,char\* Goutput,char\* Boutput) | 生成灰度图以及RGB通道分离，input是输入的P6格式的PPM图像；Grayoutput是输出的灰度图文件名，Routput、Goutput和Boutput分别是输出的R、G和B通道的图像文件名，输出都是PGM格式文件。 |
| void PGMBin(char\* input,char\* output,int threshold) | 灰度图像二值化，输入是灰度图像，输入和输出都是PGM文件，threshold是阈值，如threshold=125。 |
| void Brightening(char\* input,char\* output,int a) | 彩色图像增亮，输入和输出都是P6格式的PPM图像，a是增亮系数，如a=80。 |
| void GrayBrightening(char\* input,char\* output,int a) | 灰度图像增亮，输入和输出都是PGM图像，a是增亮系数，如a=80。 |
| void PPMFilter(char\* input,char\* output) | 彩色图像滤波，输入和输出都是P6格式的PPM文件。 |
| void PGMGrayFilter(char\* input,char\* output) | 灰度图像滤波，输入和输出都是PGM图像。 |
| void PPMtoBMP(char\* input,char\* output) | PPM图像转BMP图像，input是输入文件名，output是输出文件名。支持P6格式的PPM图像。 |
| void PGMRotated(char\* input,char\* output,int width,int height,int channels,double theta) | 图像旋转，channels是输入图像的通道，theta是旋转弧度，支持PGM图像。参考：theta=45.0\*3.1415926/180。 |
| void XCorner(char\* input,char\* output,int width,int height,int channels,double theta) | X方向角度，channels是输入图像的通道。支持PGM图像。 |
| void YCorner(char\* input,char\* output,int width,int height,int channels,double theta) | Y方向角度，channels是输入图像的通道。支持PGM图像。 |
| void Smooth(char\* input,char\* output,int width,int height,int channels,float sigma\_x,float sigma\_y,double theta) | 图像模糊，channels是输入图像的通道，sigma\_x是X方向的模糊系数，sigma\_y是Y方向的模糊系数。 |
| void PGMOtsuThreshold(string input,char\* output) | 大津阈值法，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void PGMLocalisedOtsuThreshold(string input,char\* output) | 局部大津阈值，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void PGMSauvolaThreshold(string input,char\* output,double a,double b,double c) | 索沃拉阈值，支持P5格式的PGM图像。a、b和c的参考值如：a=0.01,b=15,c=225。 |
| void PGMThreshold(string input,char\* output,int thresh) | 阈值法，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。thresh是阈值，如：thresh=5。 |
| float Repair1(char\* input,char\* output,float var,float threshold,int nbLevels,float a) | 图像修复，var是噪声方差，threshold是阈值，nbLevels是要处理的级别数，a=10。返回ISNR。 |
| float Repair2(char\* input,char\* output,float var,float threshold,int nbLevels,float a) | 图像修复，var是噪声方差，threshold是阈值，nbLevels是要处理的级别数，a=10。返回ISNR。 |
| void LowPassFilterRepair1(char\* input,char\* output,int size\_filter,float var,int nb\_iterations,int nbLevels,float a,int b) | 低通滤波图像修复，a=10，b=6，nbLevels=3，size\_filter是低通过滤器的大小，var是噪声方差，nb\_iterations是Landweber的迭代数。 |
| void LowPassFilterRepair2(char\* input,char\* output,int size\_filter,float var,int nb\_iterations,int nbLevels,float a,int b) | 低通滤波图像修复，a=10，b=6，nbLevels=3，size\_filter是低通过滤器的大小，var是噪声方差，nb\_iterations是Landweber的迭代数。 |
| float LowPassFilterRepair3(char\* input,char\* output,int size\_filter,float var,int nb\_iterations,int nbLevels,int pas,float a,int b) | 低通滤波图像修复，a=10，b=6，nbLevels=3，pas=1，size\_filter是低通过滤器的大小，var是噪声方差，nb\_iterations是Landweber的迭代数。返回ISNR。 |
| void Repair1(char\* input,char\* output,int M,float a) | 图像修复，a=0.0，M是分解的层次数，如M=3。 |
| void Repair2(char\* input,char\* output,int M,float a) | 图像修复，a=0.0，M是分解的层次数，如M=3。 |
| void MakeNoise1(char\* input,char\* output,int size\_filter) | 制造噪声，size\_filter是低通滤波器的宽度。 |
| void MakeNoise2(char\* input,char\* output,int nb\_iterations,int pas) | 制造噪声，nb\_iterations是Landweber的迭代数，pas=1。 |
| void MakeNoise3(char\* output,int height,int width,float var) | 制造噪声，height是输出图像的高，width是输出图像的宽，var是噪声方差。 |
| void MakeNoise4(char\* input,char\* output,int nb\_iterations,int pas) | 制造噪声，nb\_iterations是Landweber的迭代数，pas=1。 |
| void ImageReconstruction(char\* input,char\* output,int maxDepth,int threshold,int tx,int ty) | 图像重建，支持PGM文件。参考：maxDepth=80，threshold=50，tx=0，ty=0。 |

**YUV图像处理**

|  |  |
| --- | --- |
| void YUVsuperposition(char\* input1,char\* input2,char\* output,int width,int height,unsigned char Y\_BLACK,unsigned char U\_BLACK,unsigned char V\_BLACK) | YUV420叠加，Y\_BLACK、U\_BLACK和V\_BLACK用于将原图中的黑色变成透明，参考：Y\_BLACK=16，U\_BLACK=128，V\_BLACK=128。 |
| void YUVsuperposition(char\* input1,char\* input2,char\* output,int width,int height,unsigned char Y\_BLACK,unsigned char U\_BLACK,unsigned char V\_BLACK) | YUV444叠加，Y\_BLACK、U\_BLACK和V\_BLACK用于将原图中的黑色变成透明，参考：Y\_BLACK=16，U\_BLACK=128，V\_BLACK=128。 |
| void YUVsuperposition(char\* input1,char\* input2,char\* output,int width,int height,unsigned char Y\_BLACK,unsigned char U\_BLACK,unsigned char V\_BLACK) | yuv444p直接叠加到yuv420p上，不做转换，Y\_BLACK、U\_BLACK和V\_BLACK用于将原图中的黑色变成透明，参考：Y\_BLACK=16，U\_BLACK=128，V\_BLACK=128。 |
| void YUV444toYUV420(char\* input,char\* output,int height,int width) | YUV444转YUV420，height是输入的YUV444文件的高，width是输入的YUV444文件的宽。 |
| void YUV444toYUV420(char\* input,char\* output,int height,int width,int frames) | YUV444转YUV420，height和width是输入文件的高和宽，frames是要输入文件中操作的帧序号。 |
| void YUVsuperposition(char\* input1,char\* input2,char\* output,int width,int height,unsigned char Y\_BLACK,unsigned char U\_BLACK,unsigned char V\_BLACK) | YUV444转到YUV420上的叠加，Y\_BLACK、U\_BLACK和V\_BLACK用于将原图中的黑色变成透明，参考：Y\_BLACK=16，U\_BLACK=128，V\_BLACK=128。 |
| void YUVEdgeProcessingY(char\* input,char\* output,int width,int height,double k) | YUV边缘处理，input是输入文件名，output是输出文件名。width和height是输入图像的宽和高。参考：k=0.5。 |
| void YUVEdgeProcessingU(char\* input,char\* output,int width,int height,double k) | YUV边缘处理，input是输入文件名，output是输出文件名。width和height是输入图像的宽和高。参考：k=0.5。 |
| void YUVEdgeProcessingV(char\* input,char\* output,int width,int height,double k) | YUV边缘处理，input是输入文件名，output是输出文件名。width和height是输入图像的宽和高。参考：k=0.5。 |
| void BMPLoadedIntoYUV(char\* inputBMP,char\* inputYUV,char\* output,int YUVwidth,int YUVheight,int depth,bool mt) | YUV加载BMP，inputBMP是输入的BMP图像，inputYUV是输入的YUV图像，inputYUV起到容器的作用，YUVwidth和YUVheight是输入的YUV图像的宽和高，参考：depth=12，mt=true。 |
| void YUVEdgeProcessingHorizontalDirection(char\* input,char\* output,int width,int height,double k) | YUV仅水平方向的边缘处理，input是输入文件名，output是输出文件名。width和height是输入图像的宽和高。参考：k=0.7。 |
| void YUVVieoEdgeProcessing(char\* input,char\* output,int width,int height,int frame,int max\_frame) | YUV视频文件边缘处理，input是输入文件名，output是输出文件名。width和height是输入图像的宽和高，frame是要处理的帧序号，max\_frame是最大帧序号。 |
| void YUVScale(char\* input,char\* output,int inputWidth,int inputHeight,int outputWidth,int outputHeight) | 缩放yuv420图像，参考：inputWidth=1280，inputHeight=720，outputWidth=128，outputHeight=72。 |
| void NoiseTreatment(char\* input,char\* output,int width,int height,int TWICEwidth,int TWICEheight) | YUV噪声处理。 |
| void NoiseTreatment(char\* input,char\* output,int width,int height,int frame,int max\_frame) | YUV噪声处理。 |

**RAW图像处理**

|  |  |
| --- | --- |
| unsigned char\*\* RAWRead(char\* input,int height,int width) | 读取RAW图像。 |
| void RAWWrite(unsigned char\*\* input,char\* output,int height,int width) | 保存RAW图像。 |
| void MBVQ(char\* input,char\* output,int width,int height) | MBVQ效果，input是输入文件名，output是输出文件名。width和height是输出图像的宽和高。 |
| void RAWtoPPM\_red(char\* input,char\* output,int width, int height,DebayerAlgorithm algo) | RAW转为PPM后提取红色通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR或LINEAR。支持RAW12格式。  需引入以下枚举：  enum DebayerAlgorithm {  NEARESTNEIGHBOUR,  LINEAR  }; |
| void RAWtoPPM\_green1(char\* input,char\* output,int width, int height,DebayerAlgorithm algo) | RAW转为PPM后提取绿色1通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR或LINEAR。  支持RAW12格式。  需引入以下枚举：  enum DebayerAlgorithm {  NEARESTNEIGHBOUR,  LINEAR  }; |
| void RAWtoPPM\_green2(char\* input,char\* output,int width, int height,DebayerAlgorithm algo) | RAW转为PPM后提取绿色2通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR或LINEAR。  支持RAW12格式。  需引入以下枚举：  enum DebayerAlgorithm {  NEARESTNEIGHBOUR,  LINEAR  }; |
| void RAWtoPPM\_blue(char\* input,char\* output,int width, int height,DebayerAlgorithm algo) | RAW转为PPM后提取蓝色通道，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR或LINEAR。  支持RAW12格式。  需引入以下枚举：  enum DebayerAlgorithm {  NEARESTNEIGHBOUR,  LINEAR  }; |
| void RAWtoPPM(char\* input,char\* output,int width, int height,DebayerAlgorithm algo) | RAW转为PPM，参考：width=4096，height=3072，algo=NEARESTNEIGHBOUR或LINEAR。  支持RAW12格式。  需引入以下枚举：  enum DebayerAlgorithm {  NEARESTNEIGHBOUR,  LINEAR  }; |
| void RawPowerTransformation(char\* input,char\* output,int width,int height,int c,float v) | 幂次变换，input是输入的RAW图像文件名，output是输出的RAW图像文件名，width是输入图像的宽，height是输入图像的高。默认c=1，v=0.6。支持RAW图像。 |
| void RAWAvgFilter(char\* input,char\* output,int ROWS,int COLS,int M,float mask[3][3]) | 平均滤波器，input是输入文件名，output是输出文件名。ROWS是图像的行大小，COLS是图像的列大小，M是滤波相关参数，如M=1；mask是滤波器模板。支持RAW图像。  参考模板：  float mask[3][3] = {{0.1111,0.1111,0.1111},  {0.1111,0.1111,0.1111},  {0.1111,0.1111,0.1111}}; |
| void RawImageInversion(char\* input,char\* output,int width,int height) | 图像反相，input是输入的RAW图像文件名，output是输出的RAW图像文件名，width是输入图像的宽，height是输入图像的高。支持RAW图像。 |
| void RawHistogramEqualization(char\* input,char\* output,int width,int height) | 直方图均衡化，input是输入的RAW图像文件名，output是输出的RAW图像文件名，width是输入图像的宽，height是输入图像的高。支持RAW图像。 |
| void RAWHistogramEqualization(char\* input,char\* output,int width,int height) | RAW直方图均衡化，width和height是输入图像的宽和高。 |
| void RAWMedianFilter(char\* input,char\* output,int ROWS,int COLS,int M,int sequence[9]) | 中值滤波，input是输入文件名，output是输出文件名。ROWS是图像的行，COLS是图像的列，M是滤波相关参数，如M=1。支持RAW图像。  参考模板：  int sequence[9]={0,0,0,0,0,0,0,0,0}; |
| void RawtoBmp1(char\* input, char\* output,unsigned long Width, unsigned long Height) | RAW图像转为BMP图像，input是输入文件名，output是输出文件名。Width和Height是输入文件的宽和高。 |
| void RawToBmp(char\* input,char\* output,int imageWidth,int imageHigth) | RAW图像转为BMP图像，input是输入文件名，output是输出文件名。支持宽和高相等的图像。 |
| void RGBtoHSI(char\* input,char\* output) | RGB模型转为HIS模型，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void RAWLaplacialSharpeningFilter(char\* input,char\* output,int ROWS,int COLS,int M,float w,float mask[3][3]) | 拉普拉斯锐化滤波器，input是输入文件名，output是输出文件名。ROWS是图像的行大小，COLS是图像的列大小，M和w是滤波相关参数，如M=1，w=1；mask是滤波器模板。支持RAW图像。  参考模板：  float mask[3][3] = {{0,1,0},  {1,-4,1},  {0,1,0}}; |
| void RawLaplacianEnhancement(char\* input1,char\* output1,int width,int height) | 拉普拉斯算子增强，input1是输入的RAW图像文件名，output1是输出的RAW图像文件名，width是输入图像的宽，height是输入图像的高。支持RAW图像。 |
| void CyanGray(char\* input,char\* output,int width,int height) | 青色灰度图像。 |
| void MagentaGray(char\* input,char\* output,int width,int height) | 品红灰度图像。 |
| void YellowGray(char\* input,char\* output,int width,int height) | 黄色灰度图像。 |
| void Transfer(char\* input,char\* output,int width,int height) | 传递函数。 |
| void Homography(char\* input1,char\* input2,char\* input3,char\* output,int width,int height,int newwidth,int newheight) | 单应。 |
| void MovieEffect(char\* input,char\* output,int width,int height) | 电影效果。 |
| void FixedThresholdMethod(char\* input,char\* output,int width,int height) | 抖色处理，固定阈值法。 |
| void RandomThresholdMethod(char\* input,char\* output,int width,int height) | 抖色处理，随机阈值法。 |
| void DitherMatrixMethod(char\* input,char\* output,int width,int height,int N) | 抖色处理，抖动矩阵法，默认N=2。 |
| void NormalizedLogBuffer1(char\* input,char\* output,int width,int height) | 对数变换，规范化对数。 |
| void NormalizedLogBuffer2(char\* input,char\* output,int width,int height) | 对数变换，规范化对数。 |
| void TernaryGrayLevel1(char\* input,char\* output,int width,int height) | 三值灰度。 |
| void TernaryGrayLevel2(char\* input,char\* output,int width,int height) | 三值灰度。 |
| void BestEdgeMap1(char\* input,char\* output,int width,int height) | 最佳边贴图。 |
| void BestEdgeMap2(char\* input,char\* output,int width,int height) | 最佳边贴图。 |
| void Skeletonize(char\* input,char\* output,int width,int height) | 骨架化。 |
| void SeparableDiffusion(char\* input,char\* output,int width,int height) | 可分离扩散。 |
| void Denoising(char\* input1,char\* input2,char\* output,int width,int height) | 去除噪声。 |
| void Luminosity(char\* input,char\* output,int width,int height) | 亮度调整。 |
| void Average(char\* input,char\* output,int width,int height) | 平均化。 |
| void MinMax(char\* input,char\* output,int width,int height) | 最小与最大。 |
| void Shrink(char\* input,char\* output,int width,int height) | 收缩。 |
| void BilinearTransformation(char\* input,char\* output,int width,int height,int newwidth,int newheight) | 双线性变换。 |
| void DitherMatrixMethod(char\* input,char\* output,int width,int height,int N) | 四级抖动，默认N=2。 |
| void Dewarped1(char\* input,char\* output,int width,int height,int Offset,double a,double b) | 脱蜡。a是在输出图像中检查半径是否<=a，然后再进行扭曲，参考：Offset=256，a=256.5，b=0.5。 |
| void Dewarped2(char\* input,char\* output,int width,int height,int Offset,double a,double b,double coeffx[12],double coeffy[12]) | 脱蜡。a是在输出图像中检查半径是否<=a，然后再进行扭曲，参考：Offset=256，a=256.5，b=0.5。  脱蜡规范：  double coeffx[12] = { 1.00056776e+00, -5.68880703e-04, -1.13998357e-03,  1.00056888e+00,-5.65549579e-04,-1.13554790e-03,  9.99434446e-01 , 5.66658513e-04 , 1.13110351e-03 ,  9.99433341e-01 , 5.67767429e-04 , 1.13553921e-03 };  double coeffy[12] = {-5.67763072e-04, 1.00056888e+00, 1.13998357e-03,  5.68880703e-04, 9.99434450e-01, -1.13554790e-03,  5.65553919e-04, 9.99433341e-01, -1.13110351e-03,  -5.66658513e-04, 1.00056777e+00, 1.13553921e-03}; |
| void TextureSegmentation1(char\* input,char\* output,int width,int height,int K,int N) | 纹理分割，默认K=6，N=100。 |
| void TextureSegmentation2(char\* input,char\* output,int width,int height,int K,int N) | 纹理分割，默认K=6，N=100。 |
| void TextureClassification(vector <string> filename,char\* output,int width,int height,int K,int N,int a) | 纹理分类，a是要分类的图像的数量，如filename里有3个图像名称，则a=3；output是分类结果文件，格式为txt的文本文件；默认K=4，N=1000。 |
| void ErrorDiffusion1(char\* input,char\* output,int width,int height) | 误差扩散。 |
| void ErrorDiffusion2(char\* input,char\* output,int width,int height) | 误差扩散。 |
| void ErrorDiffusion3(char\* input,char\* output,int width,int height) | 误差扩散。 |
| void Thin(char\* input,char\* output,int width,int height) | 图像细化。 |
| void OilPainting(char\* input,char\* output,int width,int height,int N) | 油画效果，默认N=2。 |
| void OilPainting1(char\* input,char\* output,int width,int height,int N) | 油画效果，默认N=2。 |
| void AverageFiltering(char\* inputfile,char\* outputfile,int width,int height) | 3\*3平均值滤波。 |
| void GeometricMeanFiltering(char\* inputfile,char\* outputfile,int width,int height) | 3\*3几何均值滤波。 |
| void MedianFiltering(char\* inputfile,char\* outputfile,int width,int height) | 中值滤波。 |
| void FFT(char\* input,char\* output,int width,int height) | FFT函数。 |
| void LowPassOrHighPassFiltering(char\* input,char\* output,int width,int height,int LOW\_PASS,int DEGREE) | 低通或高通滤波。LOW\_PASS=1为低通滤波，否则为高通滤波，DEGREE为滤波程度，如DEGREE=0。 |
| void IFFT(char\* input,char\* output,int width,int height,int LOW\_PASS,int DEGREE) | IFFT函数。LOW\_PASS=1为低通滤波，否则为高通滤波，DEGREE为滤波程度，如DEGREE=0。 |
| void BMPtoRAW(char\* inputfile,char\* outputfile) | BMP图像转RAW图像。支持24为BMP图像。 |
| void BMPtoRAW1(char\* input,char\* output) | BMP图像转RAW图像。支持24为BMP图像。 |

**BMP图像处理**

|  |  |
| --- | --- |
| unsigned char\*\* BMPRead8(char\* input) | 读取8位BMP图像的像素。 |
| void GenerateImage8(char\* output,unsigned char\*\* color) | 生成8位BMP图像，output是生成的图像文件名，color是像素数据。 |
| BMPMat\*\* BMPRead(char\* input) | 读取24位和32位BMP图像的像素。  需要引入以下结构体：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat; |
| unsigned int BMPHeight(char\* input) | 读取BMP图像的高度。 |
| unsigned int BMPWidth(char\* input) | 读取BMP图像的宽度。 |
| void GenerateImage(char\* output,BMPMat\*\* color,unsigned short type) | 生成24位和32位BMP图像。type等于图像的位数，如type=24。  参考用例：  BMPMat\*\* color = (BMPMat\*\*)malloc(sizeof(BMPMat\*)\*1280);  for (unsigned int i = 0; i < 1280; i++)  {  color[i] = (BMPMat\*)malloc(sizeof(BMPMat)\*2450);  }  for (unsigned int i = 0; i < 1280; i++) {  for (unsigned int j = 0; j < 2450; j++) {  color[i][j].B =0;  color[i][j].G =0;  color[i][j].R =255;  }  } |
| void HistogramEqualization5(char\* input,char\* output) | 直方图均衡，支持8位和16位BMP。input是输入文件名，output是输出文件名。 |
| void Resize(char\* input,char\* output,int Height,int Width) | 图片缩放，支持8位和16位BMP。input是输入文件名，output是输出文件名。Height和Width是输出图像的高和宽。 |
| double MeanBrightness(char\* input) | 求图像的平均亮度，支持8位和16位BMP。input是输入文件名。 |
| int IsBitMap(FILE \*fp) | 判断是否是位图。 |
| int getWidth(FILE \*fp) | 获得图片的宽度。 |
| int getHeight(FILE \*fp) | 获得图片的高度。 |
| unsigned short getBit(FILE \*fp) | 获得每个像素的位数。 |
| unsigned int getOffSet(FILE \*fp) | 获得数据的起始位置。 |
| void BMPtoYUV(char\* input,char\* output, char yuvmode) | BMP图像转为YUV图像，input是输入文件名，output是输出文件名。yuvmode是YUV文件的3个模式选项，yuvmode的值可为'0'、'2'、'4'，分别为420，422，444 |
| void BMPtoYUV420I(char\* input,char\* output) | BMP图像转为YUV420图像，input是输入文件名，output是输出文件名。 |
| void BMPtoYUV420II(char\* input,char\* output) | BMP图像转为YUV420图像，input是输入文件名，output是输出文件名。 |
| void DCMtoBMP(string input,char\* output) | DCM图像转BMP图像。input是输入文件名，output是输出文件名。 |
| void Ins1977(char\* input,char\* output,int ratio) | Ins1977滤镜，input是输入文件名，output是输出文件名。参考：ratio=100。 |
| void LOMO(char\* input,char\* DarkAngleInput,char\* output,int ratio) | LOMO滤镜，DarkAngleInput是暗角模板图像名，参考：ratio=100。 |
| void PNGGray(char\* input,char\* output) | 图像灰度化，input是输入文件名，output是输出文件名。 |
| void PNGSpotlight(char\* input,char\* output,int centerX,int centerY,double a,double b,double c,double d,double e) | 聚光灯效果，input是输入文件名，output是输出文件名。焦点坐标（centerX,centerY），如：centerX=400,centerY=180；a、b、c、d、e是相关参数，默认a=100，b=100，c=160，d=80，e=0.5。 |
| void PNGIllinify(char\* input,char\* output) | 幻化效果，input是输入文件名，output是输出文件名。 |
| void PNGWaterMark(char\* input1,char\* input2,char\* output) | 图像加水印，input1和input2的尺寸必须相同。 |
| void Short(char\* input,char\* output,int a,int b,int c,double d,int depth) | 矮化特效。a=1，b=128，c=2，d=0.5，depth=24。支持24位BMP图像。 |
| void Rise(char\* input,char\* output,int a,int b,double c,int d,int depth) | 增高特效。a=1，b=128，c=0.5，d=2，depth=24。支持24位BMP图像。 |
| void Short1(char\* input,char\* output,int a,int b,double c,double d,int depth) | 矮小化特效。a=1，b=128，c=0.5，d=0.5，depth=24。支持24位BMP图像。 |
| void Handstand(char\* input,char\* output,int a,int b,double c,int depth) | 倒立特效。a=1，b=128，c=0.5，depth=24。支持24位BMP图像。 |
| void Fat(char\* input,char\* output,int a,int b,double c,int depth) | 肥胖特效。a=1，b=128，c=0.5，depth=24。支持24位BMP图像。 |
| void HighFoot(char\* input,char\* output,int a,int b,int c,double d,int depth) | 高脚特效。a=1，b=128，c=2，d=0.5，depth=24。支持24位BMP图像。 |
| void CurvedCurve(char\* input,char\* output,int a,int b,int c,int d,double e,int depth) | 弧度弯曲特效。a=1，b=128，c=4，d=2，e=0.5，depth=24。支持24位BMP图像。 |
| void Thin(char\* input,char\* output,int a,int b,double c,double d,int depth) | 细化特效。a=1，b=128，c=0.5，d=0.5，depth=24。支持24位BMP图像。 |
| void Winding(char\* input,char\* output,int lim,int a,int b,int c,int d,double e,int depth) | 弯曲特效。lim=20，a=1，b=128，c=4，d=5，e=0.5，depth=24。支持24位BMP图像。 |
| void CrossDenoising(unsigned char\*\* input,unsigned char\*\* output,double a) | 十字法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void CrossDenoising(BMPMat\*\* input,BMPMat\*\* output,double a) | 十字法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void CrossConnectionDenoising(unsigned char\*\* input,unsigned char\*\* output,double a) | 交叉法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void CrossConnectionDenoising(BMPMat\*\* input,BMPMat\*\* output,double a) | 交叉法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void MatrixDenoising(unsigned char\*\* input,unsigned char\*\* output,double a) | 矩阵法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void MatrixDenoising(BMPMat\*\* input,BMPMat\*\* output,double a) | 矩阵法剔除孤立像素点。  需引入以下结构体和声明：  typedef struct {  unsigned char B; //24位和32位BMP图像的蓝色通道分量  unsigned char G; //24位和32位BMP图像的绿色通道分量  unsigned char R; //24位和32位BMP图像的红色通道分量  unsigned char A; //仅限32位BMP图像的Alpha通道  }BMPMat;  typedef struct {  double B;  double G;  double R;  double A;  }BMPMatdouble;  void Conversion8(unsigned char\*\* input,double\*\* output);  void Conversion8(double\*\* input,unsigned char\*\* output);  void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output);  void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output); |
| void ImageFusion3(char\* input1,char\* input2,char\* output,int block\_height,int block\_width,double threshold) | 多聚焦图像的融合，支持8位BMP图像。block\_height=8，block\_width=8，threshold=1.75。 |
| void ImageFusion4(char\* input1,char\* input2,char\* output,int block\_height,int block\_width,double threshold) | 多聚焦图像的融合，支持8位BMP图像。block\_height=8，block\_width=8，threshold=1.75。 |
| void ImageFusion5(char\* input1,char\* input2,char\* MaskImage,char\* output,int dx[],int dy[],int a,double b1,int DX1,int DY1,double EPS) | 图像融合。参考：a=3，b1=4，DX1=-68，DY1=-99，EPS=1，input1="图像融合1.jpg"，input2="图像融合2.jpg"，MaskImage="掩膜.png"，output="output.jpg"。  int dx[] = {0,0,-1,1};  int dy[] = {-1,1,0,0}; |
| void Screenshot3(HWND hWnd, LPCWSTR OutputImage) | 截屏函数。hWnd是要截屏的窗口句柄，如：GetDesktopWindow()；OutputImage 截图名称。 |
| void Screenshot1(HWND hWnd, LPCWSTR OutputImage) | 截屏函数。hWnd是要截屏的窗口句柄，如：GetDesktopWindow()；OutputImage 截图名称。 |
| void Screenshot2(HWND hWnd,LPCWSTR OutputImage) | 截屏函数。hWnd是要截屏的窗口句柄，如：GetDesktopWindow()；OutputImage 截图名称。 |
| void Dark(char\* input,char\* output,int ratio) | 暗调滤镜，参考：ratio=100。 |
| void WaveFilter(char\* input,char\* output,int degree,int a) | 波浪形变特效滤镜，degree是滤镜程度（波浪扭曲度）。a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像，参考：degree=10。 |
| void PinchFilter(char\* input,char\* output,int a) | 挤压形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像。 |
| void PinchFilter(char\* input,char\* output,int cenX,int cenY,int a) | 挤压形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像，cenX是形变中心点X坐标，cenY是形变中心点Y坐标。 |
| void SpherizeFilter(char\* input,char\* output,int a) | 球面形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像。 |
| void SpherizeFilter(char\* input,char\* output,int cenX,int cenY,int a) | 球面形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像，cenX是形变中心点X坐标，cenY是形变中心点Y坐标。 |
| void SwirlFilter(char\* input,char\* output,int ratio,int a) | 旋转形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像，ratio=3。 |
| void SwirlFilter(char\* input,char\* output,int cenX,int cenY,int ratio,int a) | 旋转形变特效滤镜，a=0时生成BMP图像，a=1时生成JPG图像，a=2时生成PNG图像，a=3时生成TGA图像，ratio=3，cenX是形变中心点X坐标，cenY是形变中心点Y坐标。 |
| void ClosedOperation(char\* input,char\* output) | 闭运算，input是输入文件名，output是输出文件名。支持4位BMP图像。 |
| void ColorTransfer(char\* input1,char\* input2,char\* output) | 颜色转移。 |
| void GrayImage1(char\* input,char\* output) | 直方图均衡化。 |
| void ChannelHisteq(char\* input, char\* output) | 直方图均衡化。 |
| void HSVtoRGB(char\* input, char\* output) | HSV转到RGB。 |
| void HistogramEqualizationOnGrayImage(string input,char\* output) | 直方图均衡化。 |
| CImg<unsigned int> HistogramEqualizationOnGrayImage2(string input) | 直方图均衡化。 |
| void HistEqualColorImageOneColorChannel(string input, char\* output) | 直方图均衡化。 |
| CImg<unsigned int> HistEqualColorImageOneColorChannel1(string input) | 直方图均衡化。 |
| void HistEqualColorImageThreeColorChannels(string input, char\* output) | 直方图均衡化。 |
| CImg<unsigned int> HistEqualColorImageThreeColorChannels(string input) | 直方图均衡化。 |
| void HistEqualColorImageHSISpace(string input, char\* output) | HSI空间。 |
| CImg<unsigned int> HistEqualColorImageHSISpace(string input) | HSI空间。 |
| void ColorTransfer1(char\* sourceImage,string targetImage, char\* output) | 颜色转移。 |
| CImg<unsigned int> ColorTransfer2(string sourceImage, string targetImage) | 颜色转移。 |
| void BMPtoJPG(char\* input,char\* output,int a) | BMP图像转为JPG图像。支持24位BMP图像，尺寸必须是8的倍数，a代表文件压缩程度，数字越大，压缩后的文件体积越小，如a=100。 |
| void PartialColorRetention(char\* input,char\* output,int ratio) | 部分颜色保留滤镜，参考：ratio=60。 |
| void GrayImageConversion8(char\* input,char\* output) | 生成图像的灰度图，支持8位BMP图像。input是输入文件名，output是输出文件名。 |
| void Gray(char\* input,char\* output) | 灰度图转换，支持24位BMP图像。input是输入文件名，output是输出文件名。 |
| void GrayImageConversion(char\* input,char\* output) | 彩色图转灰度图，input是要处理的彩色图像，output是处理后生成的灰度图名称。支持24位BMP图像。 |
| void BinaryImageVerticalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 二值图像垂直镜像，input是输入图像的像素数据，output是输出图像的像素数据，w是输入图像的宽，h是输入图像的高。 |
| void GrayImageVerticalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 灰度图像垂直镜像，input是输入图像的像素数据，output是输出图像的像素数据，w是输入图像的宽，h是输入图像的高。 |
| void ColorImageVerticalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 彩色图像垂直镜像，input是输入图像的像素数据，output是输出图像的像素数据，w是输入图像的宽，h是输入图像的高。 |
| void OTSU(char\* input,char\* output,int BeforeThreshold) | 大津算法，input是输入文件名，output是输出文件名。BeforeThreshold是初始阈值，如BeforeThreshold=10。支持8位BMP图像。 |
| void LowerBrightness(char\* input,char\* output,int a,int b) | 调低亮度，input是输入文件名，output是输出文件名。支持24位BMP图像。a和b的参考值可为a=100，b=0。 |
| void HightBrightness(char\* input,char\* output,int a,int b) | 调高亮度，input是输入文件名，output是输出文件名。支持24位BMP图像。a和b的参考值可为a=100，b=0。 |
| void IterativeThresholdSelection(char\* input,char\* output) | 迭代阈值选择，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void DitheringMethod(char\* input,char\* output) | 抖动法，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void LogTransformation(char\* input,char\* output,int constant) | 对数变换，input是输入文件名，output是输出文件名。支持8位BMP图像。constant是相关参数，如constant=15。 |
| void LogarithmicTransformation(char\* input,char\* output) | 对数变换，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void HistogramEqualization(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void Binarization(char\* input,char\* output,int threshold) | 二值化，input是输入文件名，output是输出文件名。支持24位BMP图像。threshold是阈值，如：threshold=128。 |
| void Expansion(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像膨胀，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void Corrosion(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像腐蚀，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void OpenOperation(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像开运算，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void ClosedOperation(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像闭运算，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void OpenOperationToExtractContour(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像开运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void ExpansionOperationToContourExtraction(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像膨胀运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void CorrosionCalculationToContourExtraction(char\* input,char\* output,unsigned char mask[9],int c) | 二值图像腐蚀运算提取轮廓，参考：mask[9]={0,255,0,255,255,255,0,255,0}，c=128。 |
| void Glaw(char\* input,char\* output,int ratio) | 发光滤镜，参考：ratio=100。 |
| void LowPassFilter(char\* input,char\* output) | 低通滤波器，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void HighPassFilter(char\* input,char\* output) | 高通滤波器，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void Thinning(char\* input,char\* output) | 图像细化，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void ThinningLine(char\* input,char\* output) | 图像细化且线条化，input是输入文件名，output是输出文件名。支持BMP图像。 |
| void Corrosion(char\* input,char\* output) | 腐蚀，input是输入文件名，output是输出文件名。支持4位BMP图像。 |
| void Corrosion1(char\* input,char\* output,int \*TempBuf, int TempH, int TempW) | 腐蚀，input是输入文件名，output是输出文件名。支持24位BMP图像。TempBuf是腐蚀模板，TempH和TempW分别是TempBuf的高和宽，如TempH=4,TempW=4，则有TempBuf[4][4]。 |
| void Expand(char\* input,char\* output,int \*TempBuf, int TempH, int TempW) | 膨胀，input是输入文件名，output是输出文件名。支持24位BMP图像。TempBuf是膨胀模板，TempH和TempW分别是TempBuf的高和宽，如TempH=4,TempW=4，则有TempBuf[4][4]。 |
| unsigned char\*\* create2DImg(unsigned char\* input, int w, int h) | 线性存储的灰阶图像像素转化为二维。 |
| unsigned char getMaxPixelWhole(unsigned char \*\*input,int x,int y,int w,int h,int \*Kernal,int kernalW,int halfKernalW) | 图像指定区域取最大值（判断是否超出边界）。 |
| unsigned char getMaxPixelCenter(unsigned char \*\*input,int x,int y,int \*Kernal,int kernalW,int halfKernalW) | 图像指定区域取最大值（不判断是否超出边界）。 |
| unsigned char\*\* imgDilate(unsigned char \*input,int w,int h,int \*Kernal,int kernalW,int halfKernalW) | 图像膨胀。 |
| unsigned char getMinPixelWhole(unsigned char \*\*input,int x,int y,int w,int h,int \*Kernal,int kernalW,int halfKernalW) | 图像指定区域取最小值（判断是否超出边界）。 |
| unsigned char getMinPixelCenter(unsigned char \*\*input,int x,int y,int \*Kernal,int kernalW,int halfKernalW) | 图像指定区域取最小值（不判断是否超出边界）。 |
| unsigned char\*\* imgErode(unsigned char \*input,int w,int h,int \*Kernal,int kernalW,int halfKernalW) | 图像腐蚀。 |
| void Corrosion(unsigned char \*input,unsigned char \*output,int rows,int cols,int mat[5][5]) | 二值腐蚀。 |
| void Expansion(unsigned char \*input,unsigned char \*output,int rows,int cols,int mat[5][5]) | 二值膨胀。 |
| void GaussianBlurFilter(char\* input,char\* output) | 高斯滤波，支持PNG文件。 |
| void GaussianFiltering(char\* input,char\* output) | 高斯滤波，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void LaplaceEnhancement(char\* input,char\* output) | 拉普拉斯增强，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void Residual(char\* input,char\* output) | 求残差，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void SunlightFilter(char\* input,char\* output,int intensity,int radius,int x,int y) | 光照特效滤镜，intensity是光照强度，如：intensity=255；radius是光照范围，如：radius=600；x和y是光照的位置，如：x=100，y=60。 |
| void Compress(char\* input,char\* output) | 压缩，支持多种文件。input是要压缩的文件名，output是压缩后的文件名。 |
| void Decompression(char\* input, char\* output) | 解压缩，支持多种文件。input是要解压缩的文件名，output是解压缩后的文件名。 |
| void BlackWhite(char\* input,char\* output) | 黑白化，input是输入的原图像，output是输出的黑白图像。支持24位BMP图像。 |
| void Underexposure(char\* input,char\* output) | 图像欠曝光，input是输入的原图像，output是输出的欠曝光图像。支持24位BMP图像。 |
| void Overexposure(char\* input,char\* output) | 图像过曝光，input是输入的原图像，output是输出的过曝光图像。支持24位BMP图像。 |
| void Nostalgia(char\* input,char\* Mask,char\* output,int ratio) | 怀旧滤镜，input和Mask都是输入的文件名，Mask是褶皱图像路径，ratio=100。 |
| void GammaTransform(char\* input,char\* output) | 伽马变换，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void GrayScale(char\* input,char\* output) | 灰度化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void GrayImageBinarization(char\* input,char\* output,int bit,int threshold) | 灰度图二值化，bit用于设定位数，如bit=8；threshold是阈值，如threshold=200。支持8位BMP图像。 |
| void GreyPesudoColor(char\* input,char\* output) | 灰度图伪彩色化，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void CalculateCumulativeHistogramMap(char\* input,char\* outfile) | 计算累加直方图并映射，input是输入文件名，outfile是输出文件名。支持24位BMP图像。 |
| void Translation(string input,char\* output,int dx,int dy) | 图像平移，input是输入的文件，dx和dy是横向及纵向的移动距离（像素），负值是向左 / 向下移动；output是平移操作后的结果文件名。支持BMP图像。 |
| void Mirrored(string input,char\* output,char axis) | 镜像变换，input是输入的文件，output是镜像操作后的结果文件名，axis是镜像变换的方向（以X或Y表示）。支持BMP图像。 |
| void Sheared(string input,char\* output,char axis,double Coef) | 错切变换，input是输入的文件，output是错切操作后的结果文件名，axis和Coef分别是错切变换的方向（以X或Y表示）和错切系数，负值是向左 / 向下偏移。支持BMP图像。 |
| void Scaled(string input,char\* output,double cx,double cy) | 缩放操作，input是输入的文件，output是缩放操作后的结果文件名，cx和cy分别是横向及纵向的缩放系数，系数大于1表示拉伸，小于1表示压缩。支持BMP图像。 |
| void Rotated1(string input,char\* output,double angle) | 图像旋转，input是输入的文件，output是图像旋转后的结果文件名，angle是旋转角度，弧度制。支持BMP图像。 |
| void SaltNoise(char\* input,char\* output,int a,int b,int c,int d) | 添加椒盐噪声，a和b是噪声相关参数，如a=3，b=3；c和d是颜色相关参数，如c=0，d=255。支持8位BMP图像。 |
| void CrossProcess(char\* input,char\* output,int ratio) | 交叉冲印滤镜，参考：ratio=100。 |
| void Conversion8(unsigned char\*\* input,short\*\* output) | unsigned char\*\*转short\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(short\*\* input,unsigned char\*\* output) | short\*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned char\*\* input,int\*\* output) | unsigned char\*\*转int\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(int\*\* input,unsigned char\*\* output) | int\*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned char\*\* input,unsigned int\*\* output) | unsigned char\*\*转unsigned int\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned int\*\* input,unsigned char\*\* output) | unsigned int\*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned char\*\* input,float\*\* output) | unsigned char\*\*转float \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(float\*\* input,unsigned char\*\* output) | float \*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned char\*\* input,double\*\* output) | unsigned char\*\*转double \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(double\*\* input,unsigned char\*\* output) | double \*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(unsigned char\*\* input,char\*\* output) | unsigned char\*\*转char \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion8(char\*\* input,unsigned char\*\* output) | char \*\*转unsigned char\*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMat\*\* input,BMPMatshort\*\* output) | BMPMat \*\*转BMPMatshort \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMatshort\*\* input,BMPMat\*\* output) | BMPMatshort \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMat\*\* input,BMPMatint\*\* output) | BMPMat \*\*转BMPMatint \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMatint\*\* input,BMPMat\*\* output) | BMPMatint \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMat\*\* input,BMPMatfloat\*\* output) | BMPMat \*\*转BMPMatfloat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMatfloat\*\* input,BMPMat\*\* output) | BMPMatfloat \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMat\*\* input,BMPMatdouble\*\* output) | BMPMat \*\*转BMPMatdouble \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMatdouble\*\* input,BMPMat\*\* output) | BMPMatdouble \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMat\*\* input,BMPMatchar\*\* output) | BMPMat \*\*转BMPMatchar \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion24(BMPMatchar\*\* input,BMPMat\*\* output) | BMPMatchar \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMat\*\* input,BMPMatshort\*\* output) | BMPMat \*\*转BMPMatshort \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMatshort\*\* input,BMPMat\*\* output) | BMPMatshort \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMat\*\* input,BMPMatint\*\* output) | BMPMat \*\*转BMPMatint \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMatint\*\* input,BMPMat\*\* output) | BMPMatint \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMat\*\* input,BMPMatfloat\*\* output) | BMPMat \*\*转BMPMatfloat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMatfloat\*\* input,BMPMat\*\* output) | BMPMatfloat \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMat\*\* input,BMPMatdouble\*\* output) | BMPMat \*\*转BMPMatdouble \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMatdouble\*\* input,BMPMat\*\* output) | BMPMatdouble \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMat\*\* input,BMPMatchar\*\* output) | BMPMat \*\*转BMPMatchar \*\*，output用于保存结果（与input大小相同）。 |
| void Conversion32(BMPMatchar\*\* input,BMPMat\*\* output) | BMPMatchar \*\*转BMPMat \*\*，output用于保存结果（与input大小相同）。 |
| void MeanFiltering(char\* input,char\* output) | 均值滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void MeanFltering1(char\* input,char\* output) | 均值滤波，input是输入文件名，output是输出文件名。支持8位和24位BMP图像。 |
| void KapoorAlgorithm(char\* input,char\* output,int BeforeThreshold) | 卡普尔算法，input是输入文件名，output是输出文件名。BeforeThreshold是初始阈值，如BeforeThreshold=150。支持8位BMP图像。 |
| void OpenOperation(char\* input,char\* output) | 开运算，input是输入文件名，output是输出文件名。支持4位BMP图像。 |
| void Diffusion(char\* input,char\* output,int ratio) | 扩散滤镜，参考：ratio=90。 |
| void LapulasFiltering(char\* readPath,char\* writePath,float CoefArray[9],float coef) | 拉普拉斯滤波，readPath是原图像，writePath是处理后的图像文件名。支持8位BMP图像。  各参数参考值：  定义\*3的模板（拉普拉斯）：  float CoefArray[9]={1.0f,2.0f,1.0f,2.0f,4.0f,2.0f,1.0f,2.0f,1.0f};  定义模板前乘的系数（拉普拉斯）：  float coef=(float)(1.0/16.0); |
| void ImageFiltering(char\* input,char\* output,float kernel[3][3]) | 图像滤波，input是输入文件名，output是输出文件名。kernel是模糊内核。支持24位BMP图像。 |
| void ComicStrip(char\* input,char\* output,int ratio) | 连环画滤镜，参考：ratio=100。 |
| void BrightnessAdjustment1(char\* input,char\* output,int brightness,int contrast) | 亮度对比度调节，参考：brightness=-30，contrast=100。 |
| void BrightnessAdjustment2(char\* input,char\* output,int brightness,int contrast) | 亮度对比度调节，参考：brightness=-30，contrast=100。 |
| void ZeroFillingSymmetricExtension(char\* input,char\* output) | 零填充与对称扩展，支持8位和24位BMP图像。 |
| void PopArtStyle(char\* input,char\* output,int ratio) | 流行艺术风滤镜，参考：ratio=100。 |
| void LightLeakage(char\* input,char\* Mask,char\* output,int ratio) | 漏光滤镜，input和Mask都是输入的图像名，Mask是漏光模板图像，ratio=90。 |
| void LinearFiltering(char\* input,char\* output,short average[3][3]) | 线性滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。  参考模板：  short average[3][3] = {{1, 2, 1},  {2, 4, 2},  {1, 2, 1}}; |
| void MedianFiltering(char\* input,char\* output,short average[3][3]) | 中值滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。  参考模板：  short average[3][3] = {{1, 2, 1},  {2, 4, 2},  {1, 2, 1}}; |
| void SharpeningFiltering(char\* input,char\* output,short average[3][3],short sharpen[3][3]) | 锐化滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。  参考模板：  short average[3][3] = {{1, 2, 1},  {2, 4, 2},  {1, 2, 1}};  short sharpen[3][3] = {{-1, -1, -1},  {-1, 8, -1},  {-1, -1, -1}}; |
| void GradientSharpening(char\* input,char\* output,short average[3][3],short soble1[3][3],short soble2[3][3]) | 梯度锐化，input是输入文件名，output是输出文件名。支持8位BMP图像。  参考模板：  short average[3][3] = {{1, 2, 1},  {2, 4, 2},  {1, 2, 1}};  short soble1[3][3] = {{-1, -2, -1},  {0, 0, 0},  {1, 2, 1}};  short soble2[3][3] = {{-1, 0, 1},  {-2, 0, 2},  {-1, 0, 1}}; |
| void ArithmeticMeanFilter(char\* input,char\* output) | 算术平均滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void GeometricMeanFilter(char\* input,char\* output) | 几何平均滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void HarmonicMeanFilter(char\* input,char\* output) | 调和平均滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void ContraHarmonicMeanFilter(char\* input,char\* output) | 反调和平均滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Filter(char\* input,char\* output) | 滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Mosaic(char\* input,char\* output,int x) | 马赛克化图像，input是输入文件名，output是输出文件名。x是马赛克处理的块的大小。支持24位BMP图像。 |
| void MosaicFilter(char\* input,char\* output,int ratio) | 马赛克滤镜，参考：ratio=50。 |
| void Expansion(char\* input,char\* output) | 膨胀，input是输入文件名，output是输出文件名。支持4位BMP图像。 |
| void MidSmoothing(char\* input,char\* output) | 中值滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void AvgSmoothing(char\* input,char\* output) | 均值滤波器，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Averaging(char\* input1,char\* input2,char\* input3,char\* output,int a) | 图像平均化，input是输入文件名，output是输出文件名。a是平均化相关参数，如a=3。支持8位BMP图像。 |
| void PlaneSlicing(char\* input,char\* output) | 平面切片，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Translation(char\* input,char\* output,int xoffset,int yoffset) | 图像平移，参考：xoffset=-100，yoffset=-100。 |
| void SharpeningSpatialFiltering8(char\* input,char\* output,int model[9]) | 锐化空间滤波器，input是输入文件名，output是输出文件名。model是锐化模板。支持8位灰度图像。 |
| void PseudoGrayscale(char\* input,char\* output) | 伪灰度化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void TwoColors(char\* input,char\* output,int threshold,unsigned char color1,unsigned char color2) | 二色化，input是输入文件名，output是输出文件名。threshold是阈值，如threshold=115；color1和color2是要填充的两个颜色。支持24位BMP图像。 |
| void PNGImageGeneration(char\* filename,const unsigned char img[],unsigned W,unsigned H,int x) | filename是生成的PNG图像文件名；img是图像的像素数据，W是图像的宽，H是图像的高，x=0选择生成RGB图像，x=1选择生成RGBA图像。 |
| void MakeSphere(double V[3],double S[3], double r, double a, double m, int ROWS, int COLS, char\* output) | 使用反射模型在正交投影下生成球体的图像，V是摄影机的方向，output是输出的结果图像文件名，ROWS是输出图像的行数，COLS是输出图像的列数，参考：V[3] = {0.0, 0.0, 1.0}，S[3] = {0.0, 0.0, 1.0}，r=50，a=0.5，m=1。支持RAS文件。 |
| void MakeSphere(double vector\_v[3],double vector\_s[3], double r, double a, double m, int ROWS,int COLS,char\* output,double max) | 使用反射模型生成球体的图像，vector\_v是摄影机的方向，output是输出的结果图像文件名，ROWS是输出图像的行数，COLS是输出图像的列数，参考：vector\_v[3] = {0.0, 0.0, 1.0}，vector\_s[3] = {0.0, 0.0, 1.0}，r=50，a=0.5，m=1。支持RAS文件。 |
| void BilateralFiltering(string input,char\* output,double ssd, double sdid) | 双边滤波，input是输入文件名，output是输出文件名。支持24位BMP图像。ssd和sdid分别是空间域标准差与强度域标准差。 |
| void DoubleLayerErosion(char\* input,char\* output) | 具有圆形结构集的双层形态侵蚀，支持8位和24位BMP图像。 |
| void BinaryImageHorizontalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 二值图像水平镜像。 |
| void GrayImageHorizontalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 灰阶图像水平镜像。 |
| void ColorImageHorizontalMirror(unsigned char \*input,unsigned char \*output,unsigned int w,unsigned int h) | 彩色图像水平镜像。 |
| void SketchFilter(char\* input,char\* output,int ratio) | 素描滤镜，参考：ratio=100。 |
| void Zoom(char\* input,char\* output,float scaleX,float scaleY,int interpolation) | 缩放，参考：scaleX=5，scaleY=5，interpolation=0或interpolation=1。 |
| void AddGaussNoise(char\* input,char\* output) | 添加高斯噪声，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void AddSaltPepperNoise(char\* input,char\* output) | 添加椒盐噪声，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void ChannelSeparation(char\* input,char\* Routput,char\* Goutput,char\* Boutput) | 通道分离，input是输入文件名，Routput是红色通道图像，Goutput是绿色通道图像，Boutput是绿色通道图像。支持24位BMP图像。 |
| void PatternMethod(char\* input,char\* output,unsigned char Template[8][8]) | 图案法，input是输入文件名，output是输出文件名。Template是模板数组。支持8位BMP图像。 |
| void LayerAlgorithm(char\*input,char\* inputMix,char\* output,int alpha,int blendModel) | 图层算法，input是基底图层图像，inputMix是混合图层图像，参考：alpha=50，blendModel=26。  blendModel的取值对应的模式如下：  1 典型  2 溶解  3 暗化  4 多层  5 颜色加深模式  6 线性加深  7 暗调  8 亮化  9 遮盖  10 颜色减淡模式  11 线性减淡  12 浅色  13 叠加  14 柔光模式  15 强光模式  16 艳光模式  17 线性光模式  18 点光模式  19 强混合模式  20 差分  21 排除模式  22 减运算  23 图像分割  24 色相模式  25 色饱和  26 着色  27 亮度模式 |
| void BMP24LossyCompression(char\* input,char\* output) | 图像有损压缩，input是待压缩的BMP文件名，output是有损压缩后输出的文件名。支持24位BMP图像。 |
| void BMP24LossyDecompression(char\* input,char\* output) | 图像有损解压，input是待解压的文件名，output是输出解压后的BMP文件名。支持24位BMP图像。 |
| void BMP24LosslessCompression(char\* input,char\* output) | 图像无损压缩，input是待压缩的BMP文件名，output是无损压缩后输出的文件名。支持24位BMP图像。 |
| void BMP24LosslessDecompression(char\* input,char\* output) | 图像无损解压，input是待解压的文件名，output是输出解压后的BMP文件名。支持24位BMP图像。 |
| void ImageDiscoloration(char\* input,char\* output,double a,double b,double c) | 图像变色，input是输入文件名，output是输出文件名。如：a=0.2126，b=0.7152, c=0.0722。支持24位BMP图像。 |
| unsigned char\*\* HorizontalConcavity(unsigned char\*\* input,int RANGE,int height,int width) | 图像变形之水平内凹，返回处理结果。参考：RANGE=400。 |
| unsigned char\*\* HorizontalConvexity(unsigned char\*\* input,int RANGE,int height,int width) | 图像变形之水平外凸，返回处理结果。参考：RANGE=400。 |
| unsigned char\*\* TrapezoidalDeformation(unsigned char\*\* input,int height,int width,double k) | 图像变形之梯形形变，返回处理结果。参考：k=0.3。 |
| unsigned char\*\* TriangularDeformation(unsigned char\*\* input,int height,int width,double k) | 图像变形之三角形形变，返回处理结果。参考：k=0.5。 |
| unsigned char\*\* SDeformation(unsigned char\*\* input,int height,int width,int RANGE) | 图像变形之S形变，返回处理结果。参考：RANGE=450。 |
| int LsdLineDetector(unsigned char \*src, int w, int h,float scaleX, float scaleY, boundingbox\_t bbox, std::vector<line\_float\_t> &lines) | LSD直线检测器。  [in] src: 图像，单通道  [in] w: 宽  [in] h: 高  [in] scaleX: X轴上的缩小因子  [in] scaleY: Y轴上的缩小因子  [in] bbox: 要检测的边界框  [in/out] lines: 结果  return： 0:ok; 1:error  需引入以下结构体：  typedef struct  {  int x;  int y;  int width;  int height;  }boundingbox\_t;  typedef struct  {  float startx;  float starty;  float endx;  float endy;  }line\_float\_t; |
| int EdgeDrawingLineDetector(unsigned char \*src, int w, int h,float scaleX, float scaleY, boundingbox\_t bbox, std::vector<line\_float\_t> &lines) | 边缘划线检测器。  [in] src: 图像，单通道  [in] w: 宽  [in] h: 高  [in] scaleX: X轴上的缩小因子  [in] scaleY: Y轴上的缩小因子  [in] bbox: 要检测的边界框  [in/out] lines: 结果  return： 0:ok; 1:error  需引入以下结构体：  typedef struct  {  int x;  int y;  int width;  int height;  }boundingbox\_t;  typedef struct  {  float startx;  float starty;  float endx;  float endy;  }line\_float\_t; |
| int PropagatedFilter1(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst,int w, int h, int c, int r, float sigma\_s, float sigma\_r) | 传播滤波器。  [in] src: 输入图像  [in] guidance: 引导图像  [in/out]] dst: 输出图像  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1或c=3  [in] r: 局部窗口半径  [in] sigma\_s:坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r:颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  @return： 0:ok; 1:error |
| int PropagatedFilter2(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst,int w, int h, int c, int r, float sigma\_s, float sigma\_r) | 传播滤波器。  [in] src: 输入图像  [in] guidance: 引导图像  [in/out]] dst: 输出图像  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1或c=3  [in] r: 局部窗口半径  [in] sigma\_s:坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r:颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  @return： 0:ok; 1:error |
| int BoxfilterFilter(unsigned char \*src, unsigned char \*dst,int w, int h, int c, int r) | 方盒滤波。  [in] src: 输入图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1  [in] r: 局部窗口半径  return： 0:ok; 1:error |
| int BoxfilterFilter1(unsigned char \*src, unsigned char \*dst,int w, int h, int c, int r) | 方盒滤波。  [in] src: 输入图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1  [in] r: 局部窗口半径  return： 0:ok; 1:error |
| int fast\_guided\_filter(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst,int w, int h, int c, int r, float rp, float sr,float \_scale) | 快速导向滤波  [in] src: 输入图像，单通道  [in] guidance: 引导图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1  [in] r: 局部窗口半径  [in] rp: 正则化参数：eps  [in] sr: 二次采样率，sr>1：向下缩放，0<sr<1：向上缩放  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error  eg: r = 4, (try sr = r/4 to sr=r),(try rp=0.1^2, 0.2^2, 0.4^2)  try:(src,guidance,dst,w,h,1,4,0.01,4,255\*255)  condition: (MIN(w, h) / sr) > 1  condition: (int)(r / sr + 0.5f) >= 1 |
| int fast\_guided\_filter1(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst,int w, int h, int c, int r, float rp, float sr,float \_scale) | 快速导向滤波  [in] src: 输入图像，单通道  [in] guidance: 引导图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1  [in] r: 局部窗口半径  [in] rp: 正则化参数：eps  [in] sr: 二次采样率，sr>1：向下缩放，0<sr<1：向上缩放  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error  eg: r = 4, (try sr = r/4 to sr=r),(try rp=0.1^2, 0.2^2, 0.4^2)  try:(src,guidance,dst,w,h,1,4,0.01,4,255\*255)  condition: (MIN(w, h) / sr) > 1  condition: (int)(r / sr + 0.5f) >= 1 |
| int HoughLineDetector(unsigned char \*src, int w, int h,float scaleX, float scaleY, float CannyLowThresh, float CannyHighThresh,float HoughRho, float HoughTheta, float MinThetaLinelength, float MaxThetaGap, int HoughThresh,HOUGH\_LINE\_TYPE\_CODE \_type,boundingbox\_t bbox, std::vector<line\_float\_t> &lines) | 霍夫线探测器。  [in] src: 图像，单通道  [in] w: 宽  [in] h: 高  [in] scaleX: X轴上的缩小因子  [in] scaleY: Y轴上的缩小因子  [in] CannyLowThresh: canny算子中迟滞过程的低阈值  [in] CannyHighThresh: canny算子中迟滞过程的高阈值  [in] HoughRho: 累加器的距离分辨率（以像素为单位）  [in] HoughTheta: 累加器的角度分辨率（弧度）  [in] MinThetaLinelength: 标准：对于标准和多尺度hough变换，检查线条的最小角度  传播能力：最小线路长度。小于的线段被拒绝  [in] MaxThetaGap: 标准：对于标准和多尺度hough变换，检查线条的最大角度  基于概率的：连接同一条线上的点之间允许的最大间隙  [in] HoughThresh: 累加器阈值参数。只有那些获得足够选票的行才会返回（>阈值）  [in] \_type: hough线方法：hough\_line\_STANDARD或hough\_line \_PROBABILISTIC  [in] bbox: 要检测的边界框  [in/out] lines: 结果  return 0:ok; 1:error  \_type: HOUGH\_LINE\_STANDARD: 标准hough线算法  HOUGH\_LINE\_PROBABILISTIC ： 概率hough线算法    当HOUGH\_LINE\_STANDARD运行时，线点可能是图像坐标之外的位置  标准: try (src,w,h,scalex,scaley,70,150, 1, PI/180, 0, PI, 100, HOUGH\_LINE\_STANDARD, bbox, line)  基于概率的: try (src,w,h,scalex,scaley,70,150, 1, PI/180, 30, 10, 80, HOUGH\_LINE\_STANDARD, bbox, line)。  需引入以下结构体：  typedef enum \_HOUGH\_LINE\_TYPE\_CODE  {  HOUGH\_LINE\_STANDARD = 0, //standad hough line  HOUGH\_LINE\_PROBABILISTIC = 1, //probabilistic hough line  }HOUGH\_LINE\_TYPE\_CODE;  typedef struct  {  int x;  int y;  int width;  int height;  }boundingbox\_t;  typedef struct  {  float startx;  float starty;  float endx;  float endy;  }line\_float\_t; |
| void \_fast\_bilateral\_filter\_singlechannel(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst, int w, int h, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器单通道。  [in] src: 输入图像，单通道  [in] guidance: 引导图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error |
| int fast\_bilateral\_filter\_singlechannel(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst, int w, int h, int c, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器单通道。  [in] src: 输入图像，单通道  [in] guidance: 引导图像，单通道  [in/out] dst: 输出图像，单通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error |
| void \_fast\_bilateral\_filter\_color(unsigned char \*src, unsigned char \*dst, int w, int h, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器RGB通道。  [in] src: 输入图像，RGB通道  [in/out] dst: 输出图像，RGB通道  [in] w: 宽  [in] h: 高  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error |
| int fast\_bilateral\_filter\_color(unsigned char \*src, unsigned char \*dst, int w, int h, int c, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器RGB通道。  [in] src: 输入图像，RGB通道  [in/out] dst: 输出图像，RGB通道  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=3  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error |
| int FastBilateralFilter(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst, int w, int h, int c, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器。  [in] src: 输入图像  [in] guidance: 引导图像，单个通道，只有单个通道有效  [in/out] dst: 输出图像  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1或c=3  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error  如果引导为NULL，仍然可以获得滤色器 |
| int permutohedral\_bilateral\_filter(unsigned char \*src, unsigned char \*guidance, unsigned char \*dst,int w, int h, int c, float sigma\_s, float sigma\_r,float \_scale) | 快速双边滤波器。  [in] src: 输入图像  [in] guidance: 引导图像  [in/out] dst: 输出图像  [in] w: 宽  [in] h: 高  [in] c: 图像通道，仅c=1或c=3  [in] sigma\_s: 坐标空间中的滤波器西格玛。参数的值越大，意味着只要颜色足够接近，更远的像素就会相互影响（请参见sigmaColor）。当d>0时，它指定邻域大小，而不考虑sigmaSpace。否则，d与sigmaSpace成比例。  [in] sigma\_r: 颜色空间中的滤波器西格玛。该参数的值越大，意味着像素邻域（请参见sigmaSpace）内更远的颜色将混合在一起，从而产生更大的半等色区域。  如果正则化, \_scale = 1; 如果不正则化, \_scale = 255\*255  return： 0:ok; 1:error  try:(src,guidance,dst,w,h,c,1.6f,0.6f,255\*255) |
| void HighPassFilter(char\* input,char\* output,int preserve) | 高通滤波器，参考：preserve=0。 |
| void EmbossFilter(char\* input,char\* output,int preserve) | 浮雕过滤器，参考：preserve=1。 |
| void SharpenFilter(char\* input,char\* output,int preserve) | 锐化过滤器，参考：preserve=1。 |
| void Convolution(char\* input,char\* output,int w,int preserve) | 卷积，参考：w=7，preserve=1。 |
| void GaussianBlur(char\* input,char\* output,float sigma,int preserve) | 高斯模糊，参考：sigma=2，preserve=1。 |
| void HybridImage(char\* input1,char\* input2,char\* output,float sigma,int preserve) | 混合图像，参考：sigma=2，preserve=1。 |
| void LowFrequencyImage(char\* input,char\* output,float sigma,int preserve) | 低频图像，参考：sigma=2，preserve=1。 |
| void HighFrequencyImage(char\* input,char\* output,float sigma,int preserve) | 高频图像，参考：sigma=2，preserve=1。 |
| void HighFrequencyImage1(char\* input,char\* output,float sigma,int preserve) | 高频图像，参考：sigma=2，preserve=1。 |
| void Bilateral(char\* input,char\* output,float sigma1, float sigma2) | 双边滤波，参考：sigma1=3，sigma2=0.1。 |
| void SkinSmooth(char\* input,char\* output,int a,int b) | 皮肤细滑，a是平滑级别，b代表是否应用皮肤过滤器，a=2，b=1。 |
| void Resize1(char\* input,char\* output,int w, int h) | 图像模糊，w=713，h=467。 |
| void Resize2(char\* input,char\* output,int w, int h) | 图像模糊。 |
| void Shift(char\* input,char\* output,int ch, float v) | Shift函数，ch=1，v=0.1。 |
| void RGBtoHSV(char\* input,char\* output) | RGB转HSV。 |
| void HSVtoRGB(char\* input,char\* output) | HSV转RGB。 |
| void RGBtoLCH(char\* input,char\* output) | RGB转LCH。 |
| void LCHtoRGB(char\* input,char\* output) | LCH转RGB。 |
| void ColorTransfer(char\* input1,char\* input2,char\* output) | 颜色转移。 |
| void DrawText(char\* inputText,char\* output,int width,int height,int depth,int spectrum,int x,int y, unsigned char R,unsigned char G,unsigned char B,unsigned char color1[],unsigned char color2[],float opacity,unsigned int font) | 文本绘制，R=255，G=255，B=255，depth=1，spectrum=3，(x,y)是文本的坐标，color1是前景颜色，color2是背景颜色，opacity=1，font=60。 |
| void EqualizedGray(char\* input, char\* output) | 灰度图直方图均衡化。 |
| void ColorHistogramEqualization(char\* input, char\* output) | 彩色图直方图均衡化。 |
| void AverageHistogram(char\* input, char\* output) | 直方图均衡化。 |
| void HSIHist(char\* input, char\* output) | HIS直方图。 |
| void ImageCutting(char\* input,char\* output,int leftdownx,int leftdowny,int rightupx,int rightupy) | 图像裁剪，input是输入文件名，output是输出文件名。leftdownx, leftdowny, rightupx, rightupy是要裁剪的矩形区域的左下角和右上角的坐标（连续四个整数值，如50 50 300 300)。支持24位BMP图像。 |
| void ImageLayerAlgorithm(char\* input,char\* output) | 图像层算法。 |
| void RGBtoGraywithoutLUT(char\* input,char\* output) | 图像无LUT的灰度化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void RGBtoGraywithLUT(char\* input,char\* output) | 图像有LUT的灰度化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void PiecewiseLinearTransform(char\* input,char\* output) | 分段线性变换，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void PowerConvertion(char\* input,char\* output,double c,double g) | 功率转换，input是输入文件名，output是输出文件名。如: c = 1.2，g = 0.5。支持8位BMP图像。 |
| void Smooth(char\* input,char\* output) | 平滑，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Multiply(char\* input,char\* output,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3]) | 图像倍增化，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int LaplMask[3][3] = {  0,1,0,  1, -4, 1,  0, 1, 0  };  int SblMask1[3][3] = {  -1,-2,-1,  0, 0, 0,  1, 2, 1  };  int SblMask2[3][3] = {  -1, 0, 1,  -2, 0, 2,  -1, 0, 1  }; |
| void Add(char\* input,char\* output,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3]) | 图像添加，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int LaplMask[3][3] = {  0,1,0,  1, -4, 1,  0, 1, 0  };  int SblMask1[3][3] = {  -1,-2,-1,  0, 0, 0,  1, 2, 1  };  int SblMask2[3][3] = {  -1, 0, 1,  -2, 0, 2,  -1, 0, 1  }; |
| void PowerConvertion1(char\* input,char\* output,double c,double g,int N,int SblMask1[3][3],int SblMask2[3][3],int LaplMask[3][3]) | 功率变换，input是输入文件名，output是输出文件名。如: c = 1.2，g = 0.5，N=1。支持8位BMP图像。  int LaplMask[3][3] = {  0,1,0,  1, -4, 1,  0, 1, 0  };  int SblMask1[3][3] = {  -1,-2,-1,  0, 0, 0,  1, 2, 1  };  int SblMask2[3][3] = {  -1, 0, 1,  -2, 0, 2,  -1, 0, 1  }; |
| void BlackWhite(char\* input,char\* output) | 黑白化图像，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void RandomOperation(char\* input,char\* output,unsigned char treshold1,unsigned char treshold2,unsigned char treshold3,unsigned char treshold4,unsigned char treshold5,unsigned char treshold6,unsigned char red,unsigned char green,unsigned char blue,int color1,int color2,int color3,int color4,int color5,int color6,int color7,int color8) | 随意操作，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void SpecialEffects1(char\* input,char\* output,unsigned char red,unsigned char green,unsigned char blue) | 图像特效，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void GaussSmoothSharpen(char\* input,char\* output,int Template[3][3],int coefficient) | 高斯平滑，input是输入文件名，output是输出文件名。Template是高斯平滑模板，coefficient=16。支持24位BMP图像。 |
| void SmoothSharpen(char\* input,char\* output,int Template[3][3],int coefficient) | 平滑，input是输入文件名，output是输出文件名。Template是平滑模板，均一化处理，coefficient1 = 9。支持24位BMP图像。 |
| static void NonmaximumWithoutDoubleThresholding(const LPCTSTR input, const LPCTSTR output,double a,double b,double c) | 参考：a=0.33，b=0.33，c=0.33。支持24位BMP图像。 |
| void AdjustPixel(char\* input,char\* output,int a) | 调整像素值，input是输入文件名，output是输出文件名。a是用于设置图像像素的相关参数，如a=3。支持24位BMP图像。 |
| void NostalgicFilter(BMPMat\*\* input,BMPMat\*\* output) | 怀旧滤镜，支持24位BMP图像。 |
| void SizeTransformation(short\*\* input,short\*\* output,short height,short width,short out\_height,short out\_width) | 图像放缩，支持8位BMP图像。 |
| void ReverseColor(short\*\* input,short\*\* output,long height,long width,short GRAY\_LEVELS) | 图像反色。 |
| void Logarithm(short\*\* input,short\*\* output,long height,long width,short c) | 对数变换，默认c=10。 |
| void Gamma(short\*\* input,short\*\* output,long height,long width,double c) | 幂律（伽马）变换，默认c=1.2。 |
| void HistogramEqualization(short\*\* input, short\*\* output, long height, long width,short GRAY\_LEVELS) | 直方图均衡化。 |
| void SmoothLinearFiltering(short\*\* input, short\*\* output,long height, long width,short average[3][3]) | 平滑线性滤波器。 |
| void MedianFiltering(short\*\* input, short\*\* output, long height, long width) | 中值滤波器。 |
| void Laplace(short\*\* input,short\*\* output,long height,long width,short sharpen[3][3]) | 拉普拉斯算子。 |
| void Sobel(short\*\* input,short\*\* output,long height,long width,short soble1[3][3],short soble2[3][3]) | Sobel算子。 |
| void DFTRead(short\*\* input, double\*\* output,long height,long width) | 二维离散傅里叶变换，实部图像。 |
| void DFTImaginary(short\*\* input,double\*\* output,long height,long width) | 二维离散傅里叶变换，虚部图像。 |
| void FreSpectrum(short \*\*input, short \*\*output,long height,long width) | 傅里叶变换的平移。 |
| void IDFT(double\*\* re\_array,double\*\* im\_array,short\*\* output,long height,long width) | 二维离散傅里叶反变换。 |
| void AddGaussianNoise(short\*\* input,short\*\* output,long height,long width) | 添加高斯噪声。 |
| void AddSaltPepperNoise(short\*\* input, short\*\* output,long height,long width) | 添加椒盐噪声。 |
| void MeanFilter(short\*\* input,short\*\* output,long height,long width) | 均值滤波器。 |
| void GeometricMeanFilter(short\*\* input,short\*\* output,long height,long width,double product) | 几何均值滤波器，默认product=1.0。 |
| void HarmonicMeanFiltering(short\*\* input,short\*\* output,long height,long width,double sum) | 谐波均值滤波，默认sum=0。 |
| void InverseHarmonicMeanFiltering(short\*\* input,short\*\* output,long height,long width,int Q) | 逆谐波均值滤波，Q为滤波器的阶数，Q为正时，消除胡椒噪声，Q为负时消除盐粒噪声，Q=0为算术均值滤波器，Q=-1谐波均值滤波器，默认Q=2。 |
| void Threshold(short\*\* input,short\*\* output,long height,long width,int delt\_t,double T) | 基本全局阈值处理方法。 |
| void OTSU(short\*\* input,short\*\* output,long height,long width,short GRAY\_LEVELS) | Otsu方法进行最佳全局阈值处理。 |
| void MatrixGlobalAddition24(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相加。 |
| void MatrixGlobalSubtraction24(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相减。 |
| void MatrixGlobalMultiplication24(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相乘。 |
| void MatrixGlobalDivision24(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相除。 |
| void MatrixGlobalAddition32(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相加。 |
| void MatrixGlobalSubtraction32(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相减。 |
| void MatrixGlobalMultiplication32(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相乘。 |
| void MatrixGlobalDivision32(BMPMat\*\* input1,BMPMat\*\* input2,BMPMat\*\* output) | 基于模板矩阵的全局相除。 |
| void MatrixGlobalAddition8(unsigned char\*\* input1,unsigned char\*\* input2,unsigned char\*\* output) | 基于模板矩阵的全局相加。 |
| void MatrixGlobalSubtraction8(unsigned char\*\* input1,unsigned char\*\* input2,unsigned char\*\* output) | 基于模板矩阵的全局相减。 |
| void MatrixGlobalMultiplication8(unsigned char\*\* input1,unsigned char\*\* input2,unsigned char\*\* output) | 基于模板矩阵的全局相乘。 |
| void MatrixGlobalDivision8(unsigned char\*\* input1,unsigned char\*\* input2,unsigned char\*\* output) | 基于模板矩阵的全局相除。 |
| void ColorRectangleLocalSegmentation(char\* input,char\* output,int x1,int y1,int x2,int y2,BMPMat color) | 彩色图以矩形方式局部截取且其它部分填充，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。  函数源代码：  需引入以下头文件：  typedef struct {  unsigned char B;  unsigned char G;  unsigned char R;  unsigned char A;  }BMPMat;  声明：  unsigned char\*\* BMPRead8(char\* input);  void GenerateImage8(char\* output,unsigned char\*\* color);  BMPMat\*\* BMPRead(char\* input);  void GenerateImage(char\* output,BMPMat\*\* color,unsigned short type);  unsigned int BMPHeight(char\* input);  unsigned int BMPWidth(char\* input);  参考例程：  BMPMat color={255,255,255};  BMPMat\*\* input=BMPRead(inputfile);  BMPMat\*\* output=BMPRead(inputfile);  unsigned int height=BMPHeight(inputfile);  unsigned int width=BMPWidth(inputfile);  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  }  for(unsigned int i = y1;i<=y2;i++){  for(unsigned int j = x1;j<=x2;j++){  output[i][j].B=input[i][j].B;  output[i][j].G=input[i][j].G;  output[i][j].R=input[i][j].R;  }  }  GenerateImage(outputfile,output,24); |
| void GrayRectangleLocalSegmentation(char\* input,char\* output,int x1,int y1,int x2,int y2,unsigned char color) | 灰度图以矩形方式局部截取且其它部分填充，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。  函数源代码：  需引入以下头文件：  typedef struct {  unsigned char B;  unsigned char G;  unsigned char R;  unsigned char A;  }BMPMat;  声明：  unsigned char\*\* BMPRead8(char\* input);  void GenerateImage8(char\* output,unsigned char\*\* color);  BMPMat\*\* BMPRead(char\* input);  void GenerateImage(char\* output,BMPMat\*\* color,unsigned short type);  unsigned int BMPHeight(char\* input);  unsigned int BMPWidth(char\* input);  参考例程：  unsigned char color=255;  unsigned char\*\* input=BMPRead8(inputfile);  unsigned char\*\* output=BMPRead8(inputfile);  unsigned int height=BMPHeight(inputfile);  unsigned int width=BMPWidth(inputfile);  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  output[i][j]=color;  }  }  for(unsigned int i = y1;i<=y2;i++){  for(unsigned int j = x1;j<=x2;j++){  output[i][j]=input[i][j];  }  }  GenerateImage8(outputfile,output); |
| void ColorDrawRectangle(char\* input,char\* output,int x1,int y1,int x2,int y2,BMPMat color) | 彩色图画矩形，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。  函数源代码：  需引入以下头文件：  typedef struct {  unsigned char B;  unsigned char G;  unsigned char R;  unsigned char A;  }BMPMat;  声明：  unsigned char\*\* BMPRead8(char\* input);  void GenerateImage8(char\* output,unsigned char\*\* color);  BMPMat\*\* BMPRead(char\* input);  void GenerateImage(char\* output,BMPMat\*\* color,unsigned short type);  unsigned int BMPHeight(char\* input);  unsigned int BMPWidth(char\* input);  参考例程：  BMPMat color={255,255,255};  BMPMat\*\* input=BMPRead(inputfile);  BMPMat\*\* output=BMPRead(inputfile);  unsigned int height=BMPHeight(inputfile);  unsigned int width=BMPWidth(inputfile);  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  }  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  if(j>=x1&&j<=x2&&i==y1)  {  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  if(j==x1&&i>=y1&&i<=y2)  {  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  if(j==x2&&i>=y1&&i<=y2)  {  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  if(j>=x1&&j<=x2&&i==y2)  {  output[i][j].B=color.B;  output[i][j].G=color.G;  output[i][j].R=color.R;  }  }  }  GenerateImage(outputfile,output,24); |
| void GrayDrawRectangle(char\* input,char\* output,int x1,int y1,int x2,int y2,unsigned char color) | 灰度图画矩形，（x1，y1）是矩形的左上角的坐标，（x2,y2）是矩形的右下角的坐标。  函数源代码：  需引入以下头文件：  typedef struct {  unsigned char B;  unsigned char G;  unsigned char R;  unsigned char A;  }BMPMat;  声明：  unsigned char\*\* BMPRead8(char\* input);  void GenerateImage8(char\* output,unsigned char\*\* color);  BMPMat\*\* BMPRead(char\* input);  void GenerateImage(char\* output,BMPMat\*\* color,unsigned short type);  unsigned int BMPHeight(char\* input);  unsigned int BMPWidth(char\* input);  参考例程：  unsigned char color=255;  unsigned char\*\* input=BMPRead8(inputfile);  unsigned char\*\* output=BMPRead8(inputfile);  unsigned int height=BMPHeight(inputfile);  unsigned int width=BMPWidth(inputfile);  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  output[i][j]=color;  }  }  for(unsigned int i = 0;i<height;i++){  for(unsigned int j = 0;j<width;j++){  if(j>=x1&&j<=x2&&i==y1)  {  output[i][j]=color;  }  if(j==x1&&i>=y1&&i<=y2)  {  output[i][j]=color;  }  if(j==x2&&i>=y1&&i<=y2)  {  output[i][j]=color;  }  if(j>=x1&&j<=x2&&i==y2)  {  output[i][j]=color;  }  }  }  GenerateImage8(outputfile,output); |
| void Relief(BMPMat\*\* input,BMPMat\*\* output,int value) | 浮雕效果，默认value=128。 |
| void Relief(unsigned char\*\* input,unsigned char\*\* output,int value) | 浮雕效果，默认value=128。 |
| void Sharpening(BMPMat\*\* input,BMPMat\*\* output,double degree) | 图像锐化，默认degree=0.3。 |
| void Sharpening(unsigned char\*\* input,unsigned char\*\* output,double degree) | 图像锐化，默认degree=0.3。 |
| void Soften(BMPMat\*\* input,BMPMat\*\* output,int value) | 图像柔化，默认value=9。 |
| void Soften(unsigned char\*\* input,unsigned char\*\* output,int value) | 图像柔化，默认value=9。 |
| void flipX(char\* input,char\* output) | X方向翻转，支持JPG文件。 |
| void flipY(char\* input,char\* output) | Y方向翻转，支持JPG文件。 |
| void Crop(char\* input,char\* output,uint16\_t start\_x, uint16\_t start\_y, uint16\_t new\_height, uint16\_t new\_width) | 裁剪。 |
| void Resize(char\* input,char\* output,int new\_width, int new\_height) | 缩放。 |
| void Scale(char\* input,char\* output,double ratio) | 比例。 |
| void GrayscaleAvg(char\* input,char\* output) | 灰度平均值。 |
| void grayscaleLum(char\* input,char\* output) | 灰度亮度。 |
| void ColorMask(char\* input,char\* output,float r,float g,float b) | 彩色遮罩。 |
| void PixeLize(char\* input,char\* output,int strength) | 像素化，参考：strength=2。 |
| void GaussianBlur(char\* input,char\* output,int strength) | 高斯模糊，参考：strength=2。 |
| void EdgeDetection(char\* input,char\* output,double cutoff) | 边缘检测，参考：cutoff=115。 |
| void Sharpen(char\* input,char\* output) | 锐化。 |
| void CannyProcessing(char\* input, char\* output, int a) | Canny处理，a可以为1、2、3、4、5。支持BMP图像。 |
| void AverageGrayScale(char\* input,char\* output) | 平均灰度化。 |
| void SimpleBW(char\* input,char\* output) | 简易BW。 |
| void AdvancedBW(char\* input,char\* output) | 高级BW。 |
| void UniformNoise(char\* input,char\* output) | 均匀噪声。 |
| void GaussianNoise(char\* input,char\* output,double sigma) | 高斯噪声。 |
| void SaltAndPepperNoise(char\* input,char\* output) | 椒盐噪声。 |
| void MeanFilter(char\* input,char\* output,int filterSize) | 均值滤波。 |
| void GaussianFilter(char\* input,char\* output,double sigma) | 高斯滤波。 |
| void MedianFilter(char\* input,char\* output,int size) | 中值滤波。 |
| void EfficientMeanFilter(char\* input,char\* output,int filterSize) | 有效均值滤波器。 |
| double MeanSquaredError(char\* input1,char\* input2,char\* output) | 均方误差，计算图像相似度，返回值越小图像就越相似。 |
| void GrayAVS(char\* input,char\* output,float k,float b) | input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void HistogramEqualize24(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void MatrixTransformation(char\* input,char\* output) | 矩阵变换。 |
| void Binarization(char\* input,char\* output) | 二值化。 |
| void ChannelSeparation\_B(char\* input,char\* output) | 分离出蓝色通道。 |
| void ChannelSeparation\_G(char\* input,char\* output) | 分离出绿色通道。 |
| void ChannelSeparation\_R(char\* input,char\* output) | 分离出红色通道。 |
| void Inverse(char\* input,char\* output) | 反转。 |
| void HistogramEqualization8(char\* input,char\* output) | 直方图均衡化。 |
| void Smooth(char\* input,char\* output) | 平滑。 |
| void CannyEdge(char\* input,char\* output) | Canny算子。 |
| void EdgeEnhance(char\* input,char\* output) | 边缘增强。 |
| void AvrFilter(char\* input,char\* output1,char\* output2,int M,int N) | input是输入文件名，output是输出文件名。如M=21，N=1。支持8位BMP图像。 |
| void GryOppositionSSE(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void MedianFilter(char\* input,char\* output,int M,int N) | 中值滤波器，input是输入文件名，output是输出文件名。如M=5，N=5。支持8位BMP图像。 |
| void EdgeSharpeningGry(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void SJGryandRiceTest(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void TextTest(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void RedChannel(char\* input,char\* output) | 生成图像的红色通道图像，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void GreenChannel(char\* input,char\* output) | 生成图像的绿色通道图像，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void BlueChannel(char\* input,char\* output) | 生成图像的蓝色通道图像，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void HistogramStatistics(char\* input,char\* output) | 直方图统计，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void HistogramEqualization1(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void ReflectionRay(char\* input,char\* output) | 反射线，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void MeanFiltering24(char\* input,char\* output) | 均值滤波，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void MedianFiltering24(char\* input,char\* output) | 中值滤波，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void ZoomOutAndZoomIn(char\* input,char\* output,double value) | 缩放（双线性插值），input是输入文件名，output是输出文件名。value是放大倍数，如value=0.5。支持24位BMP图像。 |
| void Translation24(char\* input,char\* output,int x,int y) | 平移，input是输入文件名，output是输出文件名。x是横轴的平移量，y是纵轴的平移量，如x=-10，y=-30。支持24位BMP图像。 |
| void Mirror24(char\* input,char\* output) | 镜像，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void Rotate24(char\* input,char\* output,double degree) | 旋转，input是输入文件名，output是输出文件名。degree是旋转的度数。支持24位BMP图像。 |
| void GivenThresholdMethod(char\* input,char\* output,int threshold) | 给定阈值法处理图像，使图片黑白化，input是输入文件名，output是输出文件名。threshold是给定的阈值，如threshold=100。支持24位BMP图像。 |
| void IterativeThresholdMethod(char\* input,char\* output) | 迭代阈值法处理图像，使图片黑白化，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void OstuThresholdSegmentationMethod(char\* input,char\* output) | Ostu（大津法）阈值分割，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void Repudiation(char\* input,char\* output) | 将伪彩图片反白，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void Gray1(char\* input,char\* output) | 将彩色图片变成灰度图片，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void CorrectMethod(char\* input,char\* output) | 正确法，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void ChannelSeparation1(char\* input,char\* Routput,char\* Goutput,char\* Boutput) | 对图像分理出其中的RGB分量并分别保存为独立的图像，input是输入文件名，Routput是红色通道图像，Goutput是绿色通道图像，Boutput是绿色通道图像。支持24位BMP图像。 |
| void ReverseColor(char\* input,char\* output) | 对灰度图进行反色，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| Image1\* LoadImage1(char\* input) | BMP图像读取，input是输入文件名。支持8位和24位BMP图像。  返回Image1型数据，Image1型数据的结构如下：  typedef struct  {  int width;  int height;  int channels; //图像通道数  unsigned char\* Data; //像素数据  }Image1; |
| void SaveImage1(char\* output,Image1\* img) | 将Image1型数据保存为BMP图像，output是生成的BMP图像文件名，img是要保存的图像数据。支持8位和24位BMP图像。  Image1型数据的结构如下：  typedef struct  {  int width;  int height;  int channels; //图像通道数  unsigned char\* Data; //像素数据  }Image1; |
| void ImageContrastExtension(char\* input,char\* output,double m,double g1,double g2,double a) | 图像对比度扩展，input是输入文件名，output是输出文件名。  其中，可参考：double m=1.5,g1=100.0,g2=200.0; m对应斜率  double a=(255.0-m\*(g2-g1))/(255.0-(g2-g1));  支持8位BMP图像。 |
| void Binaryzation(char\* input,char\* output,int threshold) | 图像二值化，input是输入文件名，output是输出文件名。threshold是将灰度值转化为二值的阈值，如threshold=80。支持24位BMP图像。 |
| void GlobalBinarization(char\* input,char\* output) | 全局二值化，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void AdaptiveBinarization(char\* input,char\* output) | 自适应二值化，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void ExpansionOperation(char\* input,char\* output) | 膨胀操作，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void CorrosionOperation(char\* input,char\* output) | 腐蚀操作，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Operation1(char\* input,char\* output) | 开操作，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Closed1(char\* input,char\* output) | 闭操作，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Negative1(char\* input,char\* output) | 图像反色，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| void Negative(char\* input,char\* output) | 图像反色，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void ImageSynthesis(char\* input1,char\* input2,char\* output) | 图像合成。 |
| void BlackWhite(char\* input,char\* output,float T,int border) | 黑白化，支持8位和24位BMP图像。T是阈值，border是边界范围，如：T=50，border=0。 |
| void Mosaic(char\* input,char\* output,int w,int h) | 马赛克化图像，w和h是输出图像的宽和高。支持PNG图像。 |
| IMAGE Image\_bmp\_load(char\* filename) | 加载BMP图片。 |
| void Image\_bmp\_save(char\* filename,IMAGE im) | 保存BMP图片。 |
| IMAGE TransformShapeNearest(IMAGE input, unsigned int newWidth, unsigned int newHeight) | 缩放图片(最近邻插值法)。 |
| IMAGE TransformShapeLinear(IMAGE input, unsigned int newWidth, unsigned int newHeight) | 缩放图片(双线性插值法)。 |
| IMAGE TransformShapeWhirl(IMAGE input, float angle) | 图像的任意角度的旋转。 |
| IMAGE TransformShapeUpturn(IMAGE input, int a) | 图像的镜像翻转。 |
| void TransformColorGrayscale(IMAGE im, int GrayscaleMode) | 彩色图转灰度图，对于GrayscaleMode的值：1表示加权法，2表示最值法，3表示均值法，4表示红色分量法，5表示绿色分量法，6表示蓝色分量法。 |
| void TransformColorBWDIY(IMAGE input, unsigned char Threshold) | 二值图(自定义阈值法)。 |
| void TransformColorBWOSTU(IMAGE input) | 二值图(大津法OSTU，适用双峰直方图。) |
| void TransformColorBWTRIANGLE(IMAGE input) | 二值图（三角法TRIANGLE，适用单峰直方图。） |
| IMAGE TransformColorBWAdaptive(IMAGE input, int areaSize) | 二值图(自适应阈值法，areaSize=25较合适) |
| IMAGE TransformColorBWGrayscale(IMAGE input, int areaSize) | 二值图(用二值图表示灰度变化,areaSize=25较合适) |
| void TransformColorOpposite(IMAGE input) | 反色。 |
| IMAGE TransformColorHistogramPart(IMAGE input) | 直方图均衡化（分步计算，效果更加柔和）。 |
| IMAGE TransformColorHistogramAll(IMAGE input) | 直方图均衡化（整体计算，效果更加尖锐）。 |
| IMAGE KernelsUseDIY(IMAGE input, double\* kernels, int areaSize, double modulus) | 卷积操作（自定义）。 |
| IMAGE WavefilteringMedian(IMAGE input) | 中值滤波。 |
| IMAGE WavefilteringGauss(IMAGE input,double KERNELS\_Wave\_Gauss[9],int a,double b) | 高斯滤波。  高斯滤波卷积核：  double KERNELS\_Wave\_Gauss[9] =  {  1, 2, 1,  2, 4, 2,  1, 2 ,1  }; |
| IMAGE Wavefiltering\_LowPass(IMAGE input, double\* kernels) | 低通滤波。  // 低通滤波卷积核 LP1  double KERNELS\_Wave\_LowPass\_LP1[9] =  {  1 / 9.0, 1 / 9.0, 1 / 9.0,  1 / 9.0, 1 / 9.0, 1 / 9.0,  1 / 9.0, 1 / 9.0, 1 / 9.0  };  // 低通滤波卷积核 LP2  double KERNELS\_Wave\_LowPass\_LP2[9] =  {  1 / 10.0, 1 / 10.0, 1 / 10.0,  1 / 10.0, 1 / 5.0, 1 / 10.0,  1 / 10.0, 1 / 10.0, 1 / 10.0  };  // 低通滤波卷积核 LP3  double KERNELS\_Wave\_LowPass\_LP3[9] =  {  1 / 16.0, 1 / 8.0, 1 / 16.0,  1 / 8.0, 1 / 4.0, 1 / 8.0,  1 / 16.0, 1 / 8.0, 1 / 16.0  }; |
| IMAGE WavefilteringHighPass(IMAGE input, double\* kernels) | 高通滤波。  // 高通滤波卷积核 HP1  double KERNELS\_Wave\_HighPass\_HP1[9] =  {  -1, -1, -1,  -1, 9, -1,  -1, -1 ,-1  };  // 高通滤波卷积核 HP2  double KERNELS\_Wave\_HighPass\_HP2[9] =  {  0, -1, 0,  -1, 5, -1,  0, -1 ,0  };  // 高通滤波卷积核 HP3  double KERNELS\_Wave\_HighPass\_HP3[9] =  {  1, -2, 1,  -2, 5, -2,  1, -2 ,1  }; |
| IMAGE Wavefiltering\_Average(IMAGE input,double\* KERNELS\_Wave\_Average) | 均值滤波。  // 均值滤波卷积核  double KERNELS\_Wave\_Average[25] =  {  1, 1, 1, 1, 1,  1, 1, 1, 1, 1,  1, 1, 1, 1, 1,  1, 1, 1, 1, 1,  1, 1, 1, 1, 1  }; |
| IMAGE EdgeDetectionDifference(IMAGE input, double\* kernels) | 差分边缘检测。  // 差分垂直边缘检测卷积核  double KERNELS\_Edge\_difference\_vertical[9] =  {  0, 0, 0,  -1, 1, 0,  0, 0, 0  };  // 差分水平边缘检测卷积核  double KERNELS\_Edge\_difference\_horizontal[9] =  {  0,-1, 0,  0, 1, 0,  0, 0, 0  };  // 差分垂直和水平边缘检测卷积核  double KERNELS\_Edge\_difference\_VH[9] =  {  -1, 0, 0,  0, 1, 0,  0, 0, 0  }; |
| IMAGE KernelsUseEdgeSobel(IMAGE input, double\* kernels1, double\* kernels2) | Sobel边缘检测。  // Sobel X边缘检测卷积核  double KERNELS\_Edge\_Sobel\_X[9] =  {  -1, 0, 1,  - 2, 0, 2,  -1, 0, 1  };  // Sobel Y边缘检测卷积核  double KERNELS\_Edge\_Sobel\_Y[9] =  {  -1, -2, -1,  0, 0, 0,  1, 2, 1  }; |
| IMAGE EdgeDetectionLaplace(IMAGE input, double\* kernels) | Laplace边缘检测。  // Laplace边缘检测卷积核 LAP1  double KERNELS\_Edge\_Laplace\_LAP1[9] =  {  0, 1, 0,  1, -4, 1,  0, 1, 0  };  // Laplace边缘检测卷积核 LAP2  double KERNELS\_Edge\_Laplace\_LAP2[9] =  {  -1, -1, -1,  -1, 8, -1,  -1, -1, -1  };  // Laplace边缘检测卷积核 LAP3  double KERNELS\_Edge\_Laplace\_LAP3[9] =  {  -1, -1, -1,  -1, 9, -1,  -1, -1, -1  };  // Laplace边缘检测卷积核 LAP4  double KERNELS\_Edge\_Laplace\_LAP4[9] =  {  1, -2, 1,  -2, 8, -2,  1, -2, 1  }; |
| IMAGE MorphologyErosion(IMAGE input, double\* kernels) | 腐蚀。  // 腐蚀卷积核  double KERNELS\_Morphology\_Erosion\_cross[9] =  {  0, 1, 0,  1, 1, 1,  0, 1, 0  }; |
| IMAGE MorphologyDilation(IMAGE input, double\* kernels) | 膨胀。  // 膨胀卷积核  double KERNELS\_Morphology\_Dilation\_cross[9] =  {  0, 1, 0,  1, 1, 1,  0, 1, 0  }; |
| IMAGE Pooling(IMAGE input, int lenght) | 池化。 |
| IGIMAGE IntegralImage(IMAGE input) | 获得积分图（在此之前要保证图片是“白底黑字”）。 |
| void FaceDetection(char\* input,char\* output,double\* KERNELS\_Wave\_Average) | 人脸检测。 |
| IMAGE FaceDetection(IMAGE input1,IMAGE input2,double\* KERNELS\_Wave\_Average) | 人脸检测。  需引入以下结构体：  typedef struct tagBGRA  {  unsigned char blue;  unsigned char green;  unsigned char red;  unsigned char transparency;  }BGRA, \*PBGRA;  typedef struct tagIMAGE  {  unsigned int w;  unsigned int h;  BGRA\* color;  }IMAGE, \*PIMAGE;  声明：  IMAGE Image\_bmp\_load(char\* filename);  void Image\_bmp\_save(char\* filename,IMAGE im);  参考：  // 用于处理  IMAGE input2 = Image\_bmp\_load(inputfile);  // 用于保存  IMAGE input2= Image\_bmp\_load(inputfile);  input2=FaceDetection(input1,input2,KERNELS\_Wave\_Average);  // 保存图片  Image\_bmp\_save(outputfile, input2); |
| void IntegralDiagram(unsigned int \*input, unsigned int \*output, int width, int height) | 图像积分图。 |
| void ImageEncryption(char\* inFileName,char\* outFileName,char key) | 图像加密，支持8位、24位和32位BMP图像。inFileName是原图图像文件名，outFileName是解密图像文件名，key是密钥，如key=255。 |
| void ImageDecryption(char\* inFileName,char\* outFileName,char key) | 图像解密，inFileName是加密图像文件名，outFileName是解密图像文件名，key是密钥，如key=255。支持8位、24位和32位BMP图像。 |
| void Compress8(string input,string output) | 图像压缩，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void Decompression(string input,string output) | 图像解压，input是输入文件名，output是输出文件名。支持8位BMP图像压缩后的结果文件。 |
| void HorizontalMirror(char\* input,char\* output) | 水平镜像，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void MirrorVertically(char\* input,char\* output) | 垂直镜像，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void XMirroring(char\* input,char\* output) | X镜像，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void YMirroring(char\* input,char\* output) | Y镜像，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void ImageConvolution(char\* input,char\* output,double\*\* Kernel,int n,int m) | 图像卷积，input是输入文件名，output是输出文件名。Kernel是卷积核，如double Kernel[3][3] = {{-0.225,-0.225-0.225},{-0.225,1,-0.225},{-0.225,-0.225,-0.225}}；n是Kernel的第一维的大小，m是Kernel的第二维的大小，形如Kernel[n][m]。支持24位BMP图像。 |
| void SpatialMeanFiter(char\* input,char\* output,int radius) | 空间均值过滤器，参考：radius=3。 |
| void SpatialMedianFiter(char\* input,char\* output,int radius) | 空间中值过滤器，参考：radius=3。 |
| void SpatialMaxFiter(char\* input,char\* output,int radius) | 空间最大过滤器，参考：radius=3。 |
| void SpatialMinFiter(char\* input,char\* output,int radius) | 空间最小过滤器，参考：radius=3。 |
| void SpatialGaussFiter(char\* input,char\* output,int radius) | 空间高斯过滤器，参考：radius=3。 |
| void SpatialStatisticalFiter(char\* input,char\* output,int radius,float T) | 空间统计滤波器，参考：radius=3，T=0.2。 |
| void FFTAmp(char\* input,char\* output,bool inv) | FFT放大器，参考：inv=false。 |
| void FFTPhase(char\* input,char\* output,bool inv) | FFT相位，参考：inv=false。 |
| void STDFT1(char\* input,char\* output,bool inv) | 参考：inv=false。 |
| void STDFT2(char\* input,char\* output,bool inv) | 参考：inv=false。 |
| void SpectrumShaping(char\* input,char\* inputMsk,char\* output) | 图像频域滤波，FFT变换--相位谱，inputMsk是输入的掩膜图像名。 |
| void Translation(char\* input,char\* output,int x,int y,unsigned char color) | 图像平移，input是输入文件名，output是输出文件名。x和y是在X轴和Y轴平移的量，以右为正向，color是平移后非原图区域填充的颜色，如color=100。支持8位BMP图像。 |
| void CrossDenoising24(BMPMat\*\* input,BMPMat\*\* output,BMPMat threshold,BMPMat target) | 图像去除某种像素，output用于保存结果（与input大小相同）。 |
| void CrossDenoising8(unsigned char\*\* input,unsigned char\*\* output,unsigned char threshold,unsigned char target) | 图像去除某种像素，output用于保存结果（与input大小相同）。 |
| void ImageDecontamination(BMPMat\*\* input,BMPMat\*\* output,int x1,int y1,int x2,int y2) | 图像去污。(x1,y1)是矩形污渍区的左上角坐标，(x2,y2)是矩形污渍区的右下角坐标。 |
| void ImageDecontamination(unsigned char\*\* input,unsigned char\*\* output,int x1,int y1,int x2,int y2) | 图像去污。(x1,y1)是矩形污渍区的左上角坐标，(x2,y2)是矩形污渍区的右下角坐标。 |
| void ImageSharpening(char\* input,char\* output) | 图像锐化，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void SharpenLaplace(char\* input,char\* output,int ratio) | 拉普拉斯锐化，参考：ratio=100。 |
| void SharpenUSM(char\* input,char\* output,int radius,int amount,int threshold) | USM锐化，参考：radius=5，amount=400，threshold=50。 |
| void DrawRectangle(char\* input,char\* output,int x1,int y1,int x2,int y2,unsigned char red,unsigned char green,unsigned char blue) | 在24位BMP图像上通过传入的参数画一个矩形。input是输入文件名，output是输出文件名。(x1,y1)是矩形坐上顶点的坐标，(x2,y2)是矩形右下顶点的坐标；red是矩形线框的红色分量，green是矩形线框的绿色分量，blue是矩形的蓝色分量。 |
| void GenerateBmp(unsigned char\* pData,int width,int height,char\* filename) | 生成BMP图像，pData是图像的像素数据，width和height是图像的宽和高，filename是生成的图像的文件名。 |
| void Jpg24ImageGeneration(char\* filename,unsigned int width, unsigned int height, unsigned char\* img) | JPG图像生成，filename是生成的JPG图像文件名，width是图像的宽，height是图像的高，img是图像的像素数据。 |
| void ImageScalingNearestNeighborInterpolation(char\* input,char\* output,float lx,float ly) | 最近邻插值法去栅格，input是输入文件名，output是输出文件名。lx和ly是长和宽需要缩放的倍数。支持8位BMP图像。 |
| void ImageScalingBilinearInterpolation(char\* input,char\* output,float lx,float ly) | 双线性插值法去栅格，input是输入文件名，output是输出文件名。lx和ly是长和宽需要缩放的倍数。支持8位BMP图像。 |
| void BilinearInterpolationScaling(char\* input,char\* output,float ExpScalValue) | 双线性插值，input是输入文件名，output是输出文件名。ExpScalValue是期望的缩放倍数（允许小数）。支持BMP图像。 |
| void NearestNeighborInterpolationScaling(char\* input,char\* output,float ExpScalValue) | 最近邻插值，input是输入文件名，output是输出文件名。ExpScalValue是期望的缩放倍数（允许小数）。支持BMP图像。 |
| void ZoomImg(unsigned char \*input,unsigned char \*output,int sw,int sh,int channels,int dw,int dh) | 二次线性插值图像缩放。 |
| void CrossDenoising24(BMPMat\*\* input,BMPMat\*\* output,BMPMat target,BMPMatdouble weight) | 图像修复，output用于保存结果（与input大小相同），target是污点像素，weight是修复权重系数。 |
| void CrossDenoising8(unsigned char\*\* input,unsigned char\*\* output,unsigned char target,double weight) | 图像修复，output用于保存结果（与input大小相同），target是污点像素，weight是修复权重系数。 |
| void RotateRight90Degrees(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像，向右旋转90度。 |
| void RotateLeft90Degrees(char\* input,char\* output) | input是输入文件名，output是输出文件名。支持8位BMP图像，向左旋转90度。 |
| void ImageRotation(char\* input,char\* output,double angle) | 图像旋转，input是输入文件名，output是输出文件名。支持8位BMP图像。angle是要旋转的角度。 |
| void Rotation8(char\* input,char\* output,double Angle,int x1,int y1,int x2,int y2,unsigned char color) | 图像旋转，input是输入文件名，output是输出文件名。支持8位BMP图像。Angle是要旋转的角度数；x1、y1、x2、y2是旋转所围绕的中心点的坐标，color是旋转后非原图区域的填充颜色。 |
| void Rotation24(char\* input,char\* output,double Angle,int x1,int y1,int x2,int y2,unsigned char red,unsigned char green,unsigned char blue) | 图像旋转，input是输入文件名，output是输出文件名。支持24位BMP图像。Angle是要旋转的角度数；x1、y1、x2、y2是旋转所围绕的中心点的坐标；red、green、blue分别是旋转后非原图区域要填充的颜色的红绿蓝分量。 |
| void Rotation(char\* input,char\* output,int angle,unsigned char color) | 图像旋转，input是输入文件名，output是输出文件名。支持8位BMP图像。angle是旋转的角度，color是旋转后非原图区域填充的颜色，如color=100。 |
| void Rotate(char\* input,char\* output,int angle) | 图像旋转，input是输入文件名，output是输出文件名。支持BMP图像。angle是旋转的角度。 |
| void imgRotate90Gray(unsigned char \*input,unsigned char \*output,int sw,int sh,int \*dw,int \*dh) | 灰度图像旋转90。 |
| void imgRotate90Color(unsigned char \*input,unsigned char \*output,int sw,int sh,int \*dw,int \*dh) | 彩色图像旋转90。 |
| void imgRotate270Gray(unsigned char \*input,unsigned char \*output,int sw,int sh,int \*dw,int \*dh) | 灰阶图像旋转270。 |
| void imgRotate270Color(unsigned char \*input,unsigned char \*output,int sw,int sh,int \*dw,int \*dh) | 彩色图像旋转270。 |
| void imgRotate180Gray(unsigned char \*Img,int w,int h) | 灰阶图像旋转180，结果保存在原输入数组中。 |
| void imgRotate180Color(unsigned char \*Img,int w,int h) | 彩色图像旋转180，结果保存在原输入数组中。 |
| void imgRBExchange(unsigned char \*Img,int w,int h) | 彩色图像R、B互换，结果保存在原输入数组中。 |
| void NoiseUniform(char\* input,char\* output,double a,double b) | 均匀分布噪声，参考：a=0，b=0.2。 |
| void NoiseGauss(char\* input,char\* output,float mean,float delta) | 高斯噪声，参考：mean=0，delta=31。 |
| void NoiseRayleigh(char\* input,char\* output,float a,float b) | 瑞利噪声，参考：a=0，b=200。 |
| void NoiseExp(char\* input,char\* output,float a) | 指数噪声，参考：a=0.1。 |
| void NoiseImpulse(char\* input,char\* output,float a,float b) | 椒盐噪声，参考：a=0.2，b=0.2。 |
| void grayToColor(FILE\* input,FILE\* output) | 灰色转伪彩色，input是输入文件，output是输出文件。支持8位和24位BMP图像。 |
| void ImageThinning(char\* input,char\* output,char\*\* str,int n,int m1,int a,int b) | 图像细化，input是输入文件名，output是输出文件名。支持4位BMP图像。n是str第一维的大小，m1是第二维的大小，形如str[n][m1]；a和b是相关的调节参数，可以为a=3，b=5。  参考模板：  char str[6][8] = { { 0, 0, 0, 0, 0, 0, 0, 0, }, { 255, 0, 255, 0, 0, 255, 0, 0 },  { 255, 0, 255, 255, 0, 255, 0, 255 }, { 255, 255, 255, 0, 0, 255, 255, 255 },  { 255, 0, 255, 255, 0, 255, 255, 255 }, { 0, 255, 255, 255, 255, 255, 255, 255 } }; |
| int MinimumValueOfImagePixels(char\* filename) | 返回图像像素的最小值，filename是输入的图像文件名。支持8位和24位BMP图像。 |
| int MaximumValueOfImagePixels(char\* filename) | 返回图像像素的最大值，filename是输入的图像文件名。支持8位和24位BMP图像。 |
| float AverageValueOfImagePixels(char\* filename) | 返回图像像素的均值，filename是输入的图像文件名。支持8位和24位BMP图像。 |
| double StandardDeviationOfImagePixels(char\* filename) | 返回图像像素的标准差，filename是输入的图像文件名。支持8位和24位BMP图像。 |
| double EntropyOfImage(char\* filename) | 返回图像的熵，支持8位和24位BMP图像。 |
| float\* CountTheFrequencyOfPixels(char\* filename) | filename是输入的图像文件名。存储每个像素的频率，像素值为0~255，返回值数组中的元素序号即为像素值，该序号在数组下的值即为这个像素的频率。支持8位和24位BMP图像。 |
| void Rotate(char\* input,char\* output,int angle,int interpolation) | 图像旋转。参考：angle=80，interpolation=0或interpolation=1。 |
| void HSV(char\* input,char\* output,int h,int s,int v) | 图像色调饱和度明度调节，参考：h=120，s=60，v=20。 |
| void ColorTransfer1(char\* input1,char\* input2,char\* output) | 颜色转移，支持BMP图像。 |
| void OilpaintFilter(char\* input,char\* output,int radius,int smooth) | 油画滤镜，参考：radius=10，smooth=100。 |
| void HaloFilter(char\* input,char\* output,int ratio) | 晕角滤镜，参考：ratio=100。 |
| void GrayHistogram(char\* input,char\* output,int hWidth,int hHeight) | 灰度直方图，参考：hWidth=256，hHeight=100。 |
| void RedHistogram(char\* input,char\* output,int hWidth,int hHeight) | 红色通道直方图，参考：hWidth=256，hHeight=100。 |
| void GreenHistogram(char\* input,char\* output,int hWidth,int hHeight) | 绿色通道直方图，参考：hWidth=256，hHeight=100。 |
| void BlueHistogram(char\* input,char\* output,int hWidth,int hHeight) | 蓝色通道直方图，参考：hWidth=256，hHeight=100。 |
| void HistogramEqualization2(char\* input,char\* output,int imgBit) | 直方图均衡化，input是输入文件名，output是输出文件名。支持8位和24位BMP图像。imgBit是输入图像的位数。 |
| void HistogramEqualization3(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持8位和24位BMP图像。 |
| void HistogramEqualization4(char\* input,char\* output) | 直方图均衡化，input是输入文件名，output是输出文件名。支持8位和24位BMP图像。input是输入文件名称，out是输出文件名称。 |
| void HistogramEqualization(char\* input,char\* output,int hWidth,int hHeight) | 直方图均衡化，参考：hWidth=256，hHeight=100。 |
| void GrayHistogramEqualization(char\* input,char\* output,int hWidth,int hHeight) | 灰度直方图，参考：hWidth=256，hHeight=100。 |
| void RedHistogramEqualization(char\* input,char\* output,int hWidth,int hHeight) | 红色通道直方图，参考：hWidth=256，hHeight=100。 |
| void GreenHistogramEqualization(char\* input,char\* output,int hWidth,int hHeight) | 绿色通道直方图，参考：hWidth=256，hHeight=100。 |
| void BlueHistogramEqualization(char\* input,char\* output,int hWidth,int hHeight) | 蓝色通道直方图，参考：hWidth=256，hHeight=100。 |
| void GrayScaleStretch(char\* input,char\* output,int hWidth,int hHeight) | 灰度级拉伸，参考：hWidth=256，hHeight=100。 |
| void GrayHistagramStretch(char\* input,char\* output,int hWidth,int hHeight) | 灰度直方图拉伸，参考：hWidth=256，hHeight=100。 |
| void RedHistagramStretch(char\* input,char\* output,int hWidth,int hHeight) | 红色通道直方图，参考：hWidth=256，hHeight=100。 |
| void GreenHistagramStretch(char\* input,char\* output,int hWidth,int hHeight) | 绿色通道直方图，参考：hWidth=256，hHeight=100。 |
| void BlueHistagramStretch(char\* input,char\* output,int hWidth,int hHeight) | 蓝色通道直方图，参考：hWidth=256，hHeight=100。 |
| void MedianFiltering1(char\* input,char\* output) | 中值滤波，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void MedianFiltering2(char\* input,char\* output) | 中值滤波，input是输入文件名，output是输出文件名。支持8位和24位BMP图像。 |
| void ThresholdProcessing(char\* input,char\* output,int Threshold) | 阈值处理，input是输入文件名，output是输出文件名。支持8位BMP图像。Threshold是阈值相关参数，如Threshold=0.001。 |
| void OTSUProcessing(char\* input,char\* output) | 大津法处理，input是输入文件名，output是输出文件名。支持8位BMP图像。 |
| void OBJtoTGA(char\* input,char\* output,int width,int height) | OBJ转TGA。 |
| void ToRIM(char\* input,char\* output) | 一般图像转到RIM图像，支持PNG、JPG和TGA图像。 |
| void ToImage(char\* input,char\* output,int jpg\_quality) | RIM图像转到一般图像，支持PNG、JPG和TGA图像。jpg\_quality=25。 |
| void ImprimanteThermique(char\* input,char\* output,ARRAY3 skip\_cmd,unsigned short PRINTER\_TYPE\_BMP,unsigned char mode,unsigned int FILE\_TYPE\_AD,unsigned char a,unsigned char b) | 将1位深度的单色BMP图片转成热敏打印机的位图打印输出，支持的热敏打印机的位图打印指令为<strong>ESC \*</strong>指令。  typedef unsigned char ARRAY3[3];参考：output="output.pbin"，skip\_cmd = {0x1B, 0x4A, 0x00}，PRINTER\_TYPE\_BMP是打印机位图打印指令码标识，PRINTER\_TYPE\_BMP=(0x2A1B)，mode是打印机位图打印模式，mode=33，FILE\_TYPE\_AD是图片类型，"AD"表示广告图片，FILE\_TYPE\_AD=(0x4441)，a=0x80，b=1。 |
| void WhiteBalance(const char\* input,const char\* output) | 白平衡。 |
| void Sobel(char\* input,char\* output,double magnScale,double threshold) | Sobel算子，magnScale=0.35，threshold=130。支持PGM和PBM图像。 |
| void Canny(char\* input,char\* output,double magnScale,double lowThreshold,double highThreshold) | Canny算子，magnScale=0.35，lowThreshold=55，highThreshold=120。支持PGM和PBM图像。 |
| void BlackWhite(char\* input,char\* output,int threshold,int background) | 黑白化，threshold=100，background=0。支持PGM和PBM图像。 |
| void ConnectedComponents(char\* input,char\* output,int threshold,int background,int threshold1) | 区域连通，threshold=100，background=0，threshold1=100。支持PGM和PBM图像。 |
| void CleanImage(char\* input,char\* output) | 清洁图像。支持PGM和PBM图像。 |
| void NoiseImage(char\* input,char\* output,float probability) | 噪声化图像，probability=0.1。支持PGM和PBM图像。 |
| void HoughTransformCircle1(char\* input,char\* output,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1, double gamma1) | 圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma和kernelSize用于平滑的高斯5x5内核，sigma=1.0，kernelSize=5。  如果scale==0，则值保持不变，但如果它们低于0或高于255，则它们分别设置为0或255。  如果scale！=0，则对值进行缩放，使得最小值为零，最大值为255。  设置gamma值允许指数缩放，启用gamma==1.0。  支持PGM和PBM图像。 |
| void HoughTransformCircle2(char\* input,char\* output,int number,int minDist,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1, double gamma1) | 圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma和kernelSize用于平滑的高斯5x5内核，sigma=1.0，kernelSize=5，number=10表示图像的目视检查有10个圆圈，minDist=35。  如果scale==0，则值保持不变，但如果它们低于0或高于255，则它们分别设置为0或255。  如果scale！=0，则对值进行缩放，使得最小值为零，最大值为255。  设置gamma值允许指数缩放，启用gamma==1.0。  支持PGM和PBM图像。 |
| double\*\* HoughTransformCircle3(char\* input,char\* output,int number,int minDist,double sigma,int kernelSize,int scale,double gamma,double magnScale,double lowThreshold,double highThreshold,int scale1, double gamma1) | 圆检测。scale1=1，gamma1=1.0，magnScale=0.5，lowThreshold=85，highThreshold=150，scale=0，gamma=1.0，sigma和kernelSize用于平滑的高斯5x5内核，sigma=1.0，kernelSize=5，number=10表示图像的目视检查有10个圆圈，minDist=35。  如果scale==0，则值保持不变，但如果它们低于0或高于255，则它们分别设置为0或255。  如果scale！=0，则对值进行缩放，使得最小值为零，最大值为255。  设置gamma值允许指数缩放，启用gamma==1.0。  返回以（vCenter，hCenter）和半径（vRadius，hRadius）为中心的椭圆数据，共有number组数据，每组包含一个椭圆数据，第一个元素是vCenter，第二个元素是hCenter，第三个元素是vRadius，第四个元素是hRadius。  支持PGM和PBM图像。 |
| void ShapeEdgeDetection1(char\* input,char\* output,unsigned char CANNY\_THRESH4,int CANNY\_blur4) | 形状边缘检测，CANNY\_THRESH4=35，CANNY\_blur4=7。支持PNG图像。 |
| void ShapeEdgeDetection2(char\* input,char\* output,unsigned char CANNY\_THRESH4,int CANNY\_blur4) | 形状边缘检测，CANNY\_THRESH4=35，CANNY\_blur4=7。支持PNG图像。 |
| void ShapeEdgeDetection3(char\* input,char\* output,unsigned char CANNY\_THRESH,int CANNY\_BLUR) | 形状边缘检测，CANNY\_THRESH=50，CANNY\_BLUR=12。支持PNG图像。 |
| void ShapeEdgeDetection4(char\* input,char\* output,unsigned char CANNY\_THRESH,int CANNY\_BLUR) | 形状边缘检测，CANNY\_THRESH=50，CANNY\_BLUR=12。支持PNG图像。 |
| void ShapeEdgeDetection5(char\* input,char\* output,unsigned char CANNY\_THRESH2,int CANNY\_BLUR2) | 形状边缘检测，CANNY\_THRESH2=10，CANNY\_BLUR2=2。支持PNG图像。 |
| void ShapeEdgeDetection6(char\* input,char\* output,unsigned char CANNY\_THRESH2,int CANNY\_BLUR2) | 形状边缘检测，CANNY\_THRESH2=10，CANNY\_BLUR2=2。支持PNG图像。 |
| void ShapeEdgeDetection7(char\* input,char\* output,unsigned char CANNY\_THRESH3,int CANNY\_blur3) | 形状边缘检测，CANNY\_THRESH3=45，CANNY\_blur3=10。支持PNG图像。 |
| void ShapeEdgeDetection8(char\* input,char\* output,unsigned char CANNY\_THRESH3,int CANNY\_blur3) | 形状边缘检测，CANNY\_THRESH3=45，CANNY\_blur3=10。支持PNG图像。 |

**其他处理**

|  |  |
| --- | --- |
| void Encode(char\* input,char\* output) | 文本文件压缩，input是输入文件名，output是输出文件名。 |
| void Decode(char\* input,char\* output) | 文本文件压缩结果解压缩，input是输入文件名，output是输出文件名。 |
| void FileCompress(char \*input , char \*output) | 文件压缩，input是输入文件名，output是输出文件名。 |
| void FileDecompression(char \*input , char \*output) | 文件压缩结果解压缩，input是输入文件名，output是输出文件名。 |

**高级算子**

|  |  |
| --- | --- |
| void BlobAnalysis(char\* input,char\* output,int c1,int c2) | Blob分析，c1和c2是与颜色相关的参数，参考：c1=128，c2=127。支持BMP图像。 |
| void BlobAnalysis1(char\* input,char\* output,int c1,int c2) | Blob分析，c1和c2是与颜色相关的参数，参考：c1=128，c2=127。支持BMP图像。 |
| void VerificationCodeGeneration(char\* inputText,char\* output,int num,int foint,int a,int b1,int b2,int b3,int b4,int b5,int b6,int b7,int b8,int b9,int b10,int b11,int b12,int b13,double sigma,unsigned int noise\_type,int width,int height,int depth,int spectrum,bool shared) | 验证码生成。sigma=10,noise\_type=2,a=10,b1=128，b2=127，b3=2，b4=8，b5=12，b6=0，b7=1，b8=-100，b9=-100，b10=1，b11=3，b12=6，int b13=40，foint=30,num是字符数量,depth=1,spectrum=3,shared=0。 |
| void CornerDetection(char\* input,char\* output,float threshold, float k, float sigma,int width, int height, int channels) | 角点检测，threshold=10000，k=0.06，sigma=1.0，width=640，height=480，channels=1。支持PNM图像。 |
| vector<Keypoint> CornerDetection1(char\* input,char\* output,float threshold, float k, float sigma,int width, int height, int channels) | 角点检测，返回角点数据。threshold=10000，k=0.06，sigma=1.0，width=640，height=480，channels=1。支持PNM图像。  需引入以下结构体  typedef struct {  float x;  float y;  float score;  } Keypoint; |
| vector<Corner::Keypoint> CornerDetection(char\* input,int width,int height,int channels,float threshold, float k, float sigma) | 角点检测，返回角点数据。threshold=2000，k=1，sigma=1.2。支持PNM图像。  需引入以下命名空间：  namespace Corner {  struct Keypoint {  float x;  float y;  float score;  };  } |
| void Structure(char\* input,char\* output,float sigma) | 特征归一化统计，参考：sigma=2。支持多种图像格式。 |
| void Cornerness(char\* input,char\* output,float sigma,int method) | 角点检测，参考：sigma=2，method=0。支持多种图像格式。 |
| void Corners(char\* input,char\* output,float sigma, float thresh, int window, int nms, int corner\_method) | 角点检测，参考：sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0。支持多种图像格式。 |
| void FindMatch(char\* input1,char\* input2,char\* output,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner\_method,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2,float sigma5, float thresh5, int window5, int nms5, int corner\_method5,float sigma6, int corner\_method6, float thresh6, int window6, int nms6, float inlier\_thresh6, int iters6, int cutoff6, float acoeff6) | 特征匹配，参考：thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner\_method5=0，sigma6=2，corner\_method6=0，thresh6=0.3，window6=7，nms6=3，inlier\_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5。支持多种图像格式。 |
| vector<Descriptor> HarrisCorner(char\* input,char\* output,float sigma1, float thresh1, int window1, int nms1, int corner\_method1) | 角点检测，返回检测结果。参考：sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0。支持多种图像格式。  需引入以下结构体：  struct Point {  double x, y;    Point() : x(0), y(0) {}  Point(double x, double y) : x(x), y(y) {}  };  struct Descriptor {  Point p;  vector<float> data;    Descriptor(){}  Descriptor(const Point& p) : p(p) {}  }; |
| vector<Match> MatchDescriptors(char\* input1,char\* input2,char\* output,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2) | 描述匹配项，返回描述结果。参考：sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0。支持多种图像格式。  需引入以下结构体：  struct Point {  double x, y;    Point() : x(0), y(0) {}  Point(double x, double y) : x(x), y(y) {}  };  struct Descriptor {  Point p;  vector<float> data;    Descriptor(){}  Descriptor(const Point& p) : p(p) {}  };  struct Match {  const Descriptor\* a=nullptr;  const Descriptor\* b=nullptr;  float distance=0.f;    Match(){}  Match(const Descriptor\* a,const Descriptor\* b,float dist=0.f) : a(a), b(b), distance(dist) {}    bool operator<(const Match& other) { return distance<other.distance; }  }; |
| void DrawInliers(char\* input1,char\* input2,char\* output,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner\_method,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2,float sigma5, float thresh5, int window5, int nms5, int corner\_method5,float sigma6, int corner\_method6, float thresh6, int window6, int nms6, float inlier\_thresh6, int iters6, int cutoff6, float acoeff6) | 绘制角点，参考：thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner\_method5=0，sigma6=2，corner\_method6=0，thresh6=0.3，window6=7，nms6=3，inlier\_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5。支持多种图像格式。 |
| void PanoramaImage(char\* input1,char\* input2,char\* output,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner\_method,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2,float sigma5, float thresh5, int window5, int nms5, int corner\_method5,float sigma6, int corner\_method6, float thresh6, int window6, int nms6, float inlier\_thresh6, int iters6, int cutoff6, float acoeff6) | 制造全景图像，参考：thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner\_method5=0，sigma6=2，corner\_method6=0，thresh6=0.3，window6=7，nms6=3，inlier\_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5。支持多种图像格式。 |
| void Cylindrical(char\* input1,char\* input2,char\* output,float f1,float f2,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner\_method,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2,float sigma5, float thresh5, int window5, int nms5, int corner\_method5,float sigma6, int corner\_method6, float thresh6, int window6, int nms6, float inlier\_thresh6, int iters6, int cutoff6, float acoeff6,float sigma7, int corner\_method7, float thresh7, int window7, int nms7, float inlier\_thresh7, int iters7, int cutoff7, float acoeff7) | 角点检测。参考：f1=500，f2=500，thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner\_method5=0，sigma6=2，corner\_method6=0，thresh6=0.3，window6=7，nms6=3，inlier\_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5，sigma7=2，corner\_method7=0，thresh7=0.3，window7=7，nms7=3，inlier\_thresh7=5，iters7=1000，cutoff7=50，acoeff7=0.5。支持多种图像格式。 |
| void Spherical(char\* input1,char\* input2,char\* output,float f1,float f2,float thresh3, int k, int cutoff,float thresh4,float sigma, float thresh, int window, int nms, int corner\_method,float sigma1, float thresh1, int window1, int nms1, int corner\_method1,float sigma2, float thresh2, int window2, int nms2, int corner\_method2,float sigma5, float thresh5, int window5, int nms5, int corner\_method5,float sigma6, int corner\_method6, float thresh6, int window6, int nms6, float inlier\_thresh6, int iters6, int cutoff6, float acoeff6,float sigma7, int corner\_method7, float thresh7, int window7, int nms7, float inlier\_thresh7, int iters7, int cutoff7, float acoeff7) | 角点检测。参考：f1=500，f2=500，thresh3=5，k=10000，cutoff=50，thresh4=5，sigma=2，thresh=0.4，window=5，nms=3，corner\_method=0，sigma1=2，thresh1=0.4，window1=5，nms1=3，corner\_method1=0，sigma2=2，thresh2=0.4，window2=5，nms2=3，corner\_method2=0，sigma5=2，thresh5=0.4，window5=7，nms5=3，corner\_method5=0，sigma6=2，corner\_method6=0，thresh6=0.3，window6=7，nms6=3，inlier\_thresh6=5，iters6=1000，cutoff6=50，acoeff6=0.5，sigma7=2，corner\_method7=0，thresh7=0.3，window7=7，nms7=3，inlier\_thresh7=5，iters7=1000，cutoff7=50，acoeff7=0.5。支持多种图像格式。 |
| int\* FindLine(char\* input,char\* output,int width,int height) | 直线检测，返回直线的theta和rho，支持RAW图像。 |
| int\* FindCircle(char\* input,char\* output,int width,int height,float sigma, int tmin, int tmax) | 圆检测，返回圆心的坐标和圆的半径，支持RAW图像。sigma=1.4，tmin=70，tmax=150。 |
| void TemplateMatching(char\* input,char\* Template,char\* output,unsigned int b,double ps,double a,double a1,double b1,double c) | 模板匹配，a=0.5，b=0，a1=0.5，b1=0.5，c=0.2，ps是相似度，如ps=0.5。支持24位BMP图像。 |
| SearchResult TemplateMatching(char\* input,char\* Template) | 模板匹配，返回目标位置的左上角坐标和相似值。  支持PNG图像。  需引入以下结构体：  struct SearchResult {  int x, y;  double value;  }; |
| SearchResult TemplateMatching(uint8\_t\* input,uint8\_t\* Template,int imgWidth,int imgHeight,int imgBpp,int patWidth,int patHeight,int patBpp) | 模板匹配，返回目标位置的左上角坐标和相似值。支持PNG图像。  需引入以下头文件：  #define STB\_IMAGE\_IMPLEMENTATION  #include "stb\_image.h"  #include <cstdint>  #include <complex>  #include <vector>  参考例程：  int imgWidth, imgHeight, imgBpp;  int patWidth, patHeight, patBpp;  uint8\_t\* input = stbi\_load(inputfile, &imgWidth, &imgHeight, &imgBpp, 3);  uint8\_t\* Template = stbi\_load(Templatefile, &patWidth, &patHeight, &patBpp, 3); |
| void ImageCalibration1(char\* input,char\* output) | 图像标定。 |
| std::vector<Line> ImageCalibration2(char\* input, char\* output) | 图像标定。  需引入以下结构体：  struct Line {  double m, b;  int dist\_o; // distance to origin point when  // line(x = dist\_o) is perpendicular to x axis  int x0, x1, y0, y1; // two end points  int end\_point\_num;  Line(double \_m, double \_b, int \_dist\_o = 0, int \_x0 = 0, int \_y0 = 0,  int \_x1 = 0, int \_y1 = 0, int \_end\_point\_num = 0)  : m(\_m), b(\_b), dist\_o(\_dist\_o), x0(\_x0), x1(\_x1), y0(\_y0), y1(\_y1),  end\_point\_num(\_end\_point\_num) {}  }; |
| std::vector <CentersPoint> FindCircles(char\* input,char\* output,int size1,int size2,int size3) | 圆检测，返回检测结果。size1=5，size2=5，size3=7。支持BMP图像。  需引入以下结构体：  struct Point {  Point(int x = 0, int y = 0) { this->x = x; this->y = y; }  int x;  int y;  };  struct CentersPoint {  CentersPoint(Point point, int radius) { this->point = point; this->radius = radius; count = 1; }  Point point;  int count;  int radius;  }; |
| void ImgDisplay(char\* input,char\* WindowsName) | 图像显示，input是输入的图像文件名，WindowsName是要创建的窗口名称。 |
| vector<int> HoughTransform1(char\* input,char\* output, unsigned char color[3],int numOfLines,int fillGap,int minLength) | 霍夫直线检测，返回检测结果。如果共线线段之间的间隙小于fillGap，则连接它们，如果合并的行短于minLength，则放弃它，numOfLines是可能的线段数量，numOfLines=10，fillGap=5，minLength=60，color={ 0, 255, 0 }。支持8位BMP图像。 |
| vector<int> HoughTransform2(char\* input, char\* output, unsigned char color[3],int numOfLines,int fillGap,int minLength) | 霍夫直线检测，返回检测结果。如果共线线段之间的间隙小于fillGap，则连接它们，如果合并的行短于minLength，则放弃它，numOfLines是可能的线段数量，numOfLines=10，fillGap=5，minLength=60，color={ 0, 255, 0 }。支持8位BMP图像。 |
| void SobelGradient(char\* input,char\* output) | 用sobel算子分别求x方向,y方向的梯度。 |
| void SobelNoneMaxSpress(char\* input, char\* output) | 用sobel算子分别求x方向,y方向的梯度，再用梯度方向和非极大值抑制去掉冗余信息。 |
| void HytheresisThresholding(char\* input, char\* output, double high, double low) | Hythereis阈值处理，high=130，low=60。 |
| void HoughTransfer1(char\* input, char\* output, double high, double low, double threshold) | 霍夫变换，high=130，low=60，threshold=0.4。 |
| void HoughTransfer3(char\* input, char\* output, double high, double low, double threshold) | 霍夫变换，high=130，low=60，threshold=0.4。 |
| vector<int> HoughTransfer2(char\* input, double high, double low, double threshold) | 霍夫变换，返回角点信息，以两个元素为一组。high=130，low=60，threshold=0.4。 |
| void FeatureDetection1(char\* input,char\* output) | 特征检测。 |
| std::vector<sift::Keypoint> FeatureDetection2(char\* input, char\* output) | 特征检测。  需引入以下空间：  namespace sift  {  struct Keypoint {  //离散坐标系  int i;  int j;  int octave;  int scale; //倍频程内高斯图像的索引  //连续坐标（内插）  float x;  float y;  float sigma;  float extremum\_val; //插值DoG极值值  std::array<uint8\_t, 128> descriptor;  };  } |
| void FeatureMatching(char\* input1,char\* input2,char\* output) | 特征匹配。支持PGM图像。 |
| std::vector<std::pair<int, int>> FeatureMatching1(char\* input1,char\* input2,char\* output) | 特征匹配，返回匹配结果。 |
| std::vector<std::pair<int, int>> FeatureMatching2(char\* input1, char\* input2, char\* output) | 特征匹配，返回匹配结果。 |
| void Canny(char\* input,char\* output,int lowThreshold,int highThreshold) | Canny算子，至少支持JPG图像，input是输入文件名，output是输出文件名，参考：lowThreshold=50，highThreshold=150。 |
| void Canny(string input,string output) | Canny算子，参考：output="output"。支持BMP文件。 |
| void Canny(string input,char\* output,float sigma, float threshold) | Canny算子，参考：sigma=6.0，threshold=3.5。支持BMP文件。 |
| void Hough(char\* input, char\* output, float sigma, float threshold,double thre\_val,unsigned char\* color) | 霍夫变换，参考：sigma=6.0，threshold=3.5，thre\_val=0.5，color用于设置画出的标定点和线的颜色，如：color[3]={0,0,255}。支持BMP文件。 |
| void DES\_Encrypt(char \*PlainFile, char \*Key,char \*CipherFile) | DES加密函数，支持多种文件。PlainFile是原文件的文件名，Key是密钥字符，CipherFile是加密后的文件名。 |
| void DES\_Decrypt(char \*CipherFile, char \*Key,char \*PlainFile) | DES解密函数，支持多种文件。CipherFile是已加密文件的文件名，Key是密钥字符，PlainFile是解密后的文件名。 |
| int\* TemplateMatch(char\* input,char\* Template,char\* output,int channels,int ROTATION) | 模板匹配，input是母本图像，Template 是样本图像，output是结果图像文件名。channels是图像的像素通道数，返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的X轴坐标值，第三个元素是目标的Y轴坐标值。至少支持PNG图像。参考：channels=3，ROTATION=360或ROTATION=1。 |
| int\* TemplateMatch(image input,image Template,char\* output,int channels,int ROTATION) | 模板匹配，input是母本图像，Template 是样本图像，output是结果图像文件名。channels是图像的像素通道数，返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的X轴坐标值，第三个元素是目标的Y轴坐标值。至少支持PNG图像。参考：channels=3，ROTATION=360或ROTATION=1。  需引入以下头文件和结构体：  #include "stb\_image.h"  #include "stb\_image\_write.h"  typedef struct imageContainer {  int x,y,n;  unsigned char \*data;  } image;  参考：  image input, Template;  input.data = stbi\_load(inputFile, &input.x, &input.y, &input.n, 3); //3表示图像有3个像素通道  Template.data = stbi\_load(templateFile, &Template.x, &Template.y, &Template.n, 3); |
| int\* TemplateMatch(char\* input,char\* Template,int channels,int ROTATION) | 模板匹配，input是母本图像，Template 是样本图像。channels是图像的像素通道数，返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的X轴坐标值，第三个元素是目标的Y轴坐标值。至少支持PNG图像。参考：channels=3，ROTATION=360或ROTATION=1。 |
| int\* TemplateMatch(image input,image Template,int ROTATION) | 模板匹配，input是母本图像，Template 是样本图像。返回值数组中的第一个元素是最大匹配分数，第二个元素是目标的X轴坐标值，第三个元素是目标的Y轴坐标值。至少支持PNG图像。参考： ROTATION=360或ROTATION=1。  需引入以下头文件和结构体：  #include "stb\_image.h"  #include "stb\_image\_write.h"  typedef struct imageContainer {  int x,y,n;  unsigned char \*data;  } image;  参考：  image input, Template;  input.data = stbi\_load(inputFile, &input.x, &input.y, &input.n, 3); //3表示图像有3个像素通道  Template.data = stbi\_load(templateFile, &Template.x, &Template.y, &Template.n, 3); |
| void KMeans(string input,unsigned int Clusters,char\* output) | K-Means聚类，input是输入文件名，Clusters是聚类的种类数目，output是输出文件名。支持BMP文件。 |
| void PGMSobel(char\* input,char\* output,int Mx[3][3],int My[3][3],int max,int min) | Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM文件。  参考模板：  int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}}  int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}}  int max = -9999  int min = 9999 |
| void PGMSobelX(char\* input,char\* output,int Mx[3][3],int My[3][3],int max,int min) | X方向滤波，input是输入文件名，output是输出文件名。支持P5格式的PGM文件。  参考模板：  int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}}  int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}}  int max = -9999  int min = 9999 |
| void PGMSobelY(char\* input,char\* output,int Mx[3][3],int My[3][3],int max,int min) | Y方向滤波，input是输入文件名，output是输出文件名。支持P5格式的PGM文件。  参考模板：  int Mx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}}  int My[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}}  int max = -9999  int min = 9999 |
| void PGMSobel1(char\* input,char\* output,int min,int max,int mx[3][3],int my[3][3]) | Sobel算子，input是输入文件名，output是输出文件名。min和max是图像归一化相关参数，如min = 1000000, max = 0；mx和my分别是Sobel算子的X和Y方向模板。支持P2和P5格式的PGM。  参考模板 ：  int mx[3][3] = {  {-1, 0, 1},  {-2, 0, 2},  {-1, 0, 1}  };  int my[3][3] = {  {-1, -2, -1},  {0, 0, 0},  {1, 2, 1}  }; |
| void PGMSobelX1(char\* input,char\* output,int min,int max,int mx[3][3],int my[3][3]) | X方向梯度，input是输入文件名，output是输出文件名。min和max是图像归一化相关参数，如min = 1000000, max = 0；mx和my分别是Sobel算子的X和Y方向模板。支持P2和P5格式的PGM。  参考模板 ：  int mx[3][3] = {  {-1, 0, 1},  {-2, 0, 2},  {-1, 0, 1}  };  int my[3][3] = {  {-1, -2, -1},  {0, 0, 0},  {1, 2, 1}  }; |
| void PGMSobelY1(char\* input,char\* output,int min,int max,int mx[3][3],int my[3][3]) | Y方向梯度，input是输入文件名，output是输出文件名。min和max是图像归一化相关参数，如min = 1000000, max = 0；mx和my分别是Sobel算子的X和Y方向模板。支持P2和P5格式的PGM。  参考模板 ：  int mx[3][3] = {  {-1, 0, 1},  {-2, 0, 2},  {-1, 0, 1}  };  int my[3][3] = {  {-1, -2, -1},  {0, 0, 0},  {1, 2, 1}  }; |
| void PGMSobel2(char\* input,char\* XOutput,char\* YOutput,char\* SobelOutput,int sobel\_x[3][3],int sobel\_y[3][3],int min,int max) | Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。XOutput是输出的X方向的梯度图像，YOutput是输出的Y方向的梯度图像，SobelOutput是输出的整幅图像的Sobel算子计算结果，min和max是图像归一化的相关参数，如min=100,max=0。  参考模板：  int sobel\_x[3][3]={{-1, 0, 1},{-2, 0, 2},{-1, 0, 1}};  int sobel\_y[3][3]={{1, 2, 1}, {0, 0, 0},{-1,-2,-1}}; |
| void Sobel(char\* input,char\* output) | Sobel算子，input是输入文件名，output是输出文件名。支持PGM文件。 |
| void Laplatian(char\* input,char\* output) | Laplatian算子，input是输入文件名，output是输出文件名。支持PGM文件。 |
| void HorizSobel(char\* input,char\* output) | 水平Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void VertSobel(char\* input,char\* output) | 垂直Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。 |
| void PGMSobel1(char\* input,char\* output,int threshold) | Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。threshold是目标阈值，如threshold=80。 |
| void RAWSobelEdge(char\* input,char\* output,int ROWS,int COLS,int M,float sobelX[3][3],float sobelY[3][3]) | Sobel算子，input是输入文件名，output是输出文件名。ROWS是图像的行，COLS是图像的列，M是滤波相关参数，如M=1。支持RAW图像。  参考模板：  float sobelX[3][3] = {{-1,0,1},  {-2,0,2},  {-1,0,1}};  float sobelY[3][3] = {{-1,-2,-1},  {0,0,0},  {1,2,1}}; |
| void RAWPlaceHolder(char\* input,char\* output,int ROWS,int COLS,int M,float mask[3][3]) | 边缘检测，input是输入文件名，output是输出文件名。ROWS是图像的行，COLS是图像的列，M是滤波相关参数，如M=1。支持RAW图像。  参考模板：  float mask[3][3] = {{-1,-2,-1}, {0,0,0},{1,2,1}}; |
| void LaplacianEnhancement(char\* input,char\* output,int N,int LaplMask[3][3]) | 拉普拉斯图像增强，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int LaplMask[3][3] = {  0,1,0,  1, -4, 1,  0, 1, 0  }; |
| void LaplaceSmooth(char\* input,char\* output,int N,int LaplMask[3][3]) | 拉普拉斯平滑，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int LaplMask[3][3] = {  0,1,0,  1, -4, 1,  0, 1, 0  }; |
| void Sobel1(char\* input,char\* output,int N,int SblMask1[3][3],int SblMask2[3][3]) | Sobel算子，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int SblMask1[3][3] = {  -1,-2,-1,  0, 0, 0,  1, 2, 1  };  int SblMask2[3][3] = {  -1, 0, 1,  -2, 0, 2,  -1, 0, 1  }; |
| void SobelSmooth(char\* input,char\* output,int N,int SblMask1[3][3],int SblMask2[3][3]) | Sobel平滑，input是输入文件名，output是输出文件名。如：N=1。支持8位BMP图像。  参考模板：  int SblMask1[3][3] = {  -1,-2,-1,  0, 0, 0,  1, 2, 1  };  int SblMask2[3][3] = {  -1, 0, 1,  -2, 0, 2,  -1, 0, 1  }; |
| void Roberts(unsigned char\*\* input,unsigned char\*\* output) | Roberts算子，input是输入数据，output是输出数据。 |
| void Roberts(BMPMat\*\* input,BMPMat\*\* output) | Roberts算子，input是输入数据，output是输出数据。 |
| void SobelOperator(char\* input,char\* output) | Sobel算子，耗时较长，input是输入文件名，output是输出文件名。支持24位BMP图像。 |
| SobelImage\*\* SobelOperator(char\* input) | 返回处理后的各像素点的坐标和对应的像素值，若是边缘点则对应白色，否则对应黑色。支持24位BMP图像。  需引入以下结构体：  typedef struct {  int x;  int y;  unsigned char red;  unsigned char green;  unsigned char blue;  }SobelImage; |
| void STLSection(char\* input,char\* output,int sliceAmount, int resolution,int c) | STL切片，input是输入的STL文件，output是输出的切片文件前缀名，sliceAmount是切片量，如：sliceAmount=50，resolution是分辨率，如：resolution=260，c是执行的相关参数，如：c=5。 |
| void SURF(char\* input1,char\* input2,char\* output) | SURF算子，input1和input2是输入文件名，output是输出文件名，支持BMP图像。 |
| void EdgeDetection(char\* input,char\* output) | 边缘检测，input是输入文件名，output是输出文件名。支持4位BMP图像。 |
| void EdgeDetection1(char\* input,char\* output,short sharpen[3][3]) | 边缘检测，input是输入文件名，output是输出文件名。支持8位BMP图像。  参考模板：  short sharpen[3][3] = {{1, 1, 1},  {1, -8, 1},  {1, 1, 1}}; |
| void EdgeDetection2(char\* input,char\* output,int a) | 边缘检测，input是输入文件名，output是输出文件名。a是用于设置图像像素的相关参数，如a=3。支持24位BMP图像。 |
| void EdgeDetection3(char\* input,char\* output,int a) | 边缘检测，input是输入文件名，output是输出文件名。a是用于设置图像像素的相关参数，如a=3。支持24位BMP图像。 |
| void EdgeDetection4(char\* input,char\* output,int a) | 边缘检测，input是输入文件名，output是输出文件名。a是用于设置图像像素的相关参数，如a=3。支持24位BMP图像。 |
| void YFiltering(char\* input,char\* output,int sobel\_x[3][3],int sobel\_y[3][3]) | Y方向滤波，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。  参考模板：  int sobel\_x[3][3] = { { 1, 0, -1},  { 2, 0, -2},  { 1, 0, -1}};  int sobel\_y[3][3] = { { 1, 2, 1},  { 0, 0, 0},  {-1, -2, -1}}; |
| void XFiltering(char\* input,char\* output,int sobel\_x[3][3],int sobel\_y[3][3]) | X方向滤波，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。  参考模板：  int sobel\_x[3][3] = { { 1, 0, -1},  { 2, 0, -2},  { 1, 0, -1}};  int sobel\_y[3][3] = { { 1, 2, 1},  { 0, 0, 0},  {-1, -2, -1}}; |
| void SobelFiltering(char\* input,char\* output,int sobel\_x[3][3],int sobel\_y[3][3]) | Sobel算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。  参考模板：  int sobel\_x[3][3] = { { 1, 0, -1},  { 2, 0, -2},  { 1, 0, -1}};  int sobel\_y[3][3] = { { 1, 2, 1},  { 0, 0, 0},  {-1, -2, -1}}; |
| void PrewittFiltering(char\* input,char\* output,int prewitt\_x[3][3],int prewitt\_y[3][3]) | Prewitt算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。  参考模板：  int prewitt\_x[3][3] = { { 5, 5, 5},  { -3, 0, -3},  { -3, -3, -3}};  int prewitt\_y[3][3] = { { 5, -3, -3},  { 5, 0, -3},  {5, -3, -3}}; |
| void LaplacianFiltering(char\* input,char\* output,int laplacian[3][3]) | Laplace算子，input是输入文件名，output是输出文件名。支持P5格式的PGM图像。laplacian是Laplacian算子模板。  参考模板：  int laplacian[3][3] = { { 1, 1, 1},  { 1, -8, 1},  { 1, 1, 1}}; |
| void SobelOperation1(char\* input,char\* output,int width,int height) | Sobel算子，支持BMP图像。 |
| void SobelOperation2(char\* input,char\* output,int width,int height) | Sobel算子，支持BMP图像。 |
| void Roberts(char\* input,char\* output) | Roberts边缘检测，支持BMP图像。 |
| void Prewitt(char\* input,char\* output) | Prewitt边缘检测，支持BMP图像。 |
| void Sobel(char\* input,char\* output) | Sobel边缘检测，支持BMP图像。 |
| void Laplace(char\* input,char\* output) | Laplace边缘检测，支持BMP图像。 |
| void BoxBlurAdvanced(string input,string output,int radius) | 高级方框模糊，参考：radius=5。支持PNG文件。 |
| void HoughTransform(char\* input,char\* output,unsigned char threshold) | 霍夫变换，input是输入的RAW文件，output是输出的RAS文件，threshold=100。 |
| static void EdgeDetectionWithoutNonmaximum(const LPCTSTR input, const LPCTSTR output,double a,double b,double c) | 边缘检测，参考：a=0.33，b=0.33，c=0.33。支持24位BMP图像。 |
| static void CannyEdgeDetection(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb) | 边缘检测，参考：orank=20，oranb=80。支持24位BMP图像。 |
| static void HoughTransform(const LPCTSTR input, const LPCTSTR output,double a,double b,double c,int orank,int oranb) | 霍夫变换，参考：a=0.33，b=0.33，c=0.33，orank=20，oranb=80。支持24位BMP图像。 |
| void BoxBlurBasic(string input,string output) | 基础方框模糊，支持PNG文件。 |
| void SobelSharpen(char\* input,char\* output,int Templatex[3][3],int Templatey[3][3],int coefficient1,int coefficient2) | Sobel算子，input是输入文件名，output是输出文件名。Templatex是laplace锐化模板,4邻域，Templatey是laplace锐化模板,8邻域，coefficient1 = 9，coefficient2 = 16。支持24位BMP图像。 |
| void EncryptionDecryption(char\* input,char\* output,int Key,int a) | 图像加解密，Key是密钥，a=1时执行加密，a=0时执行解密。支持24位BMP图像。 |
| void Encryption(char\* input,char\* output,int Key) | 图像加密，input是输入文件名，output是输出文件名。Key是密钥。支持24位BMP图像。 |
| void Decryption(char\* input,char\* output,int Key) | 图像解密，input是输入文件名，output是输出文件名。Key是密钥。支持24位BMP图像。 |
| void Nesting(char\* Biginput,char\* Smallinput,char\* output) | 图像嵌套，Biginput是输入的大图，Smallinput是输入的小图。支持24位BMP图像。 |
| void Blend(char\* input1,char\* input2,char\* output) | 图像融合之混合化，input1和input2是输入的两个要融合的图像，output是输出文件名。支持24位BMP图像。 |
| void Checker(char\* input1,char\* input2,char\* output) | 图像融合之棋盘化，input1和input2是输入的两个要融合的图像，output是输出文件名。支持24位BMP图像。 |
| void Blend1(char\* input1,char\* input2,char\* output) | 图像融合之混合化，input1和input2是输入的两个要融合的图像，output是输出文件名。支持24位BMP图像。 |
| void Checker1(char\* input1,char\* input2,char\* output) | 图像融合之棋盘化，input1和input2是输入的两个要融合的图像，output是输出文件名。支持24位BMP图像。 |
| void QRCodeGeneration(char \*filename, char\* inputString) | 二维码生成，filename是生成的二维码图像文件名，inputString是二维码包含的信息。支持BMP图像。 |
| vector<float> HarrisCornerDetection(char\* input,int width,int height,int channels,int step,float threshold,float k,float sigma) | 角点检测，支持P5和P6格式的PPM文件，从返回值数组中的第一个元素开始，返回值以3个元素为一组，分别是角点的X坐标、Y坐标和分数，若返回值数组名为A，则{A[0],A[1],A[2]}是第一个角点的数据，{A[3],A[4],A[5]}是第二个角点的数据，以此类推。  input是输入的图像文件名，width和height是输入图像的宽和高，channels是输入图像的通道数，step默认=-1，threshold是Harris检测中角点的得分阈值，k是Harris评分函数中的k值，sigma是用于IxIy阵列平滑的sigma值，参考：threshold=2000，k=1，sigma=1.2。 |
| vector<float> HarrisCorner(char\* input,char\* output,int width,int height,int channels,float threshold,float k,float sigma) | 角点检测，支持P5和P6格式的PPM文件，从返回值数组中的第一个元素开始，返回值以3个元素为一组，分别是角点的X坐标、Y坐标和分数，若返回值数组名为A，则{A[0],A[1],A[2]}是第一个角点的数据，{A[3],A[4],A[5]}是第二个角点的数据，以此类推。  input是输入的图像文件名，width和height是输入图像的宽和高，channels是输入图像的通道数，threshold是Harris检测中角点的得分阈值，k是Harris评分函数中的k值，sigma是用于IxIy阵列平滑的sigma值，参考：threshold=2000，k=1，sigma=1.2。 |
| int\* TemplateMatching(char\* input1,char\* input2,char\* output,unsigned char red,unsigned char green,unsigned char blue,double MatchScore) | 模板匹配，返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标，第四个元素是匹配框的右上角的横坐标。input1是搜索图像，input2是模板图像，output是匹配结果图像，MatchScore=0.9。支持BMP图像。 |
| int\* TemplateMatching(char\* input1,char\* input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore) | 模板匹配，返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标，第四个元素是匹配框的右上角的横坐标。input1是搜索图像，input2是模板图像，output是匹配结果图像，MatchScore=0.9。支持BMP图像。 |
| struct imagine TemplateMatching(struct imagine ColorSource,struct imagine input1,struct imagine input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore) | 模板匹配，返回匹配结果的图像，input1是搜索图像，input2是模板图像，MatchScore=0.9。支持BMP图像。  需引入以下结构体：  typedef struct imagine {  unsigned char \*R,\*G,\*B,\*header;  int W, H, Wpad, size;  };  声明：  void grayscale\_image(char\* nume\_fisier\_sursa,char\* nume\_fisier\_destinatie);  struct imagine salvareBitmap (char \*destinatieFisier);  void afisare (struct imagine img, char \*destinatieSalvare);  将灰度应用于所有图像：  struct imagine output;  grayscale\_image(inputImage1, input1\_grayscale);  grayscale\_image(inputImage2, input2\_grayscale);  ColorSource=salvareBitmap(inputImage1); //inputImage1是彩色原图  input1=salvareBitmap(input1\_grayscale);  input2=salvareBitmap(input2\_grayscale);  output=TemplateMatching(ColorSource,input1,input2,red,green,blue,MatchScore);  afisare(output,outputfile); |
| int\* TemplateMatching(struct imagine input1,struct imagine input2,unsigned char red,unsigned char green,unsigned char blue,double MatchScore) | 模板匹配，返回值中第一个元素是匹配框左上角的纵坐标，第二个元素是匹配框左下角的纵坐标，第三个元素是匹配框的左上角的横坐标，第四个元素是匹配框的右上角的横坐标。input1是搜索图像，input2是模板图像，output是匹配结果图像，MatchScore=0.9。支持BMP图像。  需引入以下结构体：  typedef struct imagine {  unsigned char \*R,\*G,\*B,\*header;  int W, H, Wpad, size;  };  声明：  void grayscale\_image(char\* nume\_fisier\_sursa,char\* nume\_fisier\_destinatie);  struct imagine salvareBitmap (char \*destinatieFisier);  将灰度应用于所有图像：  struct imagine input1,input2;  grayscale\_image(inputImage1, input1\_grayscale);  grayscale\_image(inputImage2, input2\_grayscale);  input1=salvareBitmap(input1\_grayscale);  input2=salvareBitmap(input2\_grayscale); |
| void TemplateMatching(char\* input,char\* templatename,char\* output,unsigned int MaximumMatchingQuantity,double MatchScore,float suprapunereMaxima,unsigned char red,unsigned char green,unsigned char blue) | 模板匹配，suprapunereMaxima表示最大重叠率，参考：MaximumMatchingQuantity=10，MatchScore=0.8，suprapunereMaxima=0.2。支持BMP图像。 |
| int\* TemplateMatching(char\* input1,char\* input2,char\* output) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。支持BMP图像。 |
| int\* TemplateMatching1(char\* input1,char\* input2) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。支持BMP图像。 |
| int\* TemplateMatching2(char\* input1,char\* input2) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。支持BMP图像。 |
| my\_image\_comp\* TemplateMatching(my\_image\_comp input,my\_image\_comp Template,int H,int length,float\* hpf) | 模板匹配，返回匹配结果。支持BMP图像。  需引入以下结构体：  struct my\_image\_comp {  int width;  int height;  int stride;  int border;  float \*handle;  float \*buf;  my\_image\_comp()  { width = height = stride = border = 0; handle = buf = NULL; }  ~my\_image\_comp()  { if (handle != NULL) delete[] handle; }  void init(int height, int width, int border)  {  this->width = width; this->height = height; this->border = border;  stride = width + 2\*border;  if (handle != NULL)  delete[] handle;  handle = new float[stride\*(height+2\*border)];  buf = handle + (border\*stride) + border;  }  void perform\_boundary\_extension();  };  struct filt {  float\* centre;  int length;  };  声明：  int read\_bmp(char\* image, my\_image\_comp\* input\_comps, int\* num\_comps, int H);  int write\_bmp(my\_image\_comp\* output\_comps, char\* dest);  filt make\_filter(int type);  参考：  my\_image\_comp input;  my\_image\_comp Template;  filt filter = make\_filter(1);  int length = filter.length;  float\* hpf = filter.centre;  int H = (filter.length - 1) / 2;  int num\_comps = 1;  read\_bmp(inputfile, &input, &num\_comps, 0);  read\_bmp(Templatefile, &Template, &num\_comps, H);  write\_bmp(&input,outputfile); |
| int\* TemplateMatching1(my\_image\_comp input,my\_image\_comp Template,int H,int length,float\* hpf) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。支持BMP图像。  需引入以下结构体：  struct my\_image\_comp {  int width;  int height;  int stride;  int border;  float \*handle;  float \*buf;  my\_image\_comp()  { width = height = stride = border = 0; handle = buf = NULL; }  ~my\_image\_comp()  { if (handle != NULL) delete[] handle; }  void init(int height, int width, int border)  {  this->width = width; this->height = height; this->border = border;  stride = width + 2\*border;  if (handle != NULL)  delete[] handle;  handle = new float[stride\*(height+2\*border)];  buf = handle + (border\*stride) + border;  }  void perform\_boundary\_extension();  };  struct filt {  float\* centre;  int length;  };  声明：  int read\_bmp(char\* image, my\_image\_comp\* input\_comps, int\* num\_comps, int H);  int write\_bmp(my\_image\_comp\* output\_comps, char\* dest);  filt make\_filter(int type);  参考：  my\_image\_comp input;  my\_image\_comp Template;  filt filter = make\_filter(1);  int length = filter.length;  float\* hpf = filter.centre;  int H = (filter.length - 1) / 2;  int num\_comps = 1;  read\_bmp(inputfile, &input, &num\_comps, 0);  read\_bmp(Templatefile, &Template, &num\_comps, H);  write\_bmp(&input,outputfile); |
| int\* TemplateMatching2(my\_image\_comp input,my\_image\_comp Template,int H,int length,float\* hpf) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。支持BMP图像。  需引入以下结构体：  struct my\_image\_comp {  int width;  int height;  int stride;  int border;  float \*handle;  float \*buf;  my\_image\_comp()  { width = height = stride = border = 0; handle = buf = NULL; }  ~my\_image\_comp()  { if (handle != NULL) delete[] handle; }  void init(int height, int width, int border)  {  this->width = width; this->height = height; this->border = border;  stride = width + 2\*border;  if (handle != NULL)  delete[] handle;  handle = new float[stride\*(height+2\*border)];  buf = handle + (border\*stride) + border;  }  void perform\_boundary\_extension();  };  struct filt {  float\* centre;  int length;  };  声明：  int read\_bmp(char\* image, my\_image\_comp\* input\_comps, int\* num\_comps, int H);  int write\_bmp(my\_image\_comp\* output\_comps, char\* dest);  filt make\_filter(int type);  参考：  my\_image\_comp input;  my\_image\_comp Template;  filt filter = make\_filter(1);  int length = filter.length;  float\* hpf = filter.centre;  int H = (filter.length - 1) / 2;  int num\_comps = 1;  read\_bmp(inputfile, &input, &num\_comps, 0);  read\_bmp(Templatefile, &Template, &num\_comps, H);  write\_bmp(&input,outputfile); |
| int\* TemplateMatching(char\* input1,char\* input2,char\* output,float min) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。min是与匹配分数相关的参数，支持BMP图像。参考：min=65026。 |
| int\* TemplateMatching(bmp\_in input1,bmp\_in input2,float min) | 模板匹配，返回值是匹配框的左上角坐标（x，y）。min是与匹配分数相关的参数，支持BMP图像。参考：min=65026。  需引入以下结构体：  struct bmp\_in {  int num\_components, rows, cols;  int num\_unread\_rows;  int line\_bytes;  int alignment\_bytes;  FILE \*in;  };  声明：  extern int bmp\_in\_\_open(bmp\_in \*state, const char \*fname);  参考：  bmp\_in input1,input2;  bmp\_in\_\_open(&input1,input1file);  bmp\_in\_\_open(&input2,input2file); |
| double\* TemplateMatch(byte\*\*\* input,byte\*\*\* Template,char\* output,int irows,int icols,int trows,int tcols,int size,int best\_loss,double a,double b,double c,double d,int e1,int e2) | 模板匹配，支持JPG图像，返回值的第1和第2个元素是匹配框左上角顶点的横坐标和纵坐标，第3个元素是目标与模板相对的旋转角度，第4个元素是缩放比例。参考：size=1，best\_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。  声明：  #define byte unsigned char  byte \*\*\*LoadRgb(const char \*fname, int \*rows, int \*cols, int \*chan);  参考：  int irows, icols, ichan;  int trows, tcols, tchan;  byte\*\*\* input = LoadRgb(inputfile, &irows, &icols, &ichan);  byte\*\*\* Template = LoadRgb(templatefile, &trows, &tcols, &tchan); |
| double\* TemplateMatch(byte\*\*\* input,byte\*\*\* Template,int irows,int icols,int trows,int tcols,int size,int best\_loss,double a,double b,double c,double d,int e1,int e2) | 模板匹配，支持JPG图像，返回值的第1和第2个元素是匹配框左上角顶点的横坐标和纵坐标，第3个元素是目标与模板相对的旋转角度，第4个元素是缩放比例。参考：size=1，best\_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。  声明：  #define byte unsigned char  byte \*\*\*LoadRgb(const char \*fname, int \*rows, int \*cols, int \*chan);  参考：  int irows, icols, ichan;  int trows, tcols, tchan;  byte\*\*\* input = LoadRgb(inputfile, &irows, &icols, &ichan);  byte\*\*\* Template = LoadRgb(templatefile, &trows, &tcols, &tchan); |
| byte\*\*\* TemplateMatch1(byte\*\*\* input,byte\*\*\* Template,int irows,int icols,int trows,int tcols,int size,int best\_loss,double a,double b,double c,double d,int e1,int e2) | 模板匹配，支持JPG图像，返回匹配结果。参考：size=1，best\_loss=1000000000，a=0.5，b=2.1，c=0.5，d=45，e1=20，e2=20。  声明：  #define byte unsigned char  byte \*\*\*LoadRgb(const char \*fname, int \*rows, int \*cols, int \*chan);  void SaveRgbPng(byte \*\*\*in, const char \*fname, int rows, int cols);  参考：  int irows, icols, ichan;  int trows, tcols, tchan;  byte\*\*\* input = LoadRgb(inputfile, &irows, &icols, &ichan);  byte\*\*\* Template = LoadRgb(templatefile, &trows, &tcols, &tchan); |
| int ObjectFind(bmpread\_t input,bmpread\_t Template) | 模板匹配，返回匹配到的目标数量。支持BMP图像。  需引入以下头文件：  #include "bmpread1.h"  参考：  bmpread\_t input, Template;  bmpread(inputfile, BMPREAD\_BYTE\_ALIGN | BMPREAD\_ANY\_SIZE, &input);  bmpread(Templatefile, BMPREAD\_BYTE\_ALIGN | BMPREAD\_ANY\_SIZE, &Template); |
| double\* TemplateMatching1(Image2\* input, Image2\* Template,char\* output,char\* output\_txt,double threshold,int isWriteImageResult,unsigned char color,unsigned char red,unsigned char green,unsigned char blue) | 模板匹配，返回值数组：匹配框左上角顶点的X和Y坐标、模板的宽和高、差异度。参考：output是匹配结果图像名，output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持PPM文件。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| double\* TemplateMatching1(Image2\* input, Image2\* Template,char\* output\_txt,double threshold,unsigned char color,unsigned char red,unsigned char green,unsigned char blue) | 模板匹配，返回值数组：匹配框左上角顶点的X和Y坐标、模板的宽和高、差异度。参考：output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持PPM文件。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| double\* TemplateMatching2(Image2\* input, Image2\* Template,char\* output,char\* output\_txt,double threshold,int isWriteImageResult,unsigned char color,unsigned char red,unsigned char green,unsigned char blue) | 模板匹配，返回值数组：匹配框左上角顶点的X和Y坐标、模板的宽和高、差异度。参考：output是匹配结果图像名，output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持PPM文件。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| double\* TemplateMatching2(Image2\* input, Image2\* Template,char\* output\_txt,double threshold,unsigned char color,unsigned char red,unsigned char green,unsigned char blue) | 模板匹配，返回值数组：匹配框左上角顶点的X和Y坐标、模板的宽和高、差异度。参考：output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。支持PPM文件。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| Image2\* TemplateMatching3(Image2\* input, Image2\* Template, char\* output\_txt, double threshold, int isWriteImageResult, unsigned char color, unsigned char red, unsigned char green, unsigned char blue) | 模板匹配，返回匹配结果的图像数据。参考：output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| Image2\* TemplateMatching4(Image2\* input, Image2\* Template, char\* output\_txt, double threshold, int isWriteImageResult, unsigned char color, unsigned char red, unsigned char green, unsigned char blue) | 图像匹配，返回匹配结果的图像数据。参考： output\_txt是保存的匹配相关数据的文本文件，threshold=0.5，isWriteImageResult=1，color是当图像是灰度图时的匹配框颜色，red、green和blue是当图像是彩色图时的匹配框颜色的红绿蓝通道值。  需要引入以下结构体：  typedef struct Image2  {  int width;  int height;  int channel;  unsigned char\* data;  }Image2;  声明：  Image2\* readPXM(const char\* name);  参考：  Image2\* input = readPXM(inputFileName);  Image2\* Template = readPXM(templatename); |
| double\* TemplateMatching(RGB\_PACKED\_IMAGE\* input, RGB\_PACKED\_IMAGE\* Template,char\* output,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold) | 模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持PPM文件。  声明：  #ifndef \_\_P  #if defined(\_\_STDC\_\_) || defined(\_\_cplusplus)  #define \_\_P(protos) protos  #else  #define \_\_P(protos) ()  #endif  #endif  RGB\_PACKED\_IMAGE \*readRGBPackedImage \_\_P(( char\* ));  需引入以下结构体：  typedef struct rgb\_packed\_pixel {  BYTE r;  BYTE g;  BYTE b;  } RGB\_PACKED\_PIXEL;  typedef struct rgb\_packed\_image {  int cols;  int rows;  RGB\_PACKED\_PIXEL \*\*p;  RGB\_PACKED\_PIXEL \*data\_p;  } RGB\_PACKED\_IMAGE;  参考：  RGB\_PACKED\_IMAGE\* Template = readRGBPackedImage(templatename);  RGB\_PACKED\_IMAGE\* input = readRGBPackedImage(inputFileName); |
| double\* TemplateMatching1(RGB\_PACKED\_IMAGE\* input, RGB\_PACKED\_IMAGE\* Template,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold) | 模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持PPM文件。  声明：  #ifndef \_\_P  #if defined(\_\_STDC\_\_) || defined(\_\_cplusplus)  #define \_\_P(protos) protos  #else  #define \_\_P(protos) ()  #endif  #endif  RGB\_PACKED\_IMAGE \*readRGBPackedImage \_\_P(( char\* ));  需引入以下结构体：  typedef struct rgb\_packed\_pixel {  BYTE r;  BYTE g;  BYTE b;  } RGB\_PACKED\_PIXEL;  typedef struct rgb\_packed\_image {  int cols;  int rows;  RGB\_PACKED\_PIXEL \*\*p;  RGB\_PACKED\_PIXEL \*data\_p;  } RGB\_PACKED\_IMAGE;  参考：  RGB\_PACKED\_IMAGE\* Template = readRGBPackedImage(templatename);  RGB\_PACKED\_IMAGE\* input = readRGBPackedImage(inputFileName); |
| double\* TemplateMatching2(RGB\_PACKED\_IMAGE\* input, RGB\_PACKED\_IMAGE\* Template,unsigned char red,unsigned char green,unsigned char blue,double c,double threshold) | 模板匹配，返回匹配框的中心点坐标、旋转角度和缩放比例。参考：c=0.5，threshold=0.9。支持PPM文件。  声明：  #ifndef \_\_P  #if defined(\_\_STDC\_\_) || defined(\_\_cplusplus)  #define \_\_P(protos) protos  #else  #define \_\_P(protos) ()  #endif  #endif  RGB\_PACKED\_IMAGE \*readRGBPackedImage \_\_P(( char\* ));  需引入以下结构体：  typedef struct rgb\_packed\_pixel {  BYTE r;  BYTE g;  BYTE b;  } RGB\_PACKED\_PIXEL;  typedef struct rgb\_packed\_image {  int cols;  int rows;  RGB\_PACKED\_PIXEL \*\*p;  RGB\_PACKED\_PIXEL \*data\_p;  } RGB\_PACKED\_IMAGE;  参考：  RGB\_PACKED\_IMAGE\* Template = readRGBPackedImage(templatename);  RGB\_PACKED\_IMAGE\* input = readRGBPackedImage(inputFileName); |
| RGB\_PACKED\_IMAGE\* TemplateMatching(RGB\_PACKED\_IMAGE\* input, RGB\_PACKED\_IMAGE\* Template,unsigned char red, unsigned char green, unsigned char blue, double c, double threshold) | 模板匹配，返回匹配结果的图像数据。参考：c=0.5，threshold=0.9。支持PPM文件。  声明：  #ifndef \_\_P  #if defined(\_\_STDC\_\_) || defined(\_\_cplusplus)  #define \_\_P(protos) protos  #else  #define \_\_P(protos) ()  #endif  #endif  RGB\_PACKED\_IMAGE \*readRGBPackedImage \_\_P(( char\* ));  需引入以下结构体：  typedef struct rgb\_packed\_pixel {  BYTE r;  BYTE g;  BYTE b;  } RGB\_PACKED\_PIXEL;  typedef struct rgb\_packed\_image {  int cols;  int rows;  RGB\_PACKED\_PIXEL \*\*p;  RGB\_PACKED\_PIXEL \*data\_p;  } RGB\_PACKED\_IMAGE;  参考：  RGB\_PACKED\_IMAGE\* Template = readRGBPackedImage(templatename);  RGB\_PACKED\_IMAGE\* input = readRGBPackedImage(inputFileName); |
| int\* TemplateMatching(char\* input,char\* Template) | 模板匹配，支持24位BMP图像，返回匹配框的左上角坐标（x，y）。 |
| int\* TemplateMatching(bmp\_img input,bmp\_img Template) | 模板匹配，支持24位BMP图像，返回匹配框的左上角坐标（x，y）。  需引入以下结构体和声明：  enum bmp\_error  {  BMP\_FILE\_NOT\_OPENED = -4,  BMP\_HEADER\_NOT\_INITIALIZED,  BMP\_INVALID\_FILE,  BMP\_ERROR,  BMP\_OK = 0  };  typedef struct \_bmp\_img  {  bmp\_header img\_header;  bmp\_pixel \*\*img\_pixels;  } bmp\_img;  声明：  enum bmp\_error bmp\_img\_read(bmp\_img \*, const char \*);  参考例程：  bmp\_img input, Template;  bmp\_img\_read(&input, inputfile);  bmp\_img\_read(&Template, outputfile); |
| int FeatureDetection(char\* input,char\* output,float a, int b,float red, float green, float blue) | 特征检测，返回找到的特征点的数量。a是亮度值，如a=255，b与图像清晰度有关，默认b=100；red、green、blue分别是特征点指示器颜色的红色、绿色、蓝色的通道值，如red=0，green=0，blue=1。支持PPM文件。 |
| unsigned int FeatureDetection(char\* input,char\* output,unsigned char red,unsigned char green,unsigned char blue) | 特征检测，返回特征点数量。支持PPM文件。 |
| vector<int> FeatureMatching(char\* input1,char\* input2,char\* output,unsigned char red,unsigned char green,unsigned char blue) | 特征匹配，返回值中的数据格式为：第1个值是特征对的序号，从0开始，第2个值和第3个值分别是是图像1的其中一个特征点的横坐标和纵坐标，第4个值和第5个值分别是是图像2的其中一个特征点的横坐标和纵坐标并与第2个值和第3个值所在的图像1的特征点相对应，这5个值构成一组；同理，第6个值就是下一个特征对的序号，即为1，后面的值则以此类推。格式如：特征对：%d，图像1(%d, %d)->图像2：(%d, %d)。支持PPM文件。 |
| int FeatureMatching(char\* input1,char\* input2,char\* output, float a, int b, float red, float green, float blue) | 特征匹配，返回找到的特征匹配点的数量。a是亮度值，如a=255，b与图像清晰度有关，默认b=100；red、green、blue分别是匹配线颜色的红色、绿色、蓝色的通道值，如red=0，green=0，blue=1。支持PPM文件。 |
| unsigned int FeatureMatching1(char\* input1,char\* input2,char\* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor) | 从两幅输入图像中检测特征点，然后使用蛮力方法匹配两幅图像的特征。input是输入图像，output是生成的特征点图像，bExtractDescriptor=true。返回匹配的特征点数量。支持PGM文件。 |
| vector<int> FeatureMatching2(char\* input1,char\* input2,char\* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor) | 返回值中的数据格式为：第1个值是特征对的序号，从0开始，第2个值和第3个值分别是是图像1的其中一个特征点的横坐标和纵坐标，第4个值和第5个值分别是是图像2的其中一个特征点的横坐标和纵坐标并与第2个值和第3个值所在的图像1的特征点相对应，这5个值构成一组；同理，第6个值就是下一个特征对的序号，即为1，后面的值则以此类推。格式如：特征对：%d，图像1(%d, %d)->图像2：(%d, %d)。input是输入图像，output是生成的特征点图像，bExtractDescriptor=true。支持PGM文件。 |
| unsigned int FeatureExtraction1(char\* input,char\* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor) | 返回特征点数量。input是输入图像，output是生成的特征点图像，bExtractDescriptor=true。支持PGM文件。 |
| std::list<ezsift::SiftKeypoint> FeatureExtraction2(char\* input,char\* output,unsigned char red,unsigned char green,unsigned char blue,bool bExtractDescriptor) | 返回特征点列表。input是输入图像，output是生成的特征点图像，bExtractDescriptor=true。支持PGM文件。  需引入以下命名空间：  namespace ezsift {  #define DEGREE (128) //SIFT关键点: 128维  struct SiftKeypoint {  int octave; // octave数量  int layer; // layer数量  float rlayer; // layer实际数量  float r; // 归一化的row坐标  float c; // 归一化的col坐标  float scale; // 归一化的scale  float ri; // row坐标(layer)  float ci; // column坐标(layer)  float layer\_scale; // scale(layer)  float ori; // 方向(degrees)  float mag; // 模值  float descriptors[DEGREE]; //描述符  };  } |
| vector<int> DefectLocation(PGMData\* Template,PGMData\* Sample,int floor,int size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report) | 查找缺陷位置，返回材料缺陷的位置，以每4个元素为一组。Template是模板图像，Sample是样本图像，g是缺陷的有效界限的X轴长度，h是缺陷的有效界限的Y轴长度，参考：floor=80，size=10，a=64,b=64，c=16,d=16，e=2，f=4，g=65，h=65，FULL=0,EMPTY=255，report=true。  引入以下结构体：  typedef struct \_PGMData {  int row;  int col;  int max\_gray;  int \*\*matrix;  }PGMData;  若模板文件名为Template，样本文件名为Sample，使用以下代码获得合适的输入数据：  首先声明readPGM函数：  PGMData\* readPGM(const char \*file\_name, PGMData \*data);  然后执行以下代码：  PGMData\* model = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Template,model);  PGMData\* data = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Sample,data);  之后将model和data传入相应的函数。  支持P5格式的PGM文件。 |
| vector<int> DefectSize(PGMData\* Template,PGMData\* Sample,int floor,int size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report) | 缺陷尺寸，返回缺陷尺寸，以4个为一组。Template是模板图像，Sample是样本图像，g是缺陷的有效界限的X轴长度，h是缺陷的有效界限的Y轴长度，参考：floor=80，size=10，a=64,b=64，c=16,d=16，e=2，f=4，g=65，h=65，FULL=0,EMPTY=255，report=true。  引入以下结构体：  typedef struct \_PGMData {  int row;  int col;  int max\_gray;  int \*\*matrix;  }PGMData;  若模板文件名为Template，样本文件名为Sample，使用以下代码获得合适的输入数据：  首先声明readPGM函数：  PGMData\* readPGM(const char \*file\_name, PGMData \*data);  然后执行以下代码：  PGMData\* model = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Template,model);  PGMData\* data = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Sample,data);  之后将model和data传入相应的函数。  支持P5格式的PGM文件。 |
| vector<int> GoodBadQuantity(PGMData\* Template,PGMData\* Sample,int floor,int size,int a,int b,int c,int d,int e,int f,int g,int h,int FULL,int EMPTY,bool report) | 样品好坏的数量，返回结果中，第一个元素是合格的圆圈数量，第二个元素是缺陷的圆圈数量。Template是模板图像，Sample是样本图像，g是缺陷的有效界限的X轴长度，h是缺陷的有效界限的Y轴长度，参考：floor=80，size=10，a=64,b=64，c=16,d=16，e=2，f=4，g=65，h=65，FULL=0,EMPTY=255，report=true。  引入以下结构体：  typedef struct \_PGMData {  int row;  int col;  int max\_gray;  int \*\*matrix;  }PGMData;  若模板文件名为Template，样本文件名为Sample，使用以下代码获得合适的输入数据：  首先声明readPGM函数：  PGMData\* readPGM(const char \*file\_name, PGMData \*data);  然后执行以下代码：  PGMData\* model = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Template,model);  PGMData\* data = (PGMData\*)malloc(sizeof(PGMData));  readPGM(Sample,data);  之后将model和data传入相应的函数。  支持P5格式的PGM文件。 |
| struct hough\_param\_circle\* CircleDetection(char\* input,int width,int height) | 圆检测，返回找到的圆的位置和大小等相关信息。支持RAW文件。  需引入以下结构体：  struct hough\_param\_circle {  int a;  int b;  int radius;  int resolution;  int thresh;  struct point \*points;  int points\_size;  }; |
| unsigned int\* CircleDetection(char\* input) | 圆检测，返回圆心的坐标和半径，返回值数组中第一个元素是圆心X坐标，第二个元素是圆心Y坐标，第三个元素是圆半径。支持BMP图像。 |
| int Equal(char\* input1,char\* input2,double c) | 若比对图像的梯度幅相似性偏差值等于c则通过。input1和input2是要比对的两个图像。c是参考的阈值。支持24位BMP图像。 |
| int GreaterThan(char\* input1,char\* input2,double c) | 若比对图像的梯度幅相似性偏差值大于c则通过。input1和input2是要比对的两个图像。c是参考的阈值。支持24位BMP图像。 |
| int LessThan(char\* input1,char\* input2,double c) | 若比对图像的梯度幅相似性偏差值小于c则通过。input1和input2是要比对的两个图像。c是参考的阈值。支持24位BMP图像。 |
| double GMSD(char\* input1, char\* input2) | 求两幅图像的梯度幅相似性偏差值并返回结果。input1和input2是要比对的两个图像。支持24位BMP图像。 |
| vector<vector<double>> Correction(string input,char\* output, int A\_height, int A\_width,double thre\_val, double sigma, double threshold, int angle\_num, int range\_thre, double fun\_thre, int point\_thre) | 图像标定，返回各个顶角的坐标。参考：A\_height = 297，A\_width = 210，thre\_val=0.5，sigma=6.0，threshold=3.5，angle\_num=180，range\_thre=5，fun\_thre=0.8，point\_thre=3。支持BMP文件。 |
| vector<double> CalibrationAndCorrection(char\* input,char\* output1,char\* output2, double sigma, double gra\_threshold, double vote\_threshold, double peak\_dis,unsigned char\* lines\_color, unsigned char\* intersections\_color) | 图像标定，返回各个直线的交点的X轴和Y轴坐标，若以vector<double> p接收返回值则交点1的坐标为(p[0],p[1])，交点2的坐标为(p[2],p[3])，以此类推。参考：sigma=5.5，gra\_threshold=30，vote\_threshold=1000，peak\_dis=200，lines\_color[3]={0,255,0}，intersections\_color[3]={0,0,255}。至少支持BMP图像。 |
| float\* ImageMatching(char\* TargetImage,char\* Template0,char\* Template1,char\* Template2,char\* Template3,char\* Template4,char\* Template5,char\* Template6,char\* Template7,char\* Template8,char\* Template9) | 图像匹配，返回值中的前10个元素是按顺序对应的目标与模板的差异分数，最后一个元素是匹配到的模板的序号。支持BMP图像。 |
| float\* ImageMatching(char\* TargetImage, char\* Template0, char\* Template1) | 图像匹配，返回值中的前2个元素是按顺序对应的目标与模板的差异分数，最后一个元素是匹配到的模板的序号。支持BMP图像。 |
| float\* ImageMatching(Image3 TargetImage,float \*\*templates,int num\_templates) | 图像匹配，如果模板数为n，则返回值中的前n个元素是按顺序对应的目标与模板的差异分数，最后一个元素是匹配到的模板的序号。支持BMP图像。  需引入以下结构体：  typedef struct  {  float \*data;  int width;  int height;  }Image3;  声明：  Image3 load\_bmp(char \*filename);  Image3 img,TargetImage;  float \*\*templates;  int num\_templates = 10;  templates = (float \*\*) malloc(sizeof(float \*) \* num\_templates);  img = load\_bmp(Template0);  templates[0] = img.data;  img = load\_bmp(Template1);  templates[1] = img.data;  img = load\_bmp(Template2);  templates[2] = img.data;  img = load\_bmp(Template3);  templates[3] = img.data;  img = load\_bmp(Template4);  templates[4] = img.data;  img = load\_bmp(Template5);  templates[5] = img.data;  img = load\_bmp(Template6);  templates[6] = img.data;  img = load\_bmp(Template7);  templates[7] = img.data;  img = load\_bmp(Template8);  templates[8] = img.data;  img = load\_bmp(Template9);  templates[9] = img.data;  TargetImage = load\_bmp(TargetImagefile); |
| void ImageFeatures(char\* input,char\* kernel,char\* output) | 图像特征。  kernel文件内容样例：  3  1  0 -1 0  -1 5 -1  0 -1 0  其中，3表示尺寸为3\*3，1表示内核的规模 |
| void FileWrite(char\* BMP,char\* TXT) | 图像隐写之文件写入，将文本文件写入图像。支持32位BMP图像。BMP是要写入的图像文件名，TXT是要写入图像的文本文件名。 |
| void FileWriteOut(char\* BMP,char\* TXT) | 图像隐写之文件写出，将文本文件从图像中取出来。支持32位BMP图像。BMP是要写出的图像文件名，TXT是写出图像后信息保存的文本文件名。 |
| void LBP(char\* input,char\* output) | LBP图像特征提取。支持PNG图像。 |
| void LBP(char\* input,char\* output, int choice, float radius,int pointNumbers) | LBP图像特征提取。radius是取样半径（不小于1的浮点数），pointNumbers是取样点数（不小于8的整数），choice的不同取值表示：1.普通圆形LBP 2.旋转不变圆形LBP 3.等价模式圆形LBP。支持24位BMP图像，只有等价模式下取样点pointNumbers可以大于8但不得大于255。 |
| void Watershed2(char\* input,char\* inputMarqueurs,char\* output,int r,unsigned char R,unsigned char G,unsigned char B) | 图像分割之分水岭算法。inputMarqueurs是输入图像的标记图像。R=230，G=0，B=0，r=1。支持PNG图像。 |
| void EcrireImage1(char\* input,char\* output,uint32\_t rayon) | 图像分割。rayon=5。支持PNG图像。 |
| void EcrireImage2(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 图像分割。rayon=5。支持PNG图像。 |
| void EcrireLPECouleur1(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 图像分割。rayon=5。支持PNG图像。 |
| void Watershed1(char\* input,char\* inputMarqueurs,char\* output,uint32\_t rayon) | 图像分割之分水岭算法。inputMarqueurs是输入图像的标记图像。rayon=5。支持PNG图像。 |
| void EcrireImage3(char\* input,char\* inputMarqueurs,char\* output,uint16\_t rayon) | 图像分割。rayon=1。支持PNG图像。 |
| void EcrireImageCouleursAleatoires(char\* input,char\* inputMarqueurs,char\* output,uint8\_t r,uint8\_t g,uint8\_t b,uint16\_t rayon) | 图像分割。rayon=1。支持PNG图像。 |
| void Watershed(char\* input,char\* inputMarqueurs,char\* output,uint8\_t r,uint8\_t g,uint8\_t b,uint8\_t a,uint16\_t rayon) | 图像分割之分水岭算法。inputMarqueurs是输入图像的标记图像。a一般为255，rayon=1。支持PNG图像。 |
| void FloodFill(char\* input,char\* output,int x,int y,unsigned char novaCor) | 图像分割之漫水填充法。x=0，y=0，novaCor=127。支持PGM文件。 |
| void ConvertCoordinatesToGraphics(char\* input,char\* output, double IMAGE\_SIZE, double PIXEL\_PADDING, bool drawlines) | 读取点坐标并输出点之间绘制的点或线段的图像。IMAGE\_SIZE=800，PIXEL\_PADDING=25，drawlines=0或drawlines=1。  输入文件格式  如果我们将“N”表示为点数，则假设采用以下点坐标文件格式:  N  1 x坐标 y坐标  2 x坐标 y坐标  3 x坐标 y坐标  4 x坐标 y坐标  ....  N x坐标 y坐标 |
| void HumanDetection1(char\* input,char\* output, double MINH, double MAXH, double MINS, double MAXS) | 人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持24位BMP图像。 |
| void HumanDetection2(char\* input, char\* output, double MINH, double MAXH, double MINS, double MAXS) | 人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持24位BMP图像。 |
| void HumanDetection3(char\* input, char\* output, double MINH, double MAXH, double MINS, double MAXS) | 人类检测。参考：MINH=0.0， MAXH=50.0，MINS=0.23，MAXS=0.68。支持24位BMP图像。 |
| int ImageFeatureNumber(char\* input) | 计算图像的特征点数量。支持24位BMP图像。 |
| int ContentSimilarity(char\* input1, char\* input2,int KDTREE\_BBF\_MAX\_NN\_CHKS,double NN\_SQ\_DIST\_RATIO\_THR) | 返回两个图像的内容相似度。KDTREE\_BBF\_MAX\_NN\_CHKS是在BBF搜索期间要检查的关键点NN候选的最大数量，NN\_SQ\_DIST\_RATIO\_THR是NN和第二个NN之间距离平方比的阈值，参考：KDTREE\_BBF\_MAX\_NN\_CHKS=100，NN\_SQ\_DIST\_RATIO\_THR=0.49。支持24位BMP图像。 |
| Feature\* ImageFeature(char\* input) | 返回图像的特征数据。支持24位BMP图像。  需引入一下结构体：  typedef struct Point2D64f  {  double x;  double y;  }Point2D64f;  typedef struct feature  {  double x; /x坐标/  double y; /y坐标/  double a; /Oxford型仿射区域参数/  double b; /Oxford型仿射区域参数/  double c; /Oxford型仿射区域参数/  double scl; /Lowe风格特征的比例/  double ori; /Lowe风格特征的方向/  int d; /描述符长度/  double descr[128]; /描述符/  int type; /功能类型，OXFD或LOWE/  int category; /通用功能类别/  struct feature\* fwd\_match; /前向图像的匹配特征/  struct feature\* bck\_match; /从backmward图像中匹配特征/  struct feature\* mdl\_match; /匹配模型中的特征/  Point2D64f img\_pt; /图像中的位置/  Point2D64f mdl\_pt; /模型中的位置/  void\* feature\_data; /用户可定义数据/  }Feature; |
| double CharacterRecognition(char\* TargetImage,char\* TemplateFileGroup[]) | 字符匹配，支持BMP图像，返回值是目标图像匹配到的模板文件的序号，如返回值是2则说明图像与序号为2（序号从零开始）的模板匹配。  参考：TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" }; |
| double CharacterRecognition1(char\* TargetImage,char\* TemplateFileGroup[]) | 字符匹配，支持BMP图像，返回值是目标图像匹配到的模板文件的序号，如返回值是2则说明图像与序号为2（序号从零开始）的模板匹配。  参考：TemplateFileGroup[]={ "0.txt", "1.txt", "2.txt", "3.txt", "4.txt", "5.txt", "6.txt", "7.txt", "8.txt", "9.txt" }; |