# Digital Image Processing编程作业

学生姓名： 许一宁 学号： 2018222050154

### 第一章

编程1：编程将一幅图像降质为多个低空间分辨率图像

##### 源代码及重点语句的注释

|  |
| --- |
| void CImage\_ProcessingView::OnResolution(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int C = 3;  int inter = 5; //降低图像五倍分辨率  for (int c = 0; c < C; c++){  for (int j = 0; j < h; j++){  for (int k = 0; k < w; k++){  m\_Image.m\_pBits[c][j][k] = m\_Image.m\_pBits[c][j / inter \* inter][k / inter \* inter];  }  }  }  Invalidate(1); //强制调用ONDRAW函数，ONDRAW会绘制图像  } |

##### 实现结果展示

|  |  |
| --- | --- |
| 原图 | 降低分辨率后的图 |

编程2：编程将一幅256级的灰度图像分解为不同灰度分辨率（128、64、32、16、8、4、2）的图像。

##### 源代码及重点语句的注释

|  |
| --- |
| void CImage\_ProcessingView::OnGraychange(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，空图像进行操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  for (int j = 0; j < h; j++){  for (int k = 0; k < w; k++){  int ave = 0.1140 \*m\_Image.m\_pBits[0][j][k] + 0.5870 \*m\_Image.m\_pBits[1][j][k] + 0.2989 \*m\_Image.m\_pBits[2][j][k]; // change image to gray  m\_Image.m\_pBits[0][j][k] = ave;  m\_Image.m\_pBits[1][j][k] = ave;  m\_Image.m\_pBits[2][j][k] = ave;  }  }  int K = 2; //灰度分辨率等级  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int step = 256 / (K); //计算等级跨度  int value = (step << 1) - 1;  m\_Image.m\_pBits[0][i][j] = (m\_Image.m\_pBits[0][i][j] / step) \* value;  m\_Image.m\_pBits[1][i][j] = (m\_Image.m\_pBits[1][i][j] / step) \* value;  m\_Image.m\_pBits[2][i][j] = (m\_Image.m\_pBits[2][i][j] / step) \* value;  }  }  Invalidate(1); //强制调用ONDRAW函数，ONDRAW会绘制图像  } |

##### 实现结果展示

以下为原图以及分辨率依次为128,64,32,16,8,4,2。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

编程3：编程实现图像差分、多幅图像相加去噪。

##### 流程图

|  |  |
| --- | --- |
| 图片差分流程图 | 图片叠加去噪流程图 |

##### 源代码及重点语句的注释

1、图片差分

|  |
| --- |
| void CImage\_ProcessingView::OnDiff(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  MyImage\_ m\_Image2;  CFileDialog dlg(TRUE);//同样是打开一个新的对话框，存储别的输入图片  if (IDOK == dlg.DoModal()){  if (!m\_Image2.IsNull())  m\_Image2.Destroy();  m\_Image2.Load(dlg.GetPathName());  if (m\_Image2.IsNull())  return;  int w0 = m\_Image2.GetWidth();  int h0 = m\_Image2.GetHeight();  if (w0 != w || h0 != h) return;//比较两张图片大小，如果不同就return  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  m\_Image.m\_pBits[0][i][j] = m\_Image.m\_pBits[0][i][j] - m\_Image2.m\_pBits[0][i][j];  m\_Image.m\_pBits[1][i][j] = m\_Image.m\_pBits[1][i][j] - m\_Image2.m\_pBits[1][i][j];  m\_Image.m\_pBits[2][i][j] = m\_Image.m\_pBits[2][i][j] - m\_Image2.m\_pBits[2][i][j]; //逐像素计算差值  }  }  Invalidate(1);  }  } |

2、叠加去燥

|  |
| --- |
| void CImage\_ProcessingView::OnDenoiseing(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  MyImage\_ m\_Image2;  CFileDialog dlg(TRUE);//同样是打开一个新的对话框，存储别的输入图片  dlg.m\_ofn.Flags |= OFN\_ALLOWMULTISELECT;//允许选择多个文件  dlg.m\_ofn.nMaxFile = 20 \* 101;//最多可以打开20个文件，每个文件名的字符数<=100  const DWORD numberOfFileNames = 32;//最多允许32个文件  const DWORD fileNameMaxLength = MAX\_PATH + 1;  const DWORD bufferSize = (numberOfFileNames \* fileNameMaxLength) + 1;  TCHAR\* filenamesBuffer = new TCHAR[bufferSize];  filenamesBuffer[0] = NULL;//必须的  filenamesBuffer[bufferSize - 1] = NULL;  dlg.m\_ofn.lpstrFile = filenamesBuffer;  dlg.m\_ofn.nMaxFile = bufferSize;  CStringArray strArrFilePaths;  if (dlg.DoModal() == IDOK){  int k = 0;  POSITION pos = dlg.GetStartPosition();//获取第一个文件名的位置  while (pos != NULL) //GetNextPathName()返回当前pos的文件名，并将下一个文件名的位置保存到pos中，最后达到连续叠加多幅图片{  if (!m\_Image2.IsNull())  m\_Image2.Destroy();  m\_Image2.Load(dlg.GetNextPathName(pos));  if (m\_Image2.IsNull())return;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  m\_Image.m\_pBits[0][i][j] = (m\_Image.m\_pBits[0][i][j] / (k + 2))\*(k + 1) + m\_Image2.m\_pBits[0][i][j] / (k + 2);  m\_Image.m\_pBits[1][i][j] = (m\_Image.m\_pBits[1][i][j] / (k + 2))\*(k + 1) + m\_Image2.m\_pBits[1][i][j] / (k + 2);  m\_Image.m\_pBits[2][i][j] = (m\_Image.m\_pBits[2][i][j] / (k + 2))\*(k + 1) + m\_Image2.m\_pBits[2][i][j] / (k + 2); //计算叠加图片的数量并继续叠加。  }  }  k++;  }  Invalidate(1);  }  } |

##### 实现结果展示

1. 差分结果图

|  |  |
| --- | --- |
| 原图 | 相减图像 |
| 差分后图像 | |

2、叠加去噪结果图。

|  |  |
| --- | --- |
| 加入噪声后的图片 | 叠加去噪后的效果 |

### 第三章

编程1：编写程序统计任意一幅图像的直方图并显示

##### 源代码及重点语句的注释

|  |
| --- |
| void CImage\_ProcessingView::OnHistogram(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  if (bits == 24 || bits == 32) {  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  m\_Image.calcHistogram();//调用m\_Image内的直方图统计函数  paintHistDialog dlg(this);//用一个CImage\_ProcessingView的指针取初始化dlg  dlg.DoModal();  Invalidate(1);  }  //直方图统计函数  void MyImage\_::calcHistogram(void)//直方图统计{  BYTE \*lpSrc; //指向源图的指针  int w = GetWidth();  int h = GetHeight();  int arr[256] = {0}; //每个灰度值统计  int nrow = m\_CImage.GetPitch();//获得Image每一行像素的RGB所占用的存储空间的大小  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  BYTE value = m\_pBits[0][i][j];  arr[value] ++;  }  }  for (int k = 0; k < 256; k++){  hist[k] = arr[k] / (w\*h\*1.0f); //Normalization  }  } |

##### 实现结果展示

|  |  |
| --- | --- |
|  |  |

原图 归一化直方图

编程2：编程实现图像均衡化和图像规格化

##### 流程图

|  |  |
| --- | --- |
| 直方图均衡流程图 | 直方图匹配流程图 |

##### 源代码及重点语句的注释

1、直方图均衡

|  |
| --- |
| void CImage\_ProcessingView::OnHistequal(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  if (bits == 24 || bits == 32) {  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave; // change image to gray  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  m\_Image.calcHistogram();  float s[256] = { 0 };//均衡  int hist\_equal[256] = { 0 };//均衡后  for (int i = 0; i < 256; i++){  for (int j = 0; j <= i; j++){  s[i] += 255 \* m\_Image.hist[j]; //建立均衡后的映射关系  }  hist\_equal[i] = floor(s[i]); // 对映射关系去下整数  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int value = m\_Image.m\_pBits[0][i][j];  int new\_value = hist\_equal[value]; //利用映射关系对图片像素进行操作  m\_Image.m\_pBits[0][i][j] = new\_value;  m\_Image.m\_pBits[1][i][j] = new\_value;  m\_Image.m\_pBits[2][i][j] = new\_value;  }  }  m\_Image.calcHistogram();  paintHistDialog dlg(this);//用一个CImage\_ProcessingView的指针取初始化dlg  dlg.DoModal();  Invalidate(1);  } |

1. 直方图规定

|  |
| --- |
| void CImage\_ProcessingView::OnHistmatch(){  // TODO: 在此添加命令处理程序代码  MyImage\_ m\_Image\_Match;  CFileDialog dlg(TRUE);//同样是打开一个新的对话框，存储别的输入图片  if (IDOK == dlg.DoModal()) {  if (!m\_Image\_Match.IsNull())  m\_Image\_Match.Destroy();  m\_Image\_Match.Load(dlg.GetPathName());  if (m\_Image\_Match.IsNull())return;  int w = m\_Image\_Match.GetWidth();//获得图像的宽度  int h = m\_Image\_Match.GetHeight();//获得图像的高度  int arr[256] = { 0 };  int G[256] = { 0 };//存放待匹配图片的每个像素值的统计量  int bits = m\_Image\_Match.GetBPP();  if (bits == 24 || bits == 32){  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int ave = 0.1140 \*m\_Image\_Match.m\_pBits[0][i][j] + 0.5870 \*m\_Image\_Match.m\_pBits[1][i][j] + 0.2989 \*m\_Image\_Match.m\_pBits[2][i][j];  m\_Image\_Match.m\_pBits[0][i][j] = ave; // change image to gray  m\_Image\_Match.m\_pBits[1][i][j] = ave;  m\_Image\_Match.m\_pBits[2][i][j] = ave;  }  }  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  BYTE value = m\_Image\_Match.m\_pBits[0][i][j];  arr[value] ++;//计算统计直方图  }  }  if (m\_Image.IsNull()) return;//判断图像是否为空，空图像操作会出现未知的错误  int w0 = w;  int h0 = h;  w = m\_Image.GetWidth();//获得图像的宽度  h = m\_Image.GetHeight();//获得图像的高度  for (int i = 0; i < 256; i++){  for (int j = 0; j <= i; j++){  G[i] += float(arr[j]) / w0 / h0 \* w \* h;  } //计算映射到图片中每个像素值的统计量  }  bits = m\_Image.GetBPP();  int hierarchical = 2;  int n[256] = { 0 };//存放源图片的每个像素值的统计量  int nMap[256] = { 0 };//存放源图到目标图片的像素值映射表  if (bits == 24 || bits == 32){  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  BYTE value = m\_Image.m\_pBits[0][i][j];  n[value] ++;//源图的统计直方图  }  }  int count\_g = 0;  int count\_n = 0;  int sum = 0;  while (true) {  if (count\_n == 256) break;  if (sum < G[count\_g]) {  //count\_n++;  sum += n[count\_n++];  continue;  }  else {  nMap[count\_n] = count\_g;  count\_g++;  }  if (count\_g == 255) {  nMap[255] = 255;  break;  }  }  int k = nMap[255];  for (int i = 255; i >= 0; i--){  if (nMap[i] == 0) nMap[i] = k;  else k = nMap[i];  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++)  int value = m\_Image.m\_pBits[0][i][j];  m\_Image.m\_pBits[0][i][j] = nMap[value];  m\_Image.m\_pBits[1][i][j] = nMap[value];  m\_Image.m\_pBits[2][i][j] = nMap[value];  }  }  m\_Image.calcHistogram();  paintHistDialog dlg1(this);//用一个CImage\_ProcessingView的指针取初始化dlg  dlg1.DoModal();  Invalidate(1);  }  } |

##### 实现结果展示

1. 直方图均衡

|  |  |
| --- | --- |
| 均衡前原图 | 均衡前归一化直方图 |
| 均衡后图 | 均衡后归一化直方图 |

1. 直方图规范化

|  |  |
| --- | --- |
| 直方图规定前原图 | 直方图规定前原直方图 |
| 规定模板原图 | 规定模板直方图 |
| 规定后图 | 规定后直方图 |

编程3：编程实现图像均值滤波、中值滤波。

##### 源代码及重点语句的注释

1、均值滤波

|  |
| --- |
| void CImage\_ProcessingView::OnMeanfilter(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  MyImage\_ m\_Image2;  m\_Image2.Load(filename);  for (int i = 1; i < h - 1; i++){  for (int j = 1; j < w - 1; j++){//对像素点领域内的像素值进行相加求平均  m\_Image.m\_pBits[0][i][j] = m\_Image2.m\_pBits[0][i - 1][j - 1] / 9 + m\_Image2.m\_pBits[0][i][j - 1] / 9 + m\_Image2.m\_pBits[0][i + 1][j - 1] / 9 + m\_Image2.m\_pBits[0][i - 1][j] / 9 + m\_Image2.m\_pBits[0][i][j] / 9 + m\_Image2.m\_pBits[0][i + 1][j] / 9 + m\_Image2.m\_pBits[0][i - 1][j + 1] / 9 + m\_Image2.m\_pBits[0][i][j + 1] / 9 + m\_Image2.m\_pBits[0][i + 1][j + 1] / 9;  m\_Image.m\_pBits[1][i][j] = m\_Image2.m\_pBits[1][i - 1][j - 1] / 9 + m\_Image2.m\_pBits[1][i][j - 1] / 9 + m\_Image2.m\_pBits[1][i + 1][j - 1] / 9 + m\_Image2.m\_pBits[1][i - 1][j] / 9 + m\_Image2.m\_pBits[1][i][j] / 9 + m\_Image2.m\_pBits[1][i + 1][j] / 9 + m\_Image2.m\_pBits[1][i - 1][j + 1] / 9 + m\_Image2.m\_pBits[1][i][j + 1] / 9 + m\_Image2.m\_pBits[1][i + 1][j + 1] / 9;  m\_Image.m\_pBits[2][i][j] = m\_Image2.m\_pBits[2][i - 1][j - 1] / 9 + m\_Image2.m\_pBits[2][i][j - 1] / 9 + m\_Image2.m\_pBits[2][i + 1][j - 1] / 9 + m\_Image2.m\_pBits[2][i - 1][j] / 9 + m\_Image2.m\_pBits[2][i][j] / 9 + m\_Image2.m\_pBits[2][i + 1][j] / 9 + m\_Image2.m\_pBits[2][i - 1][j + 1] / 9 + m\_Image2.m\_pBits[2][i][j + 1] / 9 + m\_Image2.m\_pBits[2][i + 1][j + 1] / 9;  }  }  Invalidate(1);  } |

2、中值滤波

|  |
| --- |
| void CImage\_ProcessingView::OnMidfilter(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  int bits = m\_Image.GetBPP();  int hierarchical = 2;  MyImage\_ m\_Image2;  m\_Image2.Load(filename);  for (int i = 1; i < h - 1; i++){  for (int j = 1; j < w - 1; j++){  for (int k = 0; k < 3; k++){  int arr[9] = {m\_Image2.m\_pBits[k][i - 1][j - 1], m\_Image2.m\_pBits[k][i][j - 1], m\_Image2.m\_pBits[k][i + 1][j - 1], m\_Image2.m\_pBits[k][i - 1][j], m\_Image2.m\_pBits[k][i][j], m\_Image2.m\_pBits[k][i + 1][j], m\_Image2.m\_pBits[k][i - 1][j + 1], m\_Image2.m\_pBits[k][i][j + 1], m\_Image2.m\_pBits[k][i + 1][j + 1]}; //将像素点领域内的像素值放到数组内  sort(arr, arr + 9); //对图像像素值进行排序  m\_Image.m\_pBits[k][i][j] = arr[4]; //取排序后的中值  }  }  }  Invalidate(1);  } |

##### 实现结果展示

1. 均值滤波

|  |  |
| --- | --- |
|  |  |

原图 均值滤波后的图

1. 中值滤波

|  |  |
| --- | --- |
|  |  |

原图 中值滤波后的图

### 第四章

编程1：实现一幅图像的FFT变换，显示其频谱图像，验证其平移特性、旋转特性等。

##### 流程图

|  |
| --- |
|  |

##### 源代码及重点语句的注释

1、OnTransformfft函数

|  |
| --- |
| void CImage\_ProcessingView::OnTransformfft(){  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获取高度和宽度  int h = m\_Image.GetHeight();  if (!m\_Imagesrc.IsNull()) m\_Imagesrc.Destroy();  m\_Imagesrc.Load(filename); //记住原图文件名和原图  int bits = m\_Image.GetBPP();  if (w & w - 1 != 0 || h & h - 1 != 0) return; //不满足2n条件fft则返回  if (bits == 24 || bits == 32){ //转为灰度图  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  complex\_mat<float> F(w, h); //创建一个复数矩阵存放fft结果  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){//将图片数值转为复数型存放在F中以便调用  F.y[i][j] = complex<float>(m\_Image.m\_pBits[0][i][j], 0);  }  }  fft2<float>(F.y, w, h);//现在的F.y就是fft后的结果  fft\_shift<float>(F.y, w, h); //做fftshift将中心放在图片原点  float max = log(1 + abs(F.y[0][0]));  float min = log(1 + abs(F.y[0][0])); //保存增强图片的最大最小值以便归一化  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j]));  if (value > max) max = value;  if (value < min) min = value;  }  }  float inner = 0;  inner = (max - min) / 255;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j])); //对图片做增强  value = float((value + min) / inner);  if (value > 255) value = 255;  if (value < 0) value = 0;  m\_Image.m\_pBits[0][i][j] = int(value); //归一化后赋值到m\_Image中  m\_Image.m\_pBits[1][i][j] = int(value);  m\_Image.m\_pBits[2][i][j] = int(value);  }  }  m\_Image.flag = 1; //显示频谱图（保存在m\_Image中）和原图（保存在m\_Imagesrc中）  Invalidate(1);  } |

2、构造的complex\_mat.hpp以便fft调用

|  |
| --- |
| #ifndef COMPLEX\_MAT\_HPP  #define COMPLEX\_MAT\_HPP  #include<complex>  template <class T>  class complex\_mat {  public:  int r; // 实部  int c; // 虚部  std::complex<T> \*\*y;  complex\_mat(int r, int c) {// 构造函数  this->r = r;  this->c = c;  init();  }  ~complex\_mat() {// 析构函数  this->free();  };  void init() {  y = new complex<T>\*[r];  for (int j = 0; j < r; j++)  y[j] = new complex<T>[c];  };  void free() {  for (int j = 0; j < r; j++)  delete[] y[j];  delete[] y;  };  };  #endif // COMPLEX\_MAT\_HPP |

3、fft.hpp以便调用，内含一维fft，一维ifft，二维fft，二维fft以及fftshift函数。

|  |
| --- |
| #ifndef FFT\_TRANSFORM\_HPP  #define FFT\_TRANSFORM\_HPP  #include "complex\_mat.hpp"  using namespace std;  #define PI 3.14159265359  template <class T>  void change(complex<T> \*y, int len) {  int i, j, k;  for (i = 1, j = len / 2; i < len - 1; i++) {  if (i < j) swap(y[i], y[j]);k = len / 2;  while (j >= k) {  j = j - k;  k = k / 2;  }  if (j < k) j += k;  }  }  template <class T>//一维fft函数  void fft(complex<T> y[], int len) {  change(y, len);  for (int h = 2; h <= len; h <<= 1) {// 迭代法求fft  complex<T> w(cos(-1 \* 2 \* PI / h), sin(-1 \* 2 \* PI / h)); // W  for (int j = 0; j < len; j += h) {  complex<T> wn(1, 0);  for (int k = j; k < j + h / 2; k++) {  complex<T> u = y[k];  complex<T> t = wn \* y[k + h / 2];  y[k] = u + t;  y[k + h / 2] = u - t;  wn = wn \* w;  }  }  }  }  template <class T>//一维ifft函数  void ifft(complex<T> y[], int len) {  for (int h = 0; h < len; h++) {  y[h] /= len; // ifft还需除以序列总长度  }  change(y, len);  for (int h = 2; h <= len; h <<= 1) {// 迭代法  complex<T> w(cos(1 \* 2 \* PI / h), sin(1 \* 2 \* PI / h)); // -W和fft反相  for (int j = 0; j < len; j += h) {  complex<T> wn(1, 0);  for (int k = j; k < j + h / 2; k++) {  complex<T> u = y[k];  complex<T> t = wn \* y[k + h / 2];  y[k] = u + t;  y[k + h / 2] = u - t;  wn = wn \* w;  }  }  }  }  template <class T>// 二维fft函数  void fft2(complex<T> \*\*y, int c, int r) {  complex<T>\* row = new complex<T>[c];  complex<T>\* col = new complex<T>[r]; // 分别对每行和每列求一维fft得到二维fft结果  for (int i = 0; i < r; i++) {  for (int n = 0; n < c; n++) {  row[n] = y[i][n];  }  fft(row, c);  for (int n = 0; n < c; n++) {  y[i][n] = row[n];  }  }  for (int i = 0; i < c; i++) {  for (int n = 0; n < r; n++) {  col[n] = y[n][i];  }  fft(col, r);  for (int n = 0; n < r; n++) {  y[n][i] = col[n];  }  }  delete[] row; // 释放指针以免占用空间  delete[] col;  }  template <class T>// 二维ifft函数  void ifft2(complex<T> \*\*y, int c, int r) {  complex<T>\* row = new complex<T>[c];  complex<T>\* col = new complex<T>[r];  for (int i = 0; i < c; i++) {  for (int n = 0; n < r; n++) {  col[n] = y[n][i];  }  ifft(col, r);  for (int n = 0; n < r; n++) {  y[n][i] = col[n];  }  }  for (int i = 0; i < r; i++) {  for (int n = 0; n < c; n++) {  row[n] = y[i][n];  }  ifft(row, c);  for (int n = 0; n < c; n++) {  y[i][n] = row[n];  }  }  delete[] row;  delete[] col;  }  template <class T>// 二维fftshift函数  void fft\_shift(complex<T> \*\*y, int w, int h) {  complex\_mat<float> F\_buf(w, h);  for (int i = 0; i < w; i++){  for (int j = 0; j < h; j++){  int m = i + w / 2;  if (m > w - 1) m = i - w / 2;  F\_buf.y[i][j] = y[m][j];  }  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int m = i + h / 2;  if (m > h - 1) m = i - h / 2;  y[j][i] = F\_buf.y[j][m];  }  }// 对fft结果每行每列分别进行循环移位操作。移位为边长的二分之一  }  #endif // !FFT\_TRANSFORM\_HPP |

##### 实现结果展示

|  |
| --- |
| 经过fft变换后的图像以及原图 |
| 旋转九十度后的fft变换结果及旋转后的图（可见fft有旋转特性） |
| 对图片进行横向循环移位后得到的fft变换结果和图（可见fft平移不变） |

编程2：采用理想、巴特沃斯、高斯滤波器实现频率域低通、高通滤波功能。

##### 源代码及重点语句的注释

1、理想低通滤波器

|  |
| --- |
| void CImage\_ProcessingView::OnLowpass(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获取高度和宽度  int h = m\_Image.GetHeight();  if (!m\_Imagesrc.IsNull()) m\_Imagesrc.Destroy();  m\_Imagesrc.Load(filename); //从文件名中读取图片放在原图地址中  int bits = m\_Image.GetBPP();  if (w & w - 1 != 0 || h & h - 1 != 0) return;  if (bits == 24 || bits == 32) {  for (int i = 0; i < h; i++) {  for (int j = 0; j < w; j++) {  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  complex\_mat<float> F(w, h);  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  F.y[i][j] = complex<float>(m\_Image.m\_pBits[0][i][j], 0);  }  }  fft2<float>(F.y, w, h);// 做fft变换  fft\_shift<float>(F.y, w, h); // 做fftshift  float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 100; // 设置低通滤波器半径值  for (int i = 0; i < h; i++){// 遍历图片将距离图片中心超过半径的值置为零  for (int j = 0; j < w; j++){  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  if ( dis > pow(D0, 2)) F.y[i][j] = 0;  }  }  float max = log(1 + abs(F.y[0][0])); // 增强频谱并将频谱信息放在显示缓存中  float min = log(1 + abs(F.y[0][0]));  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j]));  if (value > max) max = value;  if (value < min) min = value;  }  }  float inner = 0;  inner = (max - min) / 255;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j]));  value = float((value + min) / inner);  if (value > 255) value = 255;  if (value < 0) value = 0;  m\_Image.m\_pBits[0][i][j] = int(value);  m\_Image.m\_pBits[1][i][j] = int(value);  m\_Image.m\_pBits[2][i][j] = int(value);  }  }  fft\_shift<float>(F.y, w, h); // 做fftshift  ifft2<float>(F.y, w, h); // ifft还原变换后的图。  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int value = abs(F.y[i][j]);  if (value > 255) value = 255;  if (value < 0) value = 0;  m\_Imagesrc.m\_pBits[0][i][j] = value;  m\_Imagesrc.m\_pBits[1][i][j] = value;  m\_Imagesrc.m\_pBits[2][i][j] = value;  }  }  m\_Image.flag = 1; // 同时显示频谱和图。  Invalidate(1);  } |

2、理想高通滤波器

|  |
| --- |
| void CImage\_ProcessingView:: OnHighpass(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获取高度和宽度  int h = m\_Image.GetHeight();  if (!m\_Imagesrc.IsNull()) m\_Imagesrc.Destroy();  m\_Imagesrc.Load(filename); //从文件名中读取图片放在原图地址中  int bits = m\_Image.GetBPP();  if (w & w - 1 != 0 || h & h - 1 != 0) return;  if (bits == 24 || bits == 32) {  for (int i = 0; i < h; i++) {  for (int j = 0; j < w; j++) {  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  complex\_mat<float> F(w, h);  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  F.y[i][j] = complex<float>(m\_Image.m\_pBits[0][i][j], 0);  }  }  fft2<float>(F.y, w, h);// 做fft变换  fft\_shift<float>(F.y, w, h); // 做fftshift  float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 100; // 设置滤波器半径值  for (int i = 0; i < h; i++){// 遍历图片将距离图片中心小于半径的值置为零  for (int j = 0; j < w; j++){  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  if ( dis < pow(D0, 2)) F.y[i][j] = 0;  }  }  float max = log(1 + abs(F.y[0][0])); // 增强频谱并将频谱信息放在显示缓存中  float min = log(1 + abs(F.y[0][0]));  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j]));  if (value > max) max = value;  if (value < min) min = value;  }  }  float inner = 0;  inner = (max - min) / 255;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float value = log(1 + abs(F.y[i][j]));  value = float((value + min) / inner);  if (value > 255) value = 255;  if (value < 0) value = 0;  m\_Image.m\_pBits[0][i][j] = int(value);  m\_Image.m\_pBits[1][i][j] = int(value);  m\_Image.m\_pBits[2][i][j] = int(value);  }  }  fft\_shift<float>(F.y, w, h); // 做fftshift  ifft2<float>(F.y, w, h); // ifft还原变换后的图。  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int value = abs(F.y[i][j]);  if (value > 255) value = 255;  if (value < 0) value = 0;  m\_Imagesrc.m\_pBits[0][i][j] = value;  m\_Imagesrc.m\_pBits[1][i][j] = value;  m\_Imagesrc.m\_pBits[2][i][j] = value;  }  }  m\_Image.flag = 1; // 同时显示频谱和图。  Invalidate(1);  } |

1. 巴特沃斯低通滤波器（关键代码）

|  |
| --- |
| float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 1000; // 设置低通滤波器半径值  int order = 5; // 设置巴特沃兹滤波器阶数  for (int i = 0; i < h; i++){// 遍历图片  for (int j = 0; j < w; j++){  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  float H = 1 / (1 + pow(dis / D0, 2 \* order));  F.y[i][j] = F.y[i][j] \* H;  }// 计算在x=i,y=j时低通变换函数H，并将fft结果和H相乘  } |

1. 巴特沃斯高通滤波器（关键代码）

|  |
| --- |
| fft2<float>(F.y, w, h);// 做fft变换  fft\_shift<float>(F.y, w, h); // 做fftshift  float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 1000; // 设置低通滤波器半径值  int order = 5; // 设置巴特沃兹滤波器阶数  for (int i = 0; i < h; i++){// 遍历图片  for (int j = 0; j < w; j++){  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  if ( dis < pow(D0, 2)) F.y[i][j] = 0;  }// 计算在x=i,y=j时低通变换函数H，并将fft结果和H相乘  } |

1. 高斯低通滤波器（关键代码）

|  |
| --- |
| fft2<float>(F.y, w, h);// 做fft变换  fft\_shift<float>(F.y, w, h); // 做fftshift  float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 100; // 设置滤波器半径值  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){// 遍历图片  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  float m = -pow(dis, 2) / (2 \* pow(D0, 2));  float e = E;  float H = pow(e, m);  F.y[i][j] = F.y[i][j] \* H;  }// 计算在x=i,y=j时低通变换函数H，并将fft结果和H相乘  } |

1. 高斯高通滤波器（关键代码）

|  |
| --- |
| fft2<float>(F.y, w, h);// 做fft变换  fft\_shift<float>(F.y, w, h); // 做fftshift  float center\_x, center\_y;  center\_x = float(w) / 2;  center\_y = float(h) / 2; // 设置变量保存中心坐标  float D0 = 100; // 设置低通滤波器半径值  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float dis = pow(i - center\_y, 2) + pow(j - center\_x, 2);  float m = -pow(dis, 2) / (2 \* pow(D0, 2));  float e = E;  float H = 1 - pow(e, m);  F.y[i][j] = F.y[i][j] \* H;  }  } |

##### 实现结果展示

|  |  |
| --- | --- |
| 原图 | 填充0后的原图 |
| 理想低通滤波器 | |
| 原图填充0后的理想低通滤波器 | |
| 理想高通滤波器 | |
| 原图填充0后的理想高通滤波器 | |
| 巴特沃兹低通滤波器 | |
| 原图填充0后的巴特沃兹低通滤波器 | |
| 巴特沃兹高通滤波器 | |
| 原图填充0后的巴特沃兹高通滤波器 | |
| 高斯低通滤波器 | |
| 原图填充0后的高斯低通滤波器 | |
| 高斯高通滤波器 | |
| 原图填充0后的高斯高通滤波器 | |

对比可以观察得到，低通滤波器可以使得图片变模糊，而高通滤波器可以很好的得到图片的边界，如果对原图填充了0，则低通滤波器可以看到图片有黑色的模糊边缘，高通滤波器可以明显看到图片的大小边界。

### 第五章

编程1：实现自适应中值滤波，要求：

1) 实现在图像上叠加至少两种噪声，其中一种为脉冲噪声。

2) 可通过人工输入设置最大允许的模板尺寸。

需要画出流程图。

##### 流程图

|  |
| --- |
|  |

##### 源代码及重点语句的注释

1、加两种噪声

|  |
| --- |
| void CImage\_ProcessingView::OnAddimpulsenoise(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得第一幅图像的宽度  int h = m\_Image.GetHeight();//获得第一幅图像的高度  int n = 0.1 \* h \* w;//盐噪声概率0.1  for (int k = 0; k < n; k++){  int i = rand() % w;  int j = rand() % h; //随机取值行列  m\_Image.m\_pBits[0][j][i] = 255;  m\_Image.m\_pBits[1][j][i] = 255;  m\_Image.m\_pBits[2][j][i] = 255;  }  int d = 0.1 \* h \* w;  for (int k = 0; k < d; k++){  int i = rand() % w;  int j = rand() % h;  m\_Image.m\_pBits[0][j][i] = 0;  m\_Image.m\_pBits[1][j][i] = 0;  m\_Image.m\_pBits[2][j][i] = 0;  }  Invalidate(1);  }  double CImage\_ProcessingView::generateGaussianNoise(double mu, double sigma){  const double epsilon = 0.00000001;//定义小值  static double z0, z1;  static bool flag = false;  flag = !flag;  if (!flag)//flag为假构造高斯随机变量X  return z1 \* sigma + mu;  double u1, u2;  do//构造随机变量{  u1 = rand() \* (1.0 / RAND\_MAX);  u2 = rand() \* (1.0 / RAND\_MAX);  } while (u1 <= epsilon);  z0 = sqrt(-2.0\*log(u1))\*cos(2 \* PI\*u2);  z1 = sqrt(-2.0\*log(u1))\*sin(2 \* PI\*u2);  return z0 \* sigma + mu; // 产生高斯分布的噪声值  }  void CImage\_ProcessingView::OnAddguaussiannoise(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得第一幅图像的宽度  int h = m\_Image.GetHeight();//获得第一幅图像的高度  int mean = 0; // 定义产生高斯噪声的均值和方差  int var = 30;  for (int j = 0; j < h; j++){  for (int i = 0; i < w; i++){// 每个三通道像素值加入随机高斯噪声  int val0 = m\_Image.m\_pBits[0][j][i] + generateGaussianNoise(mean, var);  int val1 = m\_Image.m\_pBits[1][j][i] + generateGaussianNoise(mean, var);  int val2 = m\_Image.m\_pBits[2][j][i] + generateGaussianNoise(mean, var);  m\_Image.m\_pBits[0][j][i] = val0 > 255 ? 255 : val0 < 0 ? 0 : val0;  m\_Image.m\_pBits[1][j][i] = val1 > 255 ? 255 : val1 < 0 ? 0 : val1;  m\_Image.m\_pBits[2][j][i] = val2 > 255 ? 255 : val2 < 0 ? 0 : val2;  }// 对超出范围的值进行处理  }  Invalidate(1);  } |

2、自适应中值滤波

|  |
| --- |
| void CImage\_ProcessingView::OnAdaptedmidfilter(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得第一幅图像的宽度  int h = m\_Image.GetHeight();//获得第一幅图像的高度  m\_Imagesrc.Load(filename);  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  m\_Imagesrc.m\_pBits[0][i][j] = m\_Image.m\_pBits[0][i][j];  m\_Imagesrc.m\_pBits[1][i][j] = m\_Image.m\_pBits[0][i][j];  m\_Imagesrc.m\_pBits[2][i][j] = m\_Image.m\_pBits[0][i][j];  }  }  WindowSizeDialog dlg(this); //设置对话框输入最大窗口大小  if (IDOK == dlg.DoModal()){  int W\_size = \_ttoi(dlg.str); //获取最大窗口大小  int size = 3;//初始窗口大小  int min, max, med, A1, A2, B1, B2;  int p[1024]; //初始化存放窗口内像素值的数组  int\* arr = p;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  while (true){  for (int m = -size / 2; m <= size / 2; m++){  for (int n = -size / 2; n <= size / 2; n++){  int value;  if (i + m < 0) value = 0;  else if (j + n < 0) value = 0;  else if (i + m > h - 1) value = 0;  else if (j + n > w - 1) value = 0;  else value = m\_Image.m\_pBits[0][i + m][j + n];  \*arr = value;  arr++;  }// 将窗口内像素值依次存放到数组中  }  int nums = pow(size, 2);  arr -= nums;  sort(arr, arr + nums); // 对数组进行排序  med = arr[nums / 2 + 1]; // 中值  min = arr[0]; // 最小值  max = arr[nums - 1]; // 最大值  A1 = med - min;  A2 = med - max;  if (A1 > 0 && A2 < 0) {// 判断中值是否大于最大值小于最小值  int Zxy = m\_Image.m\_pBits[0][i][j];  B1 = Zxy - min;  B2 = Zxy - max;  if (B1 > 0 && B2 < 0){//目标像素点是否大于最大值小于最小值  m\_Imagesrc.m\_pBits[0][i][j] = Zxy;  m\_Imagesrc.m\_pBits[1][i][j] = Zxy;  m\_Imagesrc.m\_pBits[2][i][j] = Zxy;  break; // 符合条件则保持源像素值  }  else {// 不符合条件取中值  m\_Imagesrc.m\_pBits[0][i][j] = med;  m\_Imagesrc.m\_pBits[1][i][j] = med;  m\_Imagesrc.m\_pBits[2][i][j] = med;  break;  }  }  else {// 判断中值不符合条件则增大窗口尺寸  size += 2;  if (size > W\_size) {  m\_Imagesrc.m\_pBits[0][i][j] = med;  m\_Imagesrc.m\_pBits[1][i][j] = med;  m\_Imagesrc.m\_pBits[2][i][j] = med;  size = 5;  break;  }  }  }  }  }  m\_Image.flag = 1;  Invalidate(1);  }  } |

##### 实现结果展示

实验结果如下：

|  |  |  |
| --- | --- | --- |
| 高斯噪声 | 脉冲噪声+高斯噪声 | |
| 最大窗口大小 | | |
| 自适应中值滤波前后对比图 | | |
| 脉冲噪声 | | 脉冲噪声 + 高斯噪声 |
| 最大窗口尺寸 | | |
| 自适应中值滤波前后对比 | | |

### 第六章

编程1：实现将一幅图像的RGB分量分别显示；将该图像从RGB空间转换到HSI空间，并分别显示H、S、I分量。

##### 源代码及重点语句的注释

1、显示R、G、B分量

|  |
| --- |
| void CImage\_ProcessingView::OnShowrgb(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  m\_Image\_r.Load(filename);  m\_Image\_g.Load(filename); // 用于存放RGB三个通道的图片  m\_Image\_b.Load(filename);  for (int j = 0; j < h; j++){  for (int k = 0; k < w; k++){  m\_Image\_r.m\_pBits[0][j][k] = 0;//B  m\_Image\_r.m\_pBits[1][j][k] = 0;//G  m\_Image\_g.m\_pBits[0][j][k] = 0;//B  m\_Image\_g.m\_pBits[2][j][k] = 0;//R  m\_Image\_b.m\_pBits[2][j][k] = 0;//R  m\_Image\_b.m\_pBits[1][j][k] = 0;//G  }  }// 用循环访问图像的像素值，将不需要的分量置为0，图像就只剩下需要的分量了  m\_Image.flag = 3;  Invalidate(1); //强制调用ONDRAW函数，ONDRAW会绘制图像  } |

2、显示H、S、I分量

|  |
| --- |
| void CImage\_ProcessingView::OnShowhsi(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  m\_Image\_h.Load(filename);  m\_Image\_s.Load(filename);  m\_Image\_i.Load(filename);  float theta, H, S, I;  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int R, G, B; //保存当前像素点RGB分量值  R = m\_Imagesrc.m\_pBits[2][i][j];  G = m\_Imagesrc.m\_pBits[1][i][j];  B = m\_Imagesrc.m\_pBits[0][i][j];  int minN = min(R, G); minN = min(minN, B); //RGB最小值  float divide = pow((R - G), 2) + (R - B)\*(G - B);  divide = sqrt(divide);  theta = 1.0 / 2 \* (2 \* R - G - B) / divide;  theta = acos(theta) / PI \* 180; //计算得到theta弧度值并转换为角度值  if (B <= G) H = int(theta);  else H = int(360 - theta); // 判断得到H值  H = H \* 255 / 360; // 将H值归一化到0-255  m\_Image\_h.m\_pBits[0][i][j] = m\_Image\_h.m\_pBits[1][i][j] = m\_Image\_h.m\_pBits[2][i][j] = H; // 赋值  S = (1 - 3.0 / (R + G + B)\*minN) \* 255; // 计算S值并归一化到0-255  m\_Image\_s.m\_pBits[0][i][j] = m\_Image\_s.m\_pBits[1][i][j] = m\_Image\_s.m\_pBits[2][i][j] = S; // 赋值  I = float(R + G + B) / 3; // 计算I值  m\_Image\_i.m\_pBits[0][i][j] = m\_Image\_i.m\_pBits[1][i][j] = m\_Image\_i.m\_pBits[2][i][j] = I;  }  }  m\_Image.flag = 4;  Invalidate(1); //强制调用ONDRAW函数，ONDRAW会绘制图像  } |

##### 实现结果展示

1、显示R、G、B分量

|  |
| --- |
| R分量 G分量 B分量 |

2、显示H、S、I分量

|  |
| --- |
| H分量 S分量 I分量 |

编程2：试验：一幅图像对RGB作图像均衡，转换到HSI 空间后，只对I分量作图像均衡，H、S分量保持不变，观察两种情况的效果。

##### 源代码及重点语句的注释

1、R、G、B上做均衡

|  |
| --- |
| void CImage\_ProcessingView::OnEqualrgb(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  m\_Imagesrc.Load(filename);  m\_Image.calcHistogram();// 计算rgb分量上的归一化直方图  float s[3][256] = { 0 };//均衡  int hist\_equal[3][256] = { 0 };//均衡后 RGB分量  for (int k = 0; k < 3; k++){  for (int i = 0; i < 256; i++){  for (int j = 0; j <= i; j++){  s[k][i] += 255 \* m\_Image.hist[k][j];  }  hist\_equal[k][i] = floor(s[k][i]); // 三个通道上的RGB对应的均衡化直方图  }  }  for (int k = 0; k < 3; k++){  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  int value = m\_Image.m\_pBits[k][i][j];  int new\_value = hist\_equal[k][value];  m\_Image.m\_pBits[k][i][j] = new\_value;  }// 分别将均衡化后的值计算后赋值给相应像素点  }  } // 在三个分量上做直方图均衡  m\_Image.flag = 1;  Invalidate(1);  } |

2、对I分量做直方图均衡

|  |
| --- |
| void CImage\_ProcessingView::OnEquali(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  m\_Imagesrc.Load(filename); //分别获取原图和HIS分量  m\_Image\_h.Load(filename);m\_Image\_s.Load(filename);m\_Image\_i.Load(filename);  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){  float theta, H, S, I;  int R, G, B;  R = m\_Imagesrc.m\_pBits[2][i][j];  G = m\_Imagesrc.m\_pBits[1][i][j];  B = m\_Imagesrc.m\_pBits[0][i][j]; // 获取原图的RGB三通道  int minN = min(R, G); minN = min(minN, B); //RGB最小值  float divide = pow((R - G), 2) + (R - B)\*(G - B);  divide = sqrt(divide);  theta = 1.0 / 2 \* (2 \* R - G - B) / divide;  theta = acos(theta) / PI \* 180; //计算得到theta弧度值并转换为角度值  if (B <= G) H = int(theta);  else H = int(360 - theta); // 判断得到H值  H = H \* 255 / 360; // 将H值归一化到0-255  m\_Image\_h.m\_pBits[0][i][j] = m\_Image\_h.m\_pBits[1][i][j] = m\_Image\_h.m\_pBits[2][i][j] = H; // 赋值  S = (1 - 3.0 / (R + G + B)\*minN) \* 255; // 计算S值并归一化到0-255  m\_Image\_s.m\_pBits[0][i][j] = m\_Image\_s.m\_pBits[1][i][j] = m\_Image\_s.m\_pBits[2][i][j] = S; // 赋值  I = float(R + G + B) / 3; // 计算I值  m\_Image\_i.m\_pBits[0][i][j] = m\_Image\_i.m\_pBits[1][i][j] = m\_Image\_i.m\_pBits[2][i][j] = I;  }  }  m\_Image\_i.calcHistogram();//计算I分量的归一化直方图  float s[256] = { 0 };//均衡  int hist\_equal[256] = { 0 };  for (int i = 0; i < 256; i++) {  for (int j = 0; j <= i; j++) {  s[i] += 255 \* m\_Image\_i.hist[0][j];  }  hist\_equal[i] = floor(s[i]); //对I分量做均衡化  }  int value, new\_value;  for (int i = 0; i < h; i++) {  for (int j = 0; j < w; j++) {  value = m\_Image\_i.m\_pBits[0][i][j];  new\_value = hist\_equal[value];  m\_Image\_i.m\_pBits[0][i][j] = new\_value;  m\_Image\_i.m\_pBits[1][i][j] = new\_value;  m\_Image\_i.m\_pBits[2][i][j] = new\_value; // 得到均衡化后的接货  }  }  for (int i = 0; i < h; i++){// 做HIS到RGB通道的转化以方便显示  for (int j = 0; j < w; j++){  float H, S, I, R, G, B;  H = m\_Image\_h.m\_pBits[0][i][j] \* 360.0 / 255.0;  S = m\_Image\_s.m\_pBits[0][i][j] / 255.0;  I = m\_Image\_i.m\_pBits[0][i][j];  if (H >= 0 && H < 120) {  B = I \* (1.0 - S);  H = H \* PI / 180;  R = I \* (1.0 + S \* cos(H) / cos(PI / 3 - H));  G = 3 \* I - (R + B);  }  else if (H >= 120 && H < 240){  H = H - 120;  R = I \* (1.0 - S);  H = H \* PI / 180;  G = I \* (1.0 + S \* cos(H) / cos(PI / 3 - H));  B = 3 \* I - (R + G);  }  else{  H = H - 240;  G = I \* (1.0 - S);  H = H \* PI / 180;  B = I \* (1.0 + S \* cos(H) / cos(PI / 3 - H));  R = 3 \* I - (G + B);  }  m\_Image.m\_pBits[0][i][j] = B > 255 ? 255 : B < 0 ? 0 : B;  m\_Image.m\_pBits[1][i][j] = G > 255 ? 255 : G < 0 ? 0 : G;  m\_Image.m\_pBits[2][i][j] = R > 255 ? 255 : R < 0 ? 0 : R;  }  }  m\_Image.flag = 1;  Invalidate(1);  } |

##### 实现结果展示

|  |
| --- |
| 对RGB三通道做均衡后结果图和原图的对比 |
| 对I通道做均衡后结果图和原图的对比 |

编程3：实现对一幅彩色图像的分割：人工画框定义一个区域，以该区域的彩色分量平均值为中心，设定一个区域距离值，将图像分割为二值图。

##### 流程图

|  |
| --- |
|  |

##### 源代码及重点语句的注释

|  |
| --- |
| void CImage\_ProcessingView::OnColorsegment(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获得图像的宽度  int h = m\_Image.GetHeight();//获得图像的高度  m\_Imagesrc.Load(filename);  float R, G, B;  int D, count;  R = 0;G = 0;B = 0;count = 0;D = 30;  PictureDialog dlg(this); // 建立一个对话框，标定选取的区域位置  if (IDOK == dlg.DoModal()) {  CPoint start, end;  start = dlg.m\_startPoint;  end = dlg.m\_endPoint; // 获取标定区域的起点和终点  float size;  size = (end.x - start.x)\*(end.y - start.y); // 计算所在区域的大小  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){// 将区域在图中标出  if (i == start.y || i == end.y) {  if (j > start.x && j <= end.x){  m\_Image.m\_pBits[0][i][j] = 0;  m\_Image.m\_pBits[1][i][j] = 0;  m\_Image.m\_pBits[2][i][j] = 0;  }  }  if (j == start.x || j == end.x) {  if (i > start.y && i <= end.y) {  m\_Image.m\_pBits[0][i][j] = 0;  m\_Image.m\_pBits[1][i][j] = 0;  m\_Image.m\_pBits[2][i][j] = 0;  }  }// 计算标定区域对应的彩色分量像素均值  if (i > start.y && i <= end.y && j > start.x && j <= end.x) {  R += m\_Imagesrc.m\_pBits[2][i][j] / size;  G += m\_Imagesrc.m\_pBits[1][i][j] / size;  B += m\_Imagesrc.m\_pBits[0][i][j] / size;  }  }  }  }  for (int i = 0; i < h; i++){  for (int j = 0; j < w; j++){// 计算像素值是否在均值范围内（球模型）  int r = m\_Imagesrc.m\_pBits[2][i][j];  int g = m\_Imagesrc.m\_pBits[1][i][j];  int b = m\_Imagesrc.m\_pBits[0][i][j];  float dis;  dis = pow(r - R, 2) + pow(g - G, 2) + pow(b - B, 2);  if (dis <= pow(D, 2)){ // 在范围内则标1  m\_Imagesrc.m\_pBits[2][i][j] = m\_Imagesrc.m\_pBits[1][i][j] = m\_Imagesrc.m\_pBits[0][i][j] = 255;  }  else{// 不在范围内则标0  m\_Imagesrc.m\_pBits[2][i][j] = m\_Imagesrc.m\_pBits[1][i][j] = m\_Imagesrc.m\_pBits[0][i][j] = 0;  }  }  }  m\_Image.flag = 1;  Invalidate(1);  } |

##### 实现结果展示

|  |
| --- |
| 区域标定对话框 |
| 标定区域和分割后结果 |

### 第十章

编程1：实现霍夫变换检测直线。

##### 流程图

|  |
| --- |
|  |

##### 源代码及重点语句的注释

1、霍夫变换函数

|  |
| --- |
| // 霍尔变换直线检测  #include <vector>  #include <utility>  #include "common.hpp"  #ifndef HOUGH\_HOUGH\_HPP  #define HOUGH\_HOUGH\_HPP  // 二值化函数  void im2bw(int\*\* src, int\*\* dst, int row, int col, int threshold=128, int maxval=255, int minval=0){  for (int i = 0; i < row; i++)  for (int j = 0; j < col; j++)  dst[i][j] = src[i][j] > threshold ? maxval : minval;  }  // 卷积用于做sobel边缘检测 其中，src为待卷积图，dst为卷积后图，kernel为卷积核  void conv2d3x3(int\*\* src, int\*\*dst, int row, int col, double \*\* kernel){  for (int i = 0; i < row; i++){  for (int j = 0; j < col; j++){  double t = 0;  for (int x = -1; x < 2; x ++){  for (int y = -1; y < 2; y ++){  if (i + y < 0 || i + y >= row || j + x < 0 || j + x >= col){  continue;  }  t += src[i + y][j + x] \* kernel[y + 1][x + 1];  }  }  dst[i][j] = t;  }  }  }  // sobel边缘检测  void sobel(int\*\* src, int\*\* dst, int row, int col){  double\*\* k1 = alloc2d<double>(3, 3);  double\*\* k2 = alloc2d<double>(3, 3);  k1[0][0] = -1; k1[0][1] = 0; k1[0][2] = 1;  k1[1][0] = -2; k1[1][1] = 0; k1[1][2] = 2;  k1[2][0] = -1; k1[2][1] = 0; k1[2][2] = 1; // sobel算子1  k2[0][0] = -1; k2[0][1] = -2; k2[0][2] = -1;  k2[1][0] = 0; k2[1][1] = 0; k2[1][2] = 0;  k2[2][0] = 1; k2[2][1] = 2; k2[2][2] = 1; // sobel算子2  int\*\* xx = alloc2d<int>(row, col);  int\*\* yy = alloc2d<int>(row, col); // 存放x方向和y方向上的卷积结果  conv2d3x3(src, xx, row, col, k1);  conv2d3x3(src, yy, row, col, k2);  for (int i = 1; i < row - 1; i++) {  for (int j = 1; j < col - 1; j++) {// 求最终的结果  dst[i][j] = sqrt(xx[i][j] \* xx[i][j] + yy[i][j] \* yy[i][j]);  }  }  free2d(k1, 3, 3);  free2d(k2, 3, 3); // 释放以免占据内存  }  // 计算hough  void drawcurve(int\*\* hough\_space, int x, int y){  for (int theta\_id = 0; theta\_id < 1000; theta\_id++){  double theta = PI \* theta\_id / 999.;  double r = x \* cos(theta) + y \* sin(theta);  int yy = 0.999 \* r + 499.5;  if (yy < 0) yy = 0;  if (yy >= 1000) yy = 999;  hough\_space[yy][theta\_id] += 1;  }  }  // hough变换函数  std::vector<std::pair<double, double>> Hough(int\*\* src, int row, int col, int val, int threshold=300){ // 其中val为取点条件，threshold为判断该点为同一直线的交点个数阈值  int\*\* hough\_space = alloc2d<int>(1000, 1000);  for(int i = 0; i < row; i++){  for (int j = 0; j < col; j ++){  if (src[i][j] == val){  drawcurve(hough\_space, j, i);  }  }  }  std::vector<std::pair<std::pair<double, double>, int>> result; //存放结果k和b值  for (int i = 0; i < 1000; i++) {  for (int j = 0; j < 1000; j++) {  if (hough\_space[i][j] > threshold){  std::pair<double, double> r\_theta\_pair((i - 499.5) / 0.999, PI \* j / 999.);  result.push\_back(std::pair<std::pair<double, double>, int>(r\_theta\_pair, hough\_space[i][j]));  }  }  }  // NMS非极大值抑制去掉冗余线  for (int i = 0; i < result.size(); i++){  std::pair<std::pair<double, double>, int> &res1 = result[i];  for (int j = i + 1; j < result.size(); j++){  std::pair<std::pair<double, double>, int> &res2 = result[j];  if (res2.second == 0) continue;  if (abs(res1.first.first - res2.first.first)<40 && abs(res1.first.second - res2.first.second) < 0.12){ // 设定阈值  if (res1.second > res2.second){  res2.second = 0;  } else{  res1.second = 0;  break;  }  }  }  }  std::vector<std::pair<double, double>> final\_result;  for(int i = 0; i < result.size(); i++){  if (result[i].second > 0){  double k = - 1. / tan(result[i].first.second);  double b = result[i].first.first / sin(result[i].first.second);  final\_result.push\_back(std::pair<double, double>(k, b));  }// 做一次转换将结果转换为k和b表示的截距式直线  }  free2d<int>(hough\_space, 1000, 1000);  return final\_result;  }  void drawline(int\*\* src, int row, int col, std::vector<std::pair<double, double>> lines){  for(int n = 0; n < lines.size(); n++){  double k = lines[n].first;  double b = lines[n].second;  double last = -1;  for(int j = 0; j < col; j++){  int y = k\*j + b;  if (y >= 0 && y < row){  if (last == -1){  src[y][j] = 128;  last = y;  }  int s = y > last ? last : y;  int l = y > last ? y : last;  for (int i = s; i <=l; i++){  src[i][j] = 128;  }  last = y;  }  }  }  }  std::vector<std::pair<double, double>> line\_det(int\*\* imgarr, int row, int col){  int\*\* bin\_img = alloc2d<int>(row, col);  sobel(imgarr, bin\_img, row, col); // sobel边缘检测  im2bw(bin\_img, bin\_img, row, col, 128); // 二值化  std::vector<std::pair<double, double>>lines = Hough(bin\_img, row, col, 255);  free2d(bin\_img, row, col); //hough变换  return lines;  } |

2、主函数

|  |
| --- |
| void CImage\_ProcessingView::OnHough(){  // TODO: 在此添加命令处理程序代码  if (m\_Image.IsNull()) return;//判断图像是否为空，对空图像操作会出现未知的错误  int w = m\_Image.GetWidth();//获取高度和宽度  int h = m\_Image.GetHeight();  if (!m\_Imagesrc.IsNull()) m\_Imagesrc.Destroy();  m\_Imagesrc.Load(filename);  int bits = m\_Image.GetBPP();  if (bits == 24 || bits == 32) {  for (int i = 0; i < h; i++) {  for (int j = 0; j < w; j++) {  int ave = 0.1140 \*m\_Image.m\_pBits[0][i][j] + 0.5870 \*m\_Image.m\_pBits[1][i][j] + 0.2989 \*m\_Image.m\_pBits[2][i][j];  m\_Image.m\_pBits[0][i][j] = ave;  m\_Image.m\_pBits[1][i][j] = ave;  m\_Image.m\_pBits[2][i][j] = ave;  }  }  }  int \*\*imgarr = alloc2d<int>(h, w);  for (int i = 0; i < h; i++) {  for (int j = 0; j < w; j++) {  imgarr[i][j] = m\_Image.m\_pBits[0][i][j];  }  }  int & H = h;  int & W = w;  std::vector<std::pair<double, double>> lines = line\_det(imgarr, H, W);  int\*\* detected = alloc2d<int>(H, W);  drawline(detected, H, W, lines);  for (int row = 0; row < H; row++) {  for (int col = 0; col < W; col++) {  if (detected[row][col] > 0) {  m\_Image.m\_pBits[0][row][col] = 0;  m\_Image.m\_pBits[1][row][col] = 0;  m\_Image.m\_pBits[2][row][col] = 255; //在该空间上画线  }  }  }  m\_Image.flag = 0;  Invalidate(1);  } |

##### 实现结果展示

|  |
| --- |
| 霍夫变换结果图 |

编程2：实现基本全局阈值算法分割图像。

##### 源代码及重点语句的注释

|  |
| --- |
| // 基本全局阈值法  #ifndef HOUGH\_BGT\_HPP  #define HOUGH\_BGT\_HPP  void bgt(int\*\* src, int\*\* dst, int row, int col){  double threshold = 128; // 设置阈值为128  double old = 0;  while(true){  double mean1 = 0;  double mean2 = 0;  double count1 = 0;  double count2 = 0; // 设定全局阈值中的均值及其所在阈值两边的像素数量  for(int i = 0; i < row; i++){  for (int j = 0; j < col; j++){  if (src[i][j] < threshold){  mean1 += src[i][j];  count1++;  }else{  mean2 += src[i][j];  count2++;  }  }  }  mean1 /= count1;  mean2 /= count2;  old = threshold;  threshold = 0.5 \* (mean1 + mean2); // 更新新的阈值  if(old - threshold < 1 && old - threshold > -1)  break; // 如果阈值不变，那么跳出循环  }  for(int i = 0; i < row; i++){  for (int j = 0; j < col; j++){  if (src[i][j] < threshold){  dst[i][j] = 0;  }else{  dst[i][j] = 255;  }// 根据新的阈值更新图像  }  }  } |

##### 实现结果展示

|  |
| --- |
|  |
|  |

分割后的图和原图对比

编程3：实现最大方差法分割图像

##### 源代码及重点语句的注释

|  |
| --- |
| // 最大方差法  #ifndef HOUGH\_OTSU\_HPP  #define HOUGH\_OTSU\_HPP  void otsu(int\*\* src, int\*\* dst, int row, int col){  int T=0;  double varValue=0;  double w0=0;  double w1=0;  double u0=0;  double u1=0; // 初始化全局方差和部分方差  double Histogram[256]={0};  for(int i=0;i<row;i++){  for(int j=0;j<col;j++){  Histogram[src[i][j]]++;  }// 计算统计直方图  }  for(int i=0;i<255;i++){//根据计算公式得到其中的threshold T  w1=0;  u1=0;  w0=0;  u0=0;  for(int j=0;j<=i;j++){  w1 += Histogram[j];  u1 += j \* Histogram[j];  }  if(w1==0) continue;  u1=u1 / w1;  w1=w1 / row / col;  for(int k=i+1;k<255;k++){  w0+=Histogram[k];  u0+=k\*Histogram[k];  }  if(w0==0) break;  u0=u0/w0;  w0=w0/row/col;  double varValueI=w0\*w1\*(u1-u0)\*(u1-u0);  if(varValue<varValueI){  varValue=varValueI;  T=i;  }  }  for(int i = 0; i < row; i++){// 根据T得到分割后的图  for (int j = 0; j < col; j++){  if (src[i][j] < T){  dst[i][j] = 0;  }else{  dst[i][j] = 255;  }  }  }  } |

##### 实现结果展示

|  |
| --- |
|  |
|  |

分割后的图和原图对比

# 附注

关键代码附在文件中，最终整个工程文件GitHub地址：

<https://github.com/JZDBB/image_processing.git>