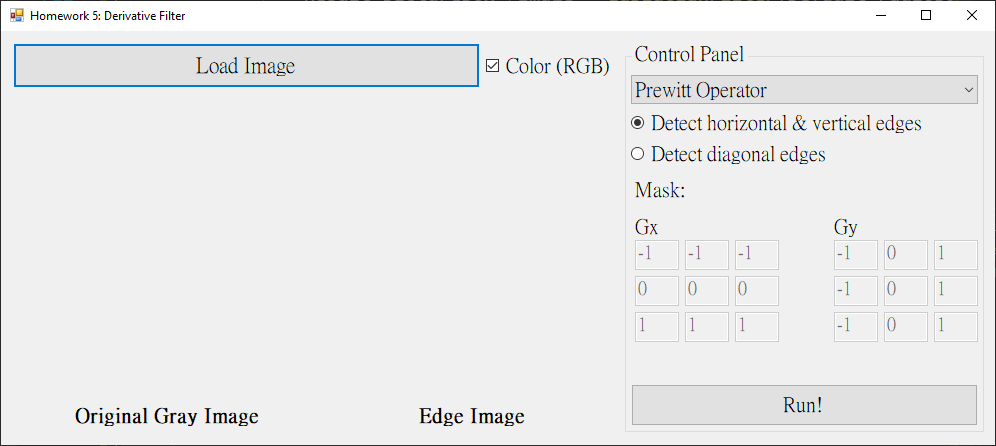
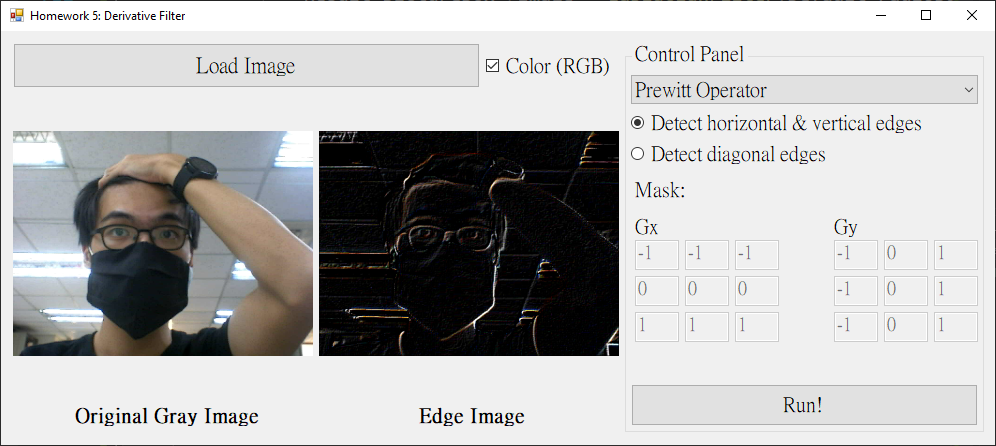
四資工三甲 C108151131 賴冠綸

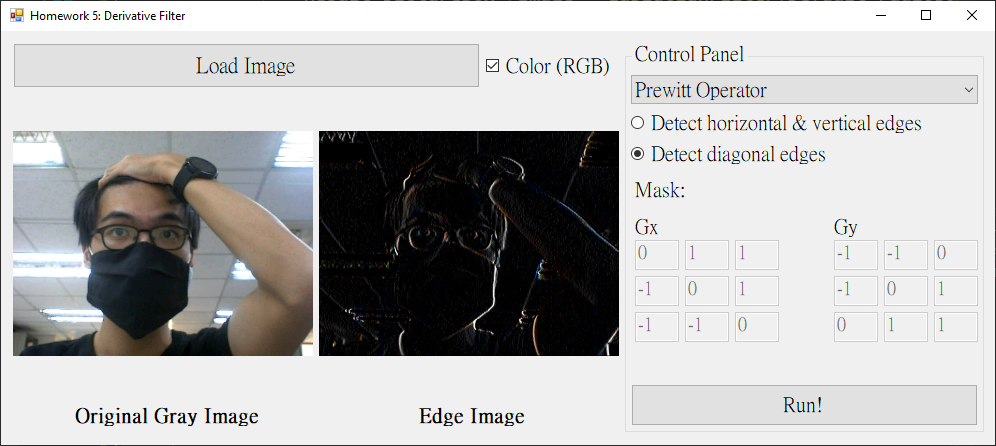
1. 程式執行畫面
   * 1. 程式起始畫面



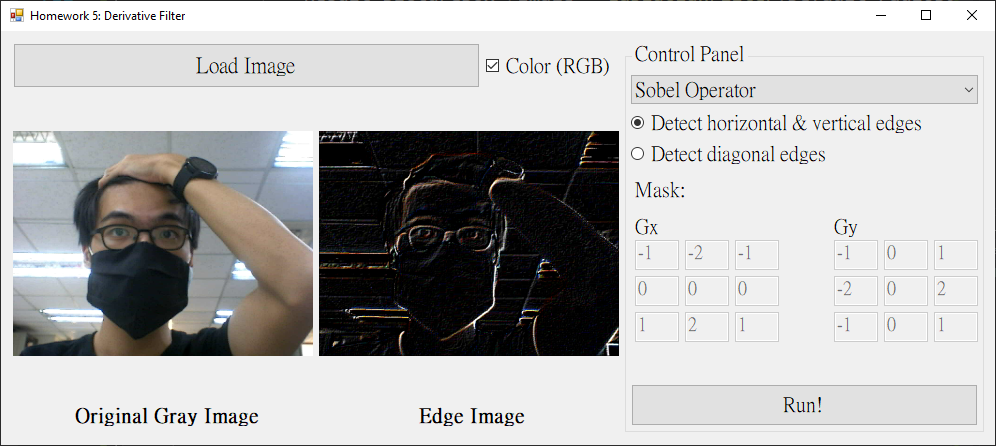
* + 1. Prewitt Operator
       1. Detect horizontal & vertical edges



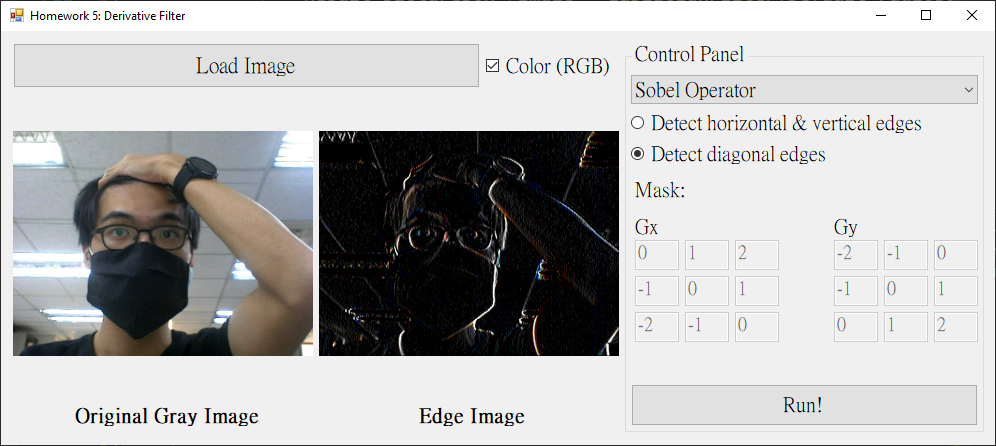
* + - 1. Detect diagonal edges



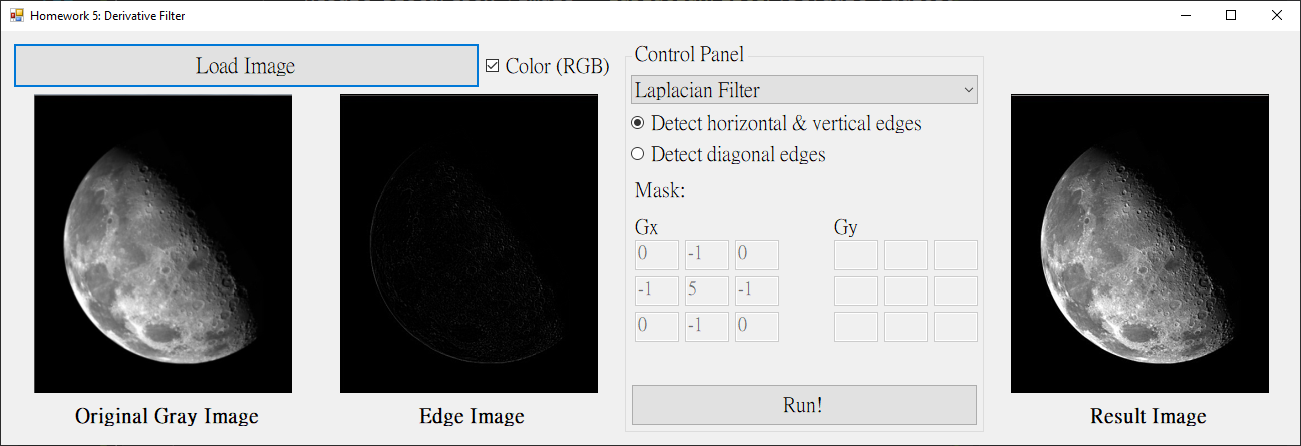
* + 1. Sobel Operator
       1. Detect horizontal & vertical edges



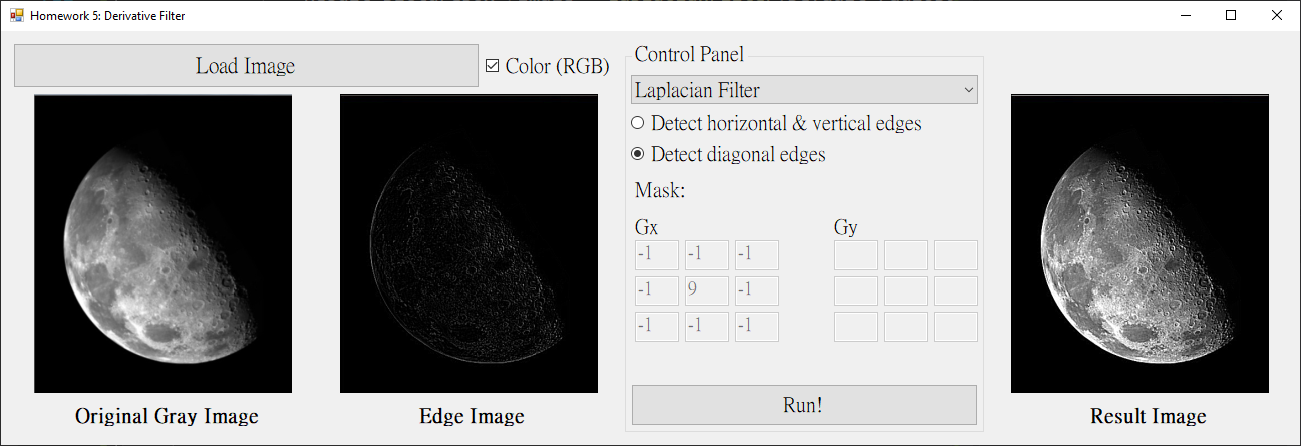
* + - 1. Detect diagonal edges



* + 1. Laplacian Filter（用自己的人頭照不明顯，因此找了個範例圖片
       1. Detect horizontal & vertical edges

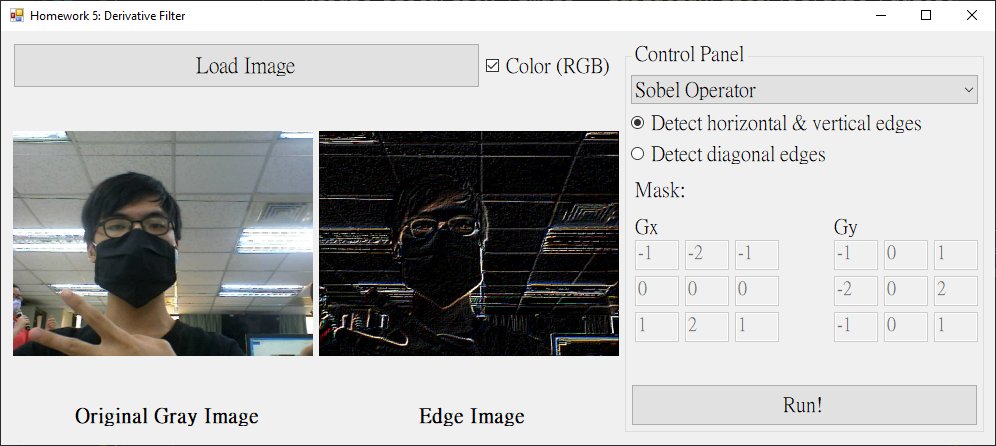


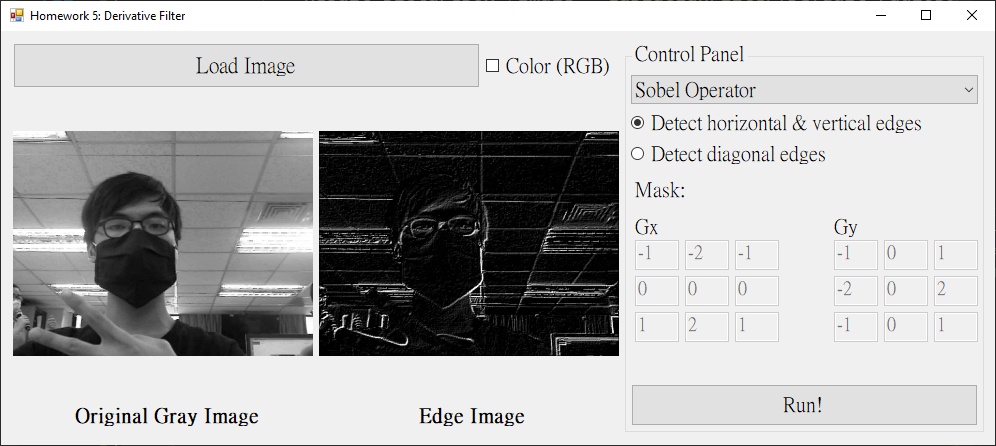
* + - 1. Detect diagonal edges

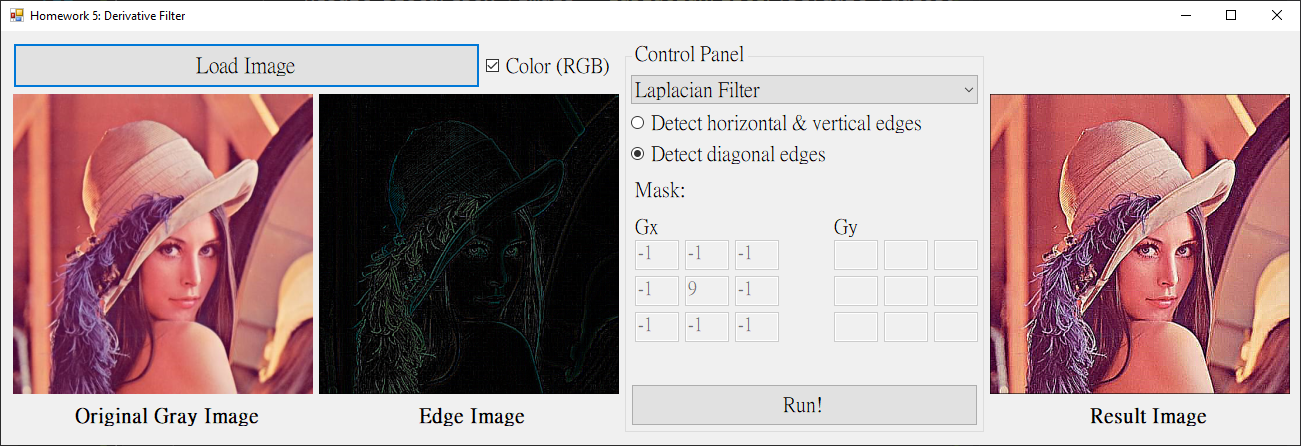


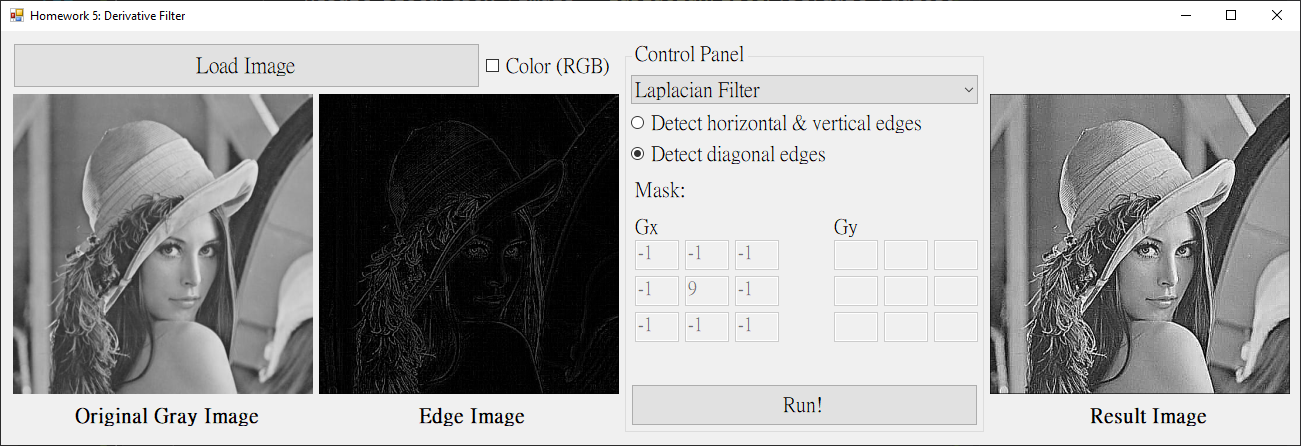
* + 1. 此程式全彩與黑白圖片皆可處理（理論上）

我獨立了一個 checkBox，可以決定該圖片是否要以全彩處理。









1. 程式碼（僅附主要程式碼）
   * 1. Prewitt Operator & Sobel Operator（我寫在同一個Function）

|  |
| --- |
| private: Bitmap^ GradientOperator(int gx[][3], int gy[][3], Bitmap^ gray)  {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadOnly, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadWrite, result->PixelFormat);  int grayByteskip = grayBD->Stride - grayBD->Width \* 3;  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  const int maskSize = 3;  const int blank = maskSize / 2;  /\* copy origin image \*/  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  resultPtr[0] = grayPtr[0];  resultPtr[1] = grayPtr[1];  resultPtr[2] = grayPtr[2];  grayPtr += 3;  resultPtr += 3;  }  grayPtr += grayByteskip;  resultPtr += resultByteskip;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  grayPtr = (Byte\*)((void\*)grayBD->Scan0);  resultPtr = (Byte\*)((void\*)resultBD->Scan0);  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  int bSum = 0, gSum = 0, rSum = 0;  // Avoid out of the range  if (x > 0 && x < grayBD->Width - blank  && y > 0 && y < grayBD->Height - blank)  {  for (int i = -blank, index\_i = 0; i <= blank; i++, index\_i++)  {  for (int j = -blank, index\_j = 0; j <= blank; j++, index\_j++)  {  Byte\* ptr = grayPtr + (x + j) \* 3 + (y + i) \* grayBD->Stride;  bSum += ptr[0] \* gx[index\_i][index\_j]  + ptr[0] \* gy[index\_i][index\_j];  gSum += ptr[1] \* gx[index\_i][index\_j]  + ptr[1] \* gy[index\_i][index\_j];  rSum += ptr[2] \* gx[index\_i][index\_j]  + ptr[2] \* gy[index\_i][index\_j];  }  }  if (bSum < 0) bSum = 0;  else if (bSum > 255) bSum = 255;  if (gSum < 0) gSum = 0;  else if (gSum > 255) gSum = 255;  if (rSum < 0) rSum = 0;  else if (rSum > 255) rSum = 255;  }  Byte\* ptr = resultPtr + x \* 3 + y \* resultBD->Stride;  ptr[0] = bSum;  ptr[1] = gSum;  ptr[2] = rSum;  }  }  gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |

* + 1. Laplacian Filter

|  |
| --- |
| private: Bitmap^ LaplacianFilter(int mask[][3], Bitmap^ gray)  {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadOnly, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadWrite, result->PixelFormat);  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  const int maskSize = 3;  const int blank = maskSize / 2;  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  double bSum = 0, gSum = 0, rSum = 0;  // Avoid out of the range  if (x > 0 && x < grayBD->Width - blank  && y > 0 && y < grayBD->Height - blank)  {  for (int i = -blank, index\_i = 0; i <= blank; i++, index\_i++)  {  for (int j = -blank, index\_j = 0; j <= blank; j++, index\_j++)  {  Byte\* ptr = grayPtr + (x + j) \* 3 + (y + i) \* grayBD->Stride;  bSum += ptr[0] \* mask[index\_i][index\_j];  gSum += ptr[1] \* mask[index\_i][index\_j];  rSum += ptr[2] \* mask[index\_i][index\_j];  }  }  if (bSum < 0) bSum = 0;  else if (bSum > 255) bSum = 255;  if (gSum < 0) gSum = 0;  else if (gSum > 255) gSum = 255;  if (rSum < 0) rSum = 0;  else if (rSum > 255) rSum = 255;  }  Byte\* ptr = resultPtr + x \* 3 + y \* resultBD->Stride;  ptr[0] = bSum;  ptr[1] = gSum;  ptr[2] = rSum;  }  }  /\* get border image \*/  Bitmap^ edgeImg = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ edgeImgBD = edgeImg->LockBits(Rectangle(0, 0, edgeImg->Width  , edgeImg->Height), ImageLockMode::ReadOnly, edgeImg->PixelFormat);  Byte\* edgeImgPtr = (Byte\*)((void\*)edgeImgBD->Scan0);    int edgeImgByteskip = edgeImgBD->Stride - edgeImgBD->Width \* 3;  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  int grayByteskip = grayBD->Stride - grayBD->Width \* 3;    for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  int b = resultPtr[0] - grayPtr[0];  int g = resultPtr[1] - grayPtr[1];  int r = resultPtr[2] - grayPtr[2];  if (b < 0) edgeImgPtr[0] = 0;  else if (b > 255) edgeImgPtr[0] = 255;  else edgeImgPtr[0] = b;  if (g < 0) edgeImgPtr[1] = 0;  else if (g > 255) edgeImgPtr[1] = 255;  else edgeImgPtr[1] = g;  if (r < 0) edgeImgPtr[2] = 0;  else if (r > 255) edgeImgPtr[2] = 255;  else edgeImgPtr[2] = r;  edgeImgPtr += 3;  resultPtr += 3;  grayPtr += 3;  }  edgeImgPtr += edgeImgByteskip;  resultPtr += resultByteskip;  grayPtr += grayByteskip;  }  edgeImg->UnlockBits(edgeImgBD);  Edge\_pictureBox->Image = edgeImg;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |