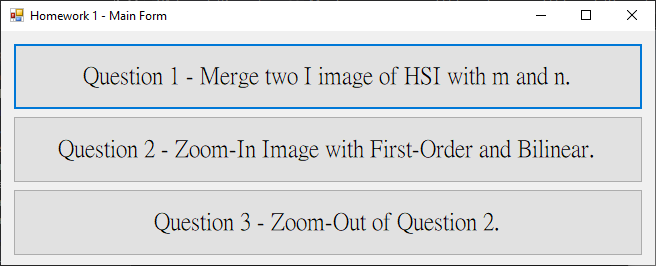
四資工三甲 C108151131 賴冠綸

1. 程式執行畫面：
   1. 主選單（用以選擇第X小題）



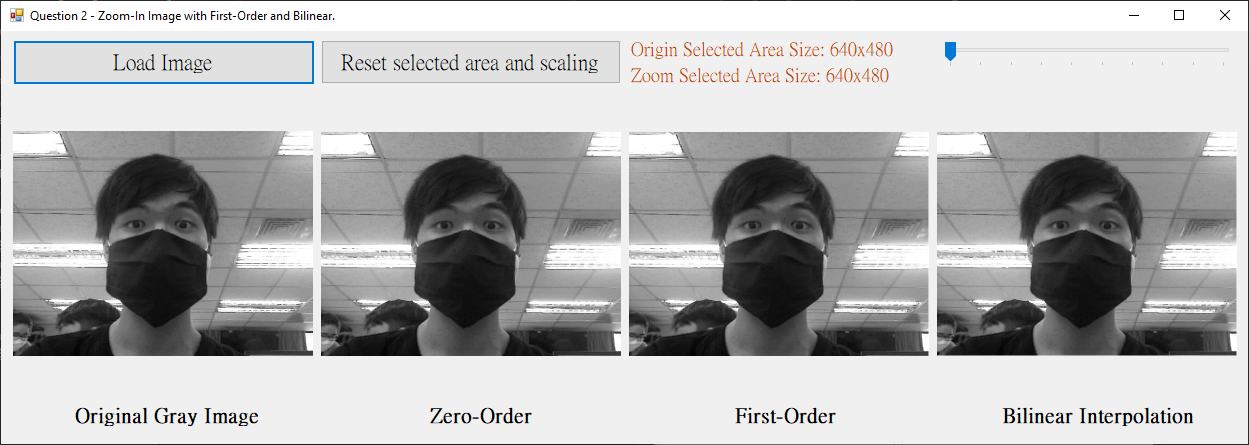
* 1. 第一小題
     1. m = 0.1, n = 0.9



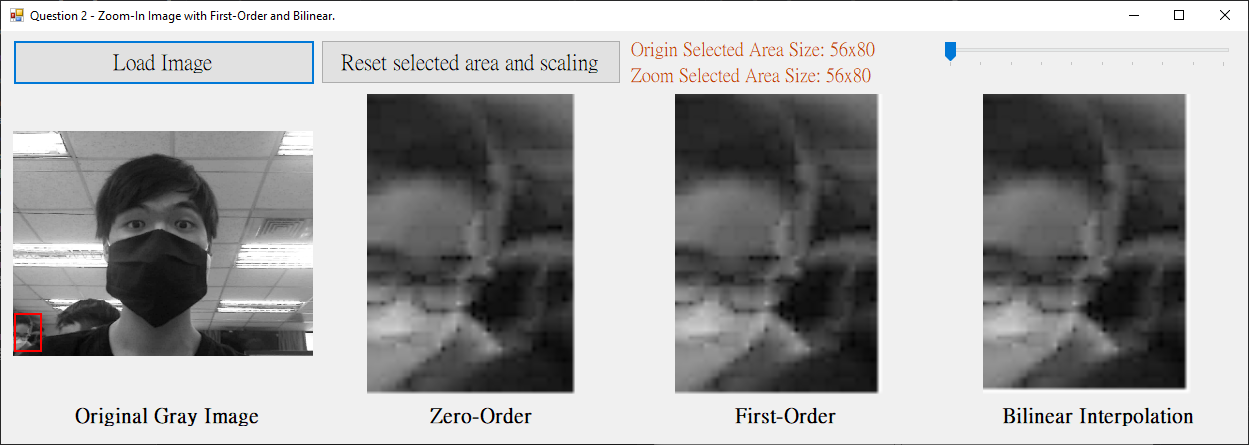
* + 1. m = 0.8, n = 0.2

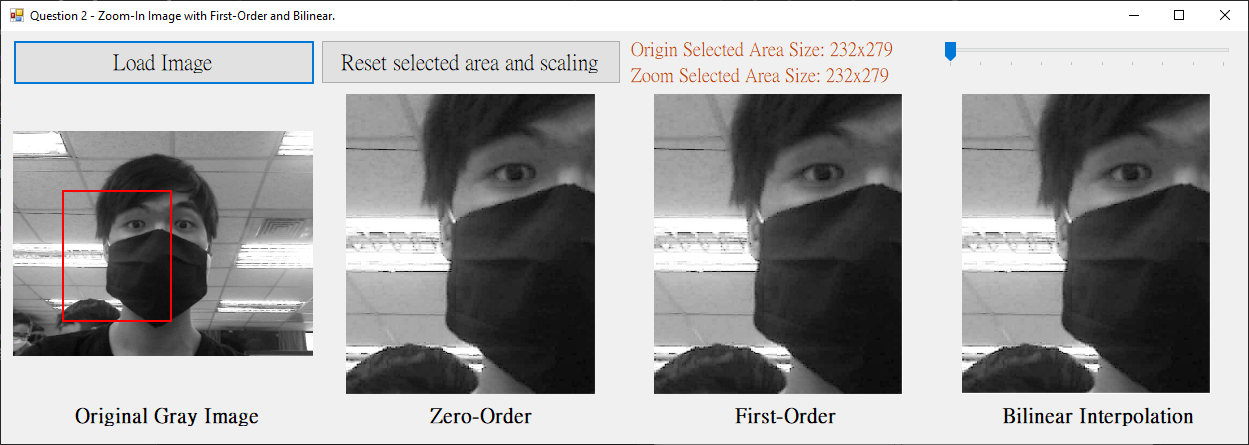


* 1. 第二小題 ─ 放大（除了First-Order與Bilinear，順便也比較了Zero-Order）
     1. 載入圖片後



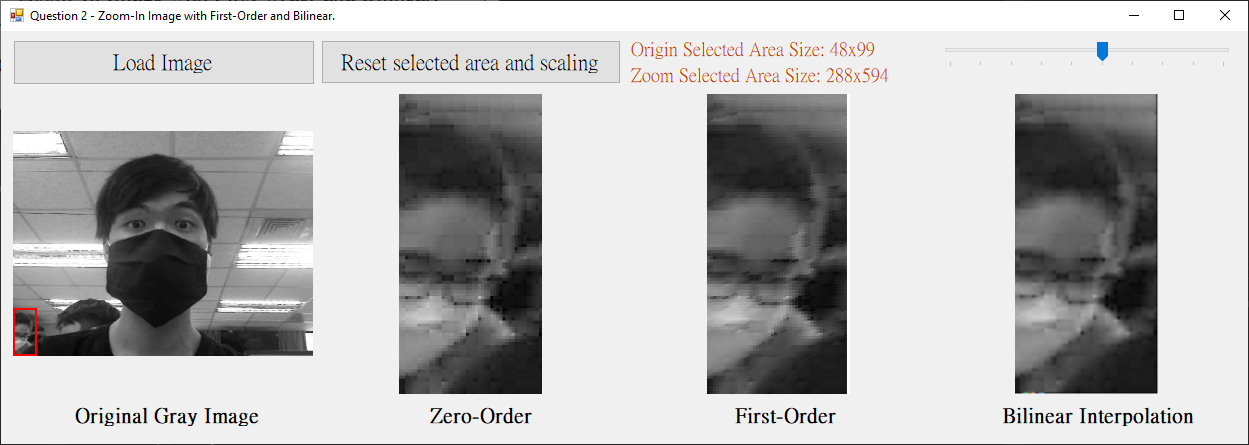
* + 1. 可以任意圈選範圍區塊（即時更新結果）



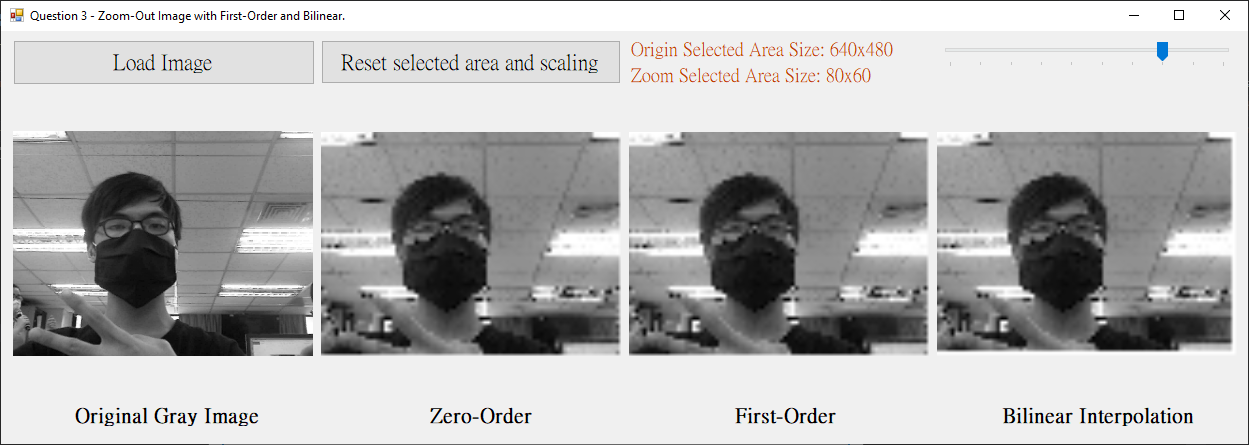


* + 1. 右上角的槓桿可以調整**放大**倍數

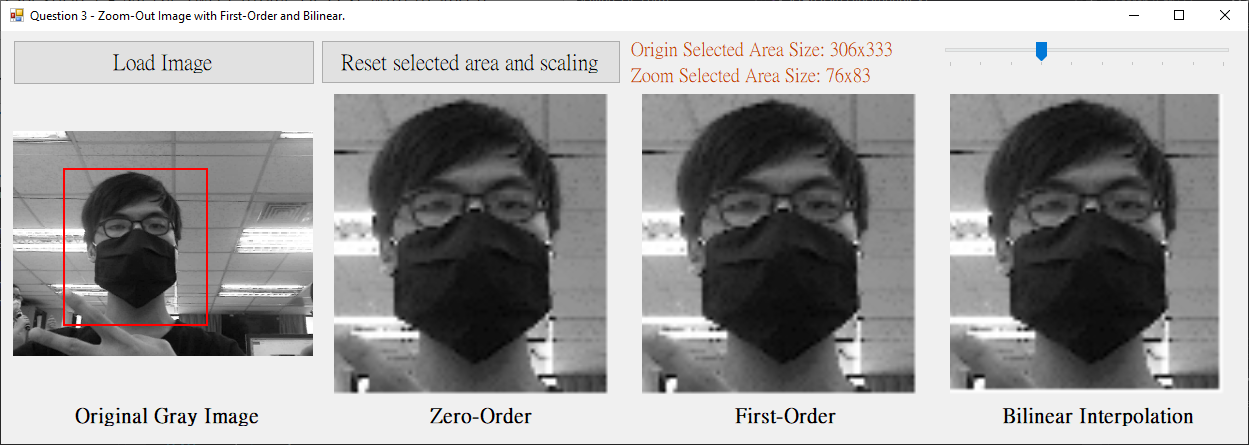
圈選範圍大小及放大後的大小皆會標示（棕色部分）



* 1. 第三小題 ─ 縮小
     1. 右上角的槓桿可以調整**縮小**倍數



* + 1. 同第二題，一樣可以圈選範圍



* + 1. 大部分功能皆與第二題相同

1. 程式碼（僅附主要程式碼）
   1. 第一小題
      1. 轉灰階（我將其包裝成一個Function，使其能重複被呼叫）

|  |
| --- |
| private: Bitmap^ Transfer(Bitmap^ rgb\_img)  {  Bitmap^ gray\_img = gcnew Bitmap(rgb\_img->Width, rgb\_img->Height, rgb\_img->PixelFormat);  BitmapData^ bd\_rgb = rgb\_img->LockBits(Rectangle(0, 0, rgb\_img->Width, rgb\_img->Height), ImageLockMode::ReadOnly, rgb\_img->PixelFormat);  BitmapData^ bd\_gray = gray\_img->LockBits(Rectangle(0, 0, gray\_img->Width, gray\_img->Height), ImageLockMode::ReadWrite, gray\_img->PixelFormat);  /\* Image maybe have unknown information \*/  int rgb\_byteskip = bd\_rgb->Stride - bd\_rgb->Width \* 3;  int gray\_byteskip = bd\_gray->Stride - bd\_gray->Width \* 3;  Byte\* rgb\_ptr = (Byte\*)((void\*)bd\_rgb->Scan0);  Byte\* gray\_ptr = (Byte\*)((void\*)bd\_gray->Scan0);  for (int y = 0; y < bd\_rgb->Height; y++)  {  for (int x = 0; x < bd\_rgb->Width; x++)  {  int pixel = ((int)rgb\_ptr[0] + (int)rgb\_ptr[1] + (int)rgb\_ptr[2]) / 3; // I image    gray\_ptr[0] = gray\_ptr[1] = gray\_ptr[2] = pixel;  // jump to next pixel  rgb\_ptr += 3;  gray\_ptr += 3;  }  // throw unknown inforamtion away  rgb\_ptr += rgb\_byteskip;  gray\_ptr += gray\_byteskip;  }  rgb\_img->UnlockBits(bd\_rgb);  gray\_img->UnlockBits(bd\_gray);  return gray\_img;  } |

* + 1. 獲取合成影像F = mI1 + nI2

|  |
| --- |
| private: System::Void GetFImgBtn\_Click(System::Object^ sender, System::EventArgs^ e)  {  float m; // n = 1 - m  switch (MN\_comboBox->SelectedIndex)  {  case 0: m = 0.1; break;  case 1: m = 0.2; break;  case 2: m = 0.3; break;  case 3: m = 0.4; break;  case 4: m = 0.5; break;  case 5: m = 0.6; break;  case 6: m = 0.7; break;  case 7: m = 0.8; break;  case 8: m = 0.9; break;  default: return; // if haven't select  }  Bitmap^ I1\_img = (Bitmap^)I1\_Img\_pictureBox->Image;  Bitmap^ I2\_img = (Bitmap^)I2\_Img\_pictureBox->Image;  Bitmap^ result\_img = gcnew Bitmap(I1\_img->Width, I1\_img->Height, I1\_img->PixelFormat);  BitmapData^ bd\_I1 = I1\_img->LockBits(Rectangle(0, 0, I1\_img->Width, I1\_img->Height)  , ImageLockMode::ReadOnly, I1\_img->PixelFormat);  BitmapData^ bd\_I2 = I2\_img->LockBits(Rectangle(0, 0, I2\_img->Width, I2\_img->Height)  , ImageLockMode::ReadOnly, I2\_img->PixelFormat);  BitmapData^ bd\_result = result\_img->LockBits(Rectangle(0, 0, result\_img->Width  , result\_img->Height), ImageLockMode::ReadWrite, result\_img->PixelFormat);  int I1\_byteskip = bd\_I1->Stride - bd\_I1->Width \* 3;  int I2\_byteskip = bd\_I2->Stride - bd\_I2->Width \* 3;  int result\_byteskip = bd\_result->Stride - bd\_result->Width \* 3;  Byte\* I1\_ptr = (Byte\*)((void\*)bd\_I1->Scan0);  Byte\* I2\_ptr = (Byte\*)((void\*)bd\_I2->Scan0);  Byte\* result\_ptr = (Byte\*)((void\*)bd\_result->Scan0);  for (int y = 0; y < bd\_I1->Height; y++)  {  for (int x = 0; x < bd\_I1->Width; x++)  {  // F = mI1 + nI2  result\_ptr[0] = m \* (int)I1\_ptr[0] + (1.f - m) \* (int)I2\_ptr[0];  result\_ptr[1] = m \* (int)I1\_ptr[1] + (1.f - m) \* (int)I2\_ptr[1];  result\_ptr[2] = m \* (int)I1\_ptr[2] + (1.f - m) \* (int)I2\_ptr[2];  I1\_ptr += 3;  I2\_ptr += 3;  result\_ptr += 3;  }  I1\_ptr += I1\_byteskip;  I2\_ptr += I2\_byteskip;  result\_ptr += result\_byteskip;  }  I1\_img->UnlockBits(bd\_I1);  I2\_img->UnlockBits(bd\_I2);  result\_img->UnlockBits(bd\_result);  Result\_pictureBox->Image = result\_img;  F\_label->Visible = true;  } |

* 1. 第二小題
     1. Zero-Order

|  |
| --- |
| private: Bitmap^ ZoomInImg\_withZeroOrder(double k)  {  Bitmap^ selectedArea = GetSelectedArea();  Bitmap^ output = gcnew Bitmap(Math::Abs(endPos\_real->X - startPos\_real->X) \* k  , Math::Abs(endPos\_real->Y - startPos\_real->Y) \* k , selectedArea->PixelFormat);  BitmapData^ selectedAreaBD = selectedArea->LockBits(Rectangle(0, 0, selectedArea->Width, selectedArea->Height), ImageLockMode::ReadOnly, selectedArea->PixelFormat);  BitmapData^ outputBD = output->LockBits(Rectangle(0, 0, output->Width, output->Height), ImageLockMode::ReadWrite, output->PixelFormat);  int selectedAreaByteskip = selectedAreaBD->Stride - selectedAreaBD->Width \* 3;  int outputByteskip = outputBD->Stride - outputBD->Width \* 3;  Byte\* selectedAreaPtr = (Byte\*)((void\*)selectedAreaBD->Scan0);  Byte\* outputPtr = (Byte\*)((void\*)outputBD->Scan0);  for (int y = 0; y < outputBD->Height; y++)  {  for (int x = 0; x < outputBD->Width; x++)  {  // get origin x and y  int src\_x = x / k;  int src\_y = y / k;  // ptr = y \* stride + x \* 3  Byte\* ptr = selectedAreaPtr + src\_y \* selectedAreaBD->Stride + src\_x \* 3;  outputPtr[0] = ptr[0];  outputPtr[1] = ptr[1];  outputPtr[2] = ptr[2];  outputPtr += 3;  }  outputPtr += outputByteskip;  }  selectedArea->UnlockBits(selectedAreaBD);  output->UnlockBits(outputBD);  return output;  } |

* + 1. First-Order

|  |
| --- |
| private: Bitmap^ ZoomInImg\_withFirstOrder(double k)  {  Bitmap^ selectedArea = GetSelectedArea();  Bitmap^ output = gcnew Bitmap(Math::Abs(endPos\_real->X - startPos\_real->X) \* k  , Math::Abs(endPos\_real->Y - startPos\_real->Y) \* k, selectedArea->PixelFormat);  BitmapData^ selectedAreaBD = selectedArea->LockBits(Rectangle(0, 0, selectedArea->Width, selectedArea->Height), ImageLockMode::ReadOnly, selectedArea->PixelFormat);  BitmapData^ outputBD = output->LockBits(Rectangle(0, 0, output->Width, output->Height), ImageLockMode::ReadWrite, output->PixelFormat);  int selectedAreaByteskip = selectedAreaBD->Stride - selectedAreaBD->Width \* 3;  int outputByteskip = outputBD->Stride - outputBD->Width \* 3;  Byte\* selectedAreaPtr = (Byte\*)((void\*)selectedAreaBD->Scan0);  Byte\* outputPtr = (Byte\*)((void\*)outputBD->Scan0);  for (int y = 0; y < outputBD->Height; y++) // scaling with X  {  for (int x = 0; x < outputBD->Width; x++)  {  // get origin x and y  int src\_x = x / k;  int src\_y = y / k;  double r = fmod(x, k);  if ((src\_x >= 0) && (src\_y >= 0) && // avoid memory out of range  (src\_x < selectedAreaBD->Width - 1) && (src\_y < selectedAreaBD->Height))  {  // ptr = y \* stride + x \* 3  Byte\* ptr = selectedAreaPtr + src\_x \* 3 + src\_y \* selectedAreaBD->Stride;  // output = (ptr \* (k - r) + (ptr + 1) \* r) / k  outputPtr[0] = (int)((\*ptr \* (k - r) + \*(ptr + 3) \* r) / k);  outputPtr[1] = (int)((\*(ptr + 1) \* (k - r) + \*(ptr + 3 + 1) \* r) / k);  outputPtr[2] = (int)((\*(ptr + 2) \* (k - r) + \*(ptr + 3 + 2) \* r) / k);  }  else  outputPtr[0] = outputPtr[1] = outputPtr[2] = 255;  outputPtr += 3;  }  outputPtr += outputByteskip;  }  outputPtr = (Byte\*)((void\*)outputBD->Scan0);  for (int y = 0; y < outputBD->Height; y++) // scaling with Y  {  for (int x = 0; x < outputBD->Width; x++)  {  // get origin x and y  int src\_x = x / k;  int src\_y = y / k;  double r = fmod(y, k);  if ((src\_x >= 0) && (src\_y >= 0) && // avoid memory out of range  (src\_x < selectedAreaBD->Width - 1) && (src\_y < selectedAreaBD->Height))  {  // ptr = y \* stride + x \* 3  Byte\* ptr = selectedAreaPtr + src\_x \* 3 + src\_y \* selectedAreaBD->Stride;  // output = (ptr \* (k - r) + (ptr + 1) \* r) / k  outputPtr[0] = (int)((\*ptr \* (k - r) + \*(ptr + 3) \* r) / k);  outputPtr[1] = (int)((\*(ptr + 1) \* (k - r) + \*(ptr + 3 + 1) \* r) / k);  outputPtr[2] = (int)((\*(ptr + 2) \* (k - r) + \*(ptr + 3 + 2) \* r) / k);  }  else  outputPtr[0] = outputPtr[1] = outputPtr[2] = 255; // set to white  outputPtr += 3;  }  outputPtr += outputByteskip;  }  selectedArea->UnlockBits(selectedAreaBD);  output->UnlockBits(outputBD);  return output;  } |

* + 1. Bilinear

|  |
| --- |
| private: Bitmap^ ZoomInImg\_withBilinearInterpolation(double k)  {  Bitmap^ selectedArea = GetSelectedArea();  Bitmap^ output = gcnew Bitmap(Math::Abs(endPos\_real->X - startPos\_real->X) \* k  , Math::Abs(endPos\_real->Y - startPos\_real->Y) \* k, selectedArea->PixelFormat);  BitmapData^ selectedAreaBD = selectedArea->LockBits(Rectangle(0, 0, selectedArea->Width, selectedArea->Height), ImageLockMode::ReadOnly, selectedArea->PixelFormat);  BitmapData^ outputBD = output->LockBits(Rectangle(0, 0, output->Width, output->Height), ImageLockMode::ReadWrite, output->PixelFormat);  int selectedAreaByteskip = selectedAreaBD->Stride - selectedAreaBD->Width \* 3;  int outputByteskip = outputBD->Stride - outputBD->Width \* 3;  Byte\* selectedAreaPtr = (Byte\*)((void\*)selectedAreaBD->Scan0);  Byte\* outputPtr = (Byte\*)((void\*)outputBD->Scan0);  for (int y = 0; y < outputBD->Height - 1; y++)  {  for (int x = 0; x < outputBD->Width - 1; x++)  {  // get origin x and y  double src\_x = (double)x / k;  double src\_y = (double)y / k;  int isrc\_x = src\_x;  int isrc\_y = src\_y;  int q\_value[4][3]; // record coord's BGR information, the order is Q11(x1, y1), 12(x1, y2), Q21(x2, y1), Q22(x2, y2)  for (int q\_x = 0; q\_x <= 1; q\_x++) // X axis, x1 and x2  {  for (int q\_y = 0; q\_y <= 1; q\_y++) // Y axis, y1 and y2  {  Byte\* ptr = selectedAreaPtr + (isrc\_x + q\_x) \* 3  + (isrc\_y + q\_y) \* selectedAreaBD->Stride;  q\_value[q\_y \* 1 + q\_x \* 2][0] = ptr[0];  q\_value[q\_y \* 1 + q\_x \* 2][1] = ptr[1];  q\_value[q\_y \* 1 + q\_x \* 2][2] = ptr[2];  }  }  outputPtr[0] = q\_value[0][0] \* ((isrc\_x + 1) - src\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[1][0] \* ((isrc\_x + 1) - src\_x) \* (src\_y - isrc\_y)  + q\_value[2][0] \* (src\_x - isrc\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[3][0] \* (src\_x - isrc\_x) \* (src\_y - isrc\_y);  outputPtr[1] = q\_value[0][1] \* ((isrc\_x + 1) - src\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[1][1] \* ((isrc\_x + 1) - src\_x) \* (src\_y - isrc\_y)  + q\_value[2][1] \* (src\_x - isrc\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[3][1] \* (src\_x - isrc\_x) \* (src\_y - isrc\_y);  outputPtr[2] = q\_value[0][2] \* ((isrc\_x + 1) - src\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[1][2] \* ((isrc\_x + 1) - src\_x) \* (src\_y - isrc\_y)  + q\_value[2][2] \* (src\_x - isrc\_x) \* ((isrc\_y + 1) - src\_y)  + q\_value[3][2] \* (src\_x - isrc\_x) \* (src\_y - isrc\_y);  outputPtr += 3;  }  outputPtr[0] = outputPtr[1] = outputPtr[2] = 255; // last column fill white pixel  outputPtr += 3; // fill last 3 information because x only run to width - 1  outputPtr += outputByteskip;  }  for (int x = 0; x < outputBD->Width; x++) // last row fill white pixel  {  outputPtr[0] = outputPtr[1] = outputPtr[2] = 255;  outputPtr += 3;  }  outputPtr += outputByteskip;  selectedArea->UnlockBits(selectedAreaBD);  output->UnlockBits(outputBD);  return output;  } |

* + 1. 整合成一個大Function，共同更新PictureBox

|  |
| --- |
| private: void UpdateAllPictureBox()  {  double k = zoomIn\_or\_zoomOut ? 1.0f / Zoom\_trackBar->Value : Zoom\_trackBar->Value;  SetAreaSizeLabel(k);  ZeroOrder\_pictureBox->Image = ZoomInImg\_withZeroOrder(k);  FirstOrder\_pictureBox->Image = ZoomInImg\_withFirstOrder(k);  Bilinear\_pictureBox->Image = ZoomInImg\_withBilinearInterpolation(k);  } |

* + 1. 滑鼠圈選範圍事件

|  |
| --- |
| /\*\* Get X Y, and draw rectangle area \*\*/  private:  Point^ startPos, ^ endPos; // select point on picture box  Point^ startPos\_real, ^ endPos\_real; // turn to real point on image  bool isDrawing, isMoving;  bool hasOriginImg;  private: void GetRealPos(int start\_x, int start\_y, int end\_x, int end\_y)  {  // transfer select point to real point (using re-scale)  if (GrayOrigin\_pictureBox->Image->Width  > GrayOrigin\_pictureBox->Image->Height)  {  int offset = (GrayOrigin\_pictureBox->Image->Width  - GrayOrigin\_pictureBox->Image->Height) / 2;  startPos\_real  = gcnew Point(start\_x \* GrayOrigin\_pictureBox->Image->Width / GrayOrigin\_pictureBox->Width , start\_y \* GrayOrigin\_pictureBox->Image->Width / GrayOrigin\_pictureBox->Width - offset);  endPos\_real = gcnew Point(end\_x \* GrayOrigin\_pictureBox->Image->Width / GrayOrigin\_pictureBox->Width, end\_y \* GrayOrigin\_pictureBox->Image->Width / GrayOrigin\_pictureBox->Width - offset);  }  else  {  int offset = (GrayOrigin\_pictureBox->Image->Height  - GrayOrigin\_pictureBox->Image->Width) / 2;  startPos\_real  = gcnew Point(start\_x \* GrayOrigin\_pictureBox->Image->Height / GrayOrigin\_pictureBox->Height - offset, start\_y \* GrayOrigin\_pictureBox->Image->Height / GrayOrigin\_pictureBox->Height);  endPos\_real  = gcnew Point(end\_x \* GrayOrigin\_pictureBox->Image->Height / GrayOrigin\_pictureBox->Height - offset, end\_y \* GrayOrigin\_pictureBox->Image->Height / GrayOrigin\_pictureBox->Height);  }  // when the selected range out of picture's range  if (startPos\_real->X < 0) startPos\_real->X = 0;  if (startPos\_real->X > GrayOrigin\_pictureBox->Image->Width) startPos\_real->X = GrayOrigin\_pictureBox->Image->Width;  if (startPos\_real->Y < 0) startPos\_real->Y = 0;  if (startPos\_real->Y > GrayOrigin\_pictureBox->Image->Height) startPos\_real->Y = GrayOrigin\_pictureBox->Image->Height;  if (endPos\_real->X < 0) endPos\_real->X = 0;  if (endPos\_real->X > GrayOrigin\_pictureBox->Image->Width) endPos\_real->X = GrayOrigin\_pictureBox->Image->Width;  if (endPos\_real->Y < 0) endPos\_real->Y = 0;  if (endPos\_real->Y > GrayOrigin\_pictureBox->Image->Height) endPos\_real->Y = GrayOrigin\_pictureBox->Image->Height;  }  private: System::Void GrayOrigin\_pictureBox\_Paint(System::Object^ sender, System::Windows::Forms::PaintEventArgs^ e)  {  Pen^ selectRed = gcnew Pen(Color::Red, 2.f);  if (isDrawing)  {  if (!GrayOrigin\_pictureBox->Image) return;  if (endPos->X - startPos->X >=0 && endPos->Y - startPos->Y >= 0)  {  GetRealPos(startPos->X, startPos->Y, endPos->X, endPos->Y);  e->Graphics->DrawRectangle(selectRed, startPos->X, startPos->Y, Math::Abs(endPos->X - startPos->X), Math::Abs(endPos->Y - startPos->Y));  }  else if (endPos->X - startPos->X < 0 && endPos->Y - startPos->Y >= 0)  {  GetRealPos(startPos->X, startPos->Y, endPos->X, endPos->Y);  e->Graphics->DrawRectangle(selectRed, endPos->X, startPos->Y, Math::Abs(endPos->X - startPos->X), Math::Abs(endPos->Y - startPos->Y));  }  else if (endPos->X - startPos->X >= 0 && endPos->Y - startPos->Y < 0)  {  GetRealPos(endPos->X, endPos->Y, startPos->X, startPos->Y);  e->Graphics->DrawRectangle(selectRed, startPos->X, endPos->Y, Math::Abs(endPos->X - startPos->X), Math::Abs(endPos->Y - startPos->Y));  }  else if (endPos->X - startPos->X < 0 && endPos->Y - startPos->Y < 0)  {  GetRealPos(endPos->X, endPos->Y, startPos->X, startPos->Y);  e->Graphics->DrawRectangle(selectRed, endPos->X, endPos->Y, Math::Abs(endPos->X - startPos->X), Math::Abs(endPos->Y - startPos->Y));  }  }  }  private: System::Void GrayOrigin\_pictureBox\_MouseDown(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e)  {  if (hasOriginImg)  {  startPos = endPos = e->Location;  isDrawing = true;  if (isMoving)  GrayOrigin\_pictureBox->Invalidate();  }  }  private: System::Void GrayOrigin\_pictureBox\_MouseUp(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e)  {  if (isDrawing)  {  isDrawing = false;  isMoving = false;  // let the endPos bigger than the startPos forever  if (startPos\_real->X > endPos\_real->X)  {  int tmp = startPos\_real->X;  startPos\_real->X = endPos\_real->X;  endPos\_real->X = tmp;  }  if (startPos\_real->Y > endPos\_real->Y)  {  int tmp = startPos\_real->Y;  startPos\_real->Y = endPos\_real->Y;  endPos\_real->Y = tmp;  }  if (endPos != startPos) // avoid bug easily  UpdateAllPictureBox();  }  }  private: System::Void GrayOrigin\_pictureBox\_MouseMove(System::Object^ sender, System::Windows::Forms::MouseEventArgs^ e)  {  if (isDrawing)  {  isMoving = true;  endPos = e->Location;  GrayOrigin\_pictureBox->Invalidate();  }  } |

* + 1. 取得圈選範圍之圖片

|  |
| --- |
| private: Bitmap^ GetSelectedArea()  {  Bitmap^ origin = (Bitmap^)GrayOrigin\_pictureBox->Image;  Bitmap^ output = gcnew Bitmap(Math::Abs(endPos\_real->X - startPos\_real->X), Math::Abs(endPos\_real->Y - startPos\_real->Y), origin->PixelFormat);  BitmapData^ originBD = origin->LockBits(Rectangle(0, 0, origin->Width, origin->Height), ImageLockMode::ReadOnly, origin->PixelFormat);  BitmapData^ outputBD = output->LockBits(Rectangle(0, 0, output->Width, output->Height), ImageLockMode::ReadWrite, output->PixelFormat);  int orignByteskip = originBD->Stride - originBD->Width \* 3;  int outputByteskip = outputBD->Stride - outputBD->Width \* 3;  Byte\* originPtr = (Byte\*)((void\*)originBD->Scan0);  Byte\* outputPtr = (Byte\*)((void\*)outputBD->Scan0);  for (int y = 0; y < originBD->Height; y++)  {  for (int x = 0; x < originBD->Width; x++)  {  if (y >= startPos\_real->Y && y < endPos\_real->Y &&  x >= startPos\_real->X && x < endPos\_real->X) // in the selected area  {  outputPtr[0] = originPtr[0];  outputPtr[1] = originPtr[1];  outputPtr[2] = originPtr[2];  outputPtr += 3;  }  originPtr += 3;  }  if (y >= startPos\_real->Y && y <= endPos\_real->Y) outputPtr += outputByteskip;  originPtr += orignByteskip;  }  origin->UnlockBits(originBD);  output->UnlockBits(outputBD);  return output;  } |

* 1. 第三小題

基本上同第二題程式碼，唯一的差別在於我在主選單傳入的參數不同，用以判別現在該縮小還是放大

|  |
| --- |
| private: System::Void Q2Btn\_Click(System::Object^ sender, System::EventArgs^ e)  {  Q2\_Form^ q2\_form = gcnew Q2\_Form();  q2\_form->zoomIn\_or\_zoomOut = 0;  q2\_form->Show();  }  private: System::Void Q3Btn\_Click(System::Object^ sender, System::EventArgs^ e)  {  Q2\_Form^ q2\_form = gcnew Q2\_Form();  q2\_form->zoomIn\_or\_zoomOut = 1;  q2\_form->Show();  } |