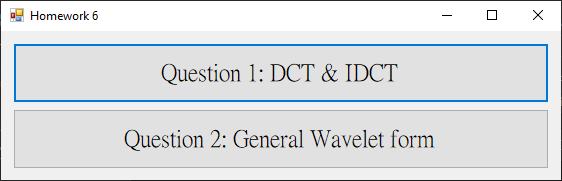
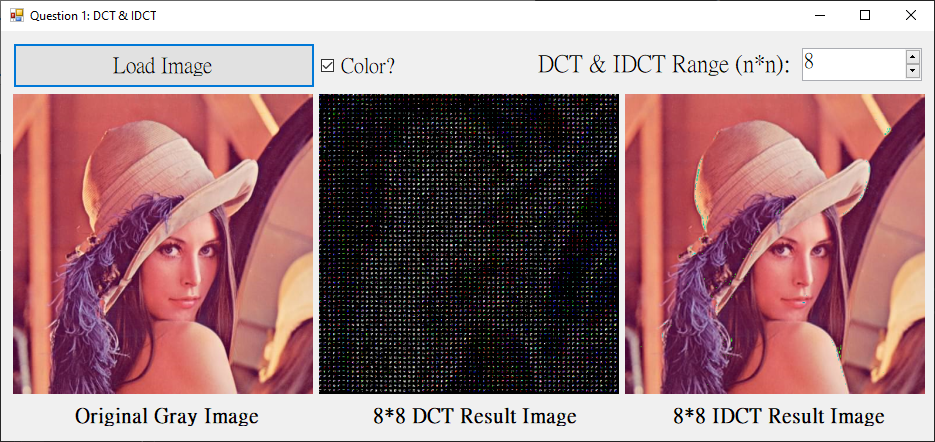
四資工三甲 C108151131 賴冠綸

1. 程式執行結果
2. 程式起始頁面（選擇做DCT or 小波轉換）

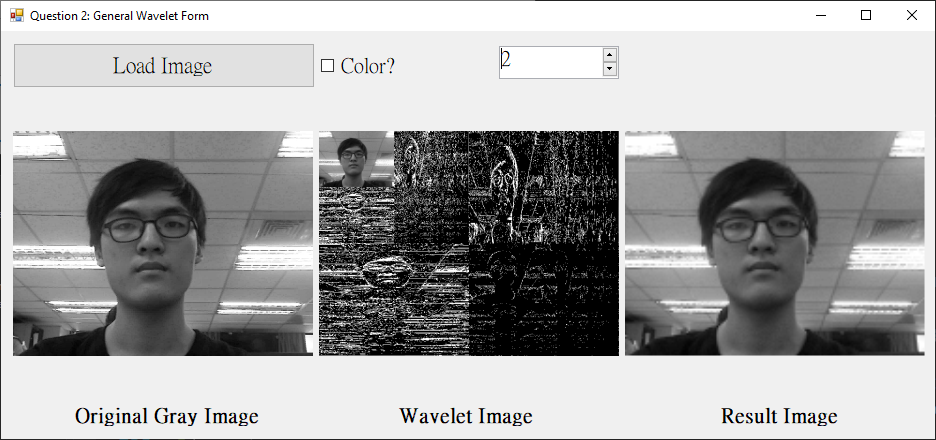


1. DCT & IDCT（可以處理全彩圖檔）

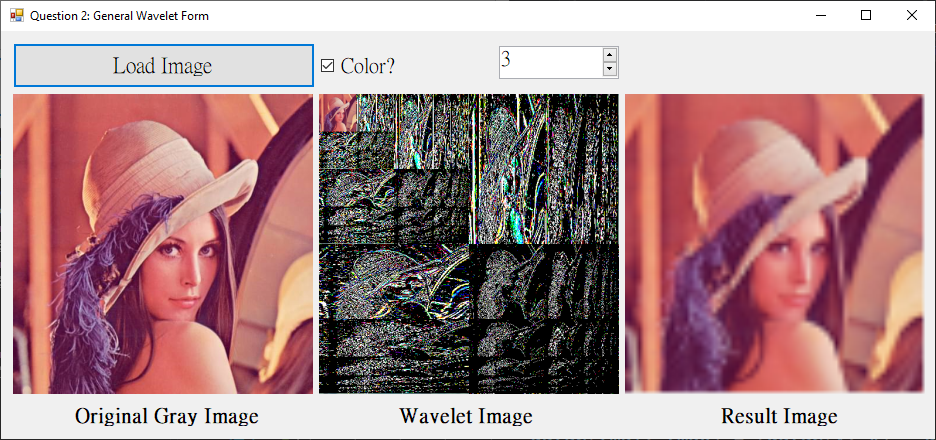


1. 小波轉換（可以處理全彩圖檔）
   * 1. 做2次（1/16）

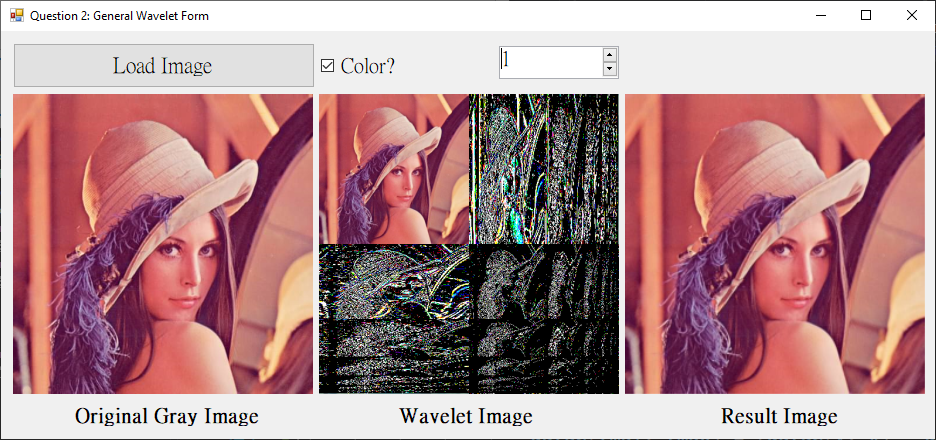
此範例用自己的人頭照，但不為正方形圖片(M\*N)。



* + 1. 做3次（1/64）



* + 1. 做1次（1/4）



1. 程式碼（僅附主要程式碼）
2. DCT & IDCT

我使用dctArr\_B、dctArr\_G、dctArr\_R存下DCT的RGB資訊（不拿圖片來做是避免溢位）。

* + 1. DCT

|  |
| --- |
| private: Bitmap^ DCT\_Transform(int n, 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);  int rowBlock = grayBD->Width / n;  int colBlock = grayBD->Height / n;  dctArr\_B = gcnew array<int, 2>(grayBD->Height, grayBD->Width);  dctArr\_G = gcnew array<int, 2>(grayBD->Height, grayBD->Width);  dctArr\_R = gcnew array<int, 2>(grayBD->Height, grayBD->Width);  for (int i = 0; i < colBlock; i++)  {  for (int j = 0; j < rowBlock; j++)  {  int offsetX = j \* n, offsetY = i \* n;  for (int v = offsetY; v < offsetY + n; v++)  {  for (int u = offsetX; u < offsetX + n; u++)  {  double alpha\_u, alpha\_v;  if (u == offsetX) alpha\_u = sqrt(1.0 / n);  else alpha\_u = sqrt(2.0 / n);  if (v == offsetY) alpha\_v = sqrt(1.0 / n);  else alpha\_v = sqrt(2.0 / n);  double bSum = 0, gSum = 0, rSum = 0;  for (int y = offsetY; y < offsetY + n; y++)  {  for (int x = offsetX; x < offsetX + n; x++)  {  Byte\* ptr = grayPtr + x \* 3 + y \* grayBD->Stride;  bSum += ((int)ptr[0] - 128)  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  gSum += ((int)ptr[1] - 128)  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  rSum += ((int)ptr[2] - 128)  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  }  }  double b = alpha\_u \* alpha\_v \* bSum;  double g = alpha\_u \* alpha\_v \* gSum;  double r = alpha\_u \* alpha\_v \* rSum;  // filter high frequency information  if (!((u % n < n / 2) && (v % n < n / 2)))  {  dctArr\_B[v, u] = 0;  dctArr\_G[v, u] = 0;  dctArr\_R[v, u] = 0;  }  else  {  dctArr\_B[v, u] = b;  dctArr\_G[v, u] = g;  dctArr\_R[v, u] = r;  }  Byte\* ptr = resultPtr + u \* 3 + v \* resultBD->Stride;  ptr[0] = (Byte)dctArr\_B[v, u];  ptr[1] = (Byte)dctArr\_G[v, u];  ptr[2] = (Byte)dctArr\_R[v, u];  }  }  }  }  gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |

* + 1. IDCT

|  |
| --- |
| private: Bitmap^ IDCT\_Transform(int n, Bitmap^ dctImg)  {  Bitmap^ result = gcnew Bitmap(dctImg->Width, dctImg->Height, dctImg->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadWrite, result->PixelFormat);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  int rowBlock = dctImgBD->Width / n;  int colBlock = dctImgBD->Height / n;  for (int i = 0; i < colBlock; i++)  {  for (int j = 0; j < rowBlock; j++)  {  int offsetX = j \* n, offsetY = i \* n;  for (int y = offsetY; y < offsetY + n; y++)  {  for (int x = offsetX; x < offsetX + n; x++)  {  double bSum = 0, gSum = 0, rSum = 0;  for (int v = offsetY; v < offsetY + n; v++)  {  for (int u = offsetX; u < offsetX + n; u++)  {  double alpha\_u, alpha\_v;  if (u == offsetX) alpha\_u = sqrt(1.0 / n);  else alpha\_u = sqrt(2.0 / n);  if (v == offsetY) alpha\_v = sqrt(1.0 / n);  else alpha\_v = sqrt(2.0 / n);  //Byte\* ptr = dctImgPtr + u \* 3 + v \* dctImgBD->Stride;  bSum += alpha\_u \* alpha\_v \* dctArr\_B[v, u]  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  gSum += alpha\_u \* alpha\_v \* dctArr\_G[v, u]  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  rSum += alpha\_u \* alpha\_v \* dctArr\_R[v, u]  \* cos(((2.0 \* (x - offsetX) + 1) \* (u - offsetX) \* PI) / (2.0 \* n))  \* cos(((2.0 \* (y - offsetY) + 1) \* (v - offsetY) \* PI) / (2.0 \* n));  }  }  // Now have some bug: have gap (high frequency) between each block  Byte\* ptr = resultPtr + x \* 3 + y \* resultBD->Stride;  ptr[0] = (Byte)(bSum + 128);  ptr[1] = (Byte)(gSum + 128);  ptr[2] = (Byte)(rSum + 128);  }  }  }  }  dctImg->UnlockBits(dctImgBD);  result->UnlockBits(resultBD);  return result;  } |

1. 小波轉換
2. 小波轉換主程式碼片段

|  |
| --- |
| private: Bitmap^ GeneralWaveletTransform(int level, Bitmap^ gray)  {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat);  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadWrite, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  array<int, 2>^ dataB = gcnew array<int, 2> (grayBD->Height, grayBD->Width);  array<int, 2>^ dataG = gcnew array<int, 2> (grayBD->Height, grayBD->Width);  array<int, 2>^ dataR = gcnew array<int, 2> (grayBD->Height, grayBD->Width);  /\* Copy origin image data \*/  for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  Byte\* ptr = grayPtr + x \* 3 + y \* grayBD->Stride;  dataB[y, x] = ptr[0];  dataG[y, x] = ptr[1];  dataR[y, x] = ptr[2];  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int curHeight = grayBD->Height;  int curWidth = grayBD->Width;  while (level--)  {  int tmpX = 0, tmpY = 0;  // handle row (low frequency)  for (int y = 0; y < curHeight; y++)  {  tmpX = 0;  for (int x = 0; x < curWidth; x += 2)  {  dataB[tmpY, tmpX] = (dataB[y, x] + dataB[y, x + 1]) / 2;  dataG[tmpY, tmpX] = (dataG[y, x] + dataG[y, x + 1]) / 2;  dataR[tmpY, tmpX] = (dataR[y, x] + dataR[y, x + 1]) / 2;  tmpX++;  }  tmpY++;  }  tmpY = 0;  // handle row (high frequency)  for (int y = 0; y < curHeight; y++)  {  tmpX = curWidth / 2;  for (int x = 0; x < curWidth; x += 2)  {  dataB[tmpY, tmpX] = (dataB[y, x] - dataB[y, x + 1]) / 2;  dataG[tmpY, tmpX] = (dataG[y, x] - dataG[y, x + 1]) / 2;  dataR[tmpY, tmpX] = (dataR[y, x] - dataR[y, x + 1]) / 2;  tmpX++;  }  tmpY++;  }  tmpY = 0;  tmpX = 0, tmpY = 0;    // handle col (low frequency)  for (int x = 0; x < curWidth; x++)  {  tmpY = 0;  for (int y = 0; y < curHeight; y += 2)  {  dataB[tmpY, tmpX] = (dataB[y, x] + dataB[y + 1, x]) / 2;  dataG[tmpY, tmpX] = (dataG[y, x] + dataG[y + 1, x]) / 2;  dataR[tmpY, tmpX] = (dataR[y, x] + dataR[y + 1, x]) / 2;  tmpY++;  }  tmpX++;  }  tmpX = 0;  // handle col (high frequency)  for (int x = 0; x < curWidth; x++)  {  tmpY = curHeight / 2;  for (int y = 0; y < curHeight; y += 2)  {  dataB[tmpY, tmpX] = (dataB[y, x] - dataB[y + 1, x]) / 2;  dataG[tmpY, tmpX] = (dataG[y, x] - dataG[y + 1, x]) / 2;  dataR[tmpY, tmpX] = (dataR[y, x] - dataR[y + 1, x]) / 2;  tmpY++;  }  tmpX++;  }  tmpX = 0;  curHeight /= 2;  curWidth /= 2;  }    for (int y = 0; y < grayBD->Height; y++)  {  for (int x = 0; x < grayBD->Width; x++)  {  resultPtr[0] = dataB[y, x];  resultPtr[1] = dataG[y, x];  resultPtr[2] = dataR[y, x];  resultPtr += 3;  }  resultPtr += resultByteskip;  }  gray->UnlockBits(grayBD);  result->UnlockBits(resultBD);  return result;  } |

1. 只取低頻區圖像之程式碼片段

|  |
| --- |
| private: Bitmap^ getRealResultImg(int level, Bitmap^ waveletImg)  {  int width = waveletImg->Width;  int height = waveletImg->Height;  while (level--)  {  width /= 2;  height /= 2;  }  if (!width || !height) return nullptr;  Bitmap^ result = gcnew Bitmap(width, height, waveletImg->PixelFormat);  BitmapData^ waveletImgBD = waveletImg->LockBits(Rectangle(0, 0, waveletImg->Width  , waveletImg->Height), ImageLockMode::ReadWrite, waveletImg->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* waveletImgPtr = (Byte\*)((void\*)waveletImgBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  for (int y = 0; y < resultBD->Height; y++)  {  for (int x = 0; x < resultBD->Width; x++)  {  Byte\* ptr = waveletImgPtr + x \* 3 + y \* waveletImgBD->Stride;  resultPtr[0] = ptr[0];  resultPtr[1] = ptr[1];  resultPtr[2] = ptr[2];  resultPtr += 3;  }  resultPtr += resultByteskip;  }  waveletImg->UnlockBits(waveletImgBD);  result->UnlockBits(resultBD);  return result;  } |