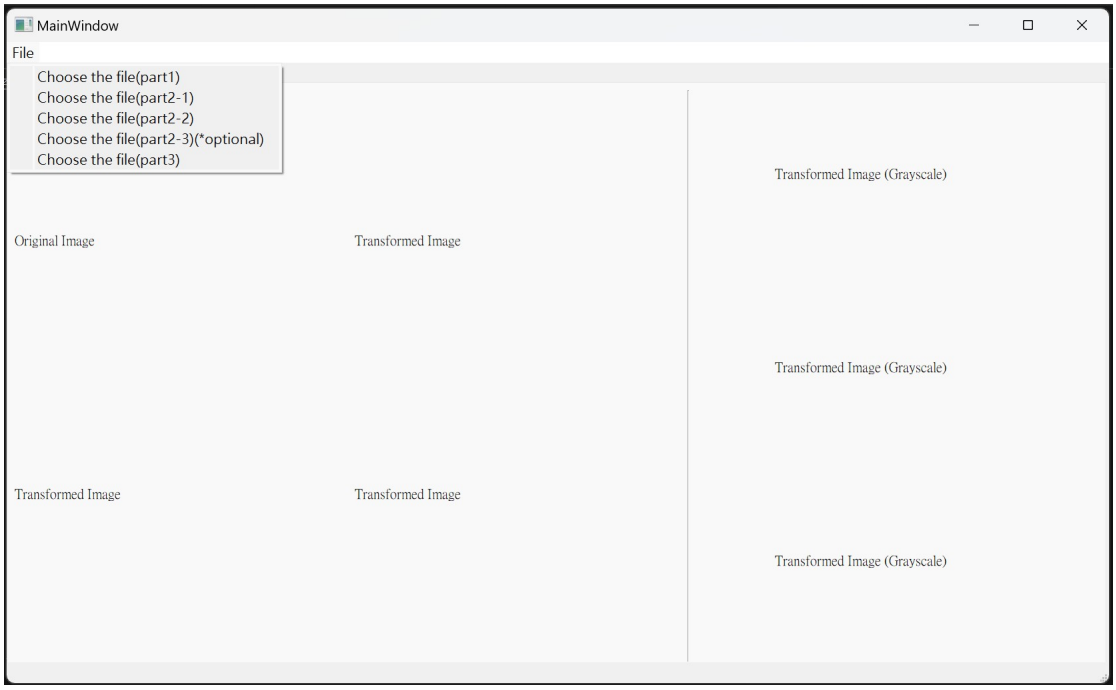
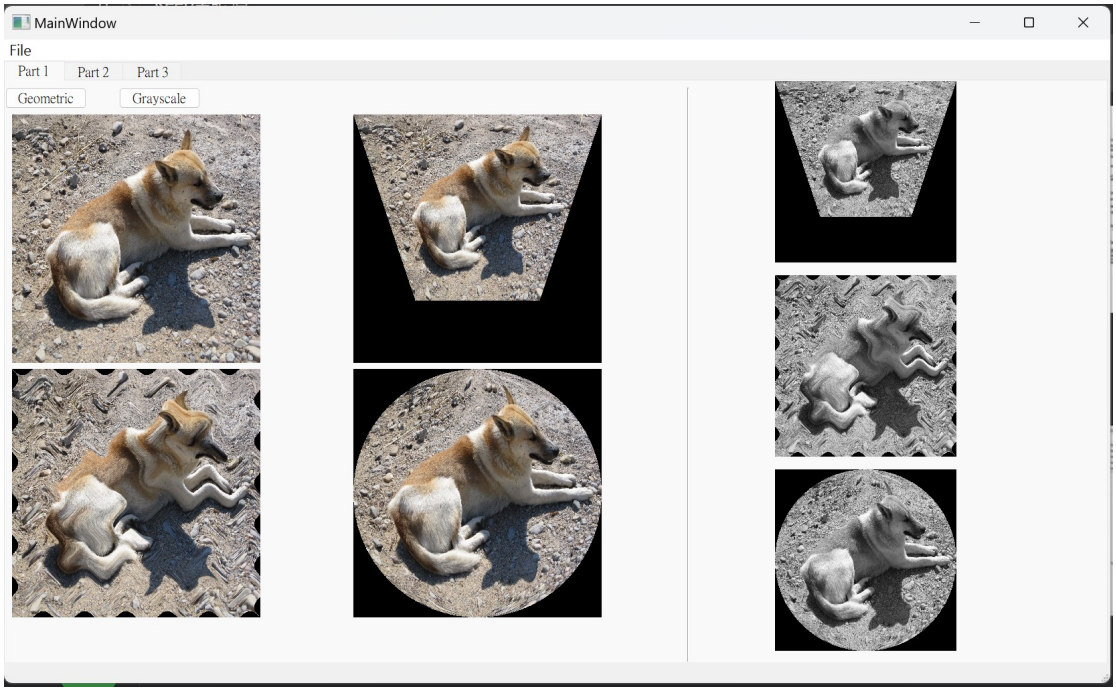
B09611007 陳柏霖

介面截圖

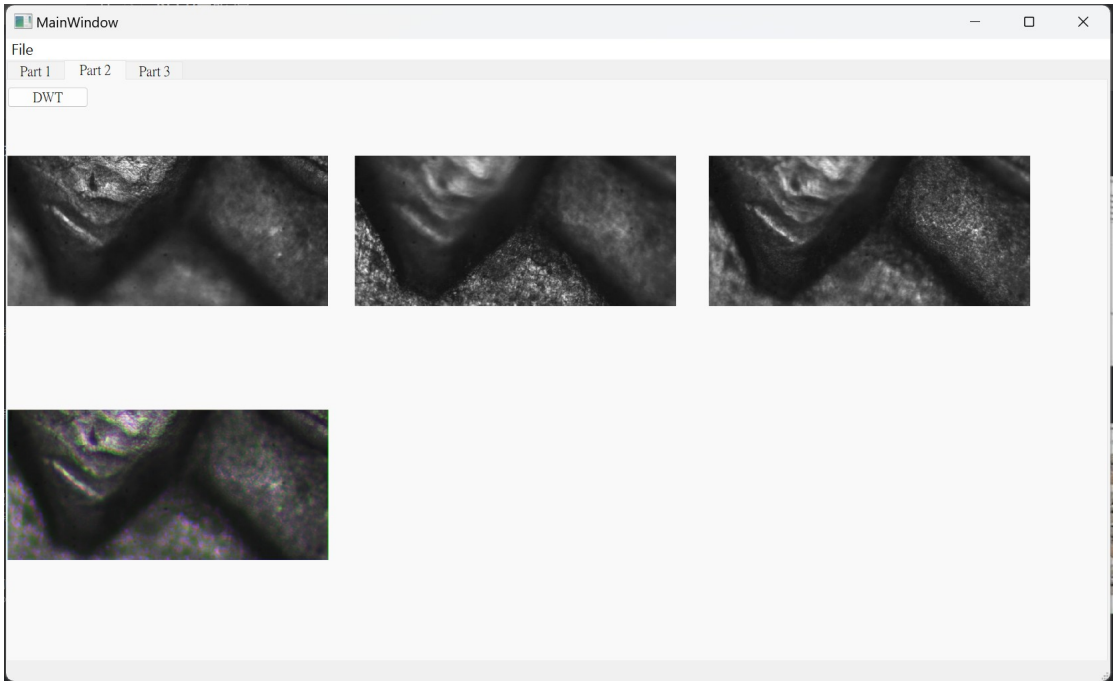
請點擊 Choose the file 以選取各 part 欲測試照片



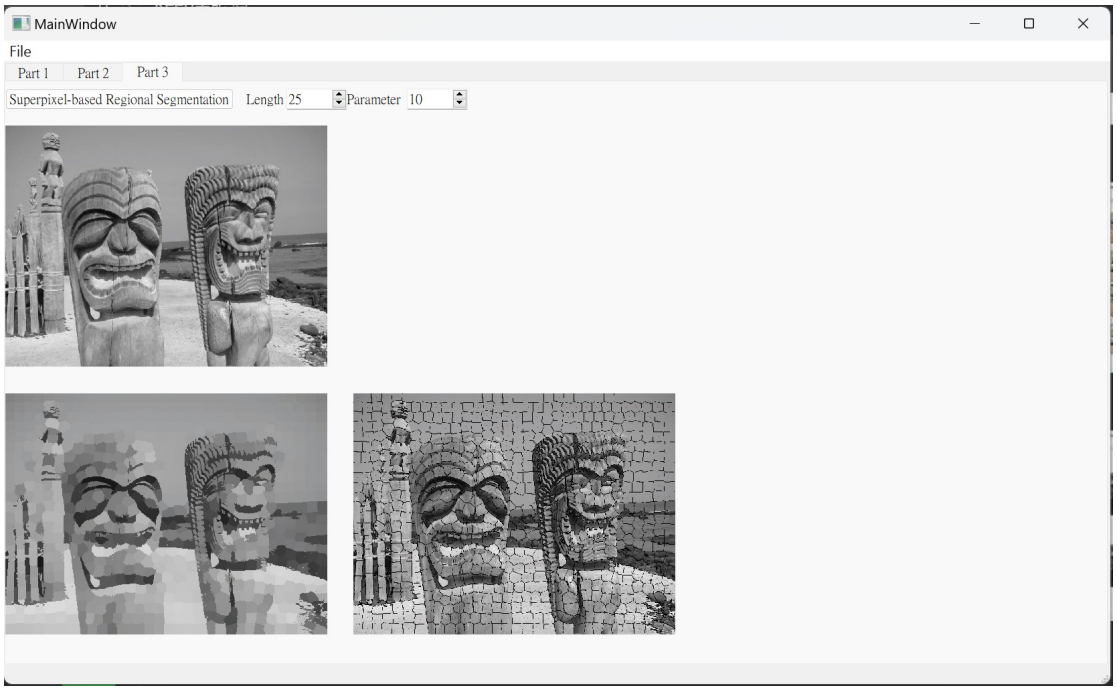
Part1: 藉由座標間的轉換讓影像產生變形，梯形是藉由數學幾何計算;波浪轉換應用三角函數公式；圓形藉由映射。按壓 geometric 按鈕進行轉換，按壓 Grayscale 按鈕以轉換為灰階影像。



Part2: 可選擇載入 2-3 張影像，按下 DWT 按鈕進行小波轉換。



Part3: 按下按鈕以使用 Simple Linear Iterative Clustering (SLIC)進行 Superpixel-based Regional Segmentation，Superpixel 的大小與 Parameter 可以調整



QImage2CvMat 函數用於將 QImage 轉換為 OpenCV 的 Mat 對象，以便進行圖像處理。

```cpp
cv::Mat MainWindow::QImage2CvMat(QImage &image)
{
    cv::Mat mat;
    //qDebug() << image.format();
    switch (image.format())
    {
    case QImage::Format_ARGB32:
        mat = cv::Mat(image.height(), image.width(), CV_8UC4, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_RGB32:
        mat = cv::Mat(image.height(), image.width(), CV_8UC3, (void*)image.constBits(), image.bytesPerLine());
        //cv::cvtColor(mat, mat, CV_BGR2RGB);
        break;
    case QImage::Format_ARGB32_Premultiplied:
        mat = cv::Mat(image.height(), image.width(), CV_8UC4, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_RGB888:
        mat = cv::Mat(image.height(), image.width(), CV_8UC3, (void*)image.constBits(), image.bytesPerLine());
        //cv::cvtColor(mat, mat, CV_BGR2RGB);
        break;
    case QImage::Format_Indexed8:
        mat = cv::Mat(image.height(), image.width(), CV_8UC1, (void*)image.constBits(), image.bytesPerLine());
        break;
    case QImage::Format_Grayscale8:
        mat = cv::Mat(image.height(), image.width(), CV_8UC1, (void*)image.constBits(), image.bytesPerLine());
        break;
    }
    return mat;
}
```

displayImageOnLabel 函數將 QImage 顯示在 QLabel 上，用於在應用程序中顯示圖像。

```cpp
void MainWindow::displayImageOnLabel(const QImage& image, QLabel* label)
{
//    label->setPixmap(QPixmap::fromImage(image));
//    label->setScaledContents(true);
//    label->show();
    const int w = label ->width();
    const int h = label ->height();
    label ->setPixmap(QPixmap::fromImage(image.scaled(w, h, Qt::KeepAspectRatio)));
}
```

cvMat_to_QImage 函數用於將 OpenCV 的 Mat 對象轉換為 QImage，以便進行圖像處理。

```cpp
QImage MainWindow::cvMat_to_QImage(const cv::Mat &mat ) {
    switch ( mat.type() )
    {
    // 8-bit, 4 channel
    case CV_8UC4:
    {
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_RGB32 );
        return image;
    }

        // 8-bit, 3 channel
    case CV_8UC3:
    {
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_RGB888 );
        return image.rgbSwapped();
    }

        // 8-bit, 1 channel
    case CV_8UC1:
    {
        static QVector<QRgb>  sColorTable;
        // only create our color table once
        if ( sColorTable.isEmpty() )
        {
            for ( int i = 0; i < 256; ++i )
                sColorTable.push_back( qRgb( i, i, i ) );
        }
        QImage image( mat.data, mat.cols, mat.rows, mat.step, QImage::Format_Indexed8 );
        image.setColorTable( sColorTable );
        return image;
    }

    default:
        qDebug("Image format is not supported: depth=%d and %d channels\n", mat.depth(), mat.channels());
        break;
    }
    return QImage();
}
```

執行三種不同的影像轉換，分別是梯形變換（performTrapezoidalTransformation）、波浪變換（performWavyTransformation）和圓形變換（performCircularTransformation）。

```
Mat MainWindow::performTrapezoidalTransformation(const Mat &srcImage)
{
    Mat trapezoidalTransformed = Mat::zeros(srcImage.size(), srcImage.type());

    for (int i = 0; i < srcImage.rows; ++i)
    {
        for (int j = 0; j < srcImage.cols; ++j)
        {
            int a = static_cast<int>(round(3 * i / 4.0));
            int b = static_cast<int>(round(j + i / 4.0 - j * i / (2.0 * srcImage.cols)));

            if (a >= 0 && a < srcImage.rows && b >= 0 && b < srcImage.cols)
            {
                trapezoidalTransformed.at<Vec3b>(a, b) = srcImage.at<Vec3b>(i, j);
            }
        }
    }

    return trapezoidalTransformed;
}
```

```
Mat MainWindow::performWavyTransformation(const Mat &srcImage)
{
    Mat wavyTransformed = Mat::zeros(srcImage.size(), srcImage.type());

    for (int i = 0; i < srcImage.rows; ++i)
    {
        for (int j = 0; j < srcImage.cols; ++j)
        {
            int a = static_cast<int>(round(i - 30 * sin(2 * CV_PI * j / 200.0)));
            int b = static_cast<int>(round(j - 30 * sin(2 * CV_PI * i / 200.0)));

            if (a >= 0 && a < srcImage.rows && b >= 0 && b < srcImage.cols)
            {
                wavyTransformed.at<Vec3b>(i, j) = srcImage.at<Vec3b>(a, b);
            }
        }
    }

    return wavyTransformed;
}
```

```cpp
Mat MainWindow::performCircularTransformation(const Mat &srcImage)
{
    Mat circularTransformed = Mat::zeros(srcImage.size(), srcImage.type());
    Point center(srcImage.cols / 2, srcImage.rows / 2);

    for (int i = 1; i < srcImage.rows; ++i)
    {
        for (int j = 1; j < srcImage.cols; ++j)
        {
            double r = srcImage.rows / 2.0;
            double r1 = sqrt(pow(r, 2) - pow(r - i, 2));

            int a = i;
            int b = static_cast<int>(round((j - r) * r / r1 + r));

            if (a >= 0 && a < srcImage.rows && b >= 0 && b < srcImage.cols)
            {
                circularTransformed.at<Vec3b>(i, j) = srcImage.at<Vec3b>(a, b);
            }
        }
    }

    return circularTransformed;
}
```

Part2:

確認是否載入了至少兩張圖片：

```
if (rgbImage3.isNull() || rgbImage2.isNull())
{       QMessageBox::warning(this, tr("Error"), tr("At least two images need to be loaded!"));
        return;
}
```

程式首先檢查是否已載入至少兩張圖片，其中 rgbImage3 和 rgbImage2 是類型為 QImage 的圖片。如果其中一張圖片為空（null），則顯示錯誤訊息並結束函式。

將 QImage 轉換為 cv::Mat：

```
Mat srcImage1 = QImageToCvMat(rgbImage3, true);
Mat srcImage2 = QImageToCvMat(rgbImage2, true);
```

使用自定義的 QImageToCvMat 函式，將兩張 QImage 轉換為對應的 cv::Mat 形式。

調整圖片尺寸至二的冪次方：

```
int width = getOptimalDFTSize(srcImage1.cols);
int height = getOptimalDFTSize(srcImage1.rows);
cv::resize(srcImage1, srcImage1, cv::Size(width, height));
cv::resize(srcImage2, srcImage2, cv::Size(width, height));
```

使用 getOptimalDFTSize 函式調整兩張圖片的寬和高至最接近的二的冪次方。接著，使用 OpenCV 的 resize 函式調整圖片大小。

確認是否載入了第三張圖片：

```
if (rgbImage1.isNull())
{
    // Image fusion with two images
    // ...
}
else
{
    // Image fusion with three images
    // ...
}
```

程式接著檢查是否載入了第三張圖片 (rgbImage1)。如果沒有，則執行具有兩張圖片的影像融合。如果有，則執行具有三張圖片的影像融合。

將第三張 QImage 轉換為 cv::Mat：

```
Mat srcImage3 = QImageToCvMat(rgbImage1, true);
```

如果有第三張圖片，將其轉換為對應的 cv::Mat。

確認所有三張圖片的大小是否相同：

```
if (srcImage1.size() != srcImage2.size() || srcImage2.size() != srcImage3.size())
{
    QMessageBox::warning(this, tr("Error"), tr("All images must have the same size for fusion!"));
    return;
}
```

程式確保三張圖片的大小相同，否則顯示錯誤訊息並結束函式。

將第三張圖片大小調整至二的冪次方：

```
cv::resize(srcImage3, srcImage3, cv::Size(width, height));
```

如果有第三張圖片，同樣調整其大小至最接近的二的冪次方。

執行影像融合（wavelet transform）：

```
Mat fusedImage = wavelet(srcImage1, srcImage2);
// or
Mat fusedImage = wavelet(srcImage1, srcImage2, srcImage3);
```

使用 wavelet 函式進行影像融合。融合後的結果存儲在 fusedImage 中。

將 cv::Mat 轉換回 QImage 並顯示：

QImage fusedQImage = cvMatToQImage(fusedImage);

displayImageOnLabel(fusedQImage, ui->Fused_Img_label);

最後，將融合後的 cv::Mat 轉換為 QImage，並使用 displayImageOnLabel 函式顯示在應用程式的 UI 中。

```
void MainWindow::on_Image_Fusion_pushButton_clicked()
{
    // Check if at least two images are loaded
    if (rgbImage3.isNull() || rgbImage2.isNull())
    {
        QMessageBox::warning(this, tr("Error"), tr("At least two images need to be loaded!"));
        return;
    }

    // Convert QImage to cv::Mat for all loaded images
    Mat srcImage1 = QImageToCvMat(rgbImage3, true);
    Mat srcImage2 = QImageToCvMat(rgbImage2, true);

    // Resize images to have power-of-two resolutions
    int width = getOptimalDFTSize(srcImage1.cols);
    int height = getOptimalDFTSize(srcImage1.rows);
    cv::resize(srcImage1, srcImage1, cv::Size(width, height));
    cv::resize(srcImage2, srcImage2, cv::Size(width, height));

    // Check if a third image is loaded
    if (rgbImage1.isNull())
    {
        // Perform image fusion using wavelet transform on two images
        Mat fusedImage = wavelet(srcImage1, srcImage2);

        // Convert cv::Mat back to QImage
        QImage fusedQImage = cvMatToQImage(fusedImage);

        // Display the fused image
        displayImageOnLabel(fusedQImage, ui->Fused_Img_label);
    }
    else
    {
        // Convert QImage to cv::Mat for the third image
        Mat srcImage3 = QImageToCvMat(rgbImage1, true);

        // Check if all three images have the same size
        if (srcImage1.size() != srcImage2.size() || srcImage2.size() != srcImage3.size())
        {
            QMessageBox::warning(this, tr("Error"), tr("All images must have the same size for fusion!"));
            return;
        }

        // Resize the third image to have power-of-two resolution
        cv::resize(srcImage3, srcImage3, cv::Size(width, height));

        // Perform image fusion using wavelet transform on three images
        Mat fusedImage = wavelet(srcImage1, srcImage2, srcImage3);

        // Convert cv::Mat back to QImage
        QImage fusedQImage = cvMatToQImage(fusedImage);

        // Display the fused image
        displayImageOnLabel(fusedQImage, ui->Fused_Img_label);
    }
}
```

Part3:

實現 SLIC（Simple Linear Iterative Clustering）的超像素分割算法。以下是主要函式的功能解釋：

1. **`clustering` 函式：**
   - 該函式執行基於 SLIC 的像素分類（clustering）。它計算每個像素到每個中心的距離，然後將像素分配到離其最近的中心。
   - `DisMask` 是一個矩陣，保存每個像素到最近中心的距離。
   - `labelMask` 是一個矩陣，保存每個像素的標籤（分配到的群組）。
   - `centers` 是一個包含群組中心位置及顏色信息的結構體的向量。
   - `len` 控制像素搜索區域的大小。
   - `m` 是控制距離和顏色相似性的權重。

2. **`updateCenter` 函式：**
   - 該函式根據當前像素標籤更新每個群組的中心。
   - `imageLAB` 是轉換為 Lab 色彩空間的輸入影像。
   - `labelMask` 包含每個像素的標籤。
   - `centers` 是群組中心的結構體向量。
   - `len` 控制像素搜索區域的大小。

3. **`SLICResult` 函式：**
   - 該函式根據最終的標籤結果將影像顏色恢復，生成最終的超像素結果。
   - `image` 是原始影像。
   - `labelMask` 包含每個像素的標籤。
   - `centers` 是群組中心的結構體向量。
   - `len` 控制像素搜索區域的大小。

4. **`SLICResult2` 函式：**
   - 該函式生成一個簡化版本的超像素結果，僅顯示區域邊界。
   - `image` 是原始影像。
   - `labelMask` 包含每個像素的標籤。
   - `centers` 是群組中心的結構體向量。
   - `len` 控制像素搜索區域的大小。

5. **`initilizeCenters` 函式：**
   - 該函式初始化群組中心，將其放置在影像的規則網格上。
   - `imageLAB` 是轉換為 Lab 色彩空間的輸入影像。
   - `centers` 是群組中心的結構體向量。
   - `len` 控制像素搜索區域的大小。

6. **`fituneCenter` 函式：**

- 該函式調整位於影像邊緣的群組中心的位置。
- `imageLAB` 是轉換為 Lab 色彩空間的輸入影像。
- `sobelGradient` 是影像的 Sobel 梯度。
- `centers` 是群組中心的結構體向量。

7. **`SLIC` 函式：**
- 該函式是 SLIC 算法的主要實現，包括初始化、clustering、update 等步驟。
- `image` 是原始影像。
- `resultLabel` 是最終的標籤結果。
- `centers` 是群組中心的結構體向量。
- `len` 控制像素搜索區域的大小。
- `m` 是控制距離和顏色相似性的權重。

8. **`SLIC_Demo` 函式：**
- 該函式是 SLIC 算法的演示。它調用 `SLIC` 函式，然後根據標籤結果生成最終的超像素結果。

9. **`on_Regional_Segmentation_pushButton_clicked` 函式：**
- 該函式是當按下 "Regional Segmentation" 按鈕時的事件處理程序。它檢查是否載入了原始影像，然後調用 `SLIC_Demo` 函式生成超像素結果，最後顯示在應用程式的 UI 中。