**Programming Language**

**Term Project(1322)**

수강과목 : 프로그래밍언어

수강번호 : 1322

조 번호 : 19조

제출일 : 2024.06.23.

전자공학과 22012007 김윤정

전자공학과 22311348 배민지

목차

1. 서론3
2. 프로젝트 개요3
3. 요구조건 분석3
4. 해결 방향3
5. 본론4
6. 클래스다이어그램4
7. 이미지 확대(Zoom In) 5
8. 이미지 회전(Rotation)8
9. 고화질 변경(Super Resolution) 12
10. 결론16
11. 부록17
12. 서론
13. 프로젝트 개요

이 프로젝트의 목적은 이미지 처리(Image Processing) 기술을 활용하여 디지털화된 영상 정보를 처리하고 변환하는 방법을 구현하고 분석하는 것입니다. 이를 통해 이미지 축소, 회전, 그리고 저해상도 영상을 고해상도로 변환하는 기술을 탐구합니다.

1. 요구조건 분석

디지털 이미지 처리에는 다양한 요구 조건들이 있습니다. 이 프로젝트에서는 다음과 같은 요구 조건을 분석하였습니다.

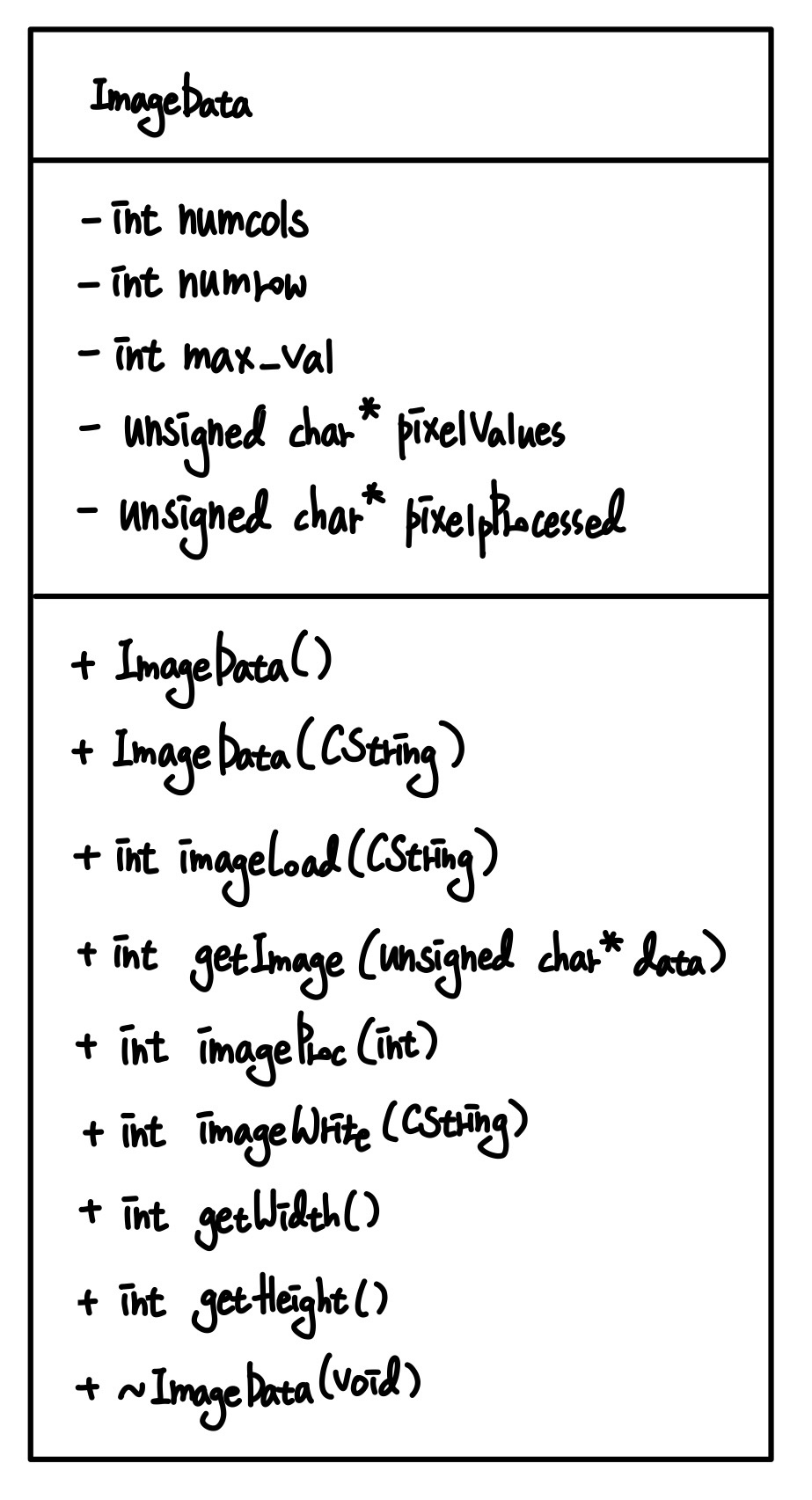
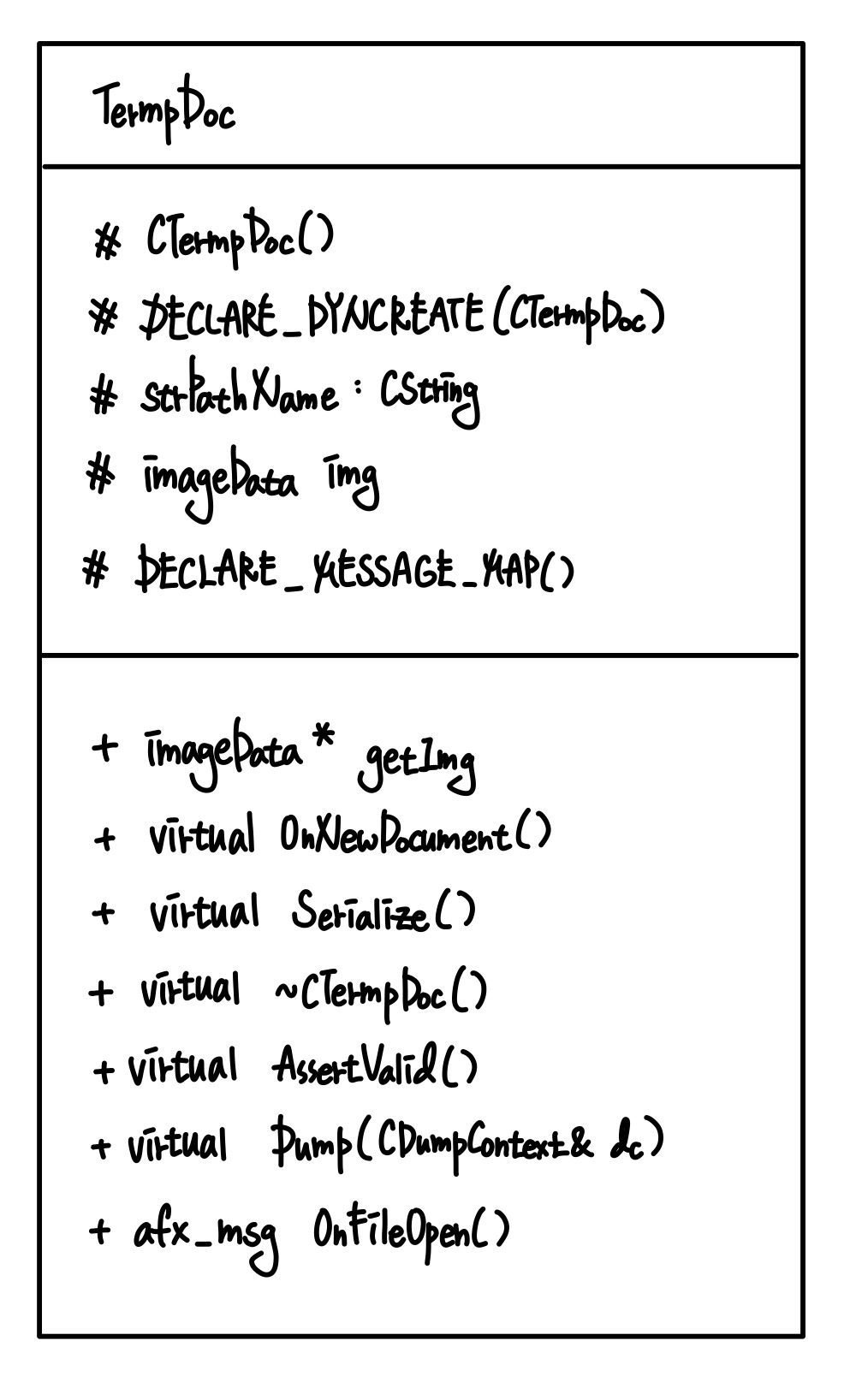
* 입력 영상 파일을 읽어 들이고 이를 처리할 수 있는 기능
* 영상의 축소와 회전을 통해 기하학적 변환을 수행할 수 있는 기능
* 저해상도의 영상을 고해상도로 변환할 수 있는 기능(ex: 명암비 조절 등)
* 처리된 영상을 저장할 수 있는 기능

1. 해결 방향

본 프로젝트에서는 다음과 같은 해결 방향을 제시합니다.

* 영상 입력: 디지털 이미지 파일을 읽어들입니다. 파일 포멧으로는 PGM(Portable GrayMap)형식을 사용합니다.
* 전처리: 입력된 영상 데이터를 구조화된 형태로 메모리에 저장합니다.
* 특수 추출: 영상의 픽셀 데이터를 분석하고 필요한 변환을 적용합니다.
* 인식 및 해석: 영상의 축소, 회전 및 명암비 조절 등의 변환을 통해 고해상도로 변환합니다.
* 출력: 변환된 영상을 파일로 저장합니다.

1. 본론
2. 클래스다이아그램



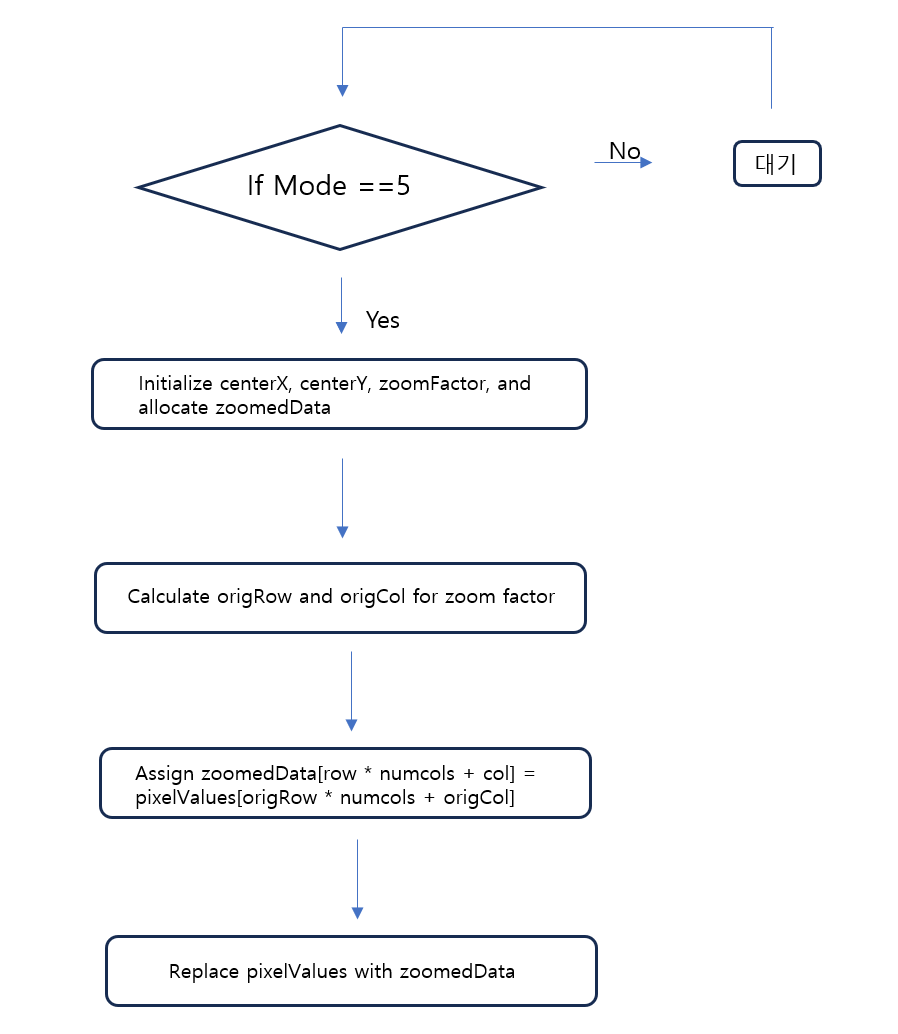
[그림1] classDiagram

Image Data 클래스는 이미지 데이터를 관리하고 변화하는 기능을 제공합니다. 먼저 멤버 변수를 소개하겠습니다. numcols는 이미지의 가로 픽셀 수, numrows는 이미지의 세로 픽셀 수, max\_val은 픽셀 값의 최대값(일반적으로 255), pixelValues는 픽셀 값을 저장하는 배열입니다. 생성자는 ImageData(CString)으로 이미지 파일을 로드하여 pixelValues 배열에 초기화합니다. 소멸자는 ~ImageData()로 동적 할당된 pixelValues 배열을 해제합니다. 마지막으로 메서드를 소개하겠습니다. int getWidth()는 이미지의 가로 크기를 반환, int getHeight()는 이미지의 세로 크기를 반환, unsigned char\* getPixelData()는 픽셀 데이터 배열을 반환합니다. void getImage(const char\* data)은 현재 이미지 데이터를 파일로 저장합니다.

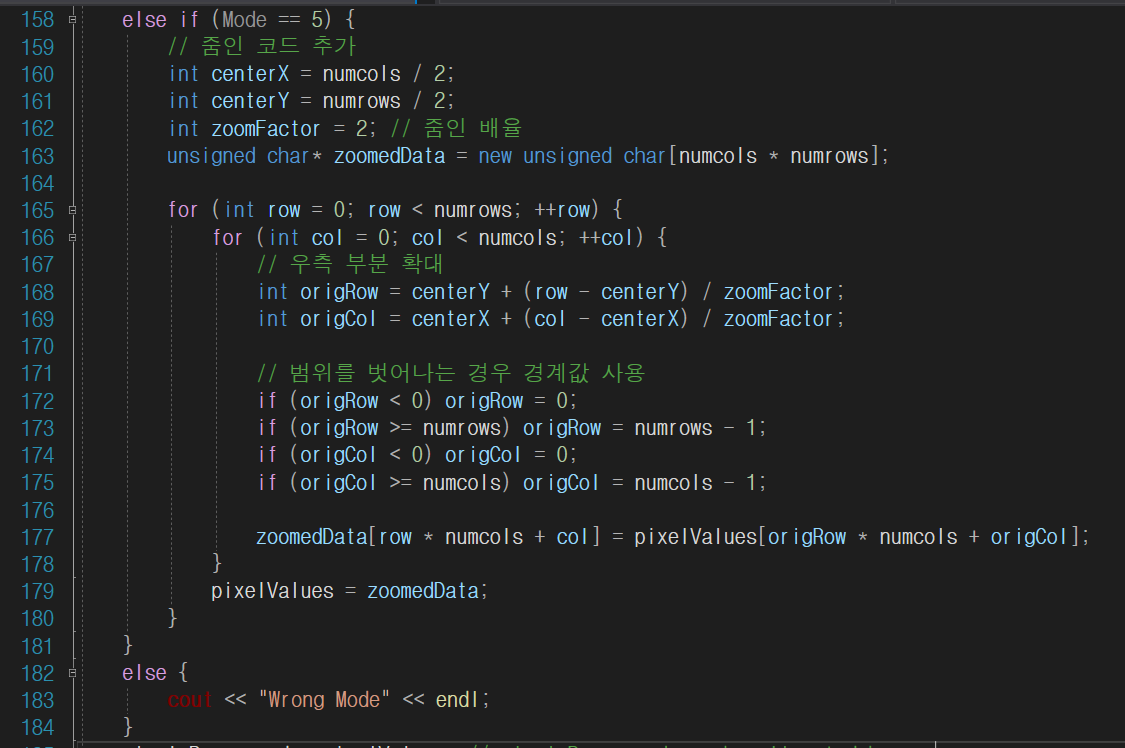
1. 이미지 확대(Zoom In)
2. 해결 방법

이미지 확대는 입력된 이미지를 일정 비율로 확대하는 과정입니다. 본 프로젝트에서는 2배 이상 확대를 구현하였습니다. 이미지의 특정 부분을 확대하여 새로운 이미지 데이터를 생성합니다.

1. 순서도

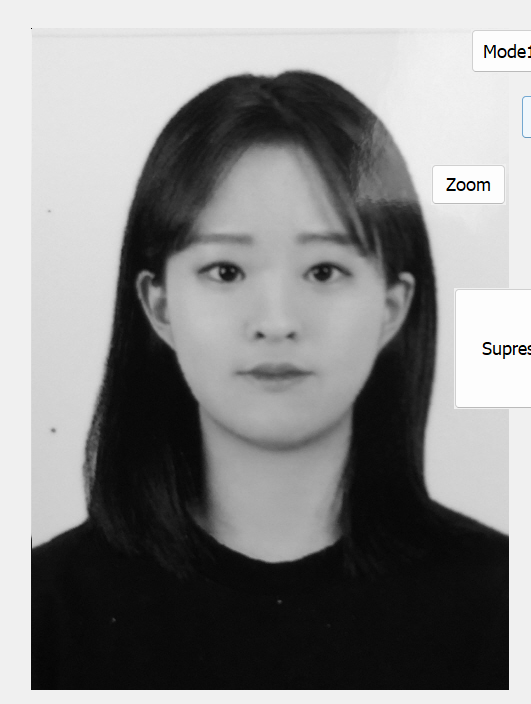


1. 프로그래밍 코드

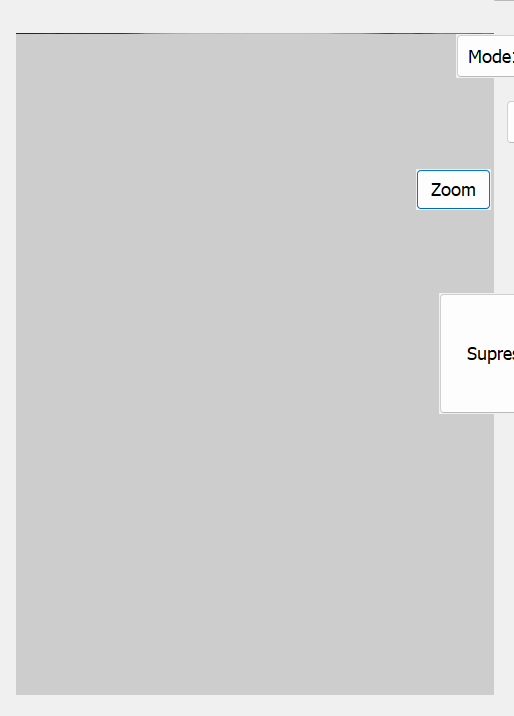


[그림2] 이미지 확대 코드

1. **중앙 좌표 계산 :** 이미지의 중앙 좌표를 계산합니다. 이는 이미지의 가로와 세로 길이를 반으로 나눈 값입니다.
2. **줌인 배율 설정 :** 확대할 배율을 설정합니다. 이 예제에서는 2배로 확대합니다.
3. **확대된 이미지 데이터를 저장할 배열 생성 :** 원본 이미지 크기와 동일한 크기의 새로운 배열을 생성합니다. 이 배열은 확대된 이미지 데이터를 저장합니다.
4. **각 픽셀에 대해 확대 계산 :** 원본 이미지의 각 픽셀을 순회하면서 확대된 이미지의 각 픽셀 값을 계산합니다.
5. **확대 계산 :** 확대된 이미지의 각 픽셀에 대응하는 원본 이미지의 중앙 부분 픽셀을 계산합니다. 중앙에서 각 픽셀의 거리를 줌인 배율로 나눠서 계산합니다.
6. **경계값 조정 :** 계산된 좌표가 이미지의 경계를 벗어나지 않도록 조정합니다. 예를 들어, 계산된 좌표가 0보다 작거나 이미지의 가로 또는 세로 길이보다 큰 경우 경계값으로 조정합니다.
7. **확대된 이미지 데이터 업데이트 :** 원본 이미지의 해당 픽셀 값을 확대된 이미지의 픽셀 값으로 설정합니다.
8. **확대된 이미지 데이터 적용 :** 확대된 이미지 데이터를 원본 이미지 데이터로 교체합니다.
9. 실험결과



[그림3] 원본



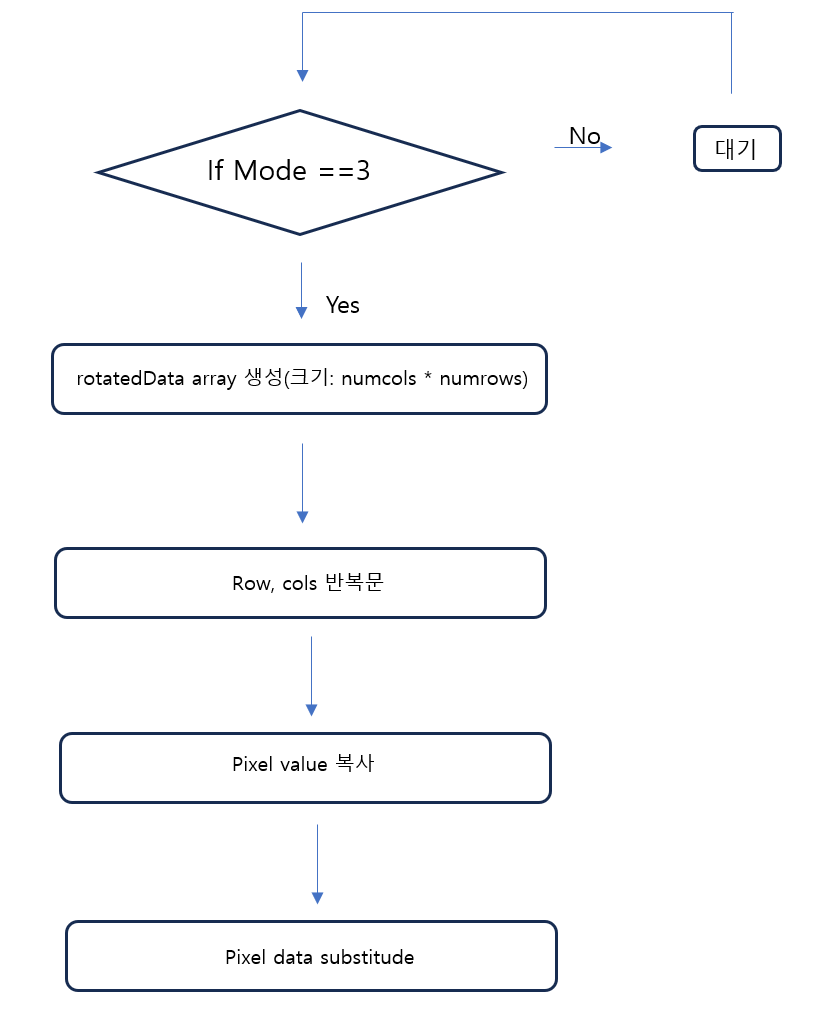
[그림4] 이미지 확대 실행 결과

이미지를 중앙을 기준으로 확대하는 과정으로 각 픽셀에 대해 중앙에서의 거리를 기반으로 계산하고, 경계값을 조정하여 안전하게 확대된 이미지를 생성한 후, 이를 원본 이미지 데이터로 교체하는 작업을 수행합니다.

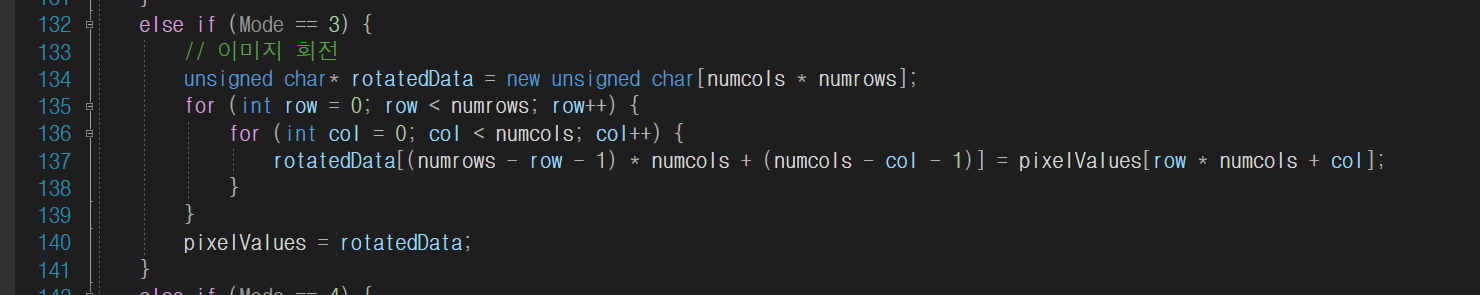
1. 이미지 회전(Rotation)
2. 해결 방법

이미지 회전은 이미지를 특정 각도로 회전시키는 과정입니다. 본 프로젝트에서는 180도의 회전을 구현하였습니다.

1. 순서도



1. 프로그래밍 코드

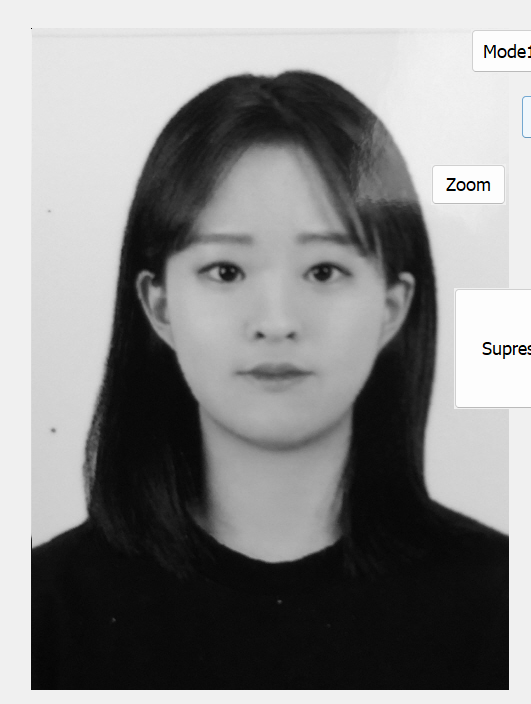


[그림6] 이미지 회전 핵심 코드(180도)

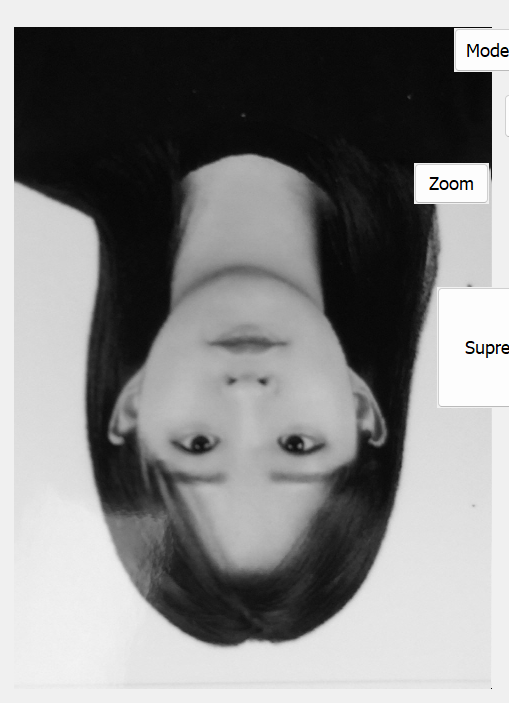
이미지를 180도 시계방향으로 회전합니다. 이를 위해 새로 회전된 이미지 데이터를 저장할 배열을 생성하고, 원본 이미지 데이터를 새 배열의 적절한 위치로 복사합니다. 이후 원본 데이터를 삭제하고 새 데이터로 대체합니다. 마지막으로 이미지의 너비와 높이를 교체하여 회전된 이미지의 올바른 크기를 설정합니다.

함수 rotateImage180은 이미지를 180도 회전합니다. 이중 for 루프를 사용하여 원본 이미지의 모든 픽셀을 순회합니다. (row, col) 위치의 픽셀을 (numrows-row-1, numcols-col-1) 위치로 이동합니다. 여기서 numrows-row-1은 행 인덱스를 반대로 계산하고, numcols-col-1은 열 인덱스를 반대로 계산합니다. 그리고 원본 배열을 삭제하고 새배열로 교체합니다. 너비와 높이는 그대로 유지됩니다.

1. 실험 결과



[그림7] 원본



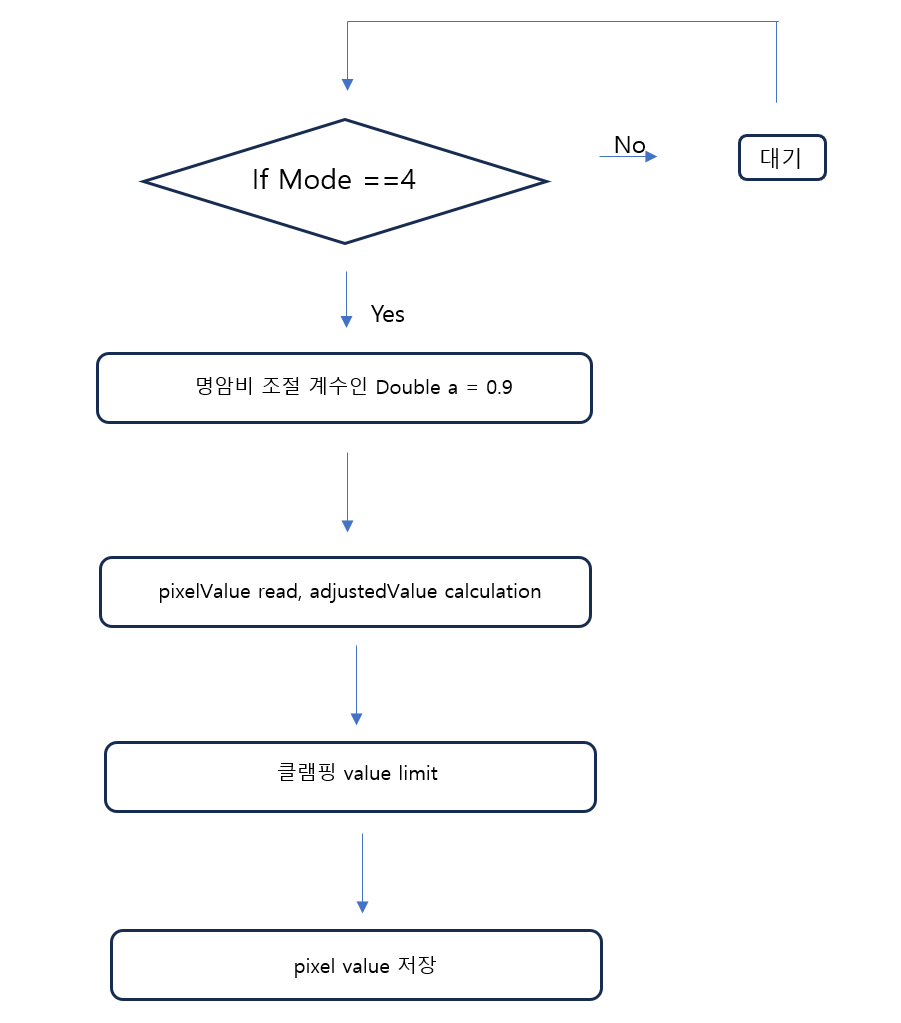
[그림8] 이미지 회전 실행 결과(180도)

이미지를 180도로 회전한 결과입니다. 결과를 통해 이미지 회전 과정을 시각적으로 이해할 수 있습니다.

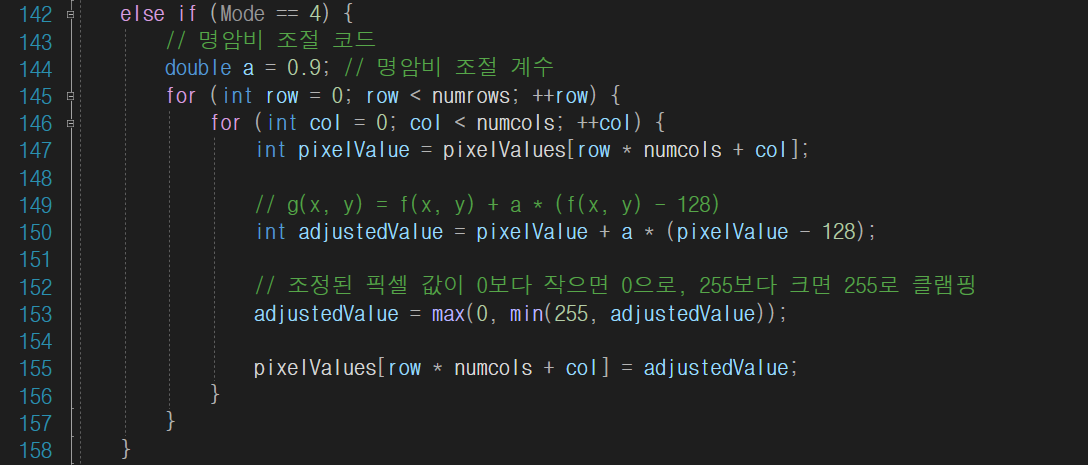
1. 고화질 변경(Super Resolution)
2. 해결 방법

고화질 변경은 저해상도 이미지를 고해상도로 변환하는 과정입니다. 이 기술은 디지털 영상 처리의 중요한 분야입니다. 이 과정에서 입력된 저해상도 영상을 전처리하고, 필요한 특성을 추출하여 영상을 인식 및 해석한 후, 최종적으로 고해상도 영상을 변환하는 과정을 포함합니다. 본 프로젝트에서는 명암비 조절을 통해 화질을 개선하는 기법을 사용하였습니다. 영상의 명암비를 조정하여 더 뚜렷하고 선명한 이미지를 생성하는 것을 목표로 합니다.

1. 순서도



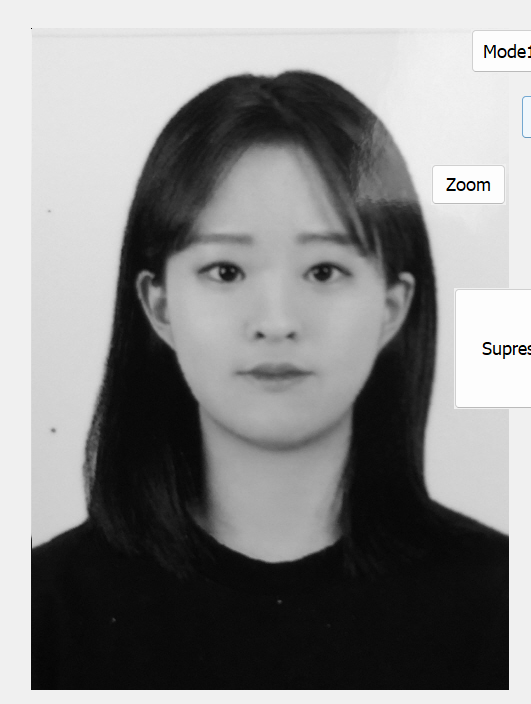
1. 프로그래밍 코드



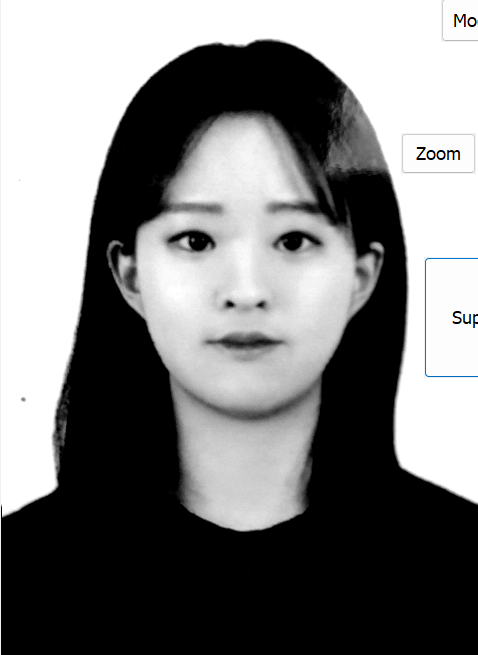
[그림9] 고화질 변경 핵심 코드1

이미지의 명암비를 조절하는 기능을 구현한다. for 루프를 사용하여 이미지 데이터의 모든 픽셀에 접근합니다. pixelValue 변수에 현재 픽셀의 값을 저장합니다. adjustedValue 변수에 새로 조정된 픽셀값을 계산합니다. 주어진 계수 a를 사용하여 명암비를 조정합니다. 공식은 g(x, y)=f(x, y)+a\*(f(x, y)-128)으로 표현됩니다. 여기서 f(x, y)는 원래 픽셀값을 나타내며, g(x, y)는 조정된 픽셀값(adjustedValue)이다. 계산된 adjustedValue 값을 0과 255 사이의 값으로 제한합니다.(클램핑) adjustedValue=max(0, min(255, adjustedValue)); 를 사용하여 조정된 픽셀 값이 0보다 작으면 0으로, 255보다 크면 255로 제한합니다. Double a는 명암비를 조절하기 위한 계수이다. 여기서 1.3을 사용하여 현재 이미지의 명암비를 1.3배로 조정하고 있다. 예를 들어, 픽셀 값이 128인 경우, adjustedValue=128+1.3\*(128-128)=128이 되어 변화가 없지만, 더 어두운 픽셀은 더 어둡게, 더 밝은 픽셀은 더 밝게 조정될 수 있습니다.

1. 실행 결과



[그림11] 원본(저해상도)



[그림12] 고화질 변경 실행 결과(고해상도)

이미지를 저해상도에서 고해상도로 변경한 결과입니다. 원본 이미지와 명암비가 조정된 이미지의 시각적인 차이를 확인할 수 있습니다. 명암비를 순서대로 0.7, 1.0, 1.3, 1.5로 변경해봄으로써 적절한 값을 찾을 수 있었습니다.



[그림13] 고화질 변경 실행 결과(좌:0.7, 우:1.0)



[그림14] 고화질 변경 실행 결과(좌:1.3, 우:1.5)

본론 요약 : 최종적으로 기존의 imageClass의 Mode0, Mode1, Mode2를 구현하고 unique image processing으로 이미지 확대와 이미지 회전을 선택했습니다. 시뮬레이터 클래스로는 명암비 조절을 선택했습니다. TermpDoc.cpp파일에 모두 다 구현을 하였기에 전체적인 순서를 말씀드리겠습니다.

우선 TermView.cpp 파일에 BEGIN\_MESSAGE\_MAP을 지정되고 CTermpView가 생성/소멸됩니다. 아래 코드를 바탕으로 ImageDisplay함수 안에 컨트롤 알림 처리기 코드를 추가함으로써 이미지 데이터를 디스플레이 가능하도록 설정합니다. 그리고 WM\_PAINT message를 활성화시켜줍니다. 그리고 CTermpView::OnPaint()를 통해 이미지를 디스플레이합니다.

void CTermpView::OnBnClickedImageDisplay()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, image display button"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

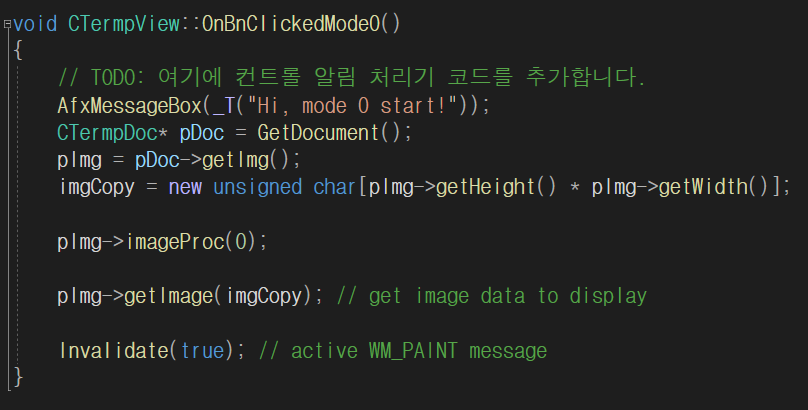
imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

그리고 OnBnClickedMode0() 부분에 메시지 박스를 출력하고 imgCopy를 동적메모리할당시켜 getImage의 인자로 입력됩니다. 그리고 그 사이에 모드를 인자로 가지는 imageProc(Mode)를 추가시켜줍니다. 아래를 참조하면 보기 쉽습니다.



나머지도 다 동일한 방식으로 적용했습니다.

두번째로, TermpDoc.cpp 파일에서 OnFileOpen()함수를 추가해줌으로써 내가 원하는 파일을 열 수 있도록 합니다. 그리고 getimg도 추가해 img의 주소를 리턴하도록 합니다.

세번째로, imageData.h 파일에서 이미지 파일을 로드하고, 다양한 이미지 처리 기능을 수행합니다. 처리된 이미지를 파일로 저장합니다.

생성자 (imageData::imageData() 및 imageData::imageData(CString fname))에서 이미지 파일을 로드하고 클래스 멤버 변수에 데이터를 저장합니다. 이미지 파일이 "P5" 포맷인지 확인하고, 데이터를 읽어와 pixelValues 배열에 저장합니다. originalPixelValues 배열에도 데이터를 복사하여 원본 이미지 데이터를 보존합니다.

이미지 로드 (imageData::imageLoad(CString fname))에서 이미지 파일을 로드하여 클래스 멤버 변수에 데이터를 저장합니다. pixelValues 배열에는 이미지 데이터가 저장되고, originalPixelValues 배열에는 원본 이미지 데이터가 보존됩니다.

이미지 가져오기 (imageData::getImage(unsigned char\* data))에서는 pixelValues 배열의 이미지 데이터를 복사하여 제공된 data 배열에 전달합니다. 이미지 폭 및 높이 얻기 (imageData::getWidth() 및 imageData::getHeight()) : 이미지의 폭과 높이를 반환합니다.

이미지 처리 (imageData::imageProc(int Mode))에서 다양한 이미지 처리 모드에 따라 이미지를 처리합니다. Mode에 따라 이미지를 반전시키거나, 이진화하거나, 명암비를 조절하거나, 회전하거나, 줌인하는 등의 처리를 수행합니다. 처리된 이미지 데이터는 pixelValues에 저장됩니다.

이미지 저장 (imageData::imageWrite(CString fname))에서는 처리된 이미지 데이터를 파일로 저장합니다. "P5" 포맷의 이미지 헤더를 작성하고, pixelpProcessed 배열의 데이터를 파일에 기록합니다.

소멸자 (imageData::~imageData())에서 동적으로 할당된 메모리를 해제합니다 (pixelValues 및 originalPixelValues).

이를 바탕으로 이미지 데이터를 로드하고, 다양한 이미지 처리를 수행하며, 전체적으로 보면 처리된 이미지를 파일로 저장하는 기능을 제공하는 imageData 클래스를 정의했습니다. 각 함수는 클래스의 멤버 함수로 구현되어 있으며, 이미지 데이터의 처리를 목적으로 합니다. 감사합니다.

1. 결론

본 프로젝트에서는 이미지 확대, 회전, 그리고 저해상도 이미지를 고해상도로 변환하는 방법을 구현하였습니다. 이를 통해 디지털 이미지 처리가 다양한 분야에서 어떻게 활용될 수 있는지 이해할 수 있었습니다.

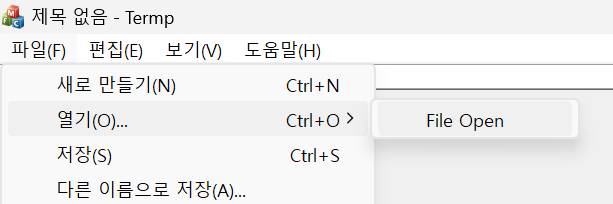
첫 번째, 이미지 축소 프로세스를 통해 주어진 이미지의 크기를 줄이고, 데이터 처리의 기술적인 측면을 탐구했습니다. 주어진 이미지를 2배 확대한 결과, 원본 이미지와 비교하여 크기가 확대되었음을 확인할 수 있습니다. 이미지 데이터를 효율적으로 처리하기 위해 메모리 할당 및 해제 과정에서 안정성을 확보했습니다. 또한 이미지의 특정 부분을 확대하여 새로운 이미지 데이터를 생성하도록 설계했습니다.

두 번째, 이미지 회전 프로세스를 통해 이미지 처리 기술의 중요성과 다양한 응용 가능성을 탐구했습니다. 주어진 이미지를 시계 방향으로 180도를 회전한 결과, 회전한 이미지는 정확히 변환되었음을 확인할 수 있습니다. 각 픽셀의 위치를 새로운 배열에 맞추어 재배열하고, 이 과정에서 원본 이미지의 데이터를 정확하게 반영하며, 회전된 이미지의 너비와 높이가 적절하게 업데이트되어 저장됩니다. 또한, 주어진 이미지를 어느 각도에서나 회전하는 경우에도 동일한 원리가 적용되었습니다.

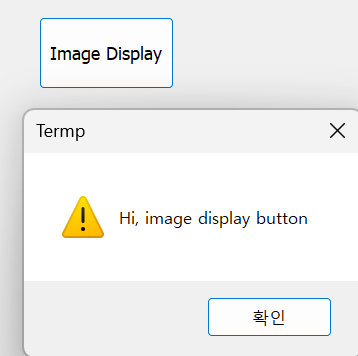
마지막, 저해상도 이미지를 고해상도로 변환하기 위해 명암비 조절 기술을 사용했습니다. 해상도 개선은 대부분 딥러닝과 관련이 있지만 딥러닝을 사용하지 않고도 C++ 코드만으로 구현할 방법을 찾다가 명암비 조절 기술을 알게되었습니다. 변경된 고해상도 이미지는 원본 이미지와 비교하여 시각적으로 세부 사항을 더욱 세밀하게 표현됩니다. 이는 데이터 처리 과정에서의 정밀성과 효율성을 고려했기 때문입니다. 향후에는 더 고도화된 기법, 예를 들어 딥러닝 기반의 초해상도(슈퍼 해상도) 기술을 적용하여 저해상도 이미지를 고해상도로 변환하는 연구를 진행할 수 있으면 좋겠습니다. 더 자연스럽고 디테일이 높은 고해상도 이미지를 생성하는 데 기여할 것으로 보입니다.

* 1. 부록\_프로그램 사용매뉴얼

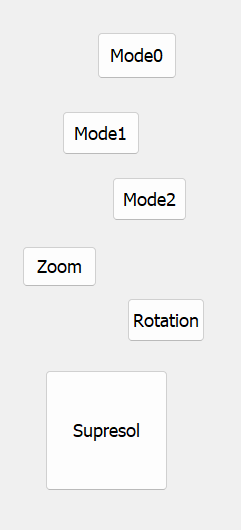
1. 우선 파일 열기로 File Open을 합니다.



1. Image Display를 클릭해 원본 이미지 파일을 불러옵니다.



1. 자기가 선택하고 싶은 버튼을 클릭합니다.



* 1. 부록 소스코드 첨부

// imageData.cpp 파일

//#include "stdafx.h"

#include "pch.h"

#include "imageData.h"

#include <string>

#include <iostream>

#include <fstream>

using namespace std;

// define class member functions

imageData::imageData()

{

AfxMessageBox(\_T("Default image constructor"));

}

imageData::imageData(CString fname)

{

CString msg;

ifstream infile(fname, ios::in | ios::binary);

string inputLine = "";

getline(infile, inputLine);

if (inputLine.compare("P5") != 0)

{

//cerr << "Version error" << endl;

msg.Format(\_T("Version error"));

AfxMessageBox(msg);

}

else

{

CA2CT str(inputLine.c\_str()); // type conversion

AfxMessageBox(str);

}

infile >> numcols >> numrows;

infile >> max\_val;

pixelValues = new unsigned char[numrows\*numcols]; // for data read

originalPixelValues = new unsigned char[numrows \* numcols]; // for storing the original data

infile.read(reinterpret\_cast<char\*>(pixelValues), numrows\*numcols \* sizeof(unsigned char));

memcpy(originalPixelValues, pixelValues, numrows \* numcols \* sizeof(unsigned char)); // copy original data

infile.close();

}

int imageData::imageLoad(CString fname)

{

CString msg;

ifstream infile(fname, ios::in | ios::binary);

string inputLine = "";

getline(infile, inputLine);

if (inputLine.compare("P5") != 0)

{

//cerr << "Version error" << endl;

msg.Format(\_T("Version error"));

AfxMessageBox(msg);

}

else

{

CA2CT str(inputLine.c\_str()); // type conversion

AfxMessageBox(str);

}

infile >> numcols >> numrows;

infile >> max\_val;

pixelValues = new unsigned char[numrows\*numcols]; // for data read

originalPixelValues = new unsigned char[numrows \* numcols]; // for storing the original data

infile.read(reinterpret\_cast<char\*>(pixelValues), numrows\*numcols \* sizeof(unsigned char));

if(infile) //data load check

AfxMessageBox(\_T("Image data loaded successfully!"));

memcpy(originalPixelValues, pixelValues, numrows \* numcols \* sizeof(unsigned char)); // copy original data

infile.close();

return(1);

}

int imageData::getImage(unsigned char\* data)

{

// copy image data

for (int i = 0; i < numcols\*numrows; i++)

data[i] = pixelValues[i];

return(1);

}

int imageData::getWidth()

{

return numcols;

}

int imageData::getHeight()

{

return numrows;

}

// define a destructor function

imageData::~imageData()

{

delete(pixelValues);

delete(originalPixelValues);

}

int imageData::imageProc(int Mode) {

memcpy(pixelValues, originalPixelValues, numrows \* numcols \* sizeof(unsigned char));

if (Mode == 1) {

for (rows = 0; rows < numrows; ++rows) {

for (cols = 0; cols < numcols; ++cols) {

cout << (int)pixelValues[rows\*numcols + cols] << ' ';

pixelValues[rows\*numcols + cols] = 255 - pixelValues[rows\*numcols + cols];

}

cout << rows << endl;

}

}

else if (Mode == 0) {

for (rows = 0; rows < numrows; ++rows) {

for (cols = 0; cols < numcols; ++cols) {

cout << (int)pixelValues[rows\*numcols + cols] << ' ';

if (pixelValues[rows\*numcols + cols] > 100) {

pixelValues[rows\*numcols + cols] = 255;

}

else {

pixelValues[rows\*numcols + cols] = 0;

}

}

cout << rows << endl;

}

}

else if (Mode == 2) {

for (rows = 0; rows < numrows; ++rows) {

for (cols = 0; cols < numcols; ++cols) {

cout << (int)pixelValues[rows\*numcols + cols] << ' ';

if (pixelValues[rows\*numcols + cols] > 200) {

pixelValues[rows\*numcols + cols] = 255;

}

else {

pixelValues[rows\*numcols + cols] = 0;

}

}

cout << rows << endl;

}

}

else if (Mode == 3) {

// 이미지 회전

unsigned char\* rotatedData = new unsigned char[numcols \* numrows];

for (int row = 0; row < numrows; row++) {

for (int col = 0; col < numcols; col++) {

rotatedData[(numrows - row - 1) \* numcols + (numcols - col - 1)] = pixelValues[row \* numcols + col];

}

}

pixelValues = rotatedData;

}

else if (Mode == 4) {

// 명암비 조절 코드

double a = 0.9; // 명암비 조절 계수

for (int row = 0; row < numrows; ++row) {

for (int col = 0; col < numcols; ++col) {

int pixelValue = pixelValues[row \* numcols + col];

// g(x, y) = f(x, y) + a \* (f(x, y) - 128)

int adjustedValue = pixelValue + a \* (pixelValue - 128);

// 조정된 픽셀 값이 0보다 작으면 0으로, 255보다 크면 255로 클램핑

adjustedValue = max(0, min(255, adjustedValue));

pixelValues[row \* numcols + col] = adjustedValue;

}

}

}

else if (Mode == 5) {

// 줌인 코드 추가

int centerX = numcols / 2;

int centerY = numrows / 2;

int zoomFactor = 2; // 줌인 배율

unsigned char\* zoomedData = new unsigned char[numcols \* numrows];

for (int row = 0; row < numrows; ++row) {

for (int col = 0; col < numcols; ++col) {

// 우측 부분 확대

int origRow = centerY + (row - centerY) / zoomFactor;

int origCol = centerX + (col - centerX) / zoomFactor;

// 범위를 벗어나는 경우 경계값 사용

if (origRow < 0) origRow = 0;

if (origRow >= numrows) origRow = numrows - 1;

if (origCol < 0) origCol = 0;

if (origCol >= numcols) origCol = numcols - 1;

zoomedData[row \* numcols + col] = pixelValues[origRow \* numcols + origCol];

}

pixelValues = zoomedData;

}

}

else {

cout << "Wrong Mode" << endl;

}

pixelpProcessed = pixelValues; // pixelpProcessed can be allocated by new

return 0;

}

int imageData::imageWrite(CString fname)

{

ofstream ofp;

ofp.open(fname, ios::binary);

if (!ofp) {

cout << "Can't open file: " << fname << endl;

exit(1);

}

// write image header

ofp << "P5 " << endl;

ofp << numcols << " " << numrows << endl;

ofp << max\_val << endl;

ofp.write(reinterpret\_cast<const char\*>(pixelpProcessed), numcols\*numrows);

if (ofp.fail()) {

cout << "Can't write image " << fname << endl;

exit(0);

}

ofp.close();

return 0;

}

//TermpDoc.cpp 파일

// TermpDoc.cpp: CTermpDoc 클래스의 구현

//

#include "pch.h"

#include "framework.h"

// SHARED\_HANDLERS는 미리 보기, 축소판 그림 및 검색 필터 처리기를 구현하는 ATL 프로젝트에서 정의할 수 있으며

// 해당 프로젝트와 문서 코드를 공유하도록 해 줍니다.

#ifndef SHARED\_HANDLERS

#include "Termp.h"

#endif

#include "TermpDoc.h"

#include <propkey.h>

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CTermpDoc

IMPLEMENT\_DYNCREATE(CTermpDoc, CDocument)

BEGIN\_MESSAGE\_MAP(CTermpDoc, CDocument)

ON\_COMMAND(ID\_32771, &CTermpDoc::OnFileOpen)

END\_MESSAGE\_MAP()

// CTermpDoc 생성/소멸

CTermpDoc::CTermpDoc() noexcept

{

// TODO: 여기에 일회성 생성 코드를 추가합니다.

}

CTermpDoc::~CTermpDoc()

{

}

BOOL CTermpDoc::OnNewDocument()

{

if (!CDocument::OnNewDocument())

return FALSE;

// TODO: 여기에 재초기화 코드를 추가합니다.

// SDI 문서는 이 문서를 다시 사용합니다.

return TRUE;

}

// CTermpDoc serialization

void CTermpDoc::Serialize(CArchive& ar)

{

if (ar.IsStoring())

{

// TODO: 여기에 저장 코드를 추가합니다.

}

else

{

// TODO: 여기에 로딩 코드를 추가합니다.

}

}

#ifdef SHARED\_HANDLERS

// 축소판 그림을 지원합니다.

void CTermpDoc::OnDrawThumbnail(CDC& dc, LPRECT lprcBounds)

{

// 문서의 데이터를 그리려면 이 코드를 수정하십시오.

dc.FillSolidRect(lprcBounds, RGB(255, 255, 255));

CString strText = \_T("TODO: implement thumbnail drawing here");

LOGFONT lf;

CFont\* pDefaultGUIFont = CFont::FromHandle((HFONT) GetStockObject(DEFAULT\_GUI\_FONT));

pDefaultGUIFont->GetLogFont(&lf);

lf.lfHeight = 36;

CFont fontDraw;

fontDraw.CreateFontIndirect(&lf);

CFont\* pOldFont = dc.SelectObject(&fontDraw);

dc.DrawText(strText, lprcBounds, DT\_CENTER | DT\_WORDBREAK);

dc.SelectObject(pOldFont);

}

// 검색 처리기를 지원합니다.

void CTermpDoc::InitializeSearchContent()

{

CString strSearchContent;

// 문서의 데이터에서 검색 콘텐츠를 설정합니다.

// 콘텐츠 부분은 ";"로 구분되어야 합니다.

// 예: strSearchContent = \_T("point;rectangle;circle;ole object;");

SetSearchContent(strSearchContent);

}

void CTermpDoc::SetSearchContent(const CString& value)

{

if (value.IsEmpty())

{

RemoveChunk(PKEY\_Search\_Contents.fmtid, PKEY\_Search\_Contents.pid);

}

else

{

CMFCFilterChunkValueImpl \*pChunk = nullptr;

ATLTRY(pChunk = new CMFCFilterChunkValueImpl);

if (pChunk != nullptr)

{

pChunk->SetTextValue(PKEY\_Search\_Contents, value, CHUNK\_TEXT);

SetChunkValue(pChunk);

}

}

}

#endif // SHARED\_HANDLERS

// CTermpDoc 진단

#ifdef \_DEBUG

void CTermpDoc::AssertValid() const

{

CDocument::AssertValid();

}

void CTermpDoc::Dump(CDumpContext& dc) const

{

CDocument::Dump(dc);

}

#endif //\_DEBUG

// CTermpDoc 명령

void CTermpDoc::OnFileOpen()

{

// TODO: 여기에 명령 처리기 코드를 추가합니다.

CString szFilter("Image (\*.pgm)|\*.pgm");

CFileDialog dlg(TRUE, NULL, NULL, OFN\_HIDEREADONLY, szFilter);

if (IDOK == dlg.DoModal())

{

strPathName = dlg.GetPathName();

}

img.imageLoad(strPathName); // load image

}

imageData\* CTermpDoc::getImg()

{

return &img;

}

// TermpView.cpp 파일

// TermpView.cpp: CTermpView 클래스의 구현

//

#include "pch.h"

#include "framework.h"

// SHARED\_HANDLERS는 미리 보기, 축소판 그림 및 검색 필터 처리기를 구현하는 ATL 프로젝트에서 정의할 수 있으며

// 해당 프로젝트와 문서 코드를 공유하도록 해 줍니다.

#ifndef SHARED\_HANDLERS

#include "Termp.h"

#endif

#include "TermpDoc.h"

#include "TermpView.h"

#ifdef \_DEBUG

#define new DEBUG\_NEW

#endif

// CTermpView

IMPLEMENT\_DYNCREATE(CTermpView, CFormView)

BEGIN\_MESSAGE\_MAP(CTermpView, CFormView)

// 표준 인쇄 명령입니다.

ON\_COMMAND(ID\_FILE\_PRINT, &CFormView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_DIRECT, &CFormView::OnFilePrint)

ON\_COMMAND(ID\_FILE\_PRINT\_PREVIEW, &CFormView::OnFilePrintPreview)

ON\_BN\_CLICKED(IDC\_IMAGE\_DISPLAY, &CTermpView::OnBnClickedImageDisplay)

ON\_WM\_PAINT()

ON\_BN\_CLICKED(IDC\_MODE0, &CTermpView::OnBnClickedMode0)

ON\_BN\_CLICKED(IDC\_MODE1, &CTermpView::OnBnClickedMode1)

ON\_BN\_CLICKED(IDC\_MODE2, &CTermpView::OnBnClickedMode2)

ON\_BN\_CLICKED(IDC\_ROTATION, &CTermpView::OnBnClickedRotation)

ON\_BN\_CLICKED(IDC\_SUPRESOL, &CTermpView::OnBnClickedSupresol)

ON\_BN\_CLICKED(IDC\_ZOOM, &CTermpView::OnBnClickedZoom)

END\_MESSAGE\_MAP()

// CTermpView 생성/소멸

CTermpView::CTermpView() noexcept

: CFormView(IDD\_TERMP\_FORM)

{

// TODO: 여기에 생성 코드를 추가합니다.

}

CTermpView::~CTermpView()

{

if (imgCopy) // delete new object

delete[] imgCopy;

}

void CTermpView::DoDataExchange(CDataExchange\* pDX)

{

CFormView::DoDataExchange(pDX);

}

BOOL CTermpView::PreCreateWindow(CREATESTRUCT& cs)

{

// TODO: CREATESTRUCT cs를 수정하여 여기에서

// Window 클래스 또는 스타일을 수정합니다.

return CFormView::PreCreateWindow(cs);

}

void CTermpView::OnInitialUpdate()

{

CFormView::OnInitialUpdate();

GetParentFrame()->RecalcLayout();

ResizeParentToFit();

}

// CTermpView 인쇄

BOOL CTermpView::OnPreparePrinting(CPrintInfo\* pInfo)

{

// 기본적인 준비

return DoPreparePrinting(pInfo);

}

void CTermpView::OnBeginPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: 인쇄하기 전에 추가 초기화 작업을 추가합니다.

}

void CTermpView::OnEndPrinting(CDC\* /\*pDC\*/, CPrintInfo\* /\*pInfo\*/)

{

// TODO: 인쇄 후 정리 작업을 추가합니다.

}

void CTermpView::OnPrint(CDC\* pDC, CPrintInfo\* /\*pInfo\*/)

{

// TODO: 여기에 사용자 지정 인쇄 코드를 추가합니다.

}

// CTermpView 진단

#ifdef \_DEBUG

void CTermpView::AssertValid() const

{

CFormView::AssertValid();

}

void CTermpView::Dump(CDumpContext& dc) const

{

CFormView::Dump(dc);

}

CTermpDoc\* CTermpView::GetDocument() const // 디버그되지 않은 버전은 인라인으로 지정됩니다.

{

ASSERT(m\_pDocument->IsKindOf(RUNTIME\_CLASS(CTermpDoc)));

return (CTermpDoc\*)m\_pDocument;

}

#endif //\_DEBUG

// CTermpView 메시지 처리기

void CTermpView::OnBnClickedImageDisplay()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, image display button"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnPaint()

{

CPaintDC dc(this); // device context for painting

// TODO: 여기에 메시지 처리기 코드를 추가합니다.

// 그리기 메시지에 대해서는 CFormView::OnPaint()을(를) 호출하지 마십시오.

if (imgCopy) // if image data exists, display the image

{

// display image using SetPixel

unsigned char value;

for (int i = 0; i < pImg->getHeight(); i++)

for (int j = 0; j < pImg->getWidth(); j++)

{

int ypos = 150 + i;

int xpos = 200 + j;

value = imgCopy[i \* pImg->getWidth() + j]; // get image pixel value

dc.SetPixel(xpos, ypos, RGB(value, value, value));

}

}

}

void CTermpView::OnBnClickedMode0()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, mode 0 start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(0);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnBnClickedMode1()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, mode 1 start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(1);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnBnClickedMode2()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, mode 2 start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(2);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnBnClickedRotation()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, ratation start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(3);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnBnClickedSupresol()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, SuperResolution start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(4);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

void CTermpView::OnBnClickedZoom()

{

// TODO: 여기에 컨트롤 알림 처리기 코드를 추가합니다.

AfxMessageBox(\_T("Hi, Zoom start!"));

CTermpDoc\* pDoc = GetDocument();

pImg = pDoc->getImg();

imgCopy = new unsigned char[pImg->getHeight() \* pImg->getWidth()];

pImg->imageProc(5);

pImg->getImage(imgCopy); // get image data to display

Invalidate(true); // active WM\_PAINT message

}

// imageData.h

#pragma once

#include <string>

#include <fstream>

//#include<iostream>

using namespace std;

// declare a new class

class imageData

{

// declare class member variables

private:

int cols = 0;

int rows = 0;

int numcols = 0;

int numrows = 0;

int max\_val;

unsigned char\* pixelValues;

unsigned char\* originalPixelValues;

unsigned char\* pixelpProcessed;

// declare class member functions

public:

imageData();

imageData(CString);

int imageLoad(CString);

int getImage(unsigned char\* data);

int imageProc(int);

int imageWrite(CString);

int getWidth();

int getHeight();

~imageData();

};

//TermpDoc.h

#include "imageData.h"

// TermpDoc.h: CTermpDoc 클래스의 인터페이스

//

#pragma once

class CTermpDoc : public CDocument

{

protected: // serialization에서만 만들어집니다.

CTermpDoc() noexcept;

DECLARE\_DYNCREATE(CTermpDoc)

CString strPathName; // file name for image read

imageData img; // image object

public: imageData\* getImg(); // get image object pointer

// 특성입니다.

public:

// 작업입니다.

public:

// 재정의입니다.

public:

virtual BOOL OnNewDocument();

virtual void Serialize(CArchive& ar);

#ifdef SHARED\_HANDLERS

virtual void InitializeSearchContent();

virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);

#endif // SHARED\_HANDLERS

// 구현입니다.

public:

virtual ~CTermpDoc();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// 생성된 메시지 맵 함수

protected:

DECLARE\_MESSAGE\_MAP()

#ifdef SHARED\_HANDLERS

// 검색 처리기에 대한 검색 콘텐츠를 설정하는 도우미 함수

void SetSearchContent(const CString& value);

#endif // SHARED\_HANDLERS

public:

afx\_msg void OnFileOpen();

};

// TermpView.h

#include "imageData.h"

// TermpView.h: CTermpView 클래스의 인터페이스

//

#pragma once

class CTermpView : public CFormView

{

private:

imageData\* pImg = NULL; // for image display

unsigned char\* imgCopy = NULL;

protected: // serialization에서만 만들어집니다.

CTermpView() noexcept;

DECLARE\_DYNCREATE(CTermpView)

public:

#ifdef AFX\_DESIGN\_TIME

enum{ IDD = IDD\_TERMP\_FORM };

#endif

// 특성입니다.

public:

CTermpDoc\* GetDocument() const;

// 작업입니다.

public:

// 재정의입니다.

public:

virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

protected:

virtual void DoDataExchange(CDataExchange\* pDX); // DDX/DDV 지원입니다.

virtual void OnInitialUpdate(); // 생성 후 처음 호출되었습니다.

virtual BOOL OnPreparePrinting(CPrintInfo\* pInfo);

virtual void OnBeginPrinting(CDC\* pDC, CPrintInfo\* pInfo);

virtual void OnEndPrinting(CDC\* pDC, CPrintInfo\* pInfo);

virtual void OnPrint(CDC\* pDC, CPrintInfo\* pInfo);

// 구현입니다.

public:

virtual ~CTermpView();

#ifdef \_DEBUG

virtual void AssertValid() const;

virtual void Dump(CDumpContext& dc) const;

#endif

protected:

// 생성된 메시지 맵 함수

protected:

DECLARE\_MESSAGE\_MAP()

public:

afx\_msg void OnBnClickedImageDisplay();

afx\_msg void OnPaint();

afx\_msg void OnBnClickedMode0();

afx\_msg void OnBnClickedMode1();

afx\_msg void OnBnClickedMode2();

afx\_msg void OnBnClickedRotation();

afx\_msg void OnBnClickedSupresol();

afx\_msg void OnBnClickedZoom();

};

#ifndef \_DEBUG // TermpView.cpp의 디버그 버전

inline CTermpDoc\* CTermpView::GetDocument() const

{ return reinterpret\_cast<CTermpDoc\*>(m\_pDocument); }

#endif

// 소스코드는 수정한 부분만 첨부하였음.