# 디지털 영상처리 연구실 연구보고서

김우헌 23.08.03

## ##영상처리를 위한 mfc프로그램

- 프로젝트이름을 MFCApplication1으로 설정하고 mfc기반 프로젝트를 생성하였습니다.

```
void CMFCApplication1Doc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        ar.Write(m_outimg, 256 * 256);
        // TODO: 여기에 저장 코드를 추가합니다.
    }
    else
    {
        CFile* infile = ar.GetFile();
        if (infile->GetLength()!=256*256)
        {
            AfxMessageBox(_T("파일크기가 256*256이 아닙니다."));
            return;

        }
        ar.Read(m_inimage, infile->GetLength());
        // TODO: 여기에 로딩 코드를 추가합니다.
    }
}
```

- doc클래스에 serialize 함수를 이용하여 영상데이터를 읽어왔습니다.

- carchive클래스는 각종 외부 문서의 데이터를 읽고 입출력하는 기능을 가진 클래스입니다.

```
CMFCApplication1View::CMFCApplication1View() noexcept
{
    height = width = 256;
    int rwsize = (((width)+31) / 32 * 4);
    BmInfo = (BITMAPINFO*)malloc(sizeof(BITMAPINFO) + 256 * sizeof(RGBQUAD));

    BmInfo->bmiHeader.biBitCount = 8;
    BmInfo->bmiHeader.biClrImportant = 256;
    BmInfo->bmiHeader.biClrUsed = 256;
    BmInfo->bmiHeader.biCompression = 0;
    BmInfo->bmiHeader.biHeight = height;
    BmInfo->bmiHeader.biPlanes = 1;
    BmInfo->bmiHeader.biSize = 40;
    BmInfo->bmiHeader.biSizeImage = rwsize * height;
    BmInfo->bmiHeader.biWidth = width;
    BmInfo->bmiHeader.biXPelsPerMeter = 0;
    BmInfo->bmiHeader.biYPelsPerMeter = 0;
    for (int i = 0; i < 256; i++) // Palette number is 256
    {
        BmInfo->bmiColors[i].rgbRed = BmInfo->bmiColors[i].rgbGreen = BmInfo->bmiColors[i].rgbBlue = i;
        BmInfo->bmiColors[i].rgbReserved = 0;
    }
```

- view 클래스 생성자에 BITMAPINFO 클래스를 이용한 BmInfo변수를 사용하여 입출력할 비트맵 정보를 설정합니다.

```
void CMFCApplication1View::OnDraw(CDC* pDC)
{
    CMFCApplication1Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++) m_RevImg[i][j] = pDoc->m_inimage[height - i - 1][j];

    SetDIBitsToDevice(pDC->GetSafeHdc(), 0, 0, width, height,
        0, 0, 0, height, m_RevImg, BmInfo, DIB_RGB_COLORS);

    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++) m_RevImg[i][j] = pDoc->m_outimg[height - i - 1][j];

    SetDIBitsToDevice(pDC->GetSafeHdc(), 300, 0, width, height,
        0, 0, 0, height, m_RevImg, BmInfo, DIB_RGB_COLORS);
```

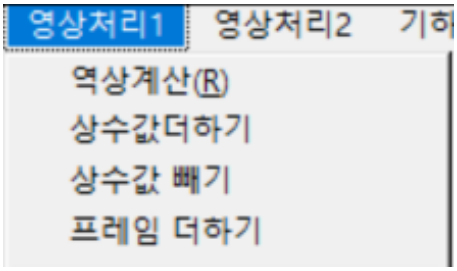
- view 클래스에 ondraw함수를 이용하여 입출력할 이미지를 디스플레이 하였습니다.
- 비트맵 방식의 이미지는 저장될 때 반대로 저장되기 때문에 m_revimg변수를 이용해서 이미지를 반전 시킨후에 setdibtodevice함수를 이용하여 inimage는 0,0에 outimage는 300,0에 디스플레잉 하였습니다.

## ##포인트 처리

- 포인트 처리란 수많은 픽셀들로 이루어진 영상에서 하나하나 단위픽셀 각각 독립적으로 연산하는 것입니다.

#상수값연산

- inimage[x][y]에 +c1,-c2,*c3,/c4를 하여 각각 픽셀에서 산술 연산이 가능합니다.




```
void CMFCApplication1View::OnConstAdd()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    CMFCApplication1Doc* pdoc = GetDocument();
    ASSERT_VALID(pdoc);
    if (!pdoc)
        return;

    for (int i = 0; i < 256; i++)
    {
        for (int j = 0; j < 256; j++) {
            int tempval = pdoc->m_inimage[i][j] - 60;
            tempval = tempval < 0 ? 0 : tempval;

            pdoc->m_outimg[i][j] = (unsigned char)tempval;

        }
    }
    Invalidate(FALSE);
```

- 상수값(60)을 빼는 코드를 작성하고 mainframe클래스에 메뉴를 작성하고 클래스마법사를 이용하여 mainframe 클래스를 호출하는 함수를 만들고 view클래스에 doc포인터를 얻은 후에 간단한 코드를 이용하여 픽셀 연산을 하였습니다.
- 조건 연산자(클립핑)을 이용하여 연산후에 픽셀값이 0~255사이의 값을 가지도록 설정하였습니다.
- image 는 전체적으로 픽셀값이 작아졌으므로 원영상에 비해 어두워진 것을 확인하였습니다.

- 상수값을 더하는 코드를 작성하여 image 를 얻었습니다. image 가 밝아진 효과가 나타났습니다.

#두영상 사이의 연산

- 앞서 단일 영상에 산술을 하였다면 두영상 사이의 산술연산또한 가능합니다. 이를 frame 연산이라하며 실용적으로 많이 응용되고 있습니다.

```
void CMFCApplication1Doc::twoimageload()
{
    // TODO: 여기에 구현 코드 추가.
    CFile file;
    CFileDialog open1(TRUE);
    if (open1.DoModal()==IDOK)
    {
        file.Open(open1.GetFileName(), CFile::modeRead);
        file.Read(m_inimage1, sizeof(m_inimage1));
        file.Close();
    }

    CFileDialog open2(TRUE); //공통 대화 상자(두번째 파일 오픈)

    if (open2.DoModal() == IDOK)
    {
        file.Open(open2.GetFileName(), CFile::modeRead);
        file.Read(m_inimage2, sizeof(m_inimage2));
        file.Close();
    }
}
```

- 두개의 image 를 불러 오기위해 doc클래스에 cfile클래스를 이용하여 m_inimage1과 m_inimage2를 입력하여 줍니다.

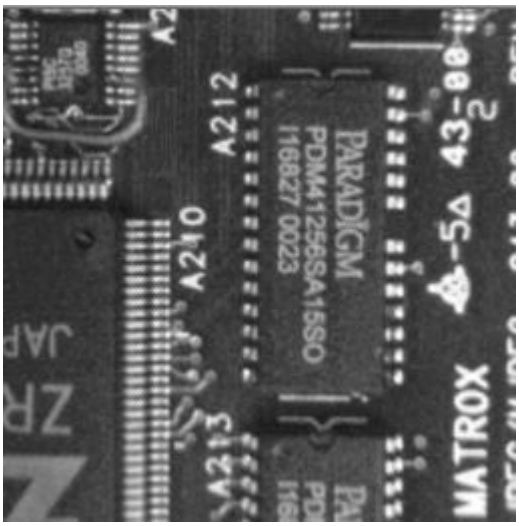
```
void CMFCApplication1View::OnFrmAdd()
{
    CMFCApplication1Doc* pdoc = GetDocument();
    ASSERT_VALID(pdoc);
    if (!pdoc)
        return;


    pdoc->twoimageload();                    // 빼기 연산을 할 두 영상을 입력한다.
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            int tempVal = pdoc->m_inimage1[i][j] + pdoc->m_inimage2[i][j];
            tempVal = tempVal > 255 ? 255 : tempVal;
            tempVal = tempVal < 0 ? 0 : tempVal;
            pdoc->m_outimg[i][j] = (unsigned char)tempVal;
        }
    }

    Invalidate(FALSE); //화면 갱신
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
}
```

- m_inimage1[i][j]+m_inimage2[i][j]에서 두개의 frame간 plus 연산이 실행되었습니다.

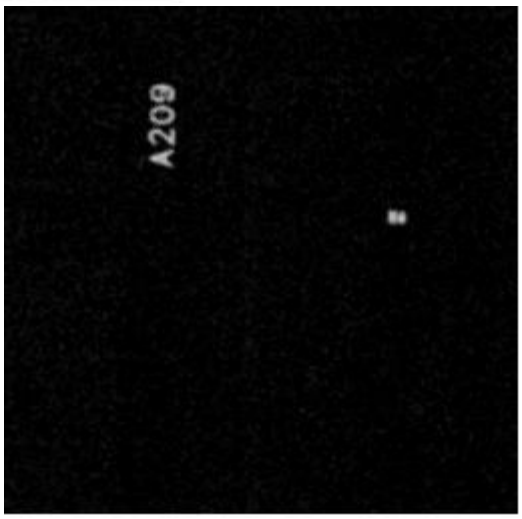
[redacted]

- Lena image 와 과일 이미지를 더하기 연산을 하여 두영상이 합쳐진 것과 같은 효과를 나타내고 전체적으로 image 가 밝아진 것을 확인하였습니다.


[redacted]




[redacted]




- image 1-image2를하여 픽셀 값에 차이가 있는 부분이 나타났습니다. 이를 통하여 결함검사,인쇄오류,납땜검사 등 여러 분야에서 응용 할 수 있습니다.

## ##히스토그램을 이용한 영상처리

#히스토그램?

- 히스토그램이란 영상의 밝기값에 대한 분포를 보여주는 그래프로서 영상의 밝기구성, 명암의 대비등 중요한 정보를 알 수 있습니다.
- 수평축은 밝기값을 나타내는 0-255이고 수직축은 각각의 밝기값이 나타난 픽셀의 빈도수입니다.

```
void CMFCApplication1Doc::m_imghisto(int height, int width)
{
    int i, j, vmax, vmin;
    for (i = 0; i < 256; i++) m_histoarr[i] = 0;

    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
            int gv = (int)m_inimage[i][j];
            m_histoarr[gv]++;
        }
    }

    // 히스토그램 크기 정규화
    vmin = 1000000; vmax = 0;
    for (i = 0; i < 256; i++)
    {
        if (m_histoarr[i] <= vmin) vmin = m_histoarr[i];
        if (m_histoarr[i] >= vmax) vmax = m_histoarr[i];
    }
    if (vmax == vmin) return;

    float vd = (float)(vmax - vmin);
    for (i = 0; i < 256; i++)
    {
        m_histoarr[i] = (int)(((float)m_histoarr[i] - vmin) * 255.0 / vd);
    }

    // 히스토그램의 출력
    for (i = 0; i < height; i++)
        for (j = 0; j < width; j++) m_outimg[i][j] = 255;//하얀색으로만듬

    for (j = 0; j < width; j++)
    {
        for (i = 0; i < m_histoarr[j]; i++) m_outimg[255 - i][j] = 0;
    }
```

- doc클래스에 변수 int heght,width를가진 m_histo함수를 추가한 후에 m_histoarr[gr]++을 이용하여 0-255사이의 x값인 gr변수로 y의값을 하나씩 더하는 방식입니다.
- 히스토그램의 정규화를 위하여 최소값과 최대값을 찾아준 후 m_histoarr값을 다시 설정한후 displaying 시켜줍니다.

```
void CMFCApplication1View::OnImghisto()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    CMFCApplication1Doc* pdoc = GetDocument();
    ASSERT_VALID(pdoc);

    pdoc->m_imghisto(256, 256);
    Invalidate(FALSE);
}
```

- doc클래스에 함수를 선언하였으므로 view클래스에서 doc클래스에 포인트를 얻어 pdoc 함수m_imhisto를 호출하고 invalidate함수를 사용하여 갱신 시켜줍니다.

> lena image histogram

#영상 이치화

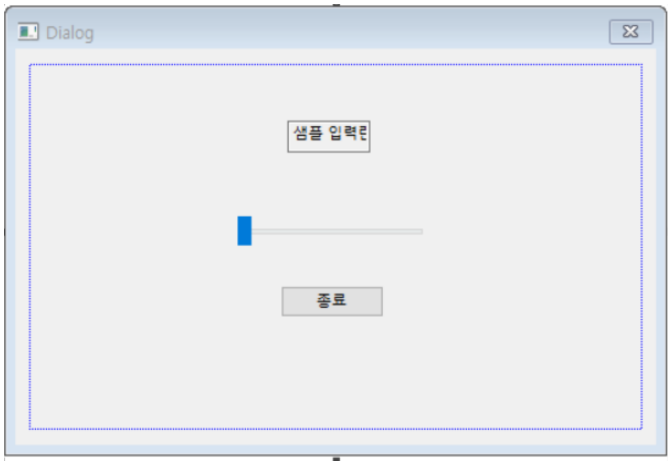
- 영상이치화란 픽셀기반 연산의 가장 단순한 예로 픽셀값을 어떤 기준(threshold)에 따라 0과 255두값중 하나로 바꾸는 것이 이치화 입니다.

```
void CMFCApplication1View::OnBinarization()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    CMFCApplication1Doc* pDoc = GetDocument(); // 다큐멘트 클래스를 참조하기 위해
    ASSERT_VALID(pDoc);                 // 인스턴스 주소를 가져옴

    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            if (pDoc->m_inimage[i][j] > 100) pDoc->m_outimg[i][j] = 255;
            else pDoc->m_outimg[i][j] = 0;
        }
    }
    Invalidate(FALSE); //화면 갱신
}
```

➔ 픽셀값 100을기준으로 100이상이면 255,100이하면 0이되는 코드와 이미지입니다.( threshold=100)

#동적이진화

-앞서 영상의 이진화를 위하여 미리 threshold 값을 설정하여 정적으로 이진화 하였지만 이번에는 동적으로 이진화가 가능한 기능을 실험해보았습니다.




- 새로운 dialog resourse 에 에디트컨트롤, 슬라이드 컨트롤, 버튼컨트롤을 추가하였습니다.
- 이후 이 dialog리소스를 기반으로한 새로운 cbincntrldlg 클래스를 생성하고 컨트롤 각각에 변수를 추가하였습니다.

```
BOOL CBINCTRLDLG::OnInitDialog()
{
    CDialogEx::OnInitDialog();


        m_ctrlslider.SetRange(0, 255);
    m_ctrlslider.SetPos(100);

    m_binval = m_ctrlslider.GetPos();
    UpdateData(FALSE);
    return TRUE;
```

```
void CBINCTRLDLG::OnClickedButton1()
{
    OnOK();
```

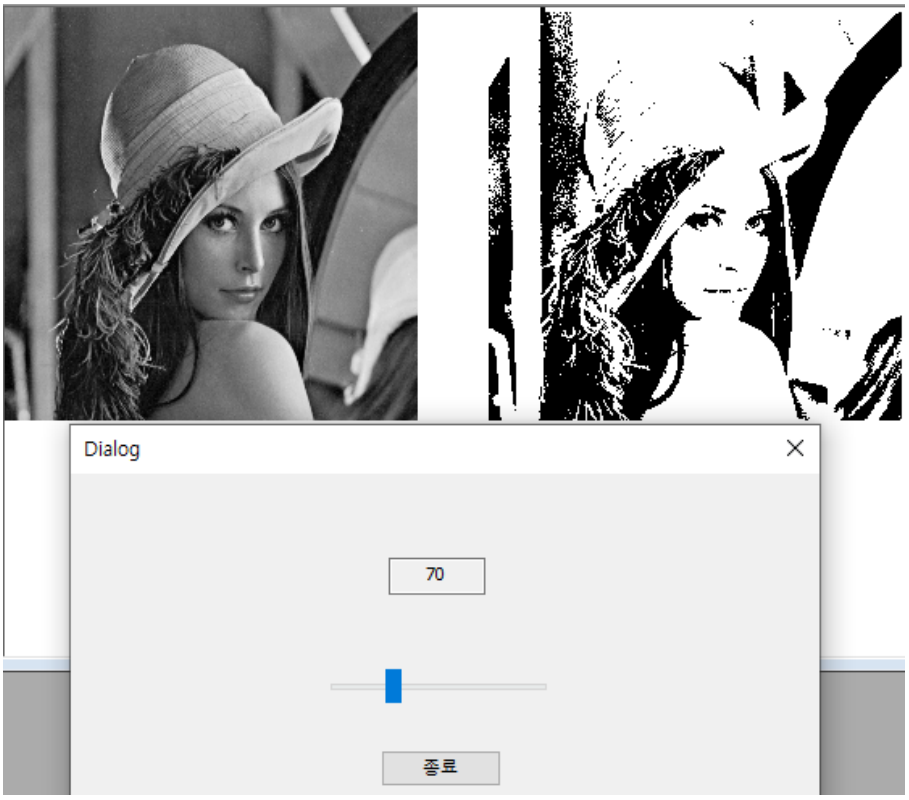
```
void CMFCApplication1Doc::m_binthres(int height, int width, int binthres)
{
    // TODO: 여기에 구현 코드 추가.

    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            if (m_inimage[i][j] > binthres) m_outimg[i][j] = 255;
            else m_outimg[i][j] = 0;
        }
    }
}
```

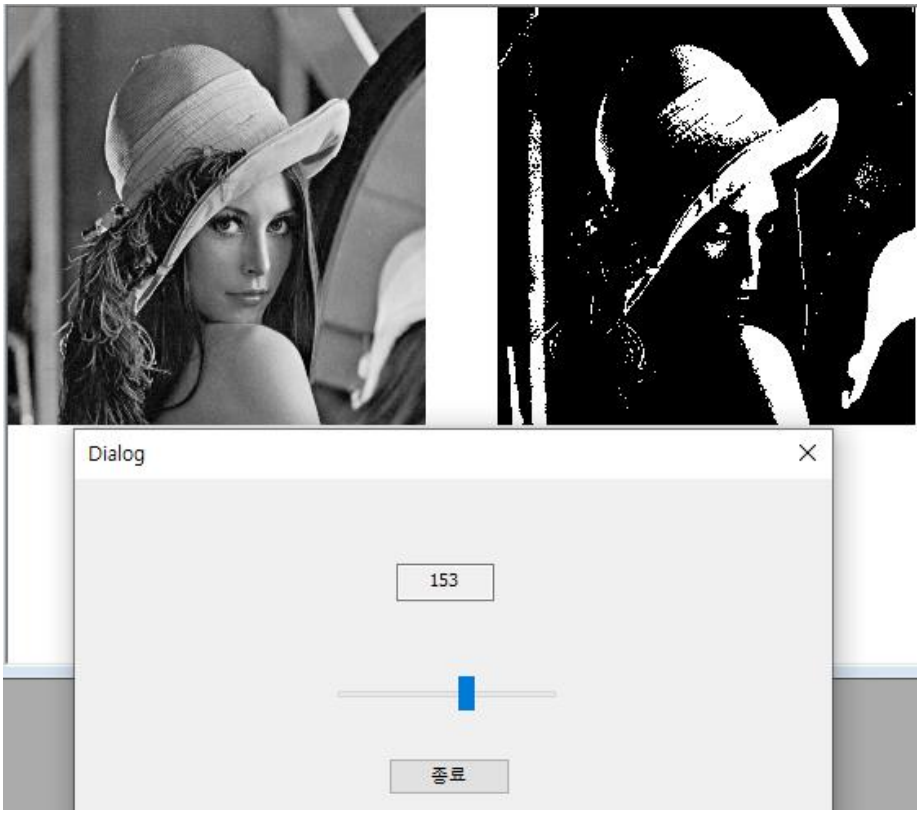
```
void CBINCTRLDLG::OnCustomdrawSlider1(NMHDR* pNMHDR, LRESULT* pResult)
{
    CMainFrame* pFrame = (CMainFrame*)AfxGetMainWnd();
    ASSERT(pFrame);
    CChildFrame* pChild = (CChildFrame*)pFrame->GetActiveFrame();
    ASSERT(pChild);
    CMFCApplication1Doc* pDoc = (CMFCApplication1Doc*)pChild->GetActiveDocument();
    ASSERT(pDoc);
    CMFCApplication1View* pView = (CMFCApplication1View*)pChild->GetActiveView();
    ASSERT(pView);


    m_binval = m_ctrlslider.GetPos();
    UpdateData(FALSE);
    pDoc->m_binthres(256, 256, m_binval);
    pView->Invalidate(FALSE);
}
```

->이과정들은 각각의 컨트롤 마다 기능을 설정한후 서로 연결시켜주는 과정입니다.

```
void CMFCApplication1View::OnBinDynamic()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    CBINCTRLDLG pcinctr;
    pcinctr.DoModal();
}
```

->이후 view클래스에 bindynamic 버튼을 누르면 dialog가 호출 되도록 설정하여 줍니다.



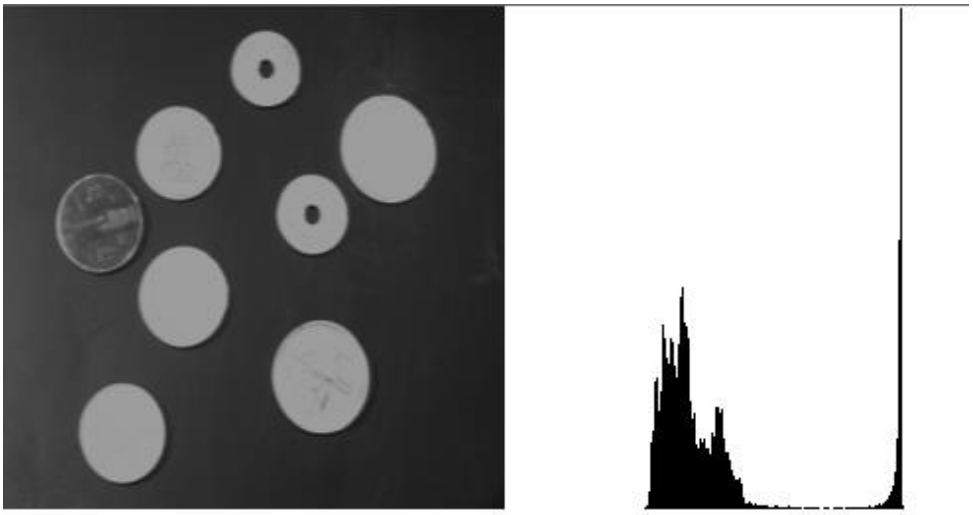




➔ slider를 통하여 threshold값을 설정하고 이를 통하여 lena image 의 이진화가 동적으로 변하는 것을 확인하였습니다.

#histogram equalization

- histogram equalization은 histogram의 형상을 분석하기 위해서 밝기분포가 특정한 부분으로 치우쳐진 것을 넓은 영역으로 밝기 분포가 존재 하도록 histogram을 펼쳐주는 것 입니다.
- 인간의 시각은 영상의 절대적 밝기의 크기보다 대비가 증가할 때 인지도가 증가하므로 histogram이 넓게 분포하는 것이 인식하기 유리하기 때문에 equalization을 하는 것 입니다.
- histogram equalization은 정규화합계산을 통하여 진행 됩니다.

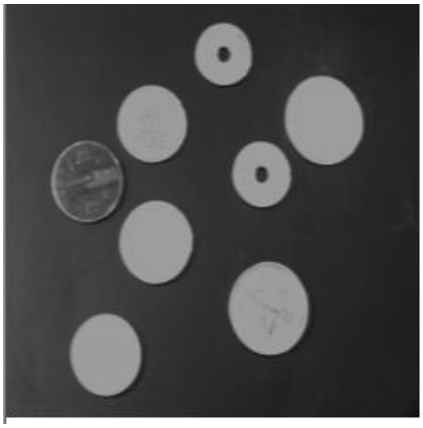
  정규화합=h(t)=(Gmax/Nt)*H(i) 로서 Gmax는 영상의 최대밝기값,Nt는 입력영상의 픽셀개수,H(i)는 원본입력영상의 누적 히스토그램 입니다.

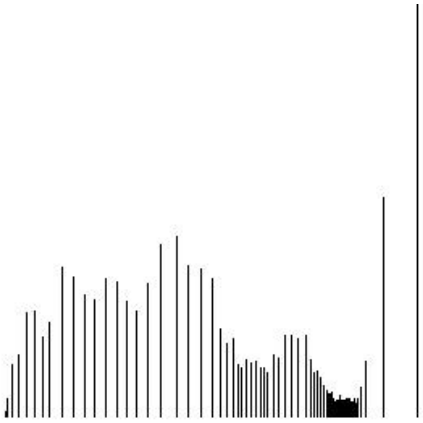


->특정 밝기에 치우쳐진 image

```
void CMFCApplication1Doc::m_histoequal(int height, int width)
{
    // TODO: 여기에 구현 코드 추가.
    int i, j;
    /// histogram연산을 위해 사용할 배열메모리를 할당
    unsigned int* histogram = new unsigned int[256];
    unsigned int* sum_hist = new unsigned int[256];

    /// histogram배열을 초기화
    for (i = 0; i < 256; i++) histogram[i] = 0;

    /// 영상의 histogram을 계산
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++) histogram[m_inimage[i][j]]++;
    }

    /// histogram의 정규화된 합을 계산
    int sum = 0;
    float scale_factor = 255.0f / (float)(height * width);

    for (i = 0; i < 256; i++)
    {
        sum += histogram[i];
        sum_hist[i] = (int)((sum * scale_factor) + 0.5);
    }

    /// LUT로써 sum_hist배열을 사용하여 영상을 변환
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++) m_outimg[i][j] = sum_hist[m_inimage[i][j]];
    }
```

- 누적화합(sum_hist)를 이용하여 outimg에 픽셀값을 변경하여 histogram equalization을 하였습니다.





-> histogram equalization 을 한후에는 image의 미세한 변화들이 눈에 확실하게 보이고 분석이 용이한 image로 변경 되었습니다. 변경된 image의 histogram을 확인해보니 밝기분포가 좁은 영역이 아닌 넓은 영역으로 존재하는 histogram을 가지고 있는 것을 확인하였습니다.

#histogram stretching

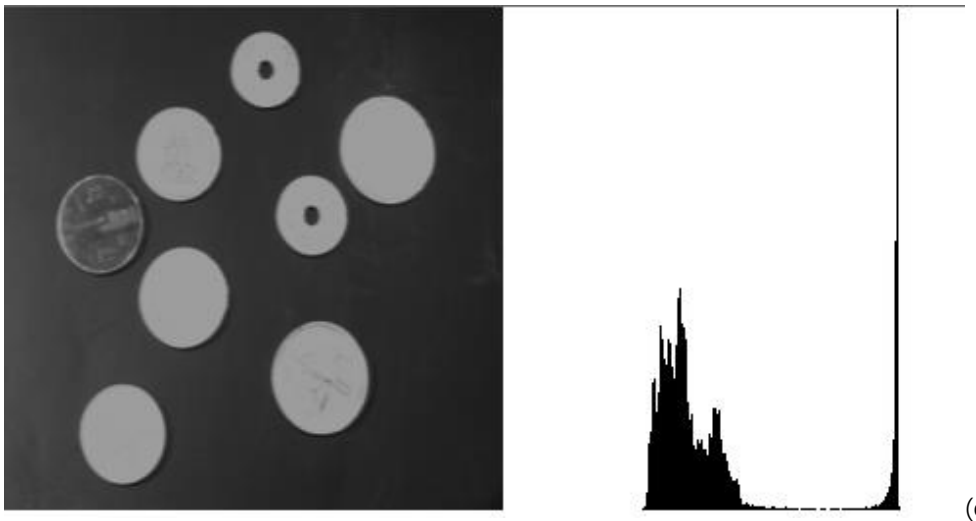
- histogram stretching은 histogram equalization과 비슷하게 histogram 을 펼쳐주는 역할을 하지만 밝기의 최대,최소값을 이용하여 고정된 비율로 낮은 밝기와 높은 밝기로 당겨준 것 입니다.
- histogram stretching은 원본영상이 low가0이고 high가 255일경우 스트레칭의 효과가 떨어지므로 ends-in search법을 사용하여 처리 하기도 합니다.

$$new\ pixel = \frac{old\ pixel - low}{high - low} \times 255$$

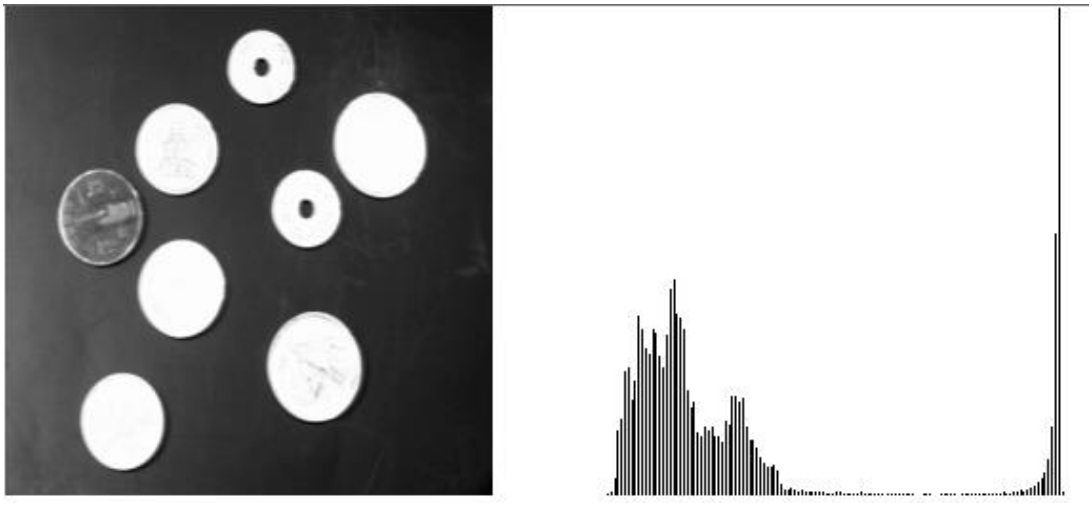
-> histogram stretching 공식

```
void CMFCApplication1Doc::m_histostrech(int height, int width)
{
    // TODO: 여기에 구현 코드 추가.
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    int i, j;
    int lowvalue = 255, highvalue = 0;


    // 밝기값이 가장 작은값 선정
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++) if (m_inimage[i][j] < lowvalue) lowvalue = m_inimage[i][j];
    }

    // 밝기값이 가장 큰값 선정
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++) if (m_inimage[i][j] > highvalue) highvalue = m_inimage[i][j];
    }

    // Histogram스트레칭 계산
    float mult = 255.0f / (float)(highvalue - lowvalue);
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
            m_outimg[i][j] = (unsigned char)((m_inimage[i][j] - lowvalue) * mult);
    }
```

- (m_inimage[i][j]-low)*mult 을 이용하여 histogram stretching을 구현 하였습니다.



(original)

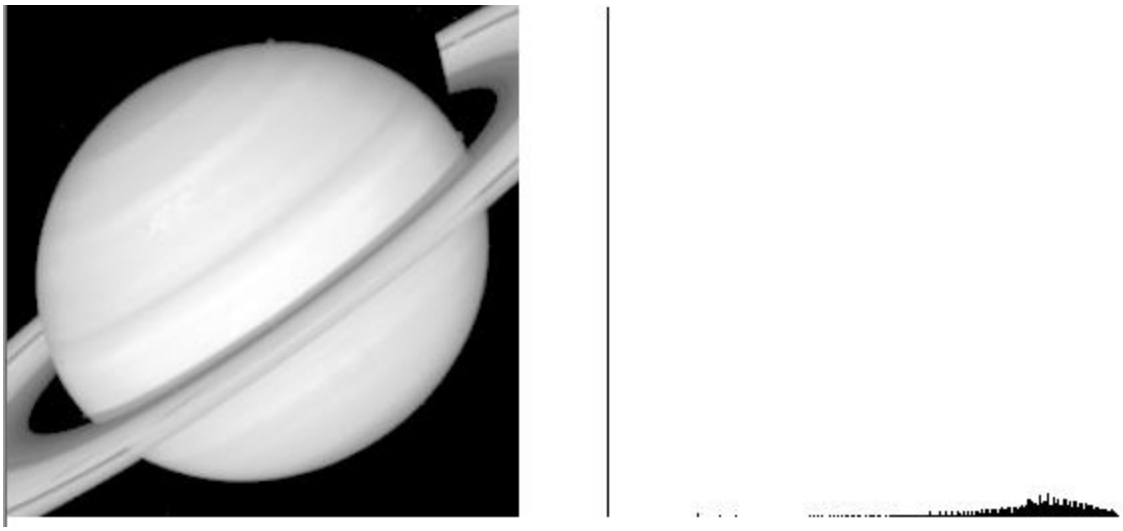


(stretching)

- 넓은 영역으로 histogram 이 변경된 것을 확인하였지만 equalization정도의 넓은 영역으로는 변하지 않는 것 확인하였고 image를 분석하는 것이 매우 좋지는 않지만 어느정도는 좋아진 것을 확인 하였습니다.

#histogram specification

- histogram specification은 원하는 영상의 histogram 모양으로 나오도록 지정하는 것입니다.



->원영상의 histogram입니다.

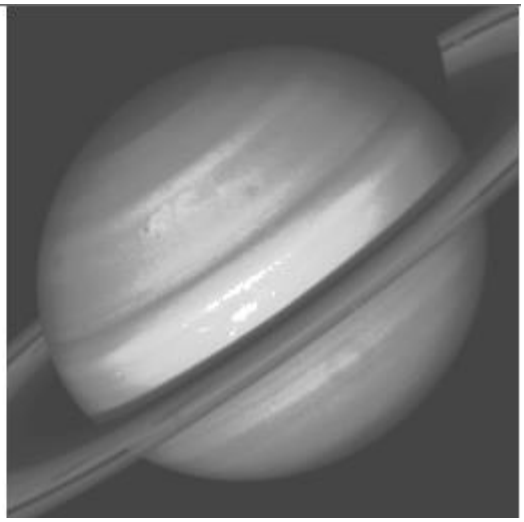
```
void CMFCApplication1Doc::m_histospec_filein()
{
    // TODO: 여기에 구현 코드 추가.
    CFileDialog opendlg1(TRUE); //공통 대화 상자(히스토그램을 지정할 영상파일 입력)
    CFile file;

    if (opendlg1.DoModal() == IDOK)
    {
        file.Open(opendlg1.GetFileName(), CFile::modeRead);
        file.Read(m_inimage1, sizeof(m_inimage1));
        file.Close();
    }
```

->이 코드를 doc클래스에 추가하여 histogram을 지정할 파일을 입력 받게 하였습니다.

- 중앙으로 고르게 포진 되어있는 레나 image를 지정 image로 입력하였습니다.

```
void CMFCApplication1View::OnHistoSpecCal()
{
    int i, j;
    CMFCApplication1Doc* pdoc = GetDocument(); // 다큐멘트 클래스를 참조하기 위해
    ASSERT_VALID(pdoc);
    unsigned int* histogram = new unsigned int[256];
    unsigned int* sum_hist = new unsigned int[256];
    unsigned int* desired_histogram = new unsigned int[256];
    unsigned int* desired_sum_hist = new unsigned int[256];
    // histogram배열을 초기화
    for (i = 0; i < 256; i++) histogram[i] = desired_histogram[i] = 0;
    // 영상의 histogram을 계산하라
    for (i = 0; i < height; i++)
    {
        for (j = 0; j < width; j++)
        {
            histogram[pdoc->m_inimage[i][j]]++;  // 입력 영상의 히스토그램
            desired_histogram[pdoc->m_inimage1[i][j]]++;  // 지정 영상의 히스토그램
        }
    }
    // histogram의 정규화된 합을 계산하라
    int sum = 0;
    float scale_factor = 255.0f / (float)(height * width);

    for (i = 0; i < 256; i++)
    {
        sum += histogram[i];
        sum_hist[i] = (int)((sum * scale_factor) + 0.5);
    }

    // desired histogram에 대한 정규화된 합을 계산
    sum = 0;
    for (i = 0; i < 256; i++)
    {
        sum += desired_histogram[i];
        desired_sum_hist[i] = (int)(sum * scale_factor);
    }

    // 가장 가까운 정규화합 히스토그램값을 주는 index를 찾음
    int* inv_hist = new int[256];
    int hist_min, hist_index, hist_diff;
    for (i = 0; i < 256; i++)
    {
        hist_min = 1000;
        for (j = 0; j < 256; j++)
        {
            hist_diff = abs((int)(sum_hist[i] - desired_sum_hist[j]));
            if (hist_diff < hist_min)
            {
                hist_min = hist_diff;
                hist_index = j;
            }
        }

        inv_hist[i] = hist_index;
    }
    // 역 히스토그램 변환
    for (i = 0; i < height; i++) for (j = 0; j < width; j++) pdoc->m_outimg[i][j] = inv_hist[pdoc->m_inimage[i][j]];
    Invalidate(FALSE);
    // 메모리 해제
    delete[]inv_hist; delete[]histogram; delete[]desired_histogram;
    delete[]sum_hist; delete[]desired_sum_hist;
}
```

- view클래스에 doc의 포인터를 얻어 코드를 작성하였습니다.
- histogram specification의 원리는 원본 이미지의 히스토그램 정규화합 값이 지정할 이미지의 히스토그램 정규화합 값이 제일 비슷한 값으로  변경되어 histogram specification연산이 진행되는 것입니다.

->histogram specification한후 최종 모습입니다. 단순하였던 이미지가 지정한 레나이미지의 히스토그램과 같이 다양한 밝기로 바뀐 이미지를 확인하였습니다.

#히스토그램을 이용한 자동 영상 이치화

- 앞에서 실험한 영상 이치화는 threshold를 사용자가 직접 입력하였지만 실제 사용과정에서 일일이 입력하는 것은 어려운 일입니다. 따라서 히스토그램의 모양을 보고 자동으로 임계치를 결정하는 알고리즘이 필요한데 이에 대해 알아보겠습니다.
- Otzu법은 자동 영상 이치화중 가장 빈번하게 사용되는 하나의 방법으로 각각의 피크를 확률분포중 하나인 정규분포로 보고 정규분포는 평균과 분산의 두 파라메터 값으로 표현되기 때문에 밝기가 집중되어있다면 작은 크기의 분산은 가집니다. 따라서 배경부와 물체부 두그룹이 가능한 낮은 분포값(작은분산)을 가지도록 threshold설정하여주면 되는 것입니다.

```
void CMFCApplication1Doc::OnAutobin()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.

    int height = 256, width = 256;
    register int i, j;

    unsigned char* orgImg = new unsigned char[height * width];
    unsigned char* outImg = new unsigned char[height * width];

    for (i = 0; i < height; i++) for (j = 0; j < width; j++) orgImg[i * width + j] = m_inimage[i][j];

    Otzu_threshold(orgImg, outImg, height, width);

    for (i = 0; i < height; i++) for (j = 0; j < width; j++) m_outimg[i][j] = outImg[i * width + j];

    delete[]orgImg;
    delete[]outImg;

    UpdateAllViews(NULL);
}
```

- 이차원 배열으로 되어있는 원본 이미지를 1차원 이미지로 바꿔준 후 Otzu_threshold 함수를 통하여 threshold를 정한 후에 다시 2차원 이미지로 displaying해주는 방식입니다.

```
void CMFCApplication1Doc::Otzu_threshold(unsigned char* orgImg, unsigned char* outImg, int height, int width)
{
    // TODO: 여기에 구현 코드 추가.
    register int i, t;

    // Histogram set
    int     hist[256];
    float   prob[256];
    for (i = 0; i < 256; i++) { hist[i] = 0; prob[i] = 0.0f; }// 초기화
    for (i = 0; i < height * width; i++) hist[(int)orgImg[i]]++;
    for (i = 0; i < 256; i++) prob[i] = (float)hist[i] / (float)(height * width);


    float wsv_min = 1000000.0f;
    float wsv_u1, wsv_u2, wsv_s1, wsv_s2;
    int wsv_t;

    for (t = 0; t < 256; t++)
    {
        // q1, q2 계산
        float q1 = 0.0f, q2 = 0.0f;

        for (i = 0; i < t; i++)   q1 += prob[i];
        for (i = t; i < 256; i++) q2 += prob[i];

        if (q1 == 0 || q2 == 0) continue;

        // u1, u2 계산
        float u1 = 0.0f, u2 = 0.0f;
        for (i = 0; i < t; i++)   u1 += i * prob[i]; u1 /= q1;
        for (i = t; i < 256; i++) u2 += i * prob[i]; u2 /= q2;

        // s1, s2 계산
        float s1 = 0.0f, s2 = 0.0f;
        for (i = 0; i < t; i++)   s1 += (i - u1) * (i - u1) * prob[i]; s1 /= q1;
        for (i = t; i < 256; i++) s2 += (i - u2) * (i - u2) * prob[i]; s2 /= q2;

        float wsv = q1 * s1 + q2 * s2;

        if (wsv < wsv_min)
        {
            wsv_min = wsv; wsv_t = t;
            wsv_u1 = u1; wsv_u2 = u2;
            wsv_s1 = s1; wsv_s2 = s2;
        }
    }

    // Display the result values
    CString strTemp;
    strTemp.Format(TEXT("Optimal Threshold: %3d₩nMean: %7.3f, %7.3f₩nVariance: %7.3f, %7.3f"), wsv_t, wsv_u1, wsv_u2, wsv_s1, wsv_s2);
    AfxMessageBox(strTemp);

    // thresholding
    for (i = 0; i < height * width; i++) if (orgImg[i] < wsv_t) outImg[i] = 0; else outImg[i] = 255;
}
```

- Hist[]는1차원으로 되어있는 이미지의 히스토그램이고 prob[]는 히스토그램을 전체 픽셀수로 나눠 확률로 바뀐 모습입니다.
- T값 즉 threshold값을 0부터 255까지 바꿔가면서 q1*s1+q2*s2의 최소값(두분포의 분산의 합)을 찾아 최적의 threshold값을 구하는 코드입니다.
- u1은 그룹1의 mean값 (평균) s1은 그룹1의 분산 값 이고 u2은 그룹2의 mean값 (평균) s2은 그룹2의 분산 값 입니다.

$$Minimize : \sigma_w^2(t) = w_1(t)\sigma_1^2(t) + w_2(t)\sigma_2^2(t)$$

$$w_1(t) = \Sigma_{i=0}^{t-1} p(i)$$

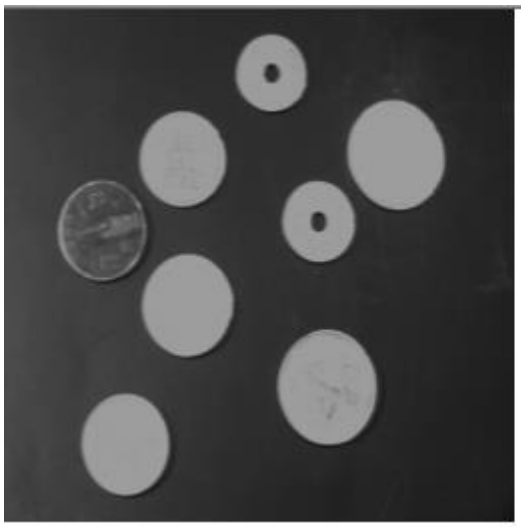
$$w_2(t) = \Sigma_{i=t}^{L} p(i)$$

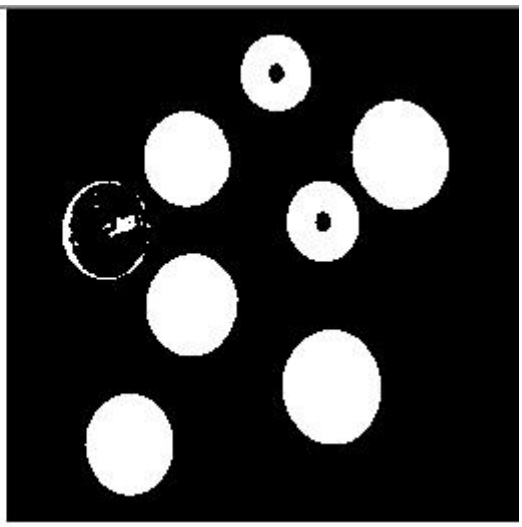
$$\mu_1(t) = \Sigma_{i=0}^{t-1} \frac{p(i)}{w_1(t)} i$$

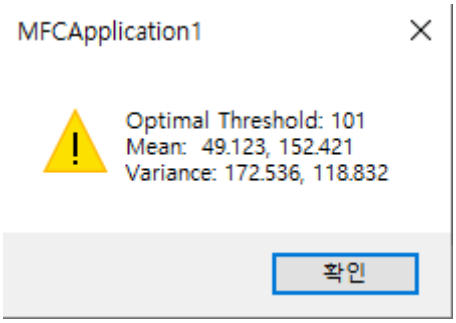
$$\mu_2(t) = \Sigma_{i=t}^{L} \frac{p(i)}{w_2(t)} i$$

$$\sigma_1(t) = \Sigma_{i=0}^{t-1} \frac{p(i)}{w_1(t)} (i - \mu_1(t))^2$$

$$\sigma_2(t) = \Sigma_{i=t}^{L} \frac{p(i)}{w_2(t)} (i - \mu_2(t))^2$$



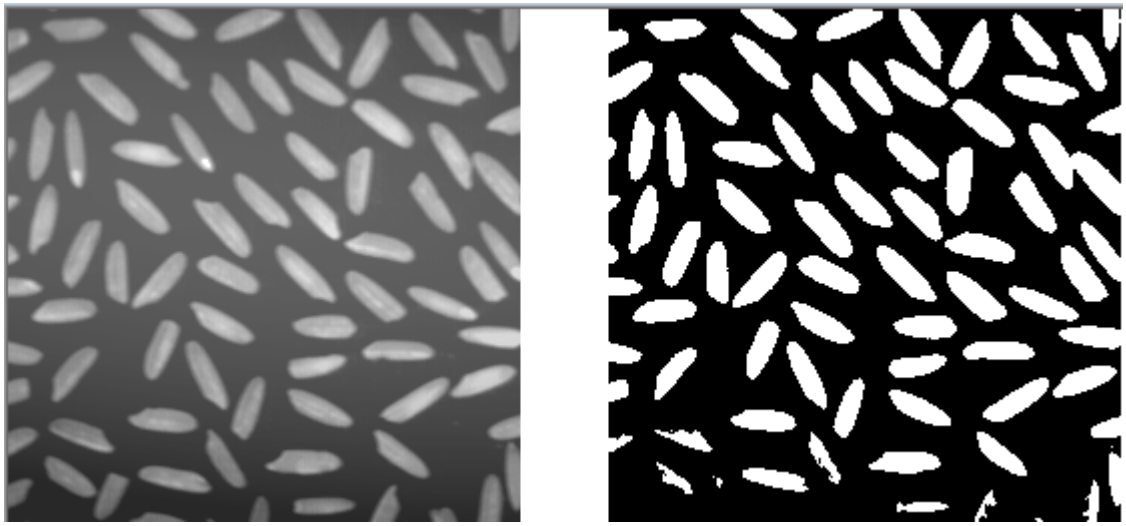





->이미지를 자동 이치화 동작 시켜보았더니 threshold값과 mean값,variance값이 자동으로 알고리즘에 의하여 적용 된 것을 확인 할 수 있었습니다.

#적응이치화 알고리즘

- otzu알고리즘 과 같은 전역이치화 알고리즘은 한프레임의 영상에 대하여 하나의threshold를 결정하기때문에 이미지내에서 배경의 밝기가 분균일 할때에는 물체 추출이 어려워집니다.



->전역 이치화의 예로서 이미지의 아랫부분은 정확하게 분석이 불가능한 이미지입니다.

- 따라서 하나의 threshold가아닌 이미지의 각각 부분에서 밝기에 따라 threshold를 결정해주어 이치화를 진행하여 분석하기 좋은 이미지를 추출합니다.

- 대표적인 적응 이진화 방법인 niblack 이치화에 대해 알아보겠습니다.

$$T = m[1 - k\left(1 - \frac{\sigma}{R}\right)]$$

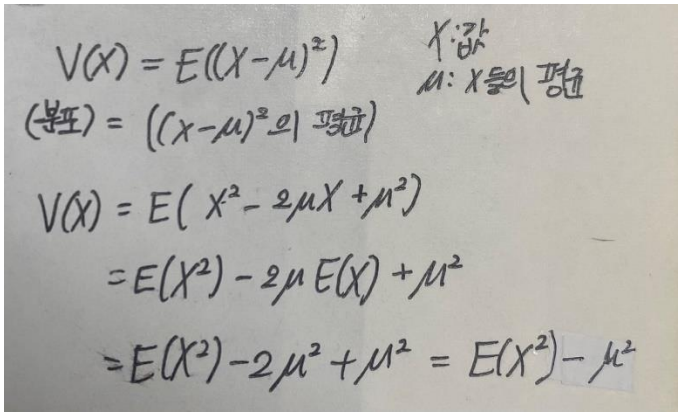
이식을 이용하여 영역내의 밝기평균과 분산을 이용하여 threshold값을 정하는 방법입니다. R은 128로 주어지는 표준편차의 최대 범위이고 k값은 상수로 사용자가 설정하며 주로 0.02를 사용합니다.

```
void CMFCApplication1Doc::OnAdapBin()
{
    // TODO: 여기에 명령 처리기 코드를 추가합니다.
    // TODO: Add your command handler code here
    int height = 256, width = 256;
    register int i, j;

    unsigned char* orgImg = new unsigned char[height * width];
    unsigned char* outImg = new unsigned char[height * width];

    for (i = 0; i < height; i++) for (j = 0; j < width; j++) orgImg[i * width + j] = m_inimage[i][j];


    AdaptiveBinarization(orgImg, outImg, height, width);


    for (i = 0; i < height; i++) for (j = 0; j < width; j++) m_outimg[i][j] = outImg[i * width + j];

    delete[]orgImg;
    delete[]outImg;

    UpdateAllViews(NULL);
```

- 전역이치화와 마찬가지로 doc클래스에는 비슷한 구조이지만 adaptivebinarization 함수를 이용하여 영역별 threshold값을 지정한 후에 displaying합니다.

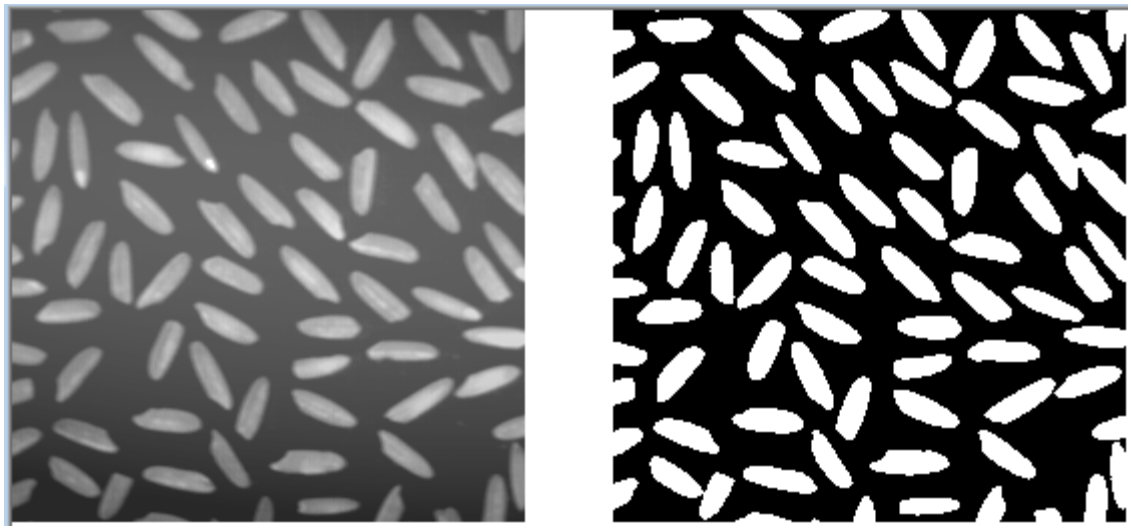
```
void CMFCApplication1Doc::AdaptiveBinarization(unsigned char* orgImg, unsigned char* outImg, int height,
{
    // TODO: 여기에 구현 코드 추가.
    register int i, j, k, l;
    int gval, index1, index2;
    float mean, vari, thres;
    int W = 20;

    for (i = 0; i < height * width; i++) outImg[i] = 255;//화이트로 초기화,일차원배열로 변경

    for (i = 0; i < height; i++)
    {
        index2 = i * width;
        for (j = 0; j < width; j++)
        {
            float gsum = 0.0f;
            float ssum = 0.0f;
            int   count = 0;

            for (k = i - W; k <= i + W; k++)
            {
                index1 = k * width;
                if (k < 0 || k >= height) continue;

                for (l = j - W; l <= j + W; l++)
                {
                    if (l < 0 || l >= width) continue;

                    gval = orgImg[index1 + l];
                    gsum += gval;
                    ssum += gval * gval;
                    count++;
                }
            }

            mean = gsum / (float)count;
            vari = ssum / (float)count - mean * mean;

            if (vari < 0) vari = 0.0f;

            //          thres   = mean+0.4f+(float)sqrt(vari);
            thres = mean * (1.0f - 0.02f * (1 - (float)sqrt(vari) / 128));

            if (orgImg[index2 + j] < thres) outImg[index2 + j] = 0;
        }
```

- 우선w값을 20으로 설정하여 영역을 지정합니다. 따라서 세로(k-20)~(k+20)과 가로 (l-20)~(l+20)의 영역안에서 의 gval(밝기값), gsum(밝기들의합), ssum(밝기들의 제곱합) ,count(영역내 픽셀수)를 얻게 되었습니다.

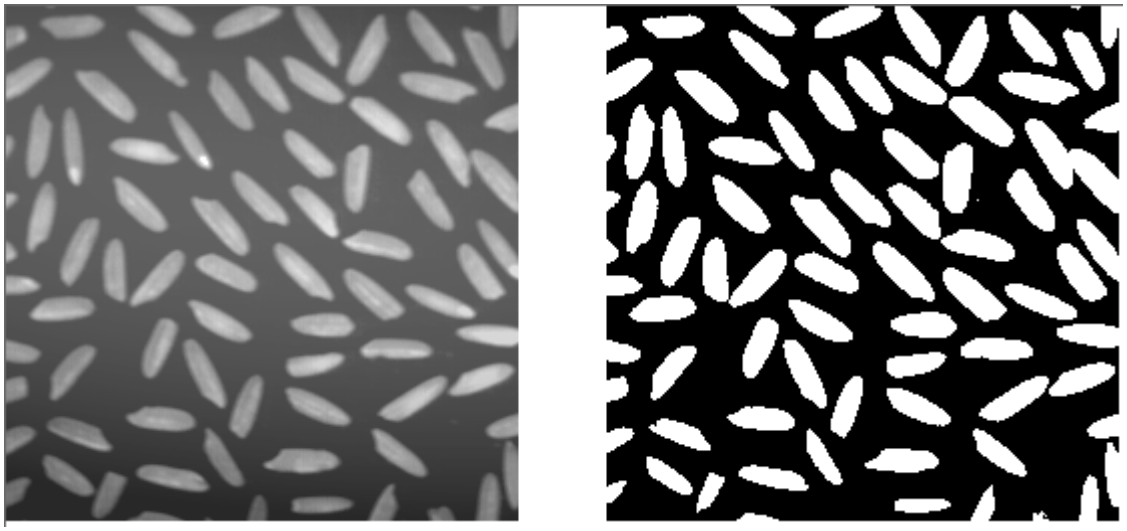



- 이식을 근거로 하여 영역내의 mean 값과 분산을 얻어 $T = m[1 - k\left(1 - \frac{\sigma}{R}\right)]$

  에 적용 시켜 threshold값을 얻고 outimg의 픽셀값을 변동시켜 이진화 하는 방식입니다.



- 따라서 전역 이치화를 실행 하였을 때 분석하기 힘들었던 아래부분도 적응 이치화를 통하여 영역별로 threshold값을 변화 시켰기 때문에 식별할 수 있는 이미지를 확인 하였습니다.



- w값을 200으로 변경한 후에 적응 이치화를 진행 시켰더니 아래쪽부분은 인식이 안되는 결과가 나왔고 실행 속도에서도 많이 느려진 것을 확인했습니다.
- 적절한 w값과 k값을 설정하는 것도 중요한 요인 인 것 같습니다.