고급 SW 실습 I
SIFT
(실습 자료)

CSE4152

서강대학교 컴퓨터공학과

SIFT

- 실습 내용
  - SIFT 소스 코드를 살펴보고, 이중 일부 함수를 직접 작성하여 기존 함수 대신 사용해 결과를 체크해 본다.
  - 작성할 함수는 customGaussianBlur, MatchDescriptor이다.
  - OpenCV 라이브러리와 VS 콘솔 프로그래밍을 사용한다.
- 주의 사항
  - 프로그램 완성 후 담당 조교에게 확인을 받아야 하고 동시에 이들을 사이버 캠퍼스에서 제출하여야 한다.
  - 제출할 파일 이름은 snnnnnnL03.cpp로 하며, 여기에 위 두 함수 모두 작성하여 제출한다.

    여기서, nnnnnn은 자신의 학번 뒤 6자리.

    (**주의**) 만일 파일의 nnnnnn을 자신의 학번 뒤 6자리로 바꾸지 않고, 그냥 snnnnnnL03_1.cpp 등으로 제출하면 **0점 처리**한다.

주의 사항 (계속)

- 실습 결과 검사
  - 담당 조교가 결과를 검사하면서 제대로 알고 작성했는지 몇 가지 작성 내용에 관한 질문을 할 수 있다.
  - 평가 사항이므로 이에 답을 제대로 못하면 감점할 수 있다.
  - 그러니, 프로그램을 작성할 때 내용을 이해하며 작성하여야 한다(질문이 있으면 주저 말고 조교에게 문의할 것)
- 숙제가 있을 경우
  - 제출 파일 이름, 마감일 등을 지정해 줄 것이다.
- 제출 마감
  - 실습 당일 담당 조교가 실습 진행 상황을 감안하여 지정해 줄 것이다. 숙제 역시 마찬가지다.
  - 주의. Late 제출은 절대 허용하지 않는다. 사이버 캠퍼스가 효과적으로 late 제출을 받지 않을 것이다.

Visual Studio Project 생성

- 생성 내용 및 방법
  - VS2017 에서의 프로젝트 생성(1)
    - 2주차 실습과 동일
    - 파일 → 새로 만들기 → 프로젝트 → Visual C++ → 일반 → 빈 프로젝트 선택
    - 프로젝트 이름 입력(예: swLab19f) → 폴더 선택 → 확인
  - OpenCV 연결
    - 1 주차 MFC 실습 때와 동일하게 x86 모드에서 설정한다(2).
    - 디버그 모드와 릴리즈 모두 설정한다(실습 때 필요하다)
    - 메모리 누수를 위한 설정도 할 것(1주차 실습 자료)

(1) VS2015, VS2019도 사용 가능할 것이다.
(2) X64에서 작업을 원한다면 이는 개인적으로 해보자. 이 경우 x86과 동일한 설정 작업을 해야 한다(라이브러리에 d가 붙어있지 않다).

SIFT in OpenCV

- Class cv::KeyPoint
  - Keypoint를 저장하는 클래스이며 생성자는 다음과 같다.

```
KeyPoint(Point2f _pt, float _size, float _angle=-1,
 float _response=0, int _octave=0, int _class_id=-1)
또는
KeyPoint(float x, float y, float _size,
 float _angle=-1, float _response=0, int _octave=0,
 int _class_id=-1)
```

  - 멤버 변수
    - `Point2f _pt` : 키포인트 좌표. pt 내에 x, y변수가 있다.
    - `float _size` : 특징점의 크기(지름)
    - `float _angle` : 키포인트의 주된 방향(범위는 [0,360)). 각도의 방향은 이미지 좌표 시스템을 따른다((반)시계방향).

✦ 멤버 변수(계속)

- **float _response** : 키포인트의 반응성. 좋은 특징점을 선별하는 용도로 사용
- **int _octave** : 키포인트가 추출된 피라미드의 옥타브.
- **int _class_id** : object 클래스(if the keypoints need to be clustered by an object they belong to)

참고: **class_id, response, size**는 키포인트에 반드시 필요한 값은 아니다.

✦ Instance 생성

```
KeyPoint kpt1;
kpt1.pt.x=5;
kpt1.pt.y=15;
kpt1.angle=24.5;
...
```

또는

```
KeyPoint kpt1(x=5,y=15,_angle=24.5,...);
```

Class SIFT(1)

- SIFT.h에 멤버 변수와 함수가 선언, SIFT.cpp에 함수가 구현되어 있다.
- 생성자

```
SIFT(int nfeatures = 0, int nOctaveLayers = 3,
     double contrastThreshold = 0.04,
     double edgeThreshold = 10, double sigma = 1.6)
```

- 멤버 변수
  - int nfeatures : 최대 키포인트 개수 설정(0이면 무제한)
  - int nOctaveLayers : 옥타브당 스케일 수(옥타브당 Gaussian scale 수는 nOctaveLayers+3)
  - double contrastThreshold : constrast 임계값
  - double edgeThreshold : edge 임계값
  - double sigma : scale sapce 구성 시 초기 시그마 값

(1) https://github.com/opencv/opencv/blob/2.4/modules/nonfree/include/opencv2/nonfree/features2d.hpp
(2) https://github.com/opencv/opencv/blob/2.4/modules/nonfree/src/sift.cpp

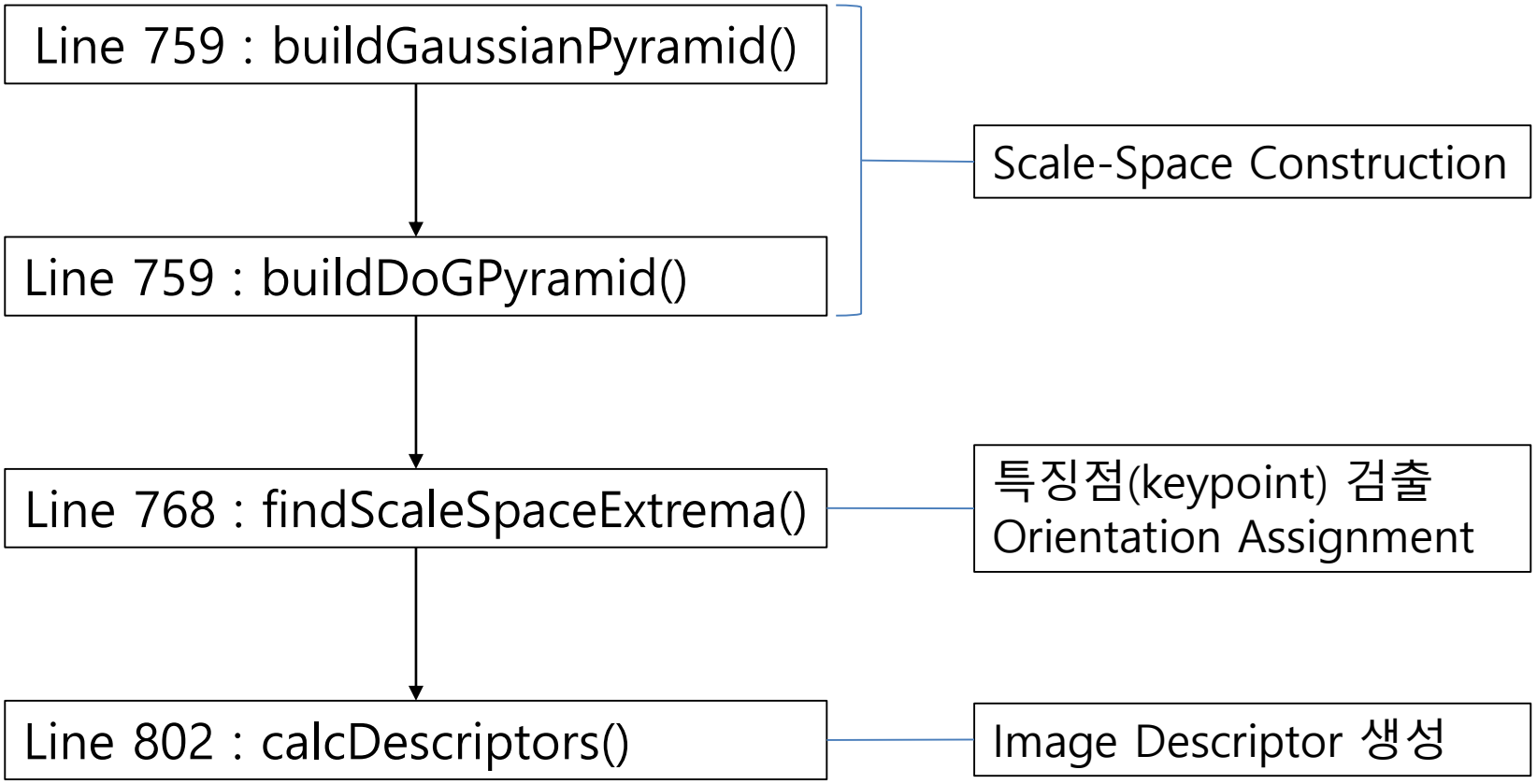
✦클래스 생성 및 SIFT 연산

```
// 먼저 SIFT instance를 아래와 같이 생성한다.
SIFT *sift = new SIFT(nfeatures, nOctaveLayers,
                      contrastThreshold, edgeThreshold, sigma);

// Keypoint를 위한 vecto와 descriptor를 선언한다.
Vector<KeyPoint> keypoints;
Mat descriptor;

// 주어진 이미지 imp에 대해 SIFT 연산을 수행한다.
// 출력은 keypoints와 descriptor이다.
sift->operator()(img, keypoints, descriptor);
```

- nfeatures가 0으로 주어진 경우, keypoints의 크기는 operator() 함수에서 연산이 진행 중에 결정된다.
- descriptor의 경우 row 수는 keypoint의 개수와 같으며 column 수는 128인데 이 역시 nfeatures가 0일 경우 operator() 내부에서 결정된다.

SIFT 클래스 operator 함수 진행과정

- sift.cpp에서 함수 operator( )의 keypoint와 descripter 추출 과정을 다음 도표로 보인다.



2D 필터

- 아래 함수는 입력 이미지에 사전 준비한 커널을 적용시켜 필터링하는 함수이다.

```
void filter2D(InputArray src, OutputArray dst,
  int ddepth, InputArray kernel,
  Point anchor = Point(-1,-1), double delta=0,
  int borderType = BORDER_DEFAULT )
```

- 함수 파라미터

- **src, dat** : 각각 입력 및 출력 이미지.
- **ddepth** : 출력 이미지의 depth(기본값 -1(입력과 동일)).

입력 이미지형태에 따라 가능한 ddepth 값은 다음과 같다:

src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
src.depth() = CV_64F, ddepth = -1/CV_64F

다음 쪽에 계속

✦함수 파라미터(계속)

- **src, dat** : 각각 입력 및 출력 이미지.
- **ddepth** : 출력 이미지의 **depth**(기본값 **-1**(입력과 동일)).
  입력 이미지형태에 따라 가능한 **ddepth** 값은 다음과 같다:
  src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F
  src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F
  src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F
  src.depth() = CV_64F, ddepth = -1/CV_64F

- **kernel** : 입력 이미지에 **convolution**할 커널. **float**형 **single** 채널 행렬.
- **anchor** : 커널의 중심점을 설정. 기본값은 커널의 중심 위치.
- **delta** : 필터 적용 후 픽셀에 추가로 더해질 값
- **borderType** : 이미지 밖 픽셀 값 설정 방법(2주차 실습 참조)

실습

실습

- SIFT에서 사용하는 두 함수를 작성하고 SIFT 연산에 적용하여 본래의 SIFT.cpp 동작과 동일하게 실행되는지 확인한다.
- 작성해야 할 두 함수는 다음과 같다
  - Gaussian 필터링을 통한 Gaussian blurring 함수 작성.
  - 두 keypoint 집합의 descriptor들 간의 비교를 통한 keypoint 매칭 함수 작성.
- 실습 방법
  - 첨부한 swLab19f_3_SIFT_Lab_Std.zip 파일을 풀어 저장.
  - snnnnnnL03.cpp의 nnnnnn을 자신의 학번 뒤 6자리로 변경.
  - 위 파일 내 두 함수를 작성한 후 실행 결과를 확인(1).
  - 담당 조교 검사 후 작성한 파일을 사이버 캠퍼스에 제출.

(1) 빌드할 때 x86 모드인지 확인하고 빌드한다.

프로그램 실행

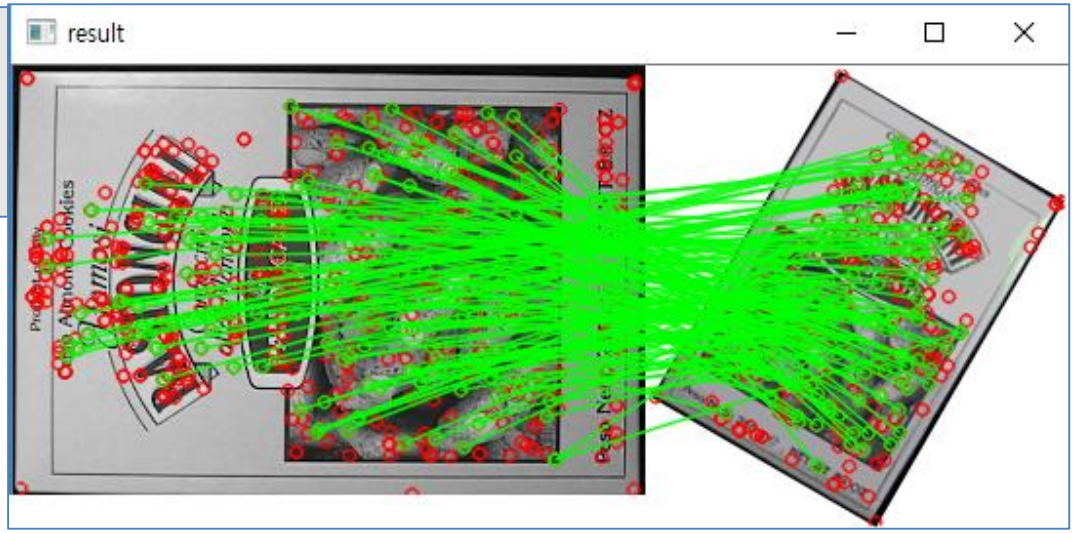
- snnnnnnL03.cpp를 작성하고 빌드 한 후 다음과 같이 실행해 보자(1).

```
...\실행파일 이미지1 이미지2
```

- 다음과 같이 실행하였을 때 그 결과는 다음과 같을 것이다.

```
...\실행파일 box.bmp box3_30.png
```

```
Image 1 size :[324 x 223], Type:8UC3
Image 2 size :[217 x 238], Type:8UC3
IMG1 Num kpt : 604
IMG2 Num kpt : 320
matches:137
```



우측 그림에서 적색 원은 매치되지 않은 keypoint 들이다.

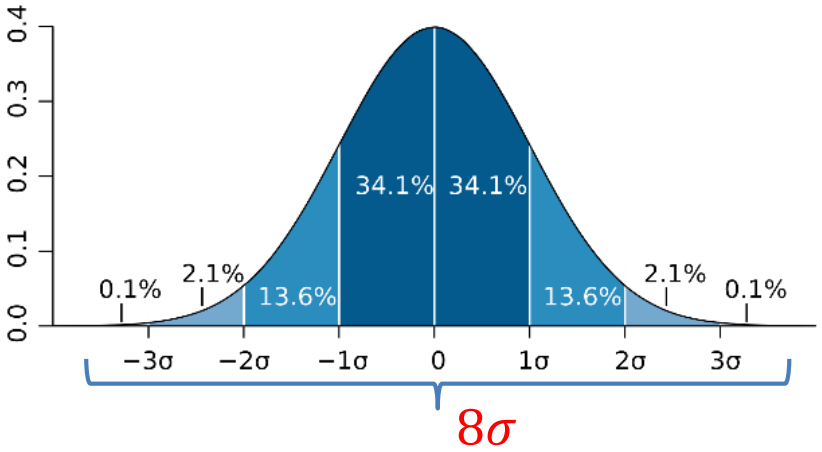
(1) 이미지는 경로까지 포함하거나 아니면 실행 파일이 있는 폴더로 복사하여 사용한다.

Gaussian blur

- snnnnnnL03.cpp에 다음 함수를 완성하자

```
void customGaussianBlur(const Mat &src, Mat &dst,
                        double sigma)
    // src    : 입력 이미지
    // dst    : 가우시안 블러링한 이미지
    // sigma : blur에 사용할 시그마 값(x, y 모두 같은 값)
```

- 함수의 기능
  - 가우시안 커널을 생성하여 이미지에 적용하는 함수.
  - 커널 크기 : 2*(4*sigma)+1 (소수점이하 올림(ceil( ) 사용))
    이는 분포의 대부분 정보를 포함하는 크기이다.



- 2차원 가우시안 수식

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- 커널 작성
  - 위 수식에 x와 y 방향으로 각각 $[-[4\sigma], +[4\sigma]]$ 범위 만큼 샘플링하여 값을 구한다(데이터 타입은 float).
  - 커널 값 계산 후 전체의 총합으로 나눠서 [0.0,1.0]사이의 값을 가지도록 정규화 한다.
- 필터링
  - 작성한 커널을 인수로 하여 함수 filter2D( )를 호출하여 블러링을 수행한다.

Keypoint Matching

- 다음과 같은 매칭 함수를 구현한다

```
int MatchDescriptor(const Mat &descriptor1,
    const Mat &descriptor2, vector<int> &machingIdx)
  // descriptor1 : 이미지 1의 디스크립터
  // descriptor2 : 이미지 2의 디스크립터
  // machingIdx  : 이미지 1의 각 키포인트와 매칭된
  //       이미지 2의 키포인트 인덱스를 저장한 STL vector
  //       매칭이 안될 경우 -1 저장.
```

- 참고: 아래 보인 두 swap 함수 모두 call by reference이다.

```
int main(void) {
  int x, y;
  x = 10;
  y = 20;
  swap1(&x, &y);
  swap2(x, y);
}
```

```
void swap1(int *x,
  int *y) {
  int tmp = *x;
  *x = *y;
  *y = tmp;
}
```

```
void swap2(int &x,
  int &y) {
  int tmp = x;
  x = y;
  y = tmp;
}
```

✦STL vector

- ◆배열과 유사한 sequence container이며 dynamic한 크기 조절이 가능하다.
- ◆다양한 멤버 함수를 제공한다(1,2).
- ◆vector instance 생성

```
vector<[data type]> [변수이름]
```

- ◆멤버 함수 resize 사용 예

```
vector<int> v;
v.resize(n); // v의 크기를 n으로(확장되는 부분은 0으로
             // 초기화)
```

- ◆본 실습에서는 keypoint 매칭 결과를 **vector machingIdx**에 저장하는데, 함수 호출시 이의 크기가 없으므로 멤버함수 **.resize( )**를 통하여 그 크기를 결정한다.

(1) http://www.cplusplus.com/reference/vector/vector/
(2) https://hyeonstorage.tistory.com/324

✦함수 작성

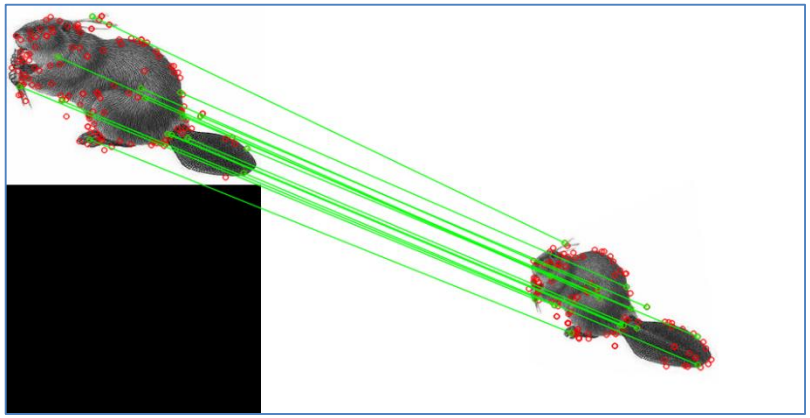
- 두 이미지의 각 keypoint에 대한 descriptor가 입력으로 주어지는데, descriptor의 rows는 keypoint 개수와 동일하고, cols는 128이다.
- Descriptor 간의 거리는 L2 norm으로 정한다.

1. Vector **machingIdx**는 멤버 함수 **.resize( )**를 통하여 이미지 1의 keypoint 개수로 그 크기를 정한다.
2. 아래 과정을 이미지 1의 모든 keypoint에 대해 수행한다.
   A. 이미지 1의 keypoint $\boldsymbol{k}$와 이미지 2의 모든 keypoint들과의 descriptor간 거리를 비교하여 가장 가까운 이미지 2의 keypoint와 두 번째 가까운 keypoint를 구한다.
   B. 가장 가까운 keypoint와의 거리를 $\boldsymbol{d_1}$, 이의 index를 $\boldsymbol{i_1}$이라고 하고, 두번째 가까운 keypoint와의 거리를 $\boldsymbol{d_2}$, 이의 index를 $\boldsymbol{i_2}$라고 하자.
   C. 만일 $\boldsymbol{i_2}$가 존재하고 $\boldsymbol{d_1/d_2}$ < **DIST_RATIO_THR**이면 $\boldsymbol{k}$의 매칭 keypoint로 $\boldsymbol{i_1}$을 선택하고 이를 vector matchingIdx의 index $\boldsymbol{k}$에 저장한다.
   D. 만일 단계 8의 테스트에 실패하면 **matchingIdx**의 index $\boldsymbol{k}$에는 $\boldsymbol{-1}$을 저장한다.

실습 결과

- 두 함수 작성을 완료한 후 빌드하여 주어진 img 폴더의 이미지에 대해 매칭을 시도해보자.
- 아래 일부 매칭 결과를 보인다

```
C:₩exe>swLab19f_3_SIFT beaver.png beaver2.png
Image 1 size :[300 x 211], Type:8UC3
Image 2 size :[640 x 480], Type:8UC3
IMG1 Num kpt : 144
IMG2 Num kpt : 114
matches:15
```



```
C:₩exe>swLab19f_3_SIFT beaver.png Butterfly_8.bmp
Image 1 size :[300 x 211], Type:8UC3
Image 2 size :[643 x 426], Type:8UC3
IMG1 Num kpt : 144
IMG2 Num kpt : 1866
matches:0
```