Lab03 ImageStitching

유형	
교수님	
학년/학기	

코드에 대한 설명은 코드에 안의 주석으로 설명해놨습니다.

약간의 쉬운 내용이나 중요도가 낮은 내용들은 코드가 아예 없거나 주석이 없습니다.

Random한 3개의 숫자 선정:

rand_array[3]에 0~10까지의 겹치지 않는 3가지의 숫자 선정

```
int   rand_array[3];
      int   a, b, c;

      // 3개의 랜덤한 숫자 선정
      for (;;) {
        a = rand()%10;
        b = rand()%10;
        c = rand()%10;
        if (a != b && b != c && a != c) break;
      }

      // 랜덤으로 선정된 m_point의 점
      std::cout << "random number is : " << a << " " << b << " " << c << "\n";
      std::cout << "random point  is : " << m_point1[a].X << " " << m_point1[a].Y << "\n";
      std::cout << "random point  is : " << m_point1[b].X << " " << m_point1[b].Y << "\n";
      std::cout << "random point  is : " << m_point1[c].X << " " << m_point1[c].Y << "\n";
      rand_array[0] = a;
      rand_array[1] = b;
      rand_array[2] = c;
```

table과 Y값에 값을 넣기 (일일이 매칭함)

```
// 6X6배열의 0~5번에 m_point의 X, Y값 넣기
// 0
table[0][0] = m_point1[rand_array[0]].X;
table[0][1] = m_point1[rand_array[0]].Y;
```

```
table[0][2] = 1;
table[0][3] = 0;
table[0][4] = 0;
table[0][5] = 0;

// 1
table[1][0] = 0;
table[1][1] = 0;
table[1][2] = 0;
table[1][3] = m_point1[rand_array[0]].X;
table[1][4] = m_point1[rand_array[0]].Y;
table[1][5] = 1;

// 2
table[2][0] = m_point1[rand_array[1]].X;
table[2][1] = m_point1[rand_array[1]].Y;
table[2][2] = 1;
table[2][3] = 0;
table[2][4] = 0;
table[2][5] = 0;

// 3
table[3][0] = 0;
table[3][1] = 0;
table[3][2] = 0;
table[3][3] = m_point1[rand_array[1]].X;
table[3][4] = m_point1[rand_array[1]].Y;
table[3][5] = 1;

// 4
table[4][0] = m_point1[rand_array[2]].X;
table[4][1] = m_point1[rand_array[2]].Y;
table[4][2] = 1;
table[4][3] = 0;
table[4][4] = 0;
table[4][5] = 0;

// 5
table[5][0] = 0;
table[5][1] = 0;
table[5][2] = 0;
table[5][3] = m_point1[rand_array[2]].X;
table[5][4] = m_point1[rand_array[2]].Y;
table[5][5] = 1;

// Y값 넣기
y[0] = m_point2[rand_array[0]].X;
y[1] = m_point2[rand_array[0]].Y;
y[2] = m_point2[rand_array[1]].X;
y[3] = m_point2[rand_array[1]].Y;
y[4] = m_point2[rand_array[2]].X;
y[5] = m_point2[rand_array[2]].Y;
```

차승제곱법

```
// 차승 제곱법 : W를 계산 하는 함수(KHU Library 내 구현된 함수)
LeastSquared(table, w, y, 6, 6, false, 0);
```

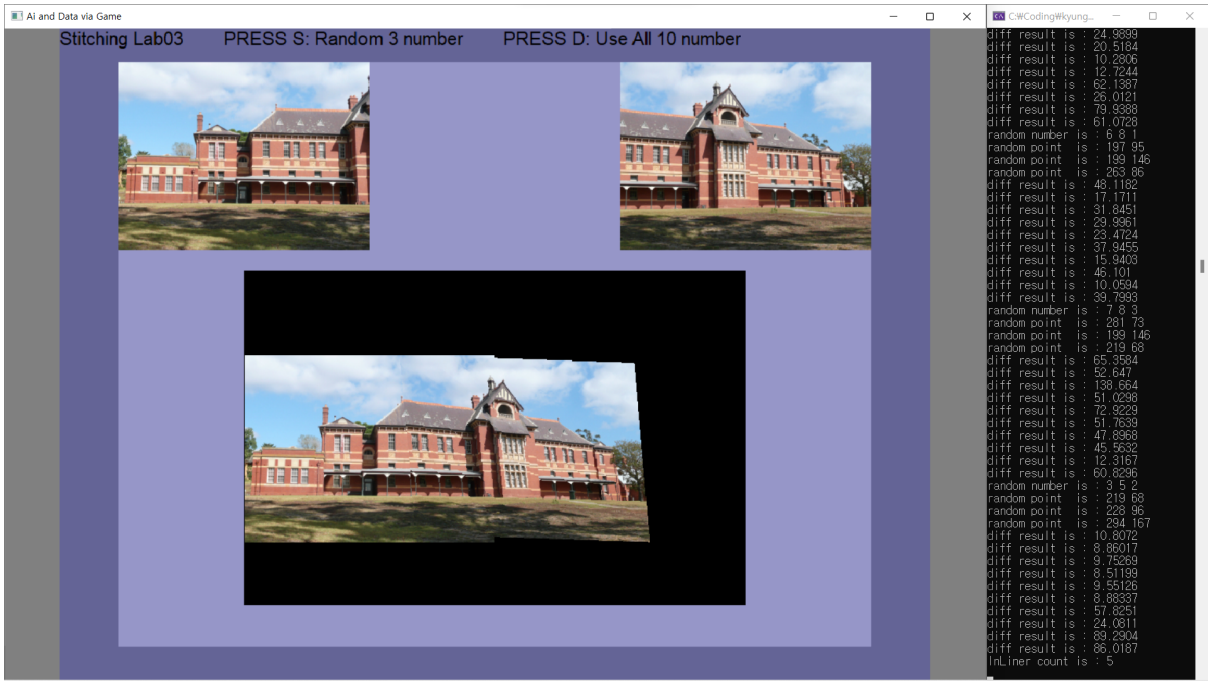
인라이너 개수 세기

```
int InLiner_count = 0;
for (int i = 0; i < 10; i++)
{
  // point1를 w에 대해서 이동시킨 sampleX, sampleY의 숫자를 실제 point2와 비교를 진행
  double sampleX = m_point1[i].X * w[0] + m_point1[i].Y * w[1] + 1 * w[2];
  double sampleY = m_point1[i].Y * w[3] + m_point2[i].Y * w[4] + 1 * w[5];

  // w 이동시킨 point1의 점과 point2의 점의 거리 비교
  double sampleX_abs = std::pow(std::abs(sampleX - m_point2[i].X), 2);
  double sampleY_abs = std::pow(std::abs(sampleY - m_point2[i].Y), 2);

  // 거리의 제곱근을 구함
  double diff = std::sqrt(sampleX_abs + sampleY_abs);

  // 내가 정한 값이 10보다 작으면 InLiner count를 증가함
  if (diff < 10)
    InLiner_count++;
  std::cout << "diff result is : " << diff << "\n";
}
```

괜찮은 데이터값을 추출할 때 까지 반복

```
// 만약 5보다 작으면, while문을 다시 돌아서 과정을 반복함.
// InLiner count가 작으면, matching이 좋지 않기 때문에 반복함
if (InLiner_count >= 5)
{
  std::cout << "InLiner count is : " << InLiner_count << "\n";
  break;
}
```

결과



10개의 숫자 모두 선택

10개의 숫자를 table에 for문을 이용해서 설정

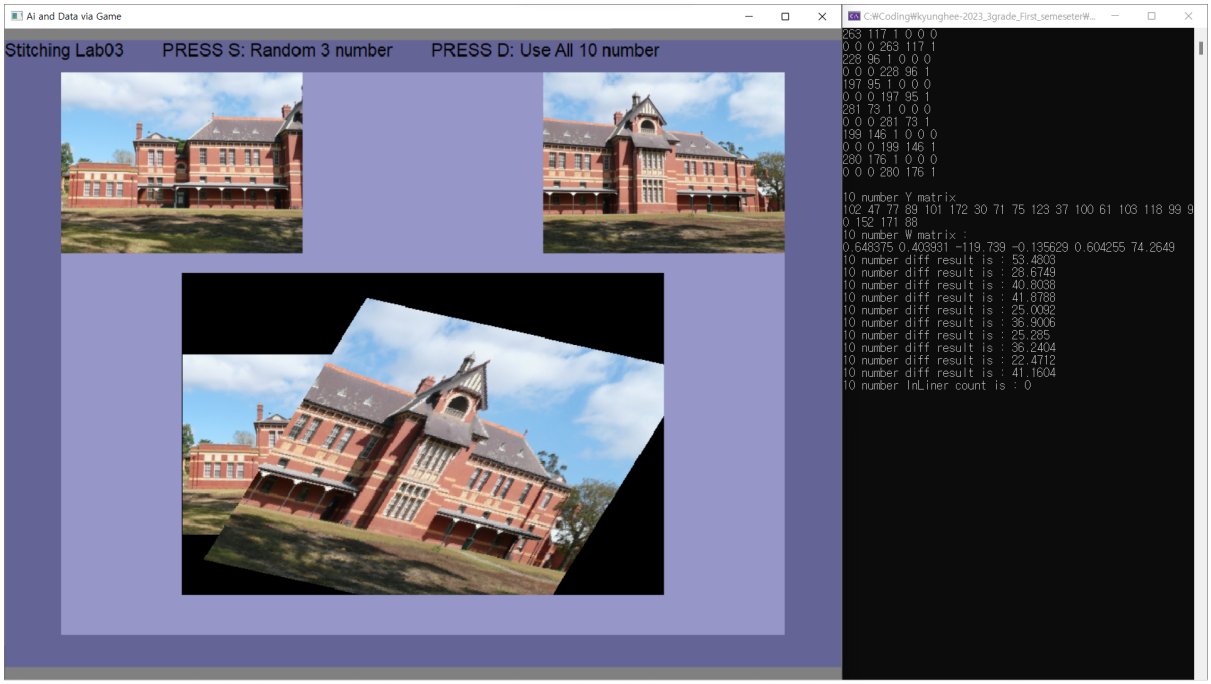
```
// 10개짜리 table를 세팅함.
for (int i = 0; i < 20; i++)
{
  if (i % 2 == 0)
  {
    // Y matrix 설정
    y[i] = m_point2[i/2].X;
    // table matrix 설정
    table_10[i][0] = m_point1[i/2].X;
    table_10[i][1] = m_point1[i/2].Y;
    table_10[i][2] = 1;
    table_10[i][3] = 0;
    table_10[i][4] = 0;
    table_10[i][5] = 0;
  }
  else
  {
    // Y matrix 설정
    y[i] = m_point2[i/2].Y;
    // table matrix 설정
    table_10[i][0] = 0;
    table_10[i][1] = 0;
```

```
    table_10[i][2] = 0;
    table_10[i][3] = m_point1[i/2].X;
    table_10[i][4] = m_point1[i/2].Y;
    table_10[i][5] = 1;
  }
}
```

전에 Random 3와 과정이 동일하지만 범위가 수정됨.

```
LeastSquared(table_10, w, y, 20, 6, false, 0);

// 앞서 진행했던 Random3d의 방법과 동일함 하지만 10개를 비교함
int InLiner_count = 0;
for (int i = 0; i < 10; i++)
{
  double sampleX = m_point1[i].X * w[0] + m_point1[i].Y * w[1] + 1 * w[2];
  double sampleY = m_point1[i].Y * w[3] + m_point2[i].Y * w[4] + 1 * w[5];

  double sampleX_abs = std::pow(std::abs(sampleX - m_point2[i].X), 2);
  double sampleY_abs = std::pow(std::abs(sampleY - m_point2[i].Y), 2);

  double diff = std::sqrt(sampleX_abs + sampleY_abs);
  if (diff < 10)
    InLiner_count++;
  std::cout << "10 number diff result is : " << diff << "\n";
}
```

결과



최종 결과

결론 : 10개를 사용하는 것보다 3개의 랜덤한 넘버를 사용하는 것이 diff값이 작아서 육안으로 보기에 더 유사한 최적인 이미지 결합의 결과를 도출할 수 있었다.

같은 방식에서 diff값이 10보다 작을 때를 비교했을 때 10개의 값에서는 0개의 InLiner가 나왔고, 3개의 Random Number를 사용할 경우 다양한 값들이 나왔지만, 최대 8개까지 InLiner의 값을 가지는 결과를 확인할 수 있었다.

내가 작성한 코드 상에서는 5개 이상의 InLiner를 발견하지 못하면 반복해서 다시 찾도록 구현해 두었다. 때문에 나의 코드를 실행한다면, 인라이너가 5미만의 매우 이상한 이미지 결합의 결과가 도출되지 않는다. (수도 코드에 이렇게 하라고 했던 것 같아서 이렇게 했다)

10개의 점을 모두 사용하였을 때 오히려 정확도가 떨어진다는 것은 이미지의 수평 및 수직또는 비율이 알맞지 않거나, 10개의 점중에 w 이동에 알맞지 않은 점들이 존재할 수도 있다는 것을 의미한다.

만약에 10개의 점이 같은 w이동을 하게 되는 일치되는 점의 집합이라면 수가 많을 수록 더 정확하고 올바른 w값을 구할 수 있게 되는데, 이 경우에서는 올바른 값들만 들어온 것이 아닌 불량값이 들어와 올바른 이미지 결합이 되지 못한 이미지 결과를 생성하게 된다.