

# 컴 퓨 터 애 니 메 이 션

## HW #01

### Natural Cubic Spline

Check list	
1) Add 10 data points (a key)	✓
2) Select/remove 3 data points (r key)	✓
3) Select/drag 2 data points (d key)	✓
4) Select edges and insert 2 data points (i key)	✓

담당 교수님: 최민규 교수님

제출날짜: 2019. 05. 10.

학 과: 컴퓨터소프트웨어학과

학 번: 2014707040

이 름: 유 진 혁

## 1) Add 10 data points (a key)



위와 같이 화면을 10 번 클릭하여 10 개의 data point 를 추가하였다.

```

coordinateX = windowW / (2 * dpi);
coordinateY = windowH / (2 * dpi);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();

glOrtho(-coordinateX, coordinateX, -coordinateY, coordinateY, nearDist, farDist);

void convertCoordinates(double* xPos, double *yPos)
{
    *xPos = *xPos - coordinateX;
    *yPos = coordinateY - *yPos;
}

case 'a':
    if (p.size() < N+1)
    {
        glfwGetCursorPos(window, &xPos, &yPos);
        convertCoordinates(&xPos, &yPos);
        addDataPoint(xPos, yPos);
    }
    break;

```

```
void addDataPoint(float xPos, float yPos)
{
    Vector3f v(xPos, yPos, 0);
    p.push_back(v);
    arrangeMatrix();
    solveLinearSystem();
}
```

인터페이스의 좌표를 모니터의 dpi에 맞춰 조정한다. 마우스 클릭 지점을 읽고 변환하여 STL의 vector에 저장한다. 이것이 Natural cubic spline을 그리는 data point가 된다. 프로그램에서 data point의 최대 개수는 12개로 설정해놓았다.

```
// Arrange elements of Matrix A and b by data points
void arrangeMatrix()
{
    int row = 0;

    b.setZero();

    for (int i = 0; i < (int)p.size()-1; i++, row += 2)
    {
        b(row, 0) = p[i][0];
        b(row, 1) = p[i][1];
        b(row, 2) = p[i][2];

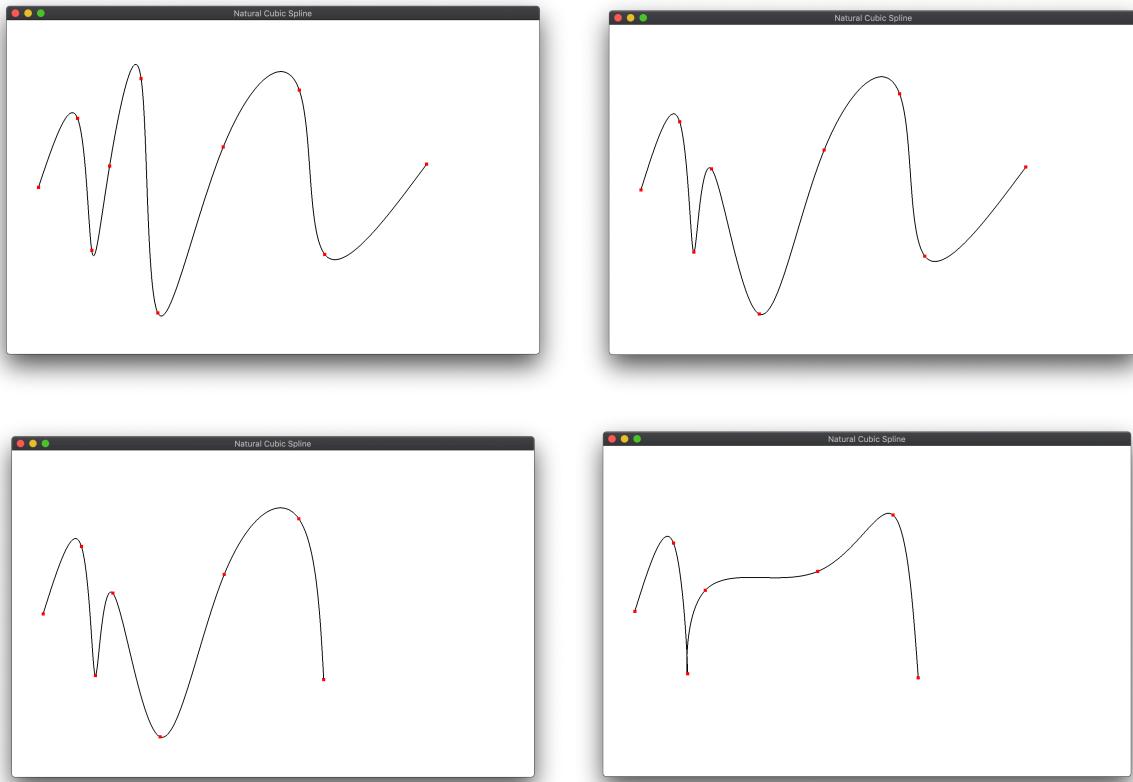
        b(row+1, 0) = p[i+1][0];
        b(row+1, 1) = p[i+1][1];
        b(row+1, 2) = p[i+1][2];
    }

    A.row(4*N-1).setZero();

    if (p.size() >= 2)
    {
        A(4*N-1, 4*(p.size()-2)+2) = 2;
        A(4*N-1, 4*(p.size()-2)+3) = 6;
    }
}
```

행렬 A에서 data point의 변화에 상관없는 부분은 프로그램이 실행될 때 미리 다 채우고, 행렬 A의 끝 점 조건 부분과 행렬 b만 점의 변화가 생길 때마다 위와 같이 새로 값을 채운다.

## 2) Select/remove 3 data points (r key)



1)의 결과에서 5 번째 점을 삭제하고, 그 다음 이전 결과에서의 9 번째 점, 5 번째 점을 차례로 삭제하였다.

```

case 'r':
    glfwGetCursorPos(window, &xPos, &yPos);
    convertCoordinates(&xPos, &yPos);
    removeDataPoint(selectPoint(xPos, yPos));
    break;

// Find the nearest point to input position
int selectPoint(double x, double y)
{
    float minDist = 15;
    int nearestPoint = -1;

    for (int i = 0; i < p.size(); i++)
    {
        float dist = sqrt((x-p[i][0]) * (x-p[i][0]) + (y-p[i][1]) * (y-p[i][1]));
        if (dist < minDist)
        {
            nearestPoint = i;
            minDist = dist;
        }
    }

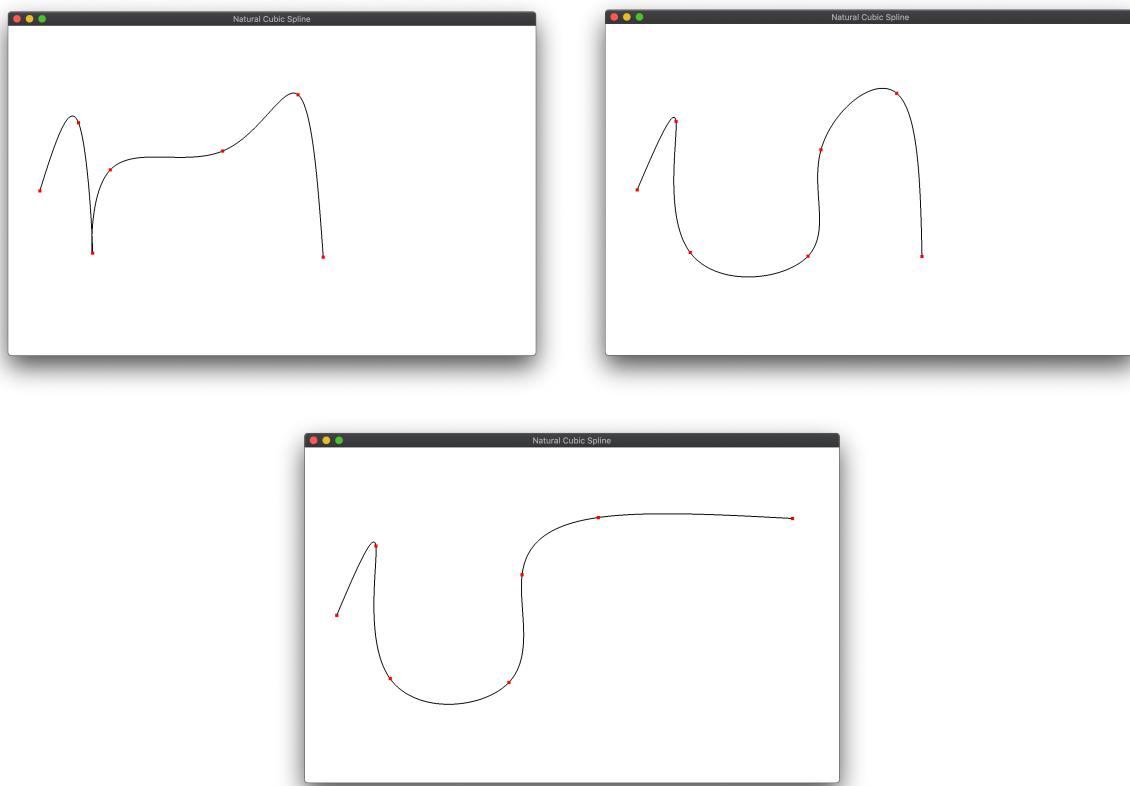
    return nearestPoint;
}

```

```
void removeDataPoint(int index)
{
    if (index >= 0)
    {
        p.erase(p.begin() + index);
        arrangeMatrix();
        solveLinearSystem();
    }
}
```

마우스로 클릭한 지점에서 범위 내 가장 가까운 data point 를 찾고 그 data point 를 vector 에서 삭제한다. 변환된 data point 들을 토대로 행렬을 다시 채우고 식을 푼다.

### 3) Select/drag 2 data points (d key)



2)의 결과에서 4 번째 점, 7 번째 점을 차례로 drag 하여 이동시켰다.

```
case 'd':
    glfwGetCursorPos(window, &xPos, &yPos);
    convertCoordinates(&xPos, &yPos);
    dragDataPoint(selectPoint(xPos, yPos));
    break;
```

```

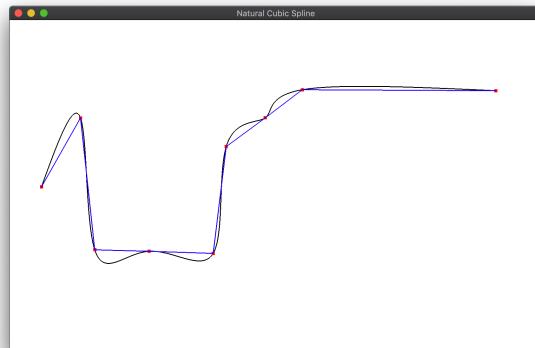
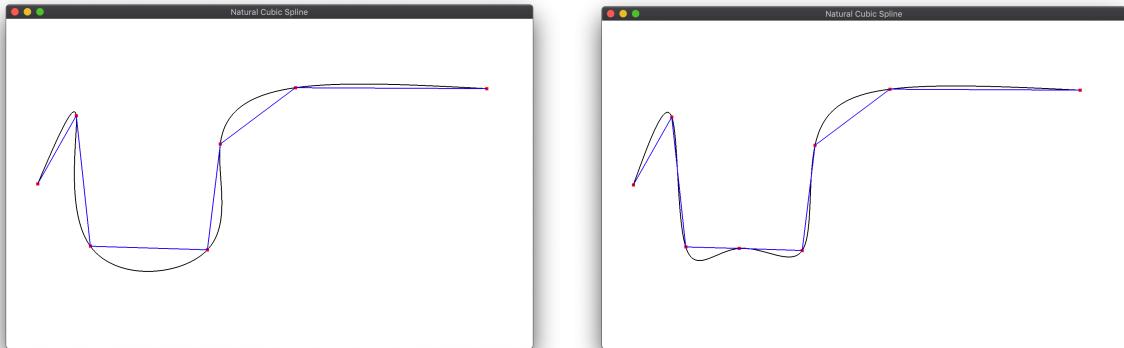
void dragDataPoint(int index)
{
    if (index >= 0)
    {
        dragPoint = index;
        isDragging = true;
    }
}

if (isDragging)           // Drag
{
    glfwGetCursorPos(window, &xPos, &yPos);
    convertCoordinates(&xPos, &yPos);
    p[dragPoint][0] = xPos;
    p[dragPoint][1] = yPos;
    arrangeMatrix();
    solveLinearSystem();
}

```

2)와 마찬가지로 마우스를 클릭한 지점에서 범위 내 가장 가까운 data point 를 찾고, 마우스 버튼을 뗄 때까지 isDragging 변수를 true 설정하여 data point 의 값을 실시간으로 바꾼다.

#### 4) Select edges and insert 2 data points (i key)



i key 를 누르면 각 data point 들을 잇는 선분이 파랗게 그려지고, 한 선분을 클릭하면 그 위치에 data point 가 삽입된다. 먼저 3 번째 점과 4 번째 점을 잇는 선분을 클릭해 그 사이에 data point 를 삽입했고, 그 다음엔 6 번째 점과 7 번째 점을 잇는 선분을 클릭해 data point 를 삽입하였다.

```

case 'i':
    if (p.size() < N+1)
    {
        glfwGetCursorPos(window, &xPos, &yPos);
        convertCoordinates(&xPos, &yPos);
        insertDataPoint(xPos, yPos);
    }
    break;

void insertDataPoint(float xPos, float yPos)
{
    Vector3f inputPos(xPos, yPos, 0);
    float minDist = 10;
    int nearestLine = -1;

    // Find the nearest line to input position
    for (int i = 0; i < (int)p.size()-1; i++)
    {
        // Line segment area
        if (((p[i+1]-p[i]).dot(inputPos-p[i]) > 0) && ((p[i]-p[i+1]).dot(inputPos-p[i+1])) > 0)
        {
            // Distance between a point and line segment
            float dist = ParametrizedLine<float, 3>::Through(p[i], p[i+1]).distance(inputPos);
            if (dist < minDist)
            {
                minDist = dist;
                nearestLine = i;
            }
        }
    }

    if (nearestLine >= 0)
    {
        p.insert(p.begin()+nearestLine+1, inputPos);
        arrangeMatrix();
        solveLinearSystem();
    }
}

```

마우스로 클릭한 지점에서 범위 내 가장 가까운 선분을 찾는다. 클릭한 지점이 각 선분들과 수직인 거리를 구할 수 있는지 확인하고, 구할 수 있으면 점과 선분 사이의 거리를 구한다. 그 거리가 선분을 클릭했다고 할 수 있을 정도로 짧으며 최소인 선분을 찾는다. 그 선분이 몇 번째 data point 들을 잇는지를 확인하여 그 사이에 data point 를 삽입한다.