

컴 퓨 터 애 니 메 이 션

HW #02

Mass-Spring System

Check list	
1. Operations: Create (r key)	✓
2. Operations: Attach (a key)	✓
3. Operations: Drag (r key)	✓
4. Operations: Nail (n key)	✓
5. Damped springs (d key)	✓
6. System instability	✓
7. Obstacles	✓

담당 교수님: 최민규 교수님

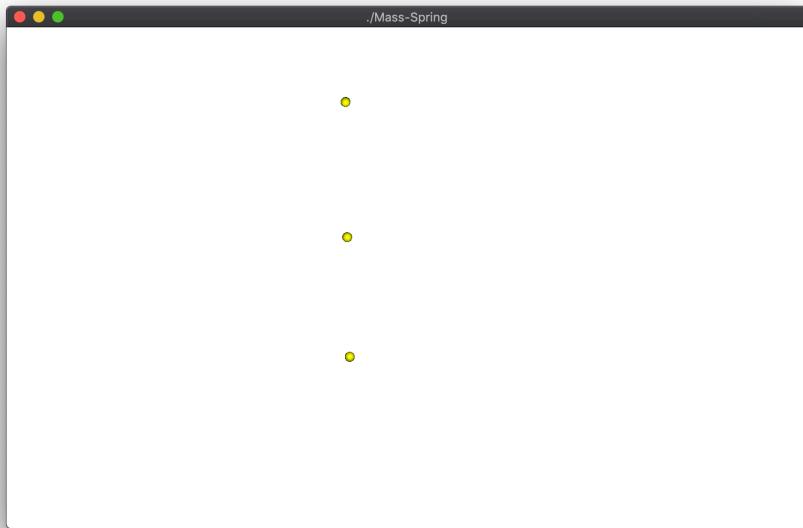
제출날짜: 2019. 06. 19.

학 과: 컴퓨터소프트웨어학과

학 번: 2014707040

이 름: 유 진 혁

1. Operations: Create (r key)



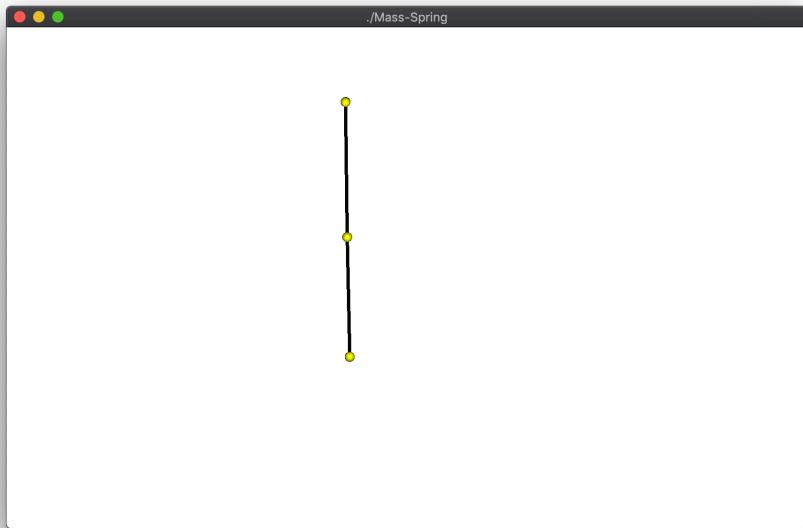
r 키를 누르면 Create/Drag 모드로 들어가게 된다. 이 때 화면을 클릭하면 해당 지점에 particle 이 생성된다. particle 세 개를 생성해보았다.

- 소스 코드

```
void
createParticle(double xpos, double ypos)
{
    p.push_back(Vector3f(xpos, ypos, 0));           // Position
    v.push_back(Vector3f::Zero());                   // Velocity
    constrained.push_back(true);                   // Constrained
    contact.push_back(false);                      // Contact
    contactN.push_back(Vector3f::Zero());           // Contact normal
}
```

마우스 클릭 지점을 받아 새 particle의 위치 상태를 추가하고, 속도, constraint, contact 상태도 초기값으로 넣어준다.

2. Operations: Attach (a key)



a 키를 누르면 Attach 모드로 들어가게 된다. 이 때 spring 으로 연결할 두 particle 을 차례로 선택하면 이를 연결하는 spring 이 생성된다. 수직으로 particle 을 연결해보았다.

- 소스 코드

```
int
selectParticle(double xpos, double ypos)
{
    float minDist = CURSOR_RANGE;    // Minimum distance
    int nearParticle = -1;           // Selected particle

    for (int i = 0; i < p.size(); i++)
    {
        float dist = (Vector3f(xpos, ypos, 0) - p[i]).norm();
        if (dist < minDist)
        {
            nearParticle = i;
            minDist = dist;
        }
    }

    return nearParticle;
}
```

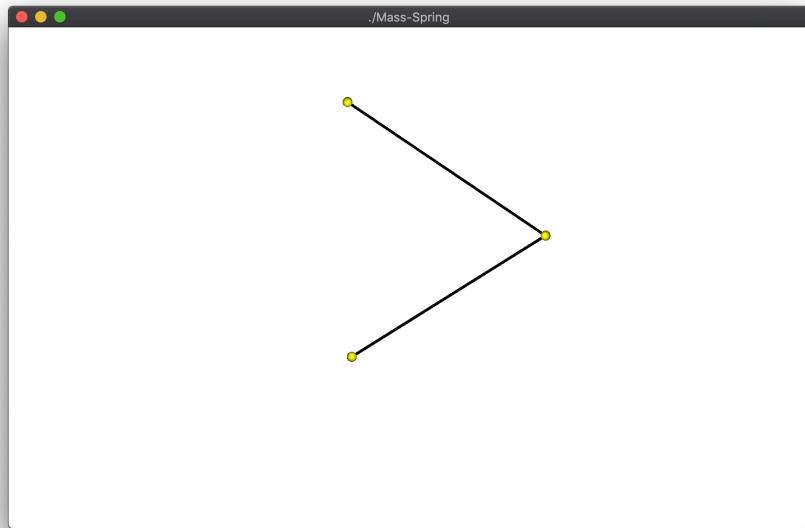
먼저, 마우스 입력 위치와 현재 생성된 particle 들 중 가장 거리가 가까운 particle 을 찾는다. selectParticle 은 가장 가까운 particle 의 배열 인덱스를 반환한다.

```
void
attachSpring(int index)
{
    selected.push_back(index);

    if (selected.size() >= 2)
    {
        if (selected[0] != selected[1])
        {
            // Connection
            e1.push_back(selected[0]);
            e2.push_back(selected[1]);
            l.push_back((p[e1.back()] - p[e2.back()]).norm()); // Rest length
            k.push_back(k0 / l.back()); // Spring constant
        }
        selected.clear();
    }
}
```

선택한 particle 의 인덱스가 들어가는 배열 selected 에 selectParticle 로부터 반환받은 값을 넣어준다. selected 배열의 길이가 2 가 되면 두 개의 particle 이 선택된 것임으로, 해당 배열의 두 값으로 e1 배열과 e2 배열을 채워준다. 이를 이용하여 l 배열과 k 배열도 적절히 계산하여 값을 채워준다.

3. Operations: Drag (r key)



r 키를 누르면 Create/Drag 모드로 들어가게 된다. 이 때 particle 을 선택하여 drag 해주면 particle 의 위치가 변하게 된다. 연결된 spring 의 길이는 초기 길이보다 늘어난다. 가운데의 particle 을 drag 하여 이동시켜보았다.

- 소스코드

```
void
dragParticle(int index)
{
    draggingParticle = index;
    isDragging = true;
}
```

Attach 와 마찬가지로 selectParticle 함수를 이용하여 마우스로 선택된 particle 의 인덱스를 가져와 전역변수로 선언된 draggingParticle 에 넣어준다. 현재 dragging 상태인 particle 을 지정해준 것이다. 그리고 마찬가지로 전역변수로 선언된 isDragging 의 값을 true 로 변경하여 현재 drag 하고 있음을 알린다.

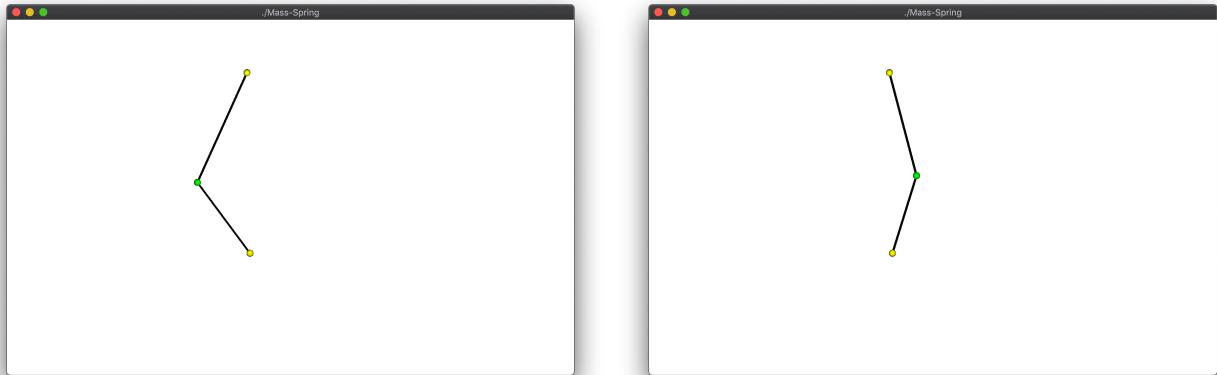
```
// Main loop
while (!glfwWindowShouldClose(window))
{
    if (!pause) update();

    render(window);           // Draw one frame
    glfwSwapBuffers(window); // Swap buffers
    glfwPollEvents();         // Events

    if (isDragging)
    {
        getCursorPos(window, &xpos, &ypos);
        p[draggingParticle][0] = xpos;
        p[draggingParticle][1] = ypos;
    }
}
```

main 함수에서의 main loop 부분이다. isDragging이 true가 되면 매 frame마다 현재 마우스 커서의 좌표를 가져와 dragging 상태의 particle 위치를 변경해준다.

4. Operations: Nail (n key)



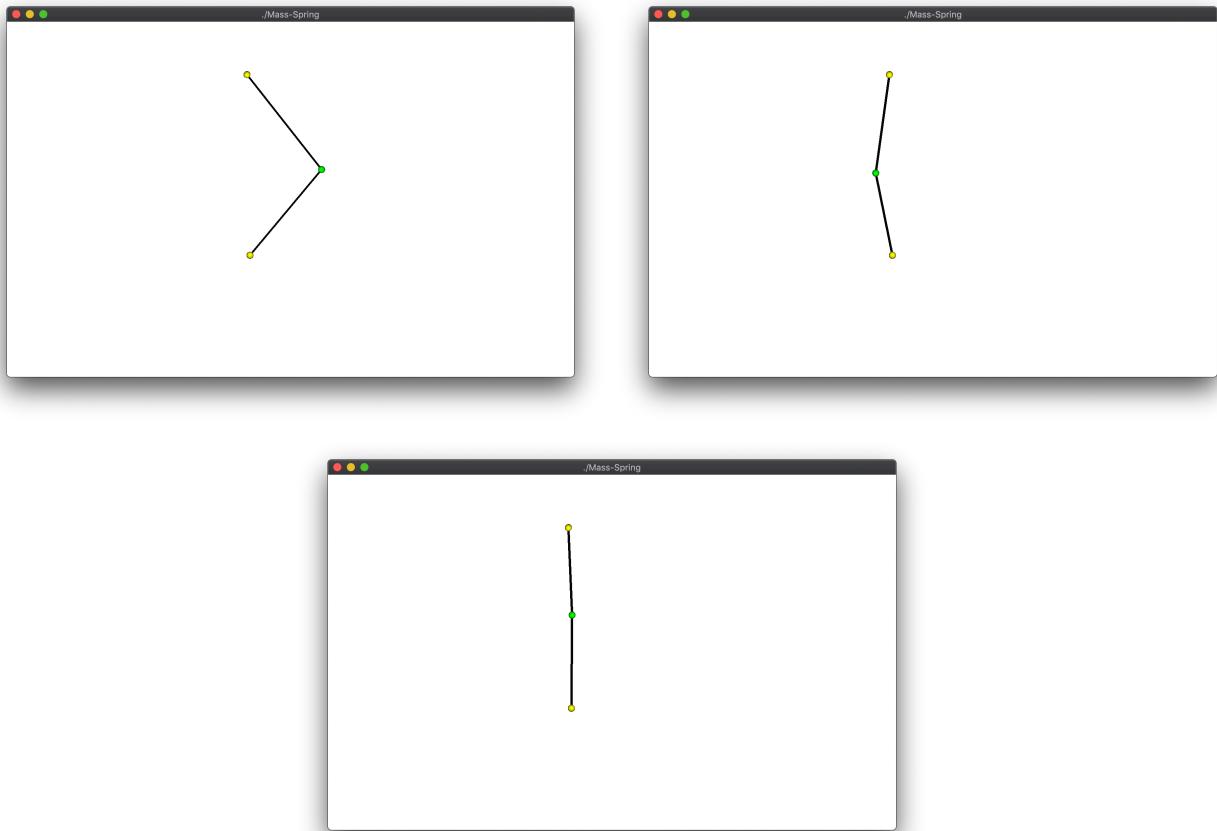
n 키를 누르면 Nail 모드에 들어가게 된다. 이 때 particle 을 클릭하면 particle 의 nail constraint 상태가 변경된다. drag 한 particle 을 선택하여 nail constraint 를 해제해보았다. 그 결과, 해당 particle 이 좌우로 진동하는 모습을 나타내었다.

- 소스코드

```
void  
nailParticle(int index)  
{  
    constrained[index] = !constrained[index];  
}
```

Attach 와 drag 와 마찬가지로, selectParticle 함수를 이용하여 선택된 particle 의 인덱스를 가져와 constraint 상태를 변경해주었다.

5. Damped springs (d key)



d 키를 누르면 damping 이 활성화된다. 스프링 방향으로 damping force 가 작용하여 진동하던 particle 이 점점 제 위치로 돌아오게 된다.

- 소스코드

```
case EULER:
    if (useDamping)
    {
        // Spring direction
        Vector3f v_s(0, 0, 0);
        for (int j = 0; j < e1.size(); j++)
            if (e1[j] == i) v_s += p[e2[j]] - p[e1[j]];
        for (int j = 0; j < e2.size(); j++)
            if (e2[j] == i) v_s += p[e1[j]] - p[e2[j]];
        v_s.normalize();

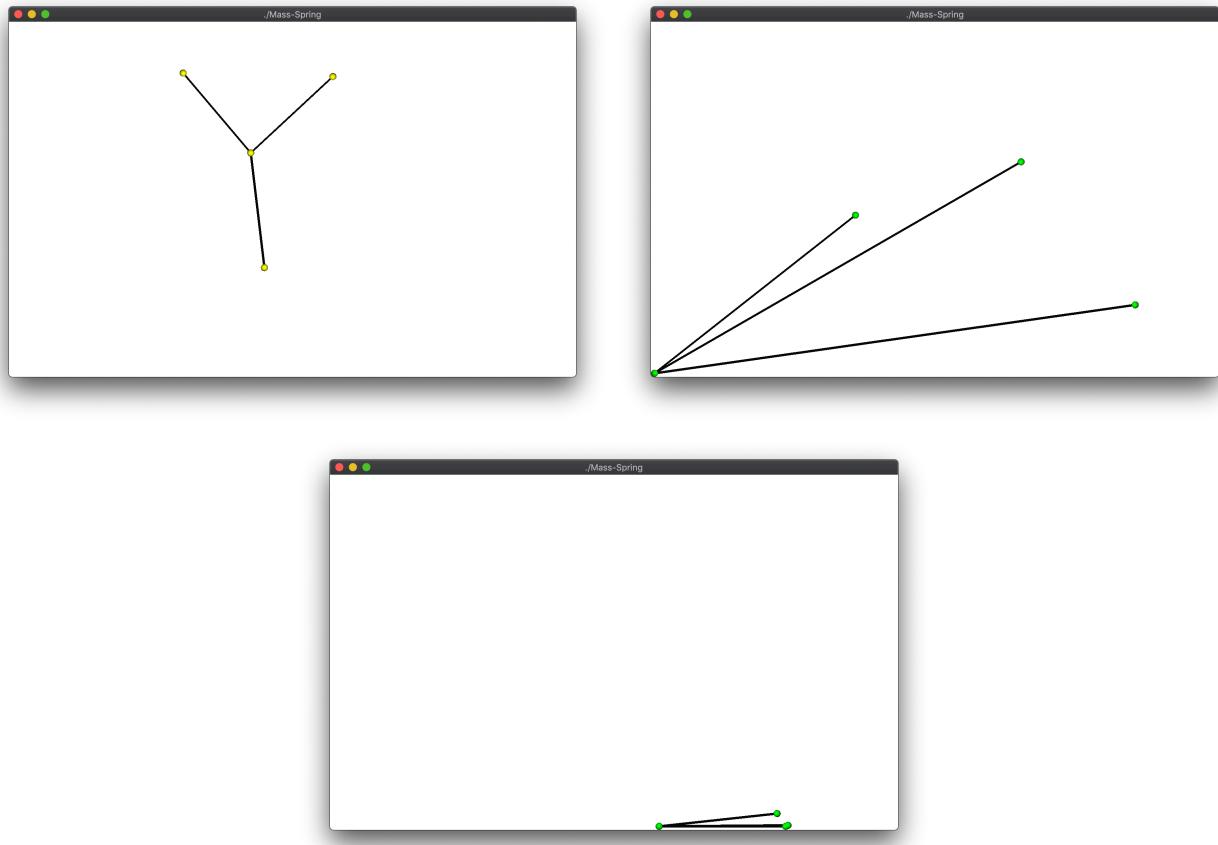
        p[i] += h * v[i];
        v[i] += h * (f[i] - c0 * v[i].dot(v_s) * v_s) / m;
    }
    else
    {
        p[i] += h * v[i];
        v[i] += h * f[i] / m;
    }
    break;
```

```
case MODIFIED_EULER:
    if (useDamping)
    {
        // Spring Direction
        Vector3f v_s(0, 0, 0);
        for (int j = 0; j < e1.size(); j++)
            if (e1[j] == i) v_s += p[e2[j]] - p[e1[j]];
        for (int j = 0; j < e2.size(); j++)
            if (e2[j] == i) v_s += p[e1[j]] - p[e2[j]];
        v_s.normalize();

        v[i] += h * (f[i] - c0 * v[i].dot(v_s) * v_s) / m;
        p[i] += h * v[i];
    }
    else
    {
        v[i] += h * f[i] / m;
        p[i] += h * v[i];
    }
    break;
```

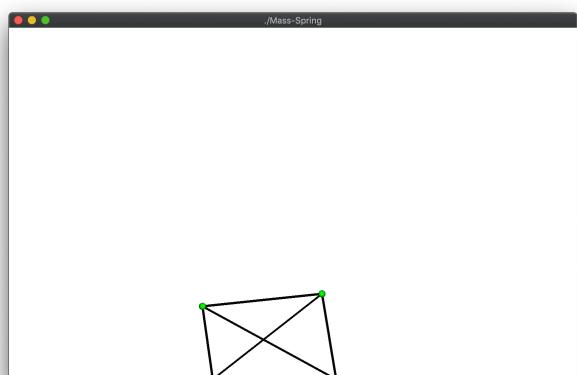
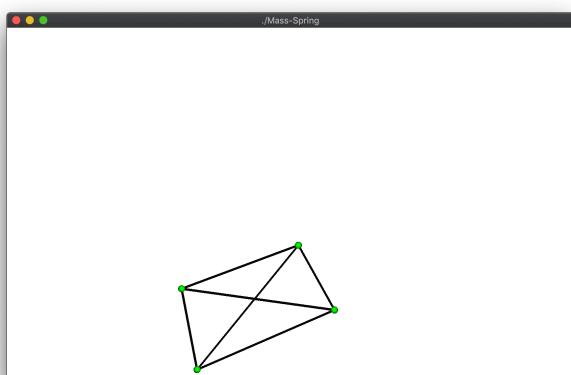
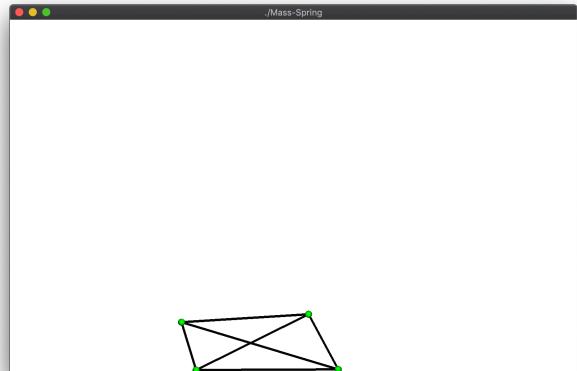
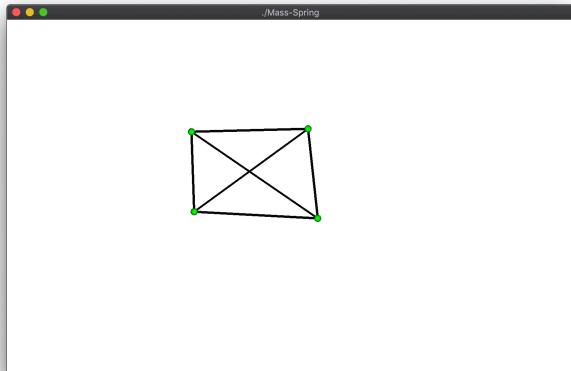
차례로 Euler method 와 Modified Euler method 의 damping 구현 소스 코드이다. 구현 방법은 동일하다. useDamping 이 true 가 되면 각 particle 들에 연결된 spring 의 방향을 구한다. 그리고 현재 particle 의 속도에서 spring 의 방향만 빼내어 이에 반대 방향으로 비례한 damping force 를 더해준다.

6. System instability



Spring 상수의 값을 높이고 Modified Euler method 대신 Euler method 를 사용하면 mass-spring system 이 불안정한 모습을 보이게 된다. 이 때, sub time step 의 횟수를 증가시켜주면 비교적 안정적인 상황으로 변화하는 것을 볼 수 있다.

7. Obstacles



위와 같이 Mass-spring model 을 만들어 자유 낙하 시켰다. Model 이 바닥 면에 충돌하여 bounce 됨을 볼 수 있었다. 시간이 지나 몇 번의 bounce 후에 바닥 면과 접촉 해있는 contact 상황까지 확인할 수 있었다.