컴퓨터 그래픽스 입문

학번 : 2016110056

학과 : 불교학부

이름 : 박승원

날짜 : 2017년 6월 1일

dongguk UNIVERSITY

Lab 13. 물리 엔진

Problem 1. Use particle system to make a fountain similarly to the one below: (5pts)
https://www.youtube.com/watch?v=rbS9VzslFU

Problem 2. Implement damping to make your cloth simulation more stable. (5pts)

제 1 절 분수

Listing 1: fountain header

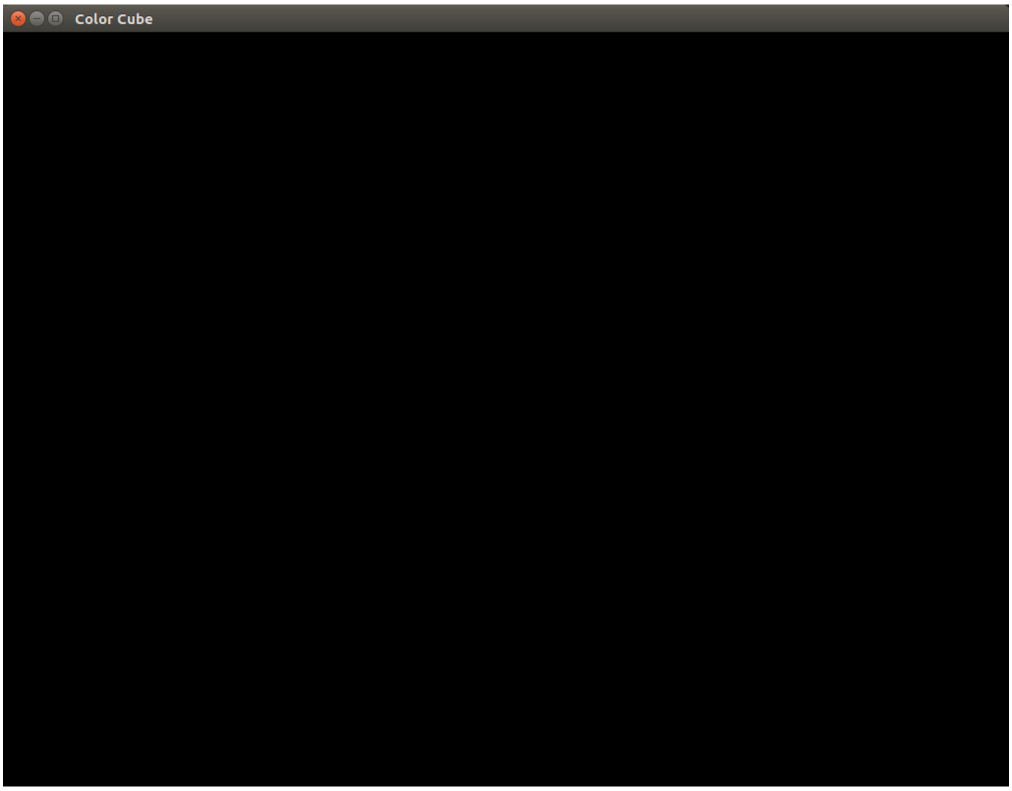
```
#include”matrix.h”
#include<random>

struct Particle
{
     Particle (float x, float y, float z);
    void time_pass(float dt = 0.01);
    Matrix<float> pos, vel, accel;
};

struct Fountain
{
    Fountain();
    std :: normal_distribution <float> ini_velx {0, .05}, ini_vely {10, 1};
    std :: random_device rd;
    std :: vector<Particle> v;
    std :: vector<Matrix<float>> pos;
    void time_pass(float dt = 0.01);
};
```

Listing 2: fountain implementation

```
#include<random>
#include”fountain.h”
using namespace std;

 Particle :: Particle (float x, float y, float z) : vel{x, y, z}, accel{0, −9.8, 0}
{}

void Particle :: time_pass(float dt)
{
    vel[1][2] −= −9.8 * dt; // y velocity
    pos = pos + vel * dt; // position
}

Fountain::Fountain()
{
    for(int i=0; i<1000; i++)
        v.push_back( Particle{ ini_velx(rd), ini_vely(rd), ini_velx(rd)});
}

void Fountain::time_pass(float dt)
{
    pos.clear();
    pos.resize(1000);
    for(auto& a : v) {
        a.time_pass(dt);
        pos.push_back(a.pos);
    }
}
```

Listing 3: fountain main

```
#include<chrono>
#include<thread>
#include<iostream>
#include”glutil.h”
#include”globj.h”
#include”fountain.h”
using namespace std;
extern Matrix<float> KeyBindMatrix;

int main()
{
    if (!glfwInit()) return −1;
    GLFWwindow* window = glfwCreateWindow(1024, 768, ”Color Cube”, NULL, NULL);
    if (!glinit(window)) return −1;

    Fountain foun;
    GLObject ob;
    ob.vertexes(foun.pos);
    ob.colors(vector<Matrix<float>>{1000, {1,0,0}});
    ob.mode(GL_POINTS);
    GLObjs stage;
    stage += ob;
    stage. transfer_all ();
    stage. light ({ // default light
        {0.1, 0.1, 0.1, 1}, //ambient
        {0.5, 0.5, 0.5, 0.5}, // diffuse
        {1, 1, 1, 1}, // specular
        {0, 0, −3, 1} // position 1 means a point 0 means a vector light
    });

    Matrix<float> m{4,4};
    float th = 0;
    while (!glfwWindowShouldClose(window)) {
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        foun.time_pass (0.01) ;

        stage. transfer_data (foun.pos, "vertexes_", stage.vbo[0]);
        stage.matrix(KeyBindMatrix * stage [0]) ;
        stage (0) ;

        glfwSwapBuffers(window);
        glfwPollEvents () ;
        this_thread :: sleep_for (50ms);
    }
    glfwTerminate ();
}
```



glEnable(GL_PROGRMA_POINT_SIZE)한 후에 vertex shader에서 glPointSize를 설정해 준다. 그러나, 검은 화면만 나옴. 버그가 있다. 2시간내에 물리엔진 두 개나 구현하기에는 조금 벅차다.

제 2 절 커튼

k=스프링상수, m=질량, x=위치, c=damping, x0=스프링이 달린 부분의 움직임,위치

$$F = ma = -k(x - x_0) - c\frac{dx}{dt}$$

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + k(x - x_0) = 0 \tag{1}$$

$$let \quad \frac{dx}{dt} = z(t)$$

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = z(t)$$

$$x(t + \Delta t) = z(t)\Delta t + x(t) \tag{2}$$

$$from(1) \quad m\frac{dz}{dt} + cz(t) + k(x - x_0) = 0$$

$$m\frac{z(t + \Delta t) - z(t)}{\Delta t} + cz(t) + k(x - x_0) = 0$$

$$z(t + \Delta t) = (cz(t) + k(x - x_0))\frac{\Delta t}{-m} + z(t) \tag{3}$$

위의 식 2,3으로부터 수치해석적으로 x(t)를 구할 수 있다.

```
float SpringModel::time_pass(float x0, float dt) {
    x = z * dt + x;
    z = (c*z + k*(x - x0)) * dt / -m + z;
}
```

Listing 4: spring system header

```
#include<valarray>
#include<complex>
#include”matrix.h”

struct SpringModel
{//F = ma = m d2x/dt2 = −k(x−x0) − c dx/dt, x = position
// mx" + cx' + k(x−x0) = 0
// x = e^at −> ma^2t^2 + cat + k = 0
    SpringModel(float damping = 0.5, float x = 0, float k = 1, float m = 1);
    float time_pass(float x0 = 0, float dt = 0.1);
    float m = 1; // mass
    float x0, x = 0; // position
    float c1 = 1, c2 = 1; // c1, c2 is determined by initial state :x(0), x'(0)
    float k; // spring constant
    float c; // damping constant
    float w; //T
    float xp = 0; //x'
};

struct SpringModel3D : public SpringModel, public Matrix<float>
{
    SpringModel3D();
    float y0=0, z0=0, yp=0, zp=0;
    void time_pass(float x0 = 0, float y0 = 0, float z0 = 0, float dt = 0.1);
    SpringModel3D& operator=(int n);
    float &x,&y,&z;
};

struct SpringConnection : public Matrix<SpringModel3D>
{
    SpringConnection(int w, int h);
    operator std::vector<Matrix<float>>();
    void time_pass(float dt);
    std::vector<unsigned> indices;

    const float W = 0.02;
    const float H = 0.04;
};
```

Listing 5: spring system implementation

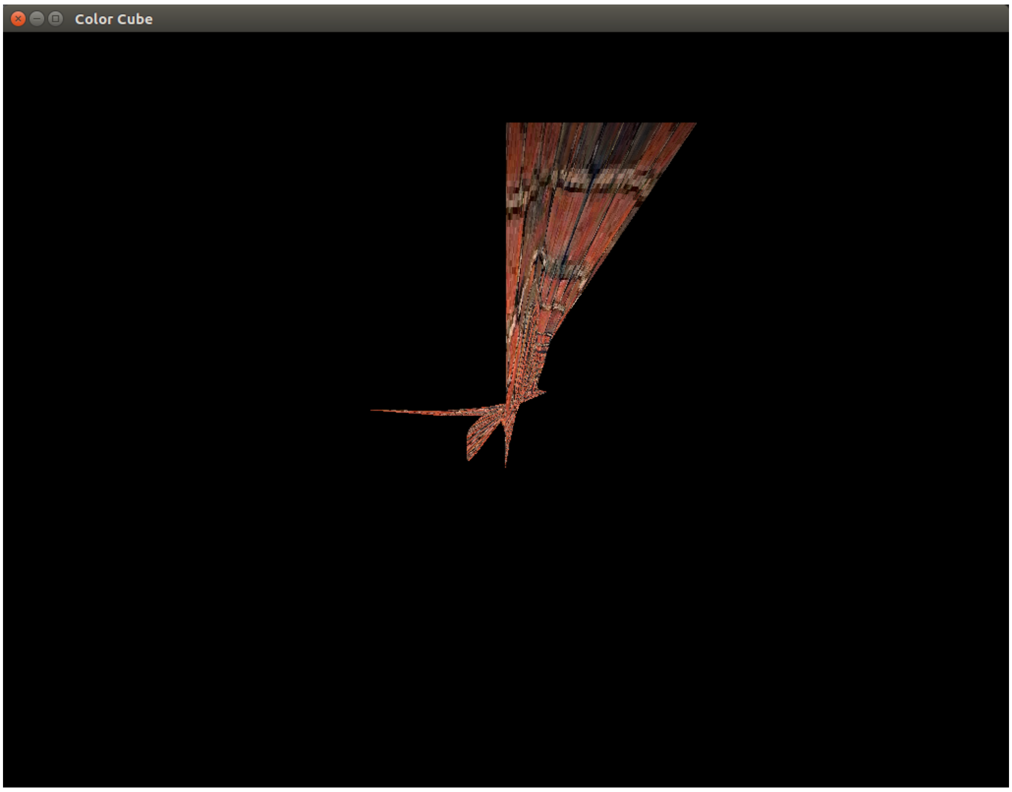
```
#include<iostream>
#include”spring.h”
```

```
using namespace std;

SpringModel::SpringModel(float damping, float x, float k, float m) {
    this->c = damping;
    this->k = k;
    this->m = m;
    this->x = x;
//  w = sqrt (4*m*k - c*c) / (2*m);// underdamping
}
float SpringModel::time_pass(float x0, float dt)
{
//  return x = exp(-c*t/2/m) * (c1 * cos(w*t) + c2 * sin(w*t));
    x = xp * dt + x;
    xp = (c*xp + k*(x - x0)) * dt / -m + xp;
    return x;
}

SpringModel3D::SpringModel3D() : x(*data()), y(*(data()+1)), z(*(data()+2)) { }

SpringModel3D& SpringModel3D::operator=(int n) { return *this; }

void SpringModel3D::time_pass(float x0, float y0, float z0, float dt)
{
    x = xp * dt + x;
    y = yp * dt + y;
    z = zp * dt + z;
    xp = (c*xp + k*(x-x0))*dt/-m + xp;
    yp = (c*yp + k*(y-y0))*dt/-m + yp;
    zp = (x*zp + k*(z-z0))*dt/-m + zp;
}


SpringConnection::operator vector<Matrix<float>>()
{
    return vector<Matrix<float>>{data(), data() + width * height};
}

SpringConnection::SpringConnection(int w, int h) : Matrix<SpringModel3D>{w, h}
{
    for(int i=0; i<w; i++) for(int j=0; j<h; j++) {
        (*this)[i+1][j+1].x = i * W;
        (*this)[i+1][j+1].x0 = i * W;
        (*this)[i+1][j+1].y = j * H;
        (*this)[i+1][j+1].y0 = j * H;
    }
    for(int i=0; i<w-1; i++) for(int j=0; j<h-1; j++) {
        indices.push_back(w* i + j);
        indices.push_back(w* i + j + w);
        indices.push_back(w* i + j + 1);
        indices.push_back(w* i + j + 1);
        indices.push_back(w* i + j + w);
        indices.push_back(w* i + j + w + 1);
    }
}

void SpringConnection::time_pass(float dt)
{
    static float th = 0;
    for(int i=0; i<width; i++) for(int j=0; j<height-1; j++) {// position
        float x0 = 0, y0 = 0, z0 = 0;
        for(int m=-1; m<2; m++) for(int n=-1; n<2; n++) {//connect around
            if(i+1+m > 0 && j+1+n > 0 && i+1+m <= width && j+1+n <= height)
                if(m != 0 || n != 0) {// border check, not itself
                    x0 += (*this)[i+1+m][j+1+n].x - (*this)[i+1][j+1].x;
                    y0 += (*this)[i+1+m][j+1+n].y - (*this)[i+1][j+1].y;
                    z0 += (*this)[i+1+m][j+1+n].z - (*this)[i+1][j+1].z;
                    (*this)[i+1][j+1].time_pass(x0, y0, z0, dt);
                }
        }
    }
}
```

Listing 6: main함수

```
#include<chrono>
#include<thread>
#include<iostream>
#include"glutil.h"
#include"globj.h"
#include"spring.h"
#define W 20
#define H 40
using namespace std;
extern Matrix<float> KeyBindMatrix;
```

```
int main()
{
    if (! glfwInit () ) return −1;
    GLFWwindow* window = glfwCreateWindow(1024, 768, ”Color Cube”, NULL, NULL);
    if (! glinit (window)) return −1;

    SpringConnection cloak{20, 20};
    GLObject ob;
    ob. vertexes (cloak) ;
    ob. indices (cloak . indices ) ;
    ob. texture_file (”brick .png”);

    GLObjs stage;
    stage += ob;
    stage . transfer_all () ;
    stage . light ({ // default light
        {0.1, 0.1, 0.1, 1}, //ambient
        {0.5, 0.5, 0.5, 0.5}, // diffuse
        {1, 1, 1, 1}, // specular
        {0, 0, −3, 1} // position 1 means a point 0 means a vector light
    });

    Matrix<float> m{4,4};
    float th = 0;
    while (!glfwWindowShouldClose(window)) {
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//      for ( int i =0; i<20; i++) {
//          cloak[ i +1][1]. y0 = 0.1* sin (th ) ;
//          cloak[ i +1][1]. z0 = 0.1* cos (th ) ;
//      }
//      th += 0.1;
        cloak . time_pass (0.005) ;
        stage . transfer_data (cloak , ”vertexes_” , stage .vbo [0]) ;
        stage .matrix(KeyBindMatrix * stage [0]) ;
        stage (0) ;

        glfwSwapBuffers(window);
        glfwPollEvents () ;
        this_thread :: sleep_for (50ms);
    }
    glfwTerminate () ;
}
```



그냥 일차원에서 시험을 할 때는 괜찮았으나 삼차원으로 확장하는 과정에서 버그가 있다. 우선 중력 계산을 빠뜨렸다. 스프링 함수에 뭔가 버그가 있다. 자연스러운 커튼이 되지 못하고 이상한 모양으로 움직인다.