COLUMBIA UNIVERSITY

MECE 4510 EVOLUTIONARY COMPUTATION AND DESIGN AUTOMATION

Assignment3 Phase A

Yifan Gui
UNI: yg2751

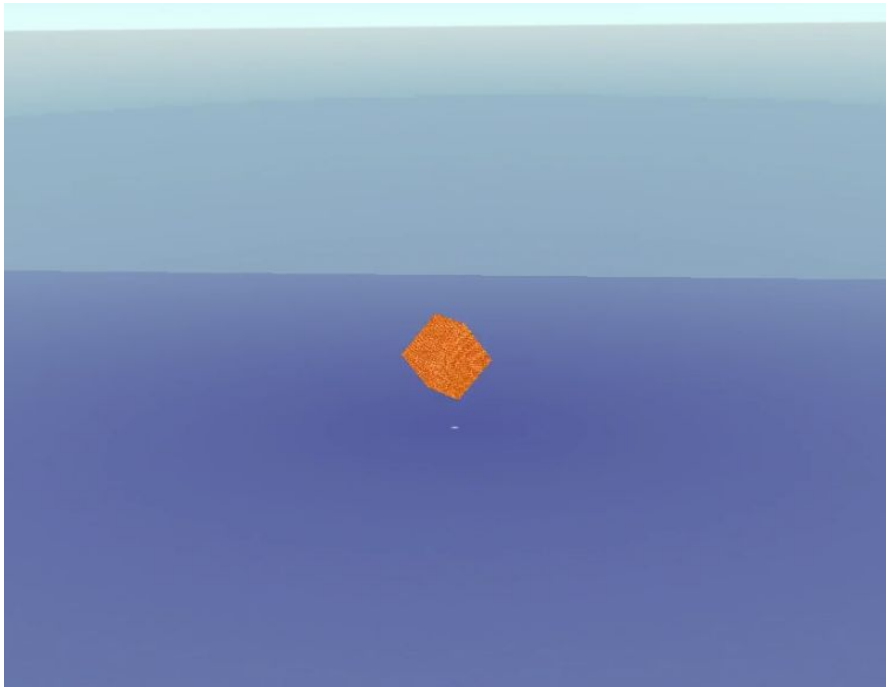
Instructor:
Dr. Hod Lipson

Grace Hour Used: 1
Grace Hour Gained: 0
Grace Hour Remaining: 92

Nov 10, 2020

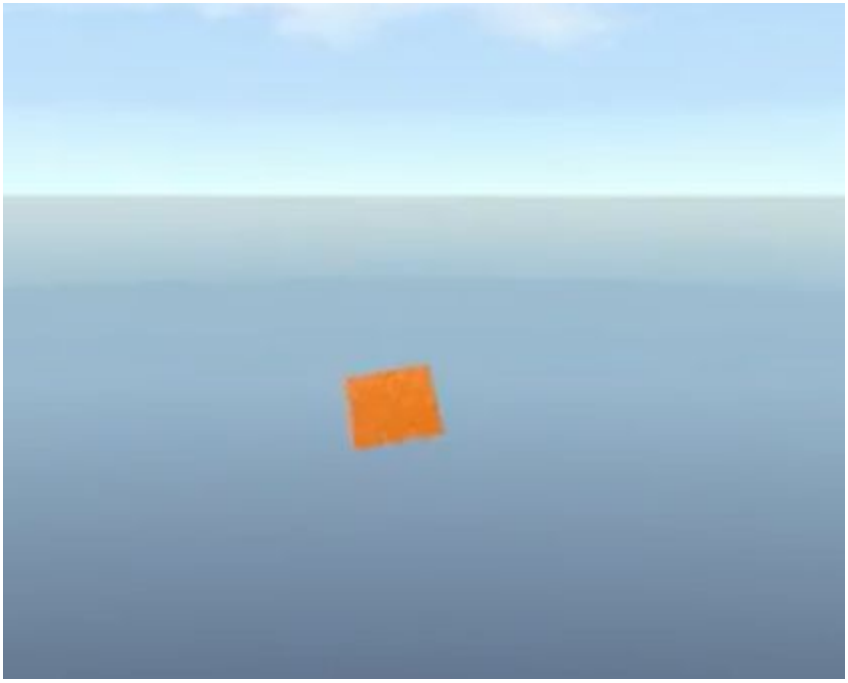
Result Summary

Bouncing Cube



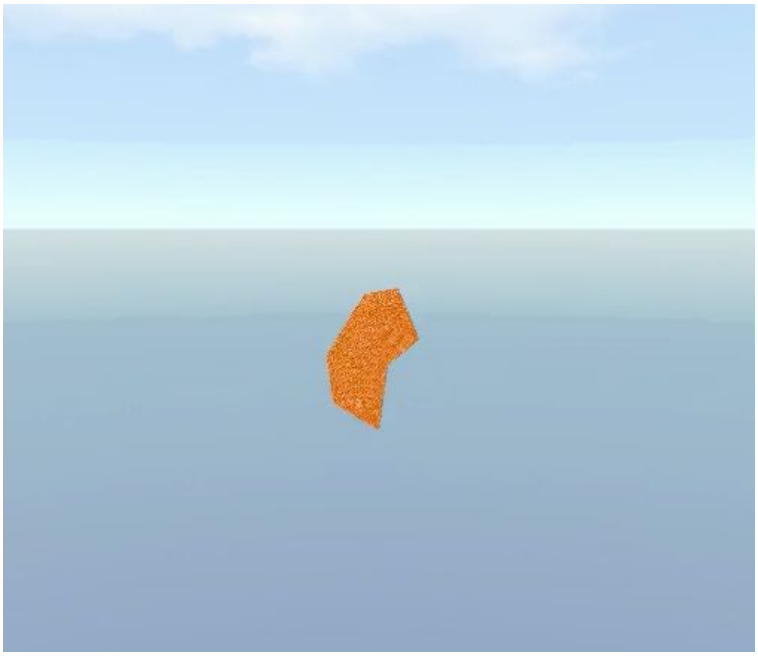
https://youtu.be/sI37rTlmBJg

Breathing Cube



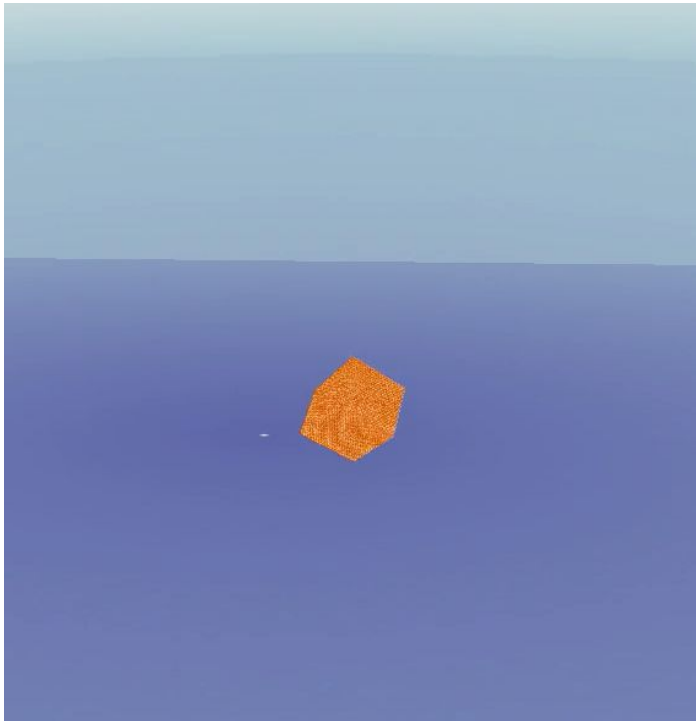
https://youtu.be/XZwx8OKLqj8

Bouncing with Spin



https://youtu.be/2EP1wbMuSOs

Bounding with Dampening



https://youtu.be/uXpsKckdzpY

Method

Description of Design

In phase A, we build a physics simulator which simulates a cube dropping from a height and bounces after it hits ground. It is also able to simulate “breathing” by changing all its springs’ length periodically. In simulation coding, OpenGL was used to build a cubic geometry and background setup, while the physics principles are accomplished by mathematical equations and computer logic statements.

Bouncing Cube parameters

- mass = 0.5 kg
- length = 0.5
- gravity = 9.81
- time step = 0.0008
- spring constant = 2000
- ground restoration constant = 200000
- damping coefficient = 0.999

Breathing Cube parameters

- mass = 0.5 kg
- length = 0.5
- gravity = 9.81
- time step = 0.0008
- spring constant = 2000
- ground restoration constant = 200000
- damping coefficient = 0.999
- breathing mode = sin(10*T)

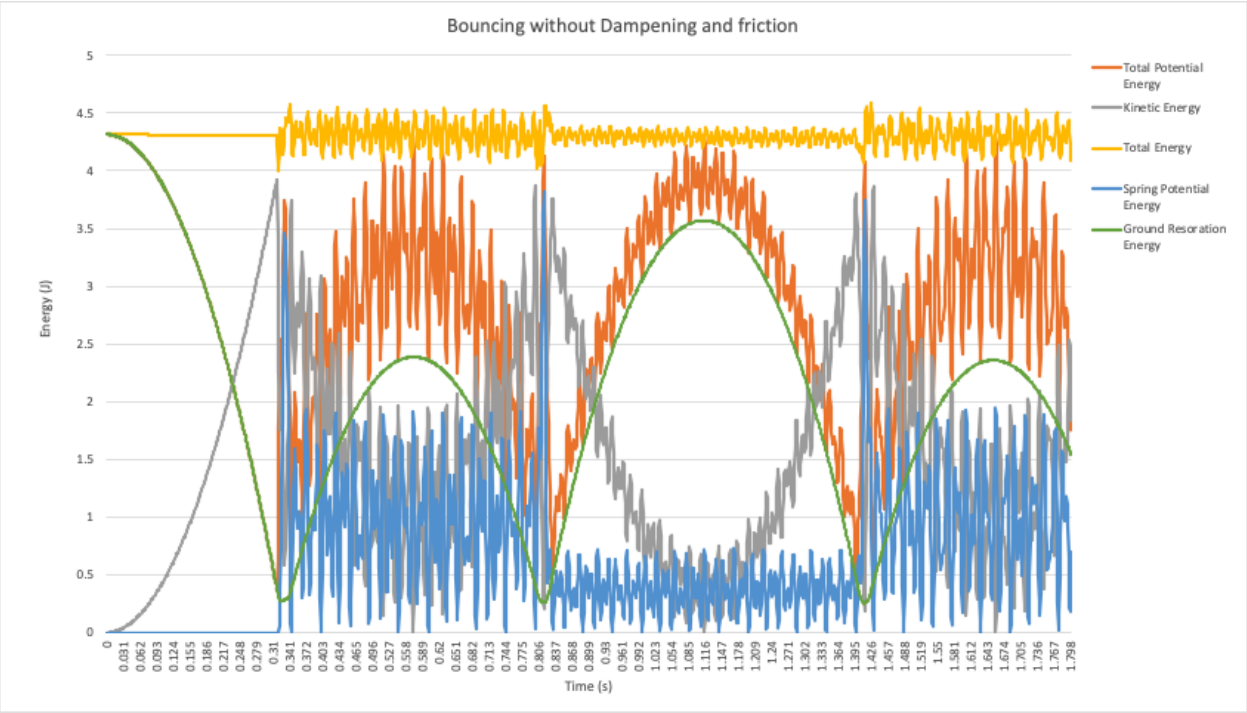
The breathing mode is defined as following:

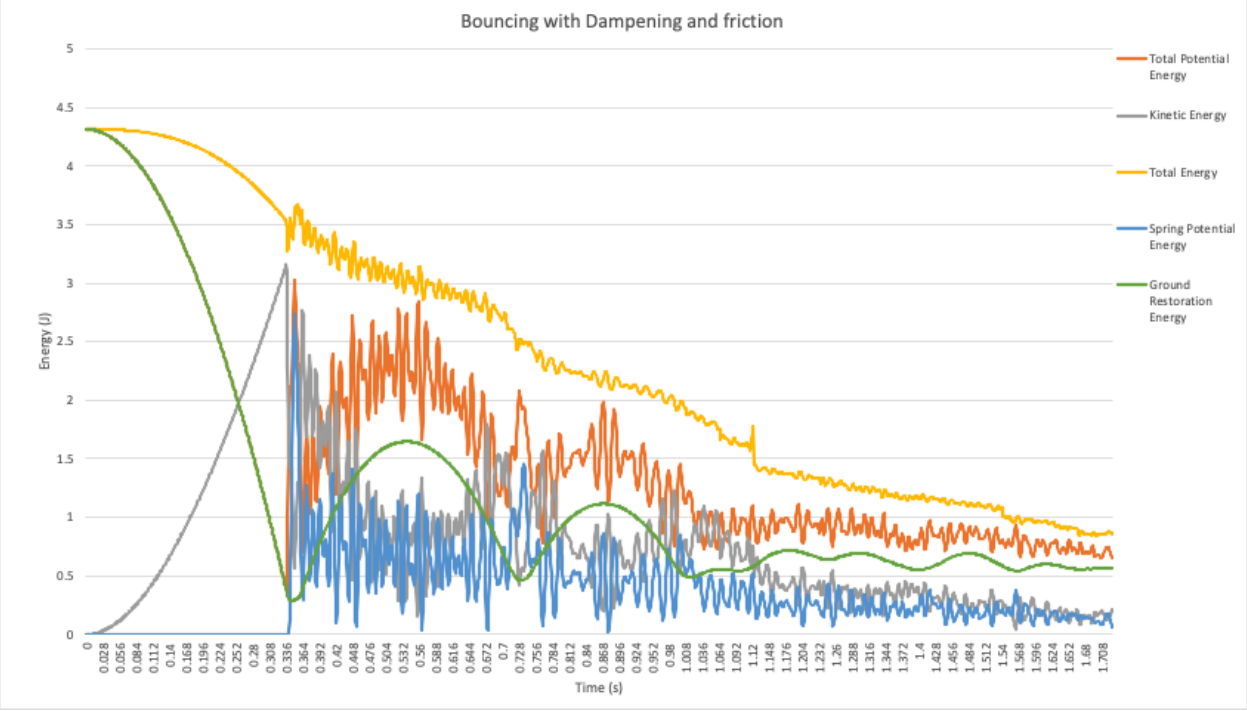
```
void cubeMove(std::vector<MASS>&Mass, std::vector<SPRING>&Spring, int
move){
    // show force

std::vector<std::vector<double>>cubeForces((int)Mass.size(),std::vect
```

```
or<double>(3));

    // calculate springs' force and encode breathing mode as a sine
function
    for (int i = 0;i < (int)Spring.size();i++){
        if (move == 1)
        {
            if (T>0.2){
                Spring[0].l_0 = 1.0*length + 0.8*length*sin(10*T);
// can change if necessary
            }
        }
```

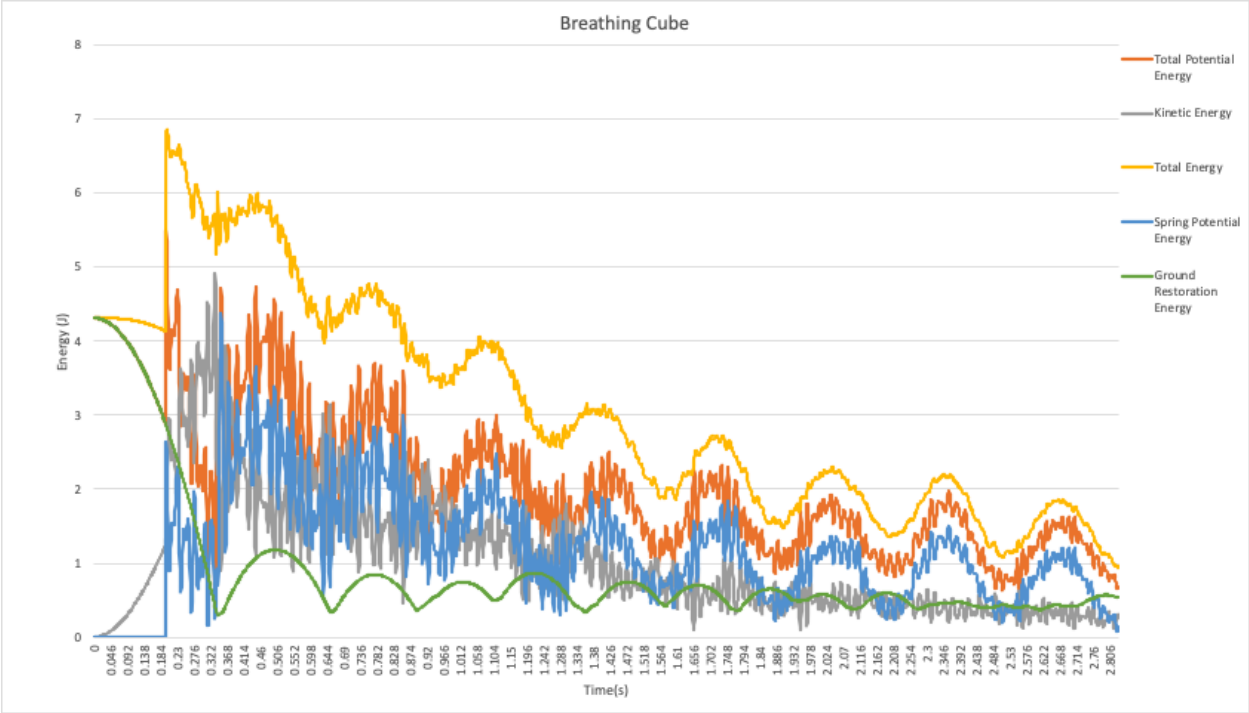
Performance Energy Plots

Bouncing Cube





Breathing Cube



Newtonian Friction

```
// calculate mass force
    for (int i; i < (int)Mass.size();i++){
        cubeForces[i][2] = cubeForces[i][2] - Mass[i].m*gravity;

        if(Mass[i].p[2] < 0){
            cubeForces[i][2] =
cubeForces[i][2]+k_ground*fabs(Mass[i].p[2]);
            groundEnergy = groundEnergy +
k_ground*pow(Mass[i].p[2],2)/2;

            double f_z = cubeForces[i][2];
            double f_xy = sqrt(pow(cubeForces[i][0],2) +
pow(cubeForces[i][1],2));
            // consider ground moving friction changes move trend and
move speed
            if (f_xy < f_z*friction_mu_k){
                cubeForces[i][0] = 0;
                cubeForces[i][1] = 0;
                Mass[i].v[0] = 0;
                Mass[i].v[1] = 0;
            }
            else {
                for(int j =0; j<2; j++){
                    if (cubeForces[i][j] < 0){
                        cubeForces[i][j] = cubeForces[i][j] + f_z*
friction_mu_k*cubeForces[i][j]/f_xy;
                        if(cubeForces[i][j] > 0) {
                            cubeForces[i][j] = 0;
                        }
                    }
                    else {
                        cubeForces[i][j] = cubeForces[i][j] - f_z*
friction_mu_k*cubeForces[i][j]/f_xy;
                        if(cubeForces[i][j] < 0){
                            cubeForces[i][j] = 0;
                        }
                    }
                }
            }
        }
```

Appendix

```
#include "hw3.h"

//physical parameters
//our setting
double mass = 0.5;
double length =  0.5;
double gravity = 9.81;
double T = 0;


double springEnergy = 0;
double gravityEnergy = 0;
double totalEnergy = 0;

double kineticEnergy = 0;
double actualCubeEnergy = 0;
double groundEnergy = 0;
double timeStep = 0.0008;

//given setting
double damping = 0.999;
double k_vertices_soft= 2000;
double k_ground  = 200000;
double friction_mu_s = 1;
double friction_mu_k = 0.8;


//GLUT parameters
int light = 1;
int fov = 45;
int th = 0;
int ph = 0;
int axes = 1;
double asp = 1;
double dim = 1.0;
double skyBoxScale = 1.0;
double res_1 = 2.0;
double res_2 = 2.0;
double res_3 = 2.0;

int emission = 100;
int ambient = 100;
int diffuse = 100;
int specular = 100;
int shininess = 100;
float shiny = 1.0;
float white[] = {1,1,1,1};
float black[] = {0,0,0,1};

unsigned int grassTexture;
unsigned int slimeTexture;
unsigned int skyBoxTexture[10];

//rotation
GLfloat worldRotation[16] = {1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1};
//record time
std::clock_t begin = std::clock();
//double time;
```

```
//define data structure
struct MASS{
    double m;
    double p[3];
    double v[3];
    double a[3];
    double f[3];
};

struct SPRING{
    double l_0;
    double k;
    int m_1;
    int m_2;
};

//
std::vector<MASS> generateMass(double mass,double length, double x, double y, double z){
    std::vector<MASS>Mass(8);
    Mass[0] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[1] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[2] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[3] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[4] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[5] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[6] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    Mass[7] = {mass,{x+length/2,y+length/2,z},{0,0,0},{0,0,0},{0,0,0}};
    return Mass;
}

std::vector<SPRING> generateSpring(double k_vertices_soft)
{
    double length = 0.1;
    double dig_1 = sqrt(pow(length,2)+pow(length,2));
    double dig_2 = sqrt(pow(length,2)+pow(length,2)+pow(length,2));
    std::vector<SPRING>Spring(28);
    Spring[0] = {k_vertices_soft, length,0,1};
    Spring[1] = {k_vertices_soft, dig_1,0,2};
    Spring[2] = {k_vertices_soft, length,0,3};
    Spring[3] = {k_vertices_soft, length,0,4};
    Spring[4] = {k_vertices_soft, dig_1,0,5};
    Spring[5] = {k_vertices_soft, dig_2,0,6};
    Spring[6] = {k_vertices_soft, dig_1,0,7};

    Spring[7] = {k_vertices_soft, length,1,2};
    Spring[8] = {k_vertices_soft, dig_1,1,3};
    Spring[9] = {k_vertices_soft, dig_1,1,4};
    Spring[10] = {k_vertices_soft, length,1,5};
    Spring[11] = {k_vertices_soft, dig_1,1,6};
    Spring[12] = {k_vertices_soft, dig_2,1,7};

    Spring[13] = {k_vertices_soft, length,2,3};
    Spring[14] = {k_vertices_soft, dig_2,2,4};
    Spring[15] = {k_vertices_soft, dig_1,2,5};
    Spring[16] = {k_vertices_soft, length,2,6};
    Spring[17] = {k_vertices_soft, dig_1,2,7};

    Spring[18] = {k_vertices_soft, dig_1,3,4};
    Spring[19] = {k_vertices_soft, dig_2,3,5};
```

```
    Spring[20] = {k_vertices_soft, dig_1,3,6};
    Spring[21] = {k_vertices_soft, length,3,7};

    Spring[22] = {k_vertices_soft, length,4,5};
    Spring[23] = {k_vertices_soft, dig_1,4,6};
    Spring[24] = {k_vertices_soft, length,4,7};

    Spring[25] = {k_vertices_soft, length,5,6};
    Spring[26] = {k_vertices_soft, dig_1,5,7};
    Spring[27] = {k_vertices_soft, length,6,7};

    return Spring;
}

double norm(double x[],std::size_t xz){
    return std::sqrt(std::inner_product(x,x+xz,x,0.0));
}

std::vector<MASS> Masses = generateMass(mass,length,1.0,1.0,1.0);

std::vector<SPRING> Springs = generateSpring(k_vertices_soft);

void showCube(std::vector<MASS>&Mass, std::vector<SPRING>&Spring)
{
    glColor3f(0,1,0); // color
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,shiny);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,white);
    glMaterialfv(GL_FRONT_AND_BACK,GL_EMISSION,black);

    glPushMatrix();
    glMultMatrixf(worldRotation);

    glEnable(GL_TEXTURE_2D);
    glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);

    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);

    // this is left side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(1,0,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[0].p[0],Mass[0].p[1],Mass[0].p[2]);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[3].p[0],Mass[3].p[1],Mass[3].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[7].p[0],Mass[7].p[1],Mass[7].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[4].p[0],Mass[4].p[1],Mass[4].p[2]);
    glEnd();

    // this is right side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(-1,0,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[2].p[0],Mass[2].p[1],Mass[2].p[2]);
```

```
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[1].p[0],Mass[1].p[1],Mass[1].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[5].p[0],Mass[5].p[1],Mass[5].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[6].p[0],Mass[6].p[1],Mass[6].p[2]);
    glEnd();

    // this is front side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(0,0,1);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[0].p[0],Mass[0].p[1],Mass[0].p[2]);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[1].p[0],Mass[1].p[1],Mass[1].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[5].p[0],Mass[5].p[1],Mass[5].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[4].p[0],Mass[4].p[1],Mass[4].p[2]);
    glEnd();


    // this is back side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(0,0,-1);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[2].p[0],Mass[2].p[1],Mass[2].p[2]);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[3].p[0],Mass[3].p[1],Mass[3].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[7].p[0],Mass[7].p[1],Mass[7].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[6].p[0],Mass[6].p[1],Mass[6].p[2]);
    glEnd();


    // this is upper side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(0,1,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[0].p[0],Mass[0].p[1],Mass[0].p[2]);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[1].p[0],Mass[1].p[1],Mass[1].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[2].p[0],Mass[2].p[1],Mass[2].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[3].p[0],Mass[3].p[1],Mass[3].p[2]);
    glEnd();

    // this is down side
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,slimeTexture);
    glBegin(GL_QUADS);
    glNormal3f(0,-1,0);
```

```
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[4].p[0],Mass[4].p[1],Mass[4].p[2]);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(Mass[5].p[0],Mass[5].p[1],Mass[5].p[2]);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(Mass[6].p[0],Mass[6].p[1],Mass[6].p[2]);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(Mass[7].p[0],Mass[7].p[1],Mass[7].p[2]);
    glEnd();

    glPopMatrix();
    glDisable(GL_TEXTURE_2D);
}

void showBack(){
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,shiny);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,white);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,black);

    glPushMatrix();

    glEnable(GL_TEXTURE_2D);
    glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);
    glColor3f(1,1,1);

    glBindTexture(GL_TEXTURE_2D,grassTexture);

    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,grassTexture);
    glBegin(GL_QUADS);
    glNormal3f(0,1,0);

    glTexCoord2f(0.0,0.0);
    glVertex3f(-1.0,+0.0,-1.0);

    glTexCoord2f(0.0,1.0);
    glVertex3f(+1.0,+0.0,-1.0);

    glTexCoord2f(1.0,1.0);
    glVertex3f(+1.0,+0.0,+1.0);

    glTexCoord2f(1.0,0.0);
    glVertex3f(-1.0,+0.0,+1.0);

    glEnd();
    glPopMatrix();
    glDisable(GL_TEXTURE_2D);
}

static void skyBox(double scale){
    glMaterialf(GL_FRONT_AND_BACK,GL_SHININESS,shiny);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,white);
    glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,black);
    glPushMatrix();
    glScaled(scale,scale,scale);
    glEnable(GL_TEXTURE_2D);
    glTexEnvi(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE);
    glColor3f(1,1,1);
    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[0]);
```

```
    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[0]);
    glBegin(GL_QUADS);
    glNormal3f(0,0,-1);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(+5,-5,+5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(-5,-5,+5);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(-5,+5,+5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(+5,+5,+5);
    glEnd();

    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[1]);
    glBegin(GL_QUADS);
    glNormal3f(0,0,+1);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(-5,-5,-5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(+5,-5,-5);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(+5,+5,-5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(-5,+5,-5);
    glEnd();

    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[2]);
    glBegin(GL_QUADS);
    glNormal3f(+1,0,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(-5,-5,+5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(-5,-5,-5);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(-5,+5,-5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(+5,+5,+5);
    glEnd();

    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[3]);
    glBegin(GL_QUADS);
    glNormal3f(-1,0,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(+5,-5,-5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(+5,-5,+5);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(+5,+5,+5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(+5,+5,-5);
    glEnd();

    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[4]);
    glBegin(GL_QUADS);
    glNormal3f(0,-1,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(+5,+5,-5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(+5,+5,+5);
    glTexCoord2f(1.0f,1.0f);
```

```
    glVertex3f(-5,+5,+5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(-5,+5,-5);
    glEnd();

    glBindTexture(GL_TEXTURE_2D,skyBoxTexture[5]);
    glBegin(GL_QUADS);
    glNormal3f(0,+1,0);
    glTexCoord2f(0.0f,0.0f);
    glVertex3f(+5,+0,-5);
    glTexCoord2f(1.0f,0.0f);
    glVertex3f(+5,+0,+5);
    glTexCoord2f(1.0f,1.0f);
    glVertex3f(-5,+0,+5);
    glTexCoord2f(0.0f,1.0f);
    glVertex3f(-5,+0,-5);
    glEnd();

    glPopMatrix();
}

static void cube(double x,double y, double z,double s){
    glPushMatrix();
    glTranslated(x,y,z);
    glScaled(s,s,s);
    glColor3f(1,1,1);
    glutSolidSphere(1.0,16,16);
    glPopMatrix();
}

void cubeMove(std::vector<MASS>&Mass, std::vector<SPRING>&Spring, int move){
    // show force
    std::vector<std::vector<double>>cubeForces((int)Mass.size(),std::vector<double>(3));

    // calculate springs' force
    for (int i = 0;i < (int)Spring.size();i++){
        if (move == 1)
        {
            if (T>0.1){
                Spring[0].l_0 = 1.0*length;   // can change if necessary
            }
        }

    MASS mass_1 = Mass[Spring[i].m_1];
    MASS mass_2 = Mass[Spring[i].m_2];

    double positionChange[3] = {mass_2.p[0]-mass_1.p[0],mass_2.p[1]-mass_1.p[1],mass_2.p[2]-
mass_1.p[2]};
    double delta_L = norm(positionChange,3);
    double force = Spring[i].k*fabs(Spring[i].l_0 - delta_L);

    double direction[3] =
{positionChange[0]/delta_L,positionChange[1]/delta_L,positionChange[2]/delta_L};

    if (delta_L < Spring[i].l_0){
        cubeForces[Spring[i].m_1][0] = cubeForces[Spring[i].m_1][0] - direction[0]*force;
        cubeForces[Spring[i].m_1][1] = cubeForces[Spring[i].m_1][1] - direction[1]*force;
        cubeForces[Spring[i].m_1][2] = cubeForces[Spring[i].m_1][2] - direction[2]*force;
        cubeForces[Spring[i].m_2][0] = cubeForces[Spring[i].m_2][0] + direction[0]*force;
        cubeForces[Spring[i].m_2][1] = cubeForces[Spring[i].m_2][1] + direction[1]*force;
```

```
        cubeForces[Spring[i].m_2][2] = cubeForces[Spring[i].m_2][2] + direction[2]*force;
    }
    else if (delta_L > Spring[i].l_0){
        cubeForces[Spring[i].m_1][0] = cubeForces[Spring[i].m_1][0] + direction[0]*force;
        cubeForces[Spring[i].m_1][1] = cubeForces[Spring[i].m_1][1] + direction[1]*force;
        cubeForces[Spring[i].m_1][2] = cubeForces[Spring[i].m_1][2] + direction[2]*force;
        cubeForces[Spring[i].m_2][0] = cubeForces[Spring[i].m_2][0] - direction[0]*force;
        cubeForces[Spring[i].m_2][1] = cubeForces[Spring[i].m_2][1] - direction[1]*force;
        cubeForces[Spring[i].m_2][2] = cubeForces[Spring[i].m_2][2] - direction[2]*force;
    }
    springEnergy = springEnergy + Spring[i].k * pow((delta_L - Spring[i].l_0),2)/2;

    }

    // calculate mass force
    for (int i; i < (int)Mass.size();i++){
        cubeForces[i][2] = cubeForces[i][2] - Mass[i].m*gravity;

        if(Mass[i].p[2] < 0){
            cubeForces[i][2] = cubeForces[i][2]+k_ground*fabs(Mass[i].p[2]);
            groundEnergy = groundEnergy + k_ground*pow(Mass[i].p[2],2)/2;

            double f_z = cubeForces[i][2];
            double f_xy = sqrt(pow(cubeForces[i][0],2) + pow(cubeForces[i][1],2));
            // consider ground moving friction changes move trend and move speed
            if (f_xy < f_z*friction_mu_k){
                cubeForces[i][0] = 0;
                cubeForces[i][1] = 0;
                Mass[i].v[0] = 0;
                Mass[i].v[1] = 0;
            }
            else {
                for(int j =0; j<2; j++){
                    if (cubeForces[i][j] < 0){
                        cubeForces[i][j] = cubeForces[i][j] + f_z*
friction_mu_k*cubeForces[i][j]/f_xy;
                        if(cubeForces[i][j] > 0) {
                            cubeForces[i][j] = 0;
                        }
                    }
                    else {
                        cubeForces[i][j] = cubeForces[i][j] - f_z*
friction_mu_k*cubeForces[i][j]/f_xy;
                        if(cubeForces[i][j] < 0){
                            cubeForces[i][j] = 0;
                        }
                    }
                }
            }
        }


        // acceleration
        Mass[i].a[0] = cubeForces[i][0]/Mass[i].m;
        Mass[i].a[1] = cubeForces[i][1]/Mass[i].m;
        Mass[i].a[2] = cubeForces[i][2]/Mass[i].m;

        // speed
        Mass[i].v[0] = (Mass[i].v[0]+Mass[i].a[0]*timeStep) * damping;
        Mass[i].v[1] = (Mass[i].v[1]+Mass[i].a[1]*timeStep) * damping;
```

```
        Mass[i].v[2] = (Mass[i].v[2]+Mass[i].a[2]*timeStep) * damping;

        // position
        Mass[i].p[0] = Mass[i].p[0] + Mass[i].v[0]*timeStep;
        Mass[i].p[1] = Mass[i].p[1] + Mass[i].v[1]*timeStep;
        Mass[i].p[2] = Mass[i].p[2] + Mass[i].v[2]*timeStep;

        gravityEnergy = gravityEnergy + Mass[i].m * gravity * Mass[i].p[2];
        kineticEnergy = kineticEnergy + Mass[i].m * pow(norm(Mass[i].v,3),2)/2;
    }

    totalEnergy = springEnergy + gravityEnergy + groundEnergy;
    actualCubeEnergy = totalEnergy + kineticEnergy;

    // output info

    showCube(Mass,Spring);
    T = T + timeStep;
    kineticEnergy = 0;
    springEnergy = 0;
    gravityEnergy =0;
    totalEnergy = 0;
    groundEnergy = 0;
    actualCubeEnergy =0;
}

void Output(const char*format,...){
    char buf[LENGTH];
    char* ch = buf;
    va_list args;
    va_start(args,format);
    vsnprintf(buf,LENGTH,format,args);
    va_end(args);
    while(*ch)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,*ch++);
}

void display(){
    const double len = 2.0;
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();

    double eye_x = -2*dim*Sin(th)*Cos(ph);
    double eye_y =  2*dim*Sin(ph);
    double eye_z =  2*dim*Cos(th)*Cos(ph);

    gluLookAt(eye_x,eye_y,eye_z,0,0,0,0,Cos(ph),0);
    if (light){
        float Ambient[] = {0.01*ambient,0.01*ambient,0.01*ambient,2.0};
        float Diffuse[] = {0.01*diffuse,0.01*diffuse,0.01*diffuse,2.0};
        float Specular[] = {0.01*specular,0.01*specular,0.01*specular,2.0};

        float position[] = {1,0.5,1,1};

        glColor3f(16,16,16);
        cube(position[0],position[1],position[2],0.001);

        glEnable(GL_NORMALIZE);
        glEnable(GL_LIGHTING);
```

```
        glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);

        glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_LIGHT0);

        glLightfv(GL_LIGHT0,GL_AMBIENT,Ambient);
        glLightfv(GL_LIGHT0,GL_DIFFUSE,Diffuse);
        glLightfv(GL_LIGHT0,GL_SPECULAR,Specular);
        glLightfv(GL_LIGHT0,GL_POSITION,position);
    }

    cubeMove(Masses,Springs,0);
    skyBox(skyBoxScale);
    glColor3f(1,1,1);
    if(axes){
        glBegin(GL_LINE);
        glVertex3d(0.0,0.0,0.0);
        glVertex3d(len,0.0,0.0);
        glVertex3d(0.0,0.0,0.0);
        glVertex3d(0.0,len,0.0);
        glVertex3d(0.0,0.0,0.0);
        glVertex3d(0.0,0.0,len);
        glEnd();
    glRasterPos3d(len,0.0,0.0);
    Output("X");
    glRasterPos3d(0.0,len,0.0);
    Output("Y");
    glRasterPos3d(0.0,0.0,len);
    Output("Z");
    }

    glFlush();
    glutSwapBuffers();
}

void unique(int key,int x, int y){
    if (key == GLUT_KEY_RIGHT)
        th = th + 5;
    else if (key == GLUT_KEY_LEFT)
        th = th -5;
    else if (key == GLUT_KEY_UP){
        if (ph + 5 < 90) {
            ph = ph + 5;
        }
    }
    else if (key == GLUT_KEY_DOWN){
        if (ph - 5 > 0){
            ph -= 5;
        }
    }

    th %= 360;
    ph %= 360;
    glutPostRedisplay();
}

void projection(double fov, double asp, double dim){
    glMatrixMode(GL_PROJECTION);

    glLoadIdentity();
```

```
    if (fov){
        gluPerspective(fov,asp,dim/16,16*dim);
    }
    else
        glOrtho(-asp*dim, asp*dim,-dim,+dim,-dim,+dim);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void key(unsigned char ch,int x,int y){
    if (ch == 27)
        exit(0);
    else if (ch == '0'){
        th = 0;
        ph = 0;
    }
    else if (ch == 'a' || ch == 'A'){
        axes = 1-axes;
    }
    else if (ch == '-' && ch > 1)
        fov ++;
    else if (ch == '=' && ch < 179)
        fov --;
    else if (ch == GLUT_KEY_PAGE_UP && dim > 1){
        dim -= 1;
    }
    else if (ch == GLUT_KEY_PAGE_DOWN){
        dim += 1;
    }
    else if (ch == 'w'){
        Masses[0].v[2] = 1;
    }

    th %= 360;
    ph %= 360;

    projection(fov,asp,dim);
    glutPostRedisplay();
}


void reshape(int width,int height){
    asp = (height>0)? (double) width/height :1;
    glViewport(0,0,width,height);
    projection(fov,asp,dim);
}

void idle(){
    glutPostRedisplay();
}



int main(int argm, char* argn[]){
    glutInit(&argm,argn);
    glutInitWindowSize(1200,800);
    glutInitDisplayMode(GLUT_RGB|GLUT_DEPTH|GLUT_DOUBLE);

    glutCreateWindow("Bouncing cube");
    glutIdleFunc(idle);
```

```
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutSpecialFunc(unique);
    glutKeyboardFunc(key);

    grassTexture = getTexture("./texture/grass.bmp");
    slimeTexture = getTexture("./texture/slime.bmp");

    skyBoxTexture[0] = getTexture("./texture/skybox1.bmp");
    skyBoxTexture[1] = getTexture("./texture/skybox2.bmp");
    skyBoxTexture[2] = getTexture("./texture/skybox3.bmp");
    skyBoxTexture[3] = getTexture("./texture/skybox4.bmp");
    skyBoxTexture[4] = getTexture("./texture/skybox5.bmp");
    skyBoxTexture[5] = getTexture("./texture/skybox6.bmp");

    //cubeTexture = getTexture("./texture/cube.bmp");
    glutMainLoop();
    return 0;

}
```