Name: Hongbo Zhu Cooperator: Chengxu Yang

UNI:hz2629 UNI: cy2554

Course name: Evolutionary Computation & Design Automation (MECS E4510)

Instructor: Hod Lipson

Date Submitted:2019/11/8/10PM.

Grace hours used:0

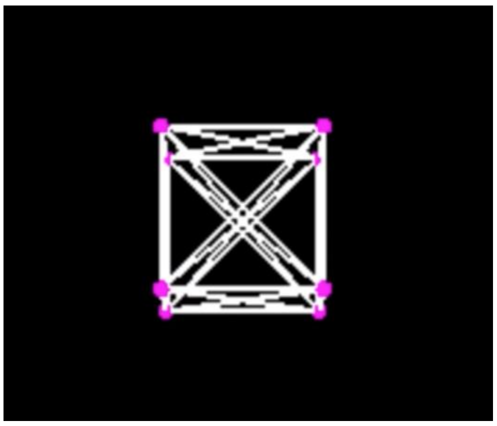
Grace hours remaining: 261.5

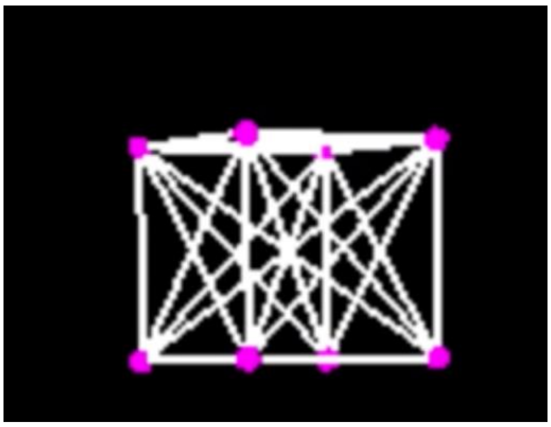
Chenxu Yang (159)

Hongbo Zhu (102.5)

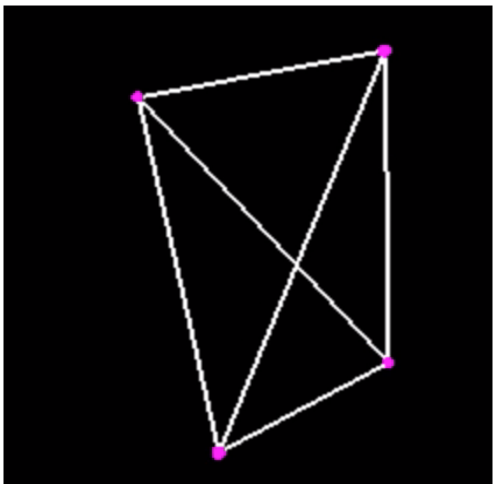
## Results summary table

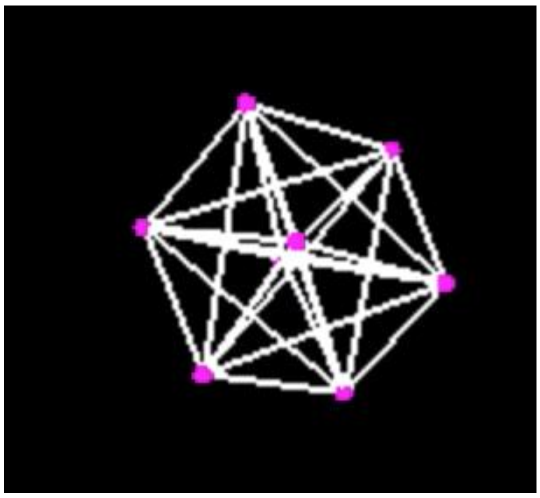
### Bouncing cube and Breathing cube



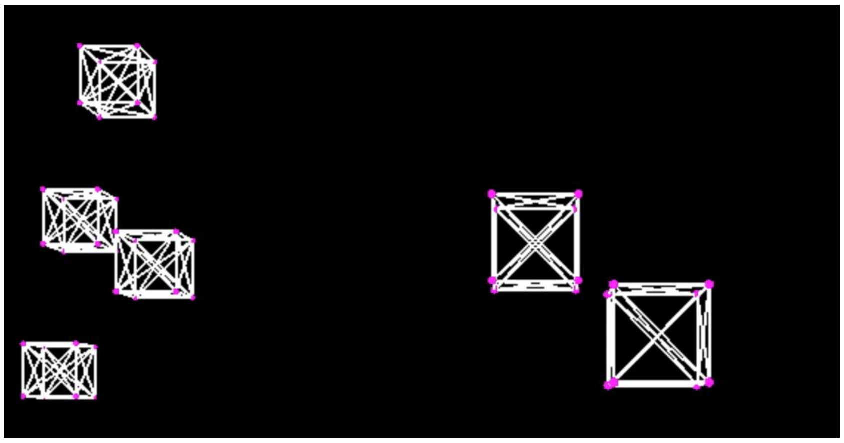


### tetrahedron and cube with spin





### Multiple cubes dropped



## Video link:

Bouncing cube:

Link:https://drive.google.com/open?id=1R0DkHFKYHdfhTJMp6fLYmZmFhRlaPFQg

Breathing cube:

Link: https://drive.google.com/open?id=1Gbms7_pbAulV7xGB8wtoAu9gA1PwdK_s

Tetrahedron:

Link: https://drive.google.com/open?id=1VWJ2rYN4ceTIRxxzU0H0iCH9SZ5pjhEO

Cube with a slight spin:

Link: https://drive.google.com/open?id=1PJegCaw9QdixJO_TMDiJ1KKYDxCQ1IgS

Multiple cubes dropped:

Link: https://drive.google.com/open?id=1Do1_YDe7PJq1IACNqInKhvz15Oznvi5V

# Description of Design:

In this assignment, a cube is emulated in a physics simulator which is created by C++ program and OpenGL. This cube is able to bounce on a flat ground and it can also “breathe” by changing the length of some springs inside it.

To emulate the movement of this cube, some coefficients are chosen:

## Bouncing Cube:

Mass: 0.1kg

Length of each spring: 0.1m

Gravity: 9.81N/kg

Time step: 0.001s

Spring constant: 10000N/m

Ground restoration constant: 100000N/m

Dampening constant: 0.9999957

**Breathing Cube:**

**Method: change the 4 longest springs’ length in a small sin wave**

Mass: 0.1kg

Length of each spring: 0.1m

Gravity: 9.81N/kg

Time step: 0.001s

Spring constant: 10000N/m

Ground restoration constant:100000N/m

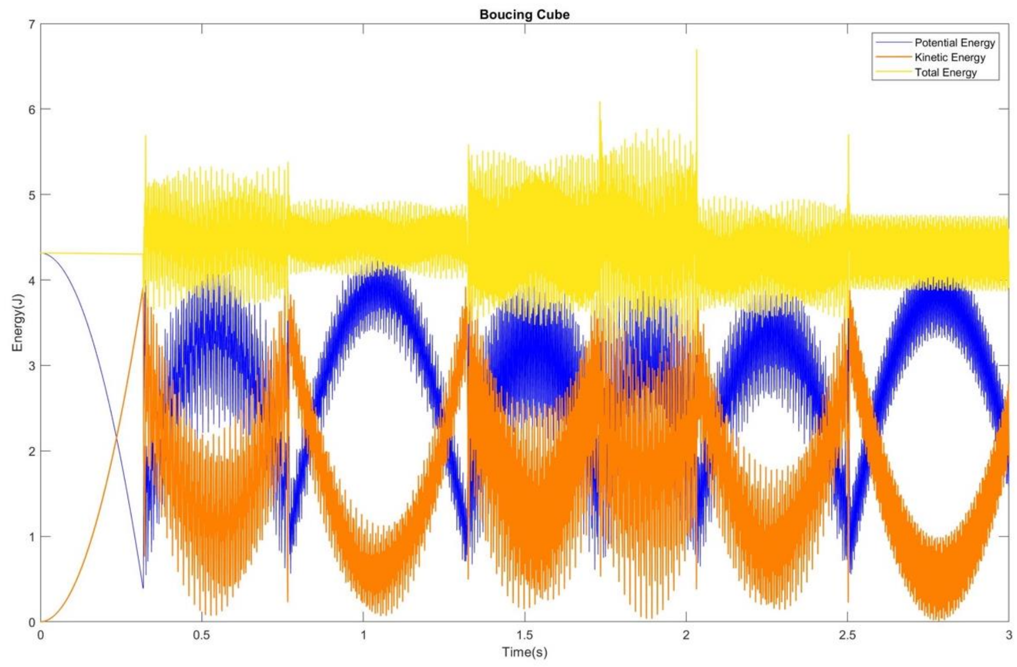
Dampening constant: 0.9999957

## Analysis:

By choosing coefficients properly, the cube is able to move in the way we want. For some coefficients, like dampening constant, if it is 1, the energy will keep increasing and, in the simulator, the velocity of the cube increase as well. If it is too small, total energy falls quickly and finally the cube stops to move and stay still.

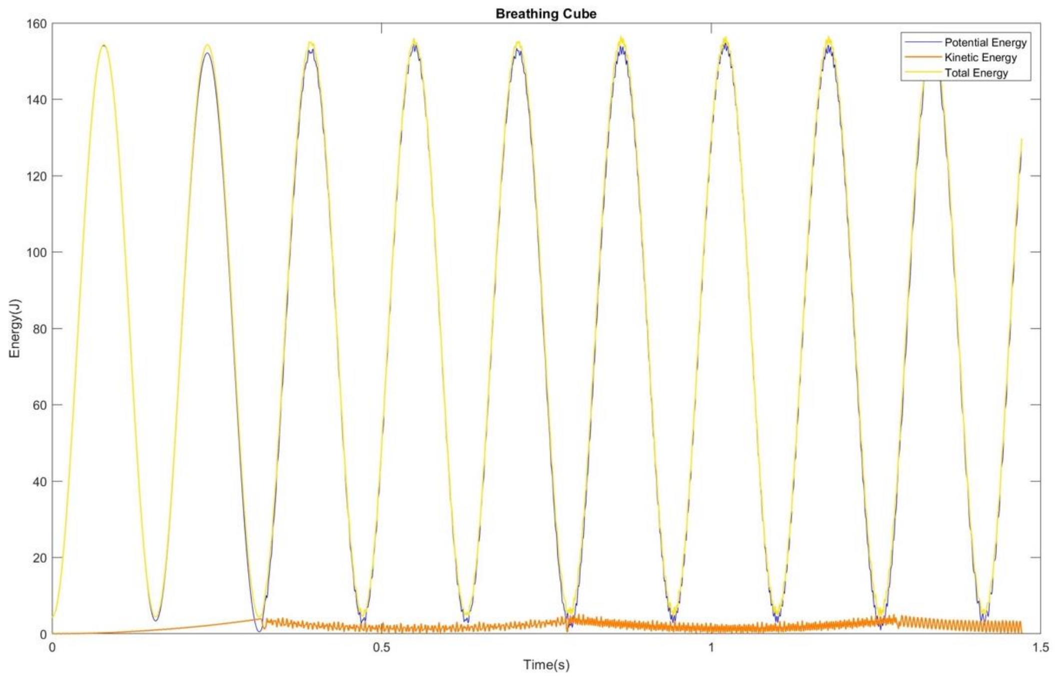
## Performance plots

### Bouncing cube energy



4228 springs per second

### Breathing cube energy



3276 springs per second

# Code

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <stdarg.h>
#include <numeric>
#include <cstdlib>
#include <ctime>
#include <cstdio>
#include <fstream>
#include <GLUT/glut.h>
using namespace std;

int cube_numbers=1;
int mass_number=8;
int spring_number=(mass_number-1)*mass_number/2;
double mass = 0.1;
double length = 0.1;
double gravity = 9.81;
double T = 0;
double timestep = 0.001;
double k_c = 100000;
double k = 10000;
double damping_constant = 0.9999957;

double asp = 1;
int fov = 45;
double dim = 1.0;
int moveX,moveY;
int spinX = 0;
int spinY = 0;
int des=0;
GLfloat world_rotation[16] = {1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 };
clock_t start = clock();

ofstream myFile("energy.txt");

struct Mass
{
    double m;
    double p[3];
    double v[3];
    double a[3];
};

struct Spring
{
    double k;
    double l0;
    int m1;
    int m2;
};
struct Cube{
    vector<Mass> Masses;
    vector<Spring> Springs;
};

vector<Mass> generate_Masses(double mass, double length, double x, double y, double z)
{
    vector<Mass> Masses(mass_number);
    Masses[0] = { mass,{x + length / 2,y + length / 2,z},{0,0,0},{0,0,0} };
    Masses[1] = { mass,{x + length / 2,y - length / 2,z},{0,0,0},{0,0,0} };
    Masses[2] = { mass,{x - length / 2,y - length / 2,z},{0,0,0},{0,0,0} };
    Masses[3] = { mass,{x - length / 2,y + length / 2,z},{0,0,0},{0,0,0} };
    Masses[4] = { mass,{x + length / 2,y + length / 2,z + length},{0,0,0},{0,0,0} };
    Masses[5] = { mass,{x + length / 2,y - length / 2,z + length},{0,0,0},{0,0,0} };
    Masses[6] = { mass,{x - length / 2,y - length / 2,z + length},{0,0,0},{0,0,0} };
    Masses[7] = { mass,{x - length / 2,y + length / 2,z + length},{0,0,0},{0,0,0} };
    return Masses;
}

//tetrahedron
/*
vector<Mass> generate_Masses(double mass, double length, double x, double y, double z)
{
    vector<Mass> Masses(mass_number);
    Masses[0] = { mass,{x + length / 2+0.1,y,z},{0,0,0},{0,0,0} };
    Masses[1] = { mass,{x - length / 2-0.1,y,z},{0,0,0},{0,0,0} };
    Masses[2] = { mass,{x,y + length+0.1 ,z},{0,0,0},{0,0,0} };
    Masses[3] = { mass,{x - length / 2-0.1,y ,z+length+0.1},{0,0,0},{0,0,0} };
    return Masses;
}
 */
double distance(Mass a, Mass b){
    return sqrt(pow(a.p[0]-b.p[0],2)+pow(a.p[1]-b.p[1],2)+pow(a.p[2]-b.p[2],2));
```

```
}
vector<Spring> generate_Springs(double springConstant, vector<Mass>& Masses)
{
    vector<Spring> Springs(spring_number);
    int count=0;
    for (int i=0;i<mass_number;i++){
        for(int j=i+1;j<mass_number;j++){
            Springs[count] = { springConstant,distance(Masses[i],Masses[j]), i,j };
            count++;
        }
    }
    return Springs;
}
vector<Cube> generate_cubes()
{
    vector<Cube>cubes(cube_numbers);
    int i=0;
    while (i<cube_numbers){
        cubes[i].Masses= generate_Masses(mass, length, 0+0.5*sin(5*i), 0-0.5*sin(5*i),
0.5+0.4*(sin(i)));
        cubes[i].Springs= generate_Springs(k, cubes[i].Masses);
        i++;
    }
    return cubes;
}
void draw_cube(vector<Mass>& Masses, vector<Spring>& Springs)
{
    GLfloat color[6][3]={{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},
                     {1.0,1.0,0.0},{1.0,0.0,1.0},{0.0,1.0,1.0}};
    GLUquadric *quad;
    quad = gluNewQuadric();
    for (int i = 0; i < (int)Masses.size(); i++)
    {
        glPushMatrix();
        glMultMatrixf(world_rotation);
        glColor3f(1, 0, 1);
        glTranslated(Masses[i].p[0],Masses[i].p[1],Masses[i].p[2]);
        gluSphere(quad,0.005,20,10);
        glPopMatrix();
    }
    for (int i = 0; i < (int)Springs.size(); i++)
    {
        glPushMatrix();
        glMultMatrixf(world_rotation);
        glLineWidth(2.0f);
        glBegin(GL_LINES);
        glColor3d(1,1,1);

glVertex3f(Masses[Springs[i].m1].p[0],Masses[Springs[i].m1].p[1],Masses[Springs[i].m1]
.p[2]);

glVertex3f(Masses[Springs[i].m2].p[0],Masses[Springs[i].m2].p[1],Masses[Springs[i].m2]
.p[2]);
        glEnd();
        glPopMatrix();
    }
}
vector<Cube> cubes=generate_cubes();
void Update_cube(vector<Mass>& Masses, vector<Spring>& Springs)
{
    double kinetic_energy = 0;
    double spring_potential_energy = 0;
    double gravity_potential_energy = 0;
    double ground_react_energy = 0;
    double total_potential_energy = 0;
    double total_energy = 0;
    bool breath= false;
    if(breath){
        Springs[0].l0=length+0.05*sin(100*T);
        //Springs[5].l0=sqrt(3)*length+0.1*sin(20*T);
        //Springs[12].l0=sqrt(3)*length+0.1*sin(20*T);
        //Springs[14].l0=sqrt(3)*length+0.1*sin(20*T);
        //Springs[19].l0=sqrt(3)*length+0.1*sin(20*T);
    }
    vector<vector<double>> forces(mass_number, vector<double>(3));
    for (int i = 0; i < spring_number; i++)
    {
        Mass mass1 = Masses[Springs[i].m1];
        Mass mass2 = Masses[Springs[i].m2];
        double position_distance[3] = { mass2.p[0] - mass1.p[0],mass2.p[1] -
mass1.p[1],mass2.p[2] - mass1.p[2] };
        double l_now = sqrt(pow(position_distance[0], 2) + pow(position_distance[1], 2)
+ pow(position_distance[2], 2));
        double this_force = Springs[i].k * fabs(Springs[i].l0 - l_now);
        int flag=1;
        if (l_now > Springs[i].l0){
            flag=-1;
        }
        forces[Springs[i].m1][0] = forces[Springs[i].m1][0] -
flag*this_force*position_distance[0] / l_now ;
        forces[Springs[i].m1][1] = forces[Springs[i].m1][1] -
flag*this_force*position_distance[1] / l_now ;
        forces[Springs[i].m1][2] = forces[Springs[i].m1][2] -
flag*this_force*position_distance[2] / l_now ;
        forces[Springs[i].m2][0] = forces[Springs[i].m2][0] +
flag*this_force*position_distance[0] / l_now ;
        forces[Springs[i].m2][1] = forces[Springs[i].m2][1] +
flag*this_force*position_distance[1] / l_now ;
        forces[Springs[i].m2][2] = forces[Springs[i].m2][2] +
flag*this_force*position_distance[2] / l_now ;
        spring_potential_energy += Springs[i].k * pow((l_now - Springs[i].l0), 2) / 2;
    }
    for (int i = 0; i < mass_number; i++)
    {
        forces[i][2] = forces[i][2] - Masses[i].m * gravity;
        if (Masses[i].p[2] <= 0)
        {
            forces[i][2] = forces[i][2] + k_c * fabs(Masses[i].p[2]);
            ground_react_energy = ground_react_energy + k_c * pow(Masses[i].p[2], 2) /
2;
            double F_H = sqrt(pow(forces[i][0], 2) + pow(forces[i][1], 2));
            double F_V = forces[i][2];


        }
        for (int j=0;j<3;j++){
            Masses[i].a[j] = forces[i][j] / Masses[i].m;
            Masses[i].v[j] = damping_constant * (Masses[i].v[j] + Masses[i].a[j] *
timestep);
            Masses[i].p[j] = Masses[i].p[j] + Masses[i].v[j] * timestep;
        }
        gravity_potential_energy += Masses[i].m * gravity * Masses[i].p[2];
        double
velocity=sqrt(pow(Masses[i].v[0],2)+pow(Masses[i].v[1],2)+pow(Masses[i].v[2],2));
        kinetic_energy += Masses[i].m * pow(velocity, 2) / 2;
    }
    total_potential_energy = spring_potential_energy + gravity_potential_energy +
ground_react_energy;
    total_energy = total_potential_energy + kinetic_energy;
    myFile << T << " " << total_potential_energy << " " << kinetic_energy << " " <<
total_energy << endl;
    cout << "total_potential_energy:" << total_potential_energy << "  " << "kenetic:"
```

```
<< kinetic_energy << endl;
    cout << "total_energy:" << total_energy << endl;
    draw_cube(Masses, Springs);
    T = T + timestep;

}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glLoadIdentity();
    gluLookAt(-1.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glPushMatrix();
    glRotated(spinX, 0, 1, 0);
    glRotated(spinY, 0, 0, 1);
    glTranslated(0, 0, des);
    for(int i=0;i<cube_numbers;i++){
        Update_cube(cubes[i].Masses, cubes[i].Springs);
    }
    glPopMatrix();
    glutSwapBuffers();
}
void Project(double fov, double asp, double dim)
{

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (fov)
        gluPerspective(fov, asp, dim /1, 16 * dim);
    else
        glOrtho(-asp * dim, asp * dim, -dim, +dim, -dim, +dim);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void mouseMove(int x, int y) {
    int dx = x - moveX;
    int dy = y - moveY;
    printf("dx;%dx,dy:%dy\n",dx,dy);
    spinX += dx;
    spinY += dy;
    glutPostRedisplay();
    moveX = x;
    moveY = y;
```

```
}
void key_pressed(unsigned char ch, int x, int y)
{
   if (ch == 27)
      exit(0);
   else if (ch=='a')
      spinX +=5;
   else if (ch=='d')
      spinX -=5;
   else if (ch=='w')
      spinY +=5;
   else if (ch=='s')
      spinY -=5;
   Project(fov, asp, dim);
   glutPostRedisplay();
}
void reshape(int width, int height)
{
   asp =  (double)width / height;
   glViewport(0, 0, width, height);
   Project(fov, asp, dim);
}
void idle()
{
   glutPostRedisplay();
}

int main(int argc, char* argv[])
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
   glutInitWindowPosition(100, 100);
   glutInitWindowSize(1400, 1400);
   glutCreateWindow("cubes");
   glutIdleFunc(idle);
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMotionFunc(mouseMove);
   glutKeyboardFunc(key_pressed);
   glutMainLoop();
   return 0;
};
```