물리기반 모델링

당구게임

동명대학교
강영민

Particle.h - 당구공

```
enum DrawMode {
    POINT_DRAW,
    SPHERE_DRAW
};

class CParticle {
public:
   int type;
   double radius;
   double mass;

   CVec3d loc, vel, force, gravity;
   CVec3d color;

private:
    void forceIntegration(double dt, double et);

public:
   CParticle();
```

```
   void setPosition(double x, double y, double z);
   void setVelocity(double vx, double vy, double vz);
   void setMass(double m);
   void setRadius(double r);
   void setColor(double r, double g, double b);

   CVec3d getPosition();
   CVec3d getVelocity();
   double getMass();
   double getRadius();


   void resetForce(void);
   void addForce(CVec3d &f);

   void drawWithGL(int drawMode = SPHERE_DRAW);
   void simulate(double dt, double eT);
};
```

Particle.cpp - 당구공

```
CParticle::CParticle() {
    radius = 1.0f;
    loc.set(0.0, 0.0, 0.0);
}
void CParticle::setPosition(double x, double y, double z) {
    loc.set(x,y,z);
}
void CParticle::setVelocity(double vx, double vy, double vz) {
   vel.set(vx,vy,vz);
}
void CParticle::setMass  (double m) { mass = m; }
void CParticle::setRadius(double r) { radius = r; }
void CParticle::setColor (double r, double g, double b) { color.set(r,g,b); }

CVec3d CParticle::getPosition() { return loc ; }
CVec3d CParticle::getVelocity() { return vel ; }
double CParticle::getMass()     { return mass; }
double CParticle::getRadius()   { return radius; }
```

Particle.cpp - 당구공

```
void CParticle::drawWithGL(int drawMode) {
    glColor3f(color.x, color.y, color.z);

    glPushMatrix();
    glTranslated(loc[0], loc[1], loc[2]);
    if (drawMode == SPHERE_DRAW) {
        glutWireSphere(radius, 30, 30);
    }
    else {
        glBegin(GL_POINTS);
        glVertex3f(0,0,0);
        glEnd();
    }
    glPopMatrix();
}
void CParticle::forceIntegration(double dt, double et) {
    if(dt>0.1) dt=0.1;
    vel = vel + dt*((1.0/mass) * force );
    loc = loc + dt*vel;
}
void CParticle::simulate(double dt, double et) {
    forceIntegration(dt, et);
    if(this->vel.len()<10) vel.set(0.0,0.0,0.0);
}
void CParticle::resetForce(void) {  this->force.set(0.0, 0.0, 0.0); }
void CParticle::addForce(CVec3d &f) { this->force = this->force + f; }
```

main.cpp - Game Control

```
void key_ready(unsigned char key) {
    switch (key) {
    case 's': // start game
            Simulator->start(); myWatch.start();
            ((CDynamicSimulator *)Simulator)->setMode(AIMING);
            break;
    }
}
void key_aiming(unsigned char key) {
    switch (key) {
        case '.': ((CDynamicSimulator *)Simulator)->rotateAim( 0.05);break;
        case ',': ((CDynamicSimulator *)Simulator)->rotateAim(-0.05);break;
        case 'm': ((CDynamicSimulator *)Simulator)->rotateAim(-0.01);break;
        case '/': ((CDynamicSimulator *)Simulator)->rotateAim( 0.01);break;
        case ' ': ((CDynamicSimulator *)Simulator)->shot();break;
    }
}
void key_simulating(unsigned char key) {
    switch (key) {
        case 'p': myWatch.pause(); Simulator->pause(); break;
        case 'r': myWatch.resume(); break;
        case ' ': ((CDynamicSimulator *)Simulator)->turnOver();break;
        default: break;
    }
}

void KEY_turnover(unsigned char key) {
    switch (key) {
        case ' ':((CDynamicSimulator *)Simulator)->setMode(AIMING); break;
    }
}
void keyboardFunction(unsigned char key, int x, int y) {
    if (key == 27) exit(0);
    gameMode mode = ((CDynamicSimulator *)Simulator)->getMode();
    switch(mode) {
    case READY: key_ready(key); break;
    case AIMING: key_aiming(key); break;
    case SIMULATING: key_simulating(key); break;
    case TURNOVER: KEY_turnover(key); break;
    default: break;
    }
}
```

main.cpp - Display/Idle function

```
void displayFunction(void) {


   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   setupCamera(0, 2500, 0, 0, 0, 0, 1, 0, 0);

   // check DT (in microsecond) from StopWatch and store it to "deltaTime" (in seconds)
   deltaTime = myWatch.checkAndComputeDT() / 1000000.0;
   currentTime = myWatch.getTotalElapsedTime() / 1000000.0;

   Simulator->actions(deltaTime, currentTime);
    // actions < doBeforeSimulation, doSimulation, doAfterSimulation >

   glutSwapBuffers();
}
```

DynamicSimulator.h

```
#include "Simulator.h"
#include "Particle.h"

#define NUMBALLS 4
#define TABLE_W 1420
#define TABLE_H 2840
#define BALL_RADIUS 40.75
```

```
typedef enum MODE {
    READY,
    AIMING,
    SIMULATING,
    TURNOVER
} gameMode;

enum TURNS {
    PLAYER1,
    PLAYER2,
    NUMPLAYERS
};
```

Definitions and enumerations

DynamicSimulator.h - 클래스

```
class CDynamicSimulator : public CSimulator {
    CParticle balls[NUMBALLS];
    TURNS turn;
    MODE mode;
    CVec3d aim;
    float aimAngle;
public:
    CDynamicSimulator();
    void init(void);
    void clean(void);
    MODE getMode(void);
    void setMode(MODE m);
    void rotateAim(double angle);
    void shot(void);
    void turnOver(void);
private:
    void doBeforeSimulation(double dt, double currentTime);
    void doSimulation(double dt, double currentTime);
    void doAfterSimulation(double dt, double currentTime);
    void visualize(void);

    CVec3d computeAttraction(int i, int j);
   void collisionHandler(int i, int j);
    void floorDrag(void);
    void cushion(void);
};
```

DynamicSimulator.cpp

```
void CDynamicSimulator::init() {

    turn = PLAYER1;
    mode = READY;

    for(int i=0;i<NUMBALLS;i++) {
        balls[i].setRadius(BALL_RADIUS);
        balls[i].setMass(0.16);
        balls[i].setVelocity(0.0, 0.0, 0.0);
    }
    balls[0].setPosition( TABLE_W/20.0, BALL_RADIUS, 3.0*TABLE_H/8.0);
    balls[0].setColor(1.0, 1.0, 1.0);
    balls[1].setPosition(-TABLE_W/20.0, BALL_RADIUS, 3.0*TABLE_H/8.0);
    balls[1].setColor(1.0, 0.0, 0.0);
    balls[2].setPosition( 0, BALL_RADIUS,-3.0*TABLE_H/8.0);
    balls[2].setColor(1.0, 1.0, 0.0);
    balls[3].setPosition( 0, BALL_RADIUS,-2.0*TABLE_H/8.0);
    balls[3].setColor(1.0, 0.0, 0.0);

    aim.set(1.0, 0.0, 0.0);
}
```

DynamicSimulator.cpp

```
void CDynamicSimulator::doSimulation(double dt, double currentTime) {

    if(dt>0.01)dt=0.01; // maximum dt

    if(mode!=SIMULATING) return;

    floorDrag();

    for (int i=0; i<NUMBALLS; i++) {
        balls[i].simulate(dt, currentTime);
    }

    cushion();


    for (int i=0; i<NUMBALLS; i++) {
        for (int j=i+1; j<NUMBALLS; j++) {
            collisionHandler(i, j);
        }
    }
}

void CDynamicSimulator::doAfterSimulation(double dt, double currentTime) {
    for(int i=0;i<NUMBALLS;i++) {
    balls[i].resetForce();
   }
}
```

DynamicSimulator.cpp

```
void CDynamicSimulator::visualize(void) {
    // Draw Table
    glColor3f(0.0, 0.5, 0.0);
    glBegin(GL_QUADS);
    glVertex3f(-TABLE_W/2.0, 0.0,-TABLE_H/2.0);
    glVertex3f(-TABLE_W/2.0, 0.0, TABLE_H/2.0);
    glVertex3f( TABLE_W/2.0, 0.0, TABLE_H/2.0);
    glVertex3f( TABLE_W/2.0, 0.0,-TABLE_H/2.0);
    glEnd();

    for(int i=0;i<NUMBALLS;i++) {
        balls[i].drawWithGL(SPHERE_DRAW);
    }

    if (mode == AIMING) {
        CVec3d pos; pos = balls[turn*2].getPosition();
        glBegin(GL_LINES);
        glVertex3f(pos.x, pos.y, pos.z);
        glVertex3f(pos.x+aim.x*2000.0, pos.y+aim.y*2000.0, pos.z+aim.z*2000.0);
        glEnd();
    }
}
```

DynamicSimulator.cpp

```
void CDynamicSimulator::collisionHandler(int i, int j) {
   // collision detect
    CVec3d p1; p1 = balls[i].getPosition();
    CVec3d p2; p2 = balls[j].getPosition();
    CVec3d N ; N = p1 - p2;
   double dist = N.len();
    double e = 0.9;
   if(dist < balls[i].getRadius() + balls[j].getRadius()) {
     double penetration = balls[i].getRadius() + balls[j].getRadius() - dist;
    // collision detected
    N.normalize();
    CVec3d v1; v1 = balls[i].getVelocity();
    CVec3d v2; v2 = balls[j].getVelocity();
    double v1N = v1 ^ N; // velocity along the line of action
    double v2N = v2 ^ N; // velocity along the line of action
    double m1 = balls[i].getMass();
    double m2 = balls[j].getMass();
    // approaching ?
    if( v1N-v2N < 0 ) { // approaching
        double vr = v1N - v2N;
        double J = -vr*(e+1.0)/(1.0/m1 + 1.0/m2);
        double v1New = v1N + J/m1;
        double v2New = v2N - J/m2;
        v1 = v1 - v1N * N + v1New*N;
        v2 = v2 - v2N * N + v2New*N;
        balls[i].setVelocity(v1.x, v1.y, v1.z);
        balls[j].setVelocity(v2.x, v2.y, v2.z);
    }
     p1 = p1 + 0.5*((1.0+e)*penetration)*N;
     p2 = p2 - 0.5*((1.0+e)*penetration)*N;
     balls[i].setPosition(p1.x, p1.y, p1.z);
     balls[j].setPosition(p2.x, p2.y, p2.z);
   }
}
```

DynamicSimulator.cpp

```
void CDynamicSimulator::floorDrag(void) {
    CVec3d vel, dragForce;
    double drag = 0.05;
    for(int i=0;i<NUMBALLS;i++) {
        vel = balls[i].getVelocity();
        dragForce = -drag*vel;
        balls[i].addForce(dragForce);
    }
}
MODE CDynamicSimulator::getMode(void)   { return mode; }
void CDynamicSimulator::setMode(MODE m) { mode = m; }

void CDynamicSimulator::rotateAim(double angle) {
    aimAngle+=angle;
    if(aimAngle>3.141592*2.0) aimAngle-=3.141592*2.0;
    aim.set(cos(aimAngle), 0.0, sin(aimAngle));
}
void CDynamicSimulator::shot(void) {
    balls[turn*2].setVelocity(5000*aim.x, 0.0, 5000*aim.z);
    mode = SIMULATING;
}
void CDynamicSimulator::turnOver(void) {
    for(int i=0;i<NUMBALLS;i++) balls[i].setVelocity(0.0, 0.0, 0.0);
    turn = turn==PLAYER1?PLAYER2:PLAYER1;
    mode = TURNOVER;
}
```

DynamicSimulator.cpp

```
void CDynamicSimulator::cushion(void) {
    // collision detect
    for(int i=0;i<NUMBALLS; i++) {
        CVec3d pos; pos = balls[i].getPosition();
        CVec3d vel; vel = balls[i].getVelocity();
        CVec3d N;
        double r = balls[i].getRadius();
        double pene = 0.0;
        if(pos.x + r > TABLE_W/2.0) {
            pene = pos.x + r - TABLE_W/2.0;
            N.set(-1.0, 0, 0);
        }
        else if(pos.x - r < -TABLE_W/2.0) {
            pene = -TABLE_W/2.0 - pos.x + r;
            N.set(1.0,0.0,0.0);
        }
        else if(pos.z + r > TABLE_H/2.0) {
            pene = pos.z + r - TABLE_H/2.0;
            N.set(0.0, 0.0, -1.0);
        }
        else if(pos.z - r < -TABLE_H/2.0) {
            pene = -TABLE_H/2.0 - pos.z + r;
            N.set(0.0, 0.0, 1.0);
        }
        double vN = vel^N;
        if (vN<0.0) { // penetrating
            vel = vel - (2.0 * vN)*N;
        }
        pos = pos + (2.0*pene)*N;
        balls[i].setVelocity(vel.x, vel.y, vel.z);
        balls[i].setPosition(pos.x, pos.y, pos.z);
    }
}
```

Result