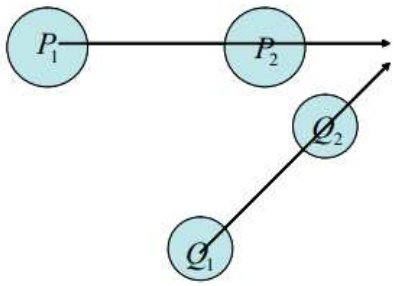
6.7 구-구 교차

직선 운동을 하는 두 개의 공들의 충돌시점과 충돌지점을 찾고 그에 상응하는 충돌반응을 계산하도록 한다. 첫 번째 공 P는 r_1을 반지름으로 하고, 시간이 $t=0$일 때 구의 중심이 P_1에 있고 시간이 $t=1$일 때 구의 중심이 P_2이 되도록 하는 운동을 하고 있다. 두 번째 공 Q는 r_2을 반지름으로 하고, 시간이 $t=0$일 때 구의 중심이 Q_1에 있고 시간이 $t=1$일 때 구의 중심이 Q_2이 되도록 하는 운동을 하고 있다. 두 공이 충돌한다는 것은 두 공의 중심 사이의 거리가 반지름의 합과 일치할 때임을 알 수 있다. 이를 이용하여 충돌시점을 먼저 구하도록 한다.



첫 번째 구와 두 번째 구의 중심의 이동방정식은 다음과 같다.

$$P(t) = P_1 + t(P_2 - P_1)$$
$$Q(t) = Q_1 + t(Q_2 - Q_1)$$

두 중심 사이의 거리는 다음의 조건을 만족하여야 한다.

$$d(P(t), Q(t)) = \| P(t) - Q(t) \| = r_1 + r_2$$

계산의 편의를 위하여 거리의 제곱을 사용한다.

$$\| P(t) - Q(t) \|^2 = \| P_1 + t V_P - Q_1 - t V_Q \|^2$$

이때

$$V_P = P_2 - P_1, \;\; V_Q = Q_2 - Q_1$$

그러면

$$\| P(t)-Q(t) \|^2 = \| P_1-Q_1+t(V_P-V_Q) \|^2$$

$$= \| A+tB \|^2$$

따라서

$$d^2 = \| A \|^2+2(A \cdot B)t+ \| B \|^2 t^2$$

충돌하는 조건은 다음과 같다.

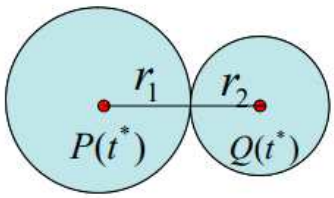
$$\| A \|^2+2(A \cdot B)t+ \| B \|^2 t^2 = (r_1+r_2)^2$$

근의 공식에 의해서 두 개의 시점을 구할 수 있는데 그 중에서 먼저 발생하는 것을 의미한다.

$$t^* = \frac{-(A \cdot B)-\sqrt{(A \cdot B)^2- \| B \|^2(\| A \|^2-(r_1+r_2)^2)}}{\| B \|^2}$$

이제는 충돌하는 지점을 찾도록 하자. 충돌시점에서의 두 개의 구의 중심은 다음과 같다.

$$P(t^*) = P_1+t^* V_P, \quad Q(t^*) = Q_1+t^* V_Q$$



충돌지점은 이 두 점의 $r_1 : r_2$로 내분하는 지점임을 알 수 있다. 그러므로 최종적으로 찾으려고 하는 충돌지점은 다음과 같다.

$$I = \frac{r_2}{r_1+r_2}P(t^*)+\frac{r_1}{r_1+r_2}Q(t^*)$$

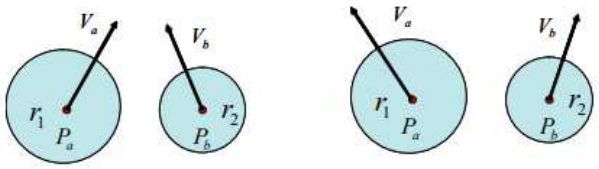
6.8 구-구 충돌반응

움직이는 두 개의 공이 향후 충돌할 지 아니면 충돌하지 않을 지를 빠르게 판단하는 방법을 생각해 보자.

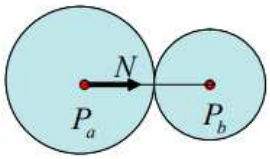
먼저 두 중심 사이의 관계를 나타내는 벡터와 두 공의 움직이는 상태 속도를 정의하자.

$$n = P_b - P_a,\ V_{ab} = V_a - V_b$$

만약 $V_{ab} \cdot n < 0$ 이면 두 개의 공은 서로 멀어지는 경향이 있음을 알 수 있다. 그러므로 충돌이 발생하지 않게 되는 것이다. 그림에서는 오른쪽의 경우인 셈이다.



그렇지 않다면 6.7절에서 설명된 바와 같이 충돌지점을 찾을 수 있다.



이때 단위벡터 N을 다음과 같이 정의한다.

$$N = \frac{n}{\| n \|}$$

충돌하는 두 물체의 고유의 성질인 반발계수(ϵ)를 반영하여 움직이는 두 물체의 충돌 전/후의 상대속도는 다음과 같은 관계식을 만족하여야 한다.

$$(V_a' - V_b') \cdot N = -\epsilon (V_a - V_b) \cdot N \qquad (1)$$

반발계수가 1이면 완전탄성충돌이며 반발계수가 0이면 찰흙과 같이 비탄성충돌을 의미한다.

충돌하는 두 물체 사이에는 전체 운동량이 보존되어야 하는 것이 원칙이다. 운동량은 질량과 속도의 곱으로 표현되는데 다음의 수식을 따라야 한다는 것이다.

$$m_a V_a + m_b V_b = m_a V_a^{'} + m_b V_b^{'}$$

이러한 관계식은 다음과 같이 분리하여 정리할 수 있다.

$$m_a V_a + \alpha N = m_a V_a^{'}$$
$$m_b V_b - \alpha N = m_b V_b^{'}$$

위 식을 정리하면 다음과 같다.

$$V_a^{'} = V_a + \frac{\alpha}{m_a} N$$

$$V_b^{'} = V_b - \frac{\alpha}{m_b} N$$

위 식을 수식 (1)에 대입하여 α를 구할 수 있다.

$$\alpha = -\frac{(1+\epsilon)(V_a - V_b) \cdot N}{(\frac{1}{m_a} + \frac{1}{m_b})}$$

그러므로 충돌 후의 속도는 다음과 같다.

$$V_a^{'} = V_a - \frac{(1+\epsilon) m_b (V_a - V_b) \cdot N}{m_a + m_b} N$$

$$V_b^{'} = V_b + \frac{(1+\epsilon) m_a (V_a - V_b) \cdot N}{m_a + m_b} N$$

6.9 Ball Collision & Response Program

```
#include <windows.h>
#include <iostream.h>
#include <math.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h> // (or others, depending on the system in use)

#define     PI      3.1415926
#define     epsilon  1.0

float       radius = 5.0;
int         width = 400;
int         height = 400;
float       point_size = 3.0;
int         n = 0;


int         num = 36;
int         c_num = 30;

float       delta;
GLenum      draw_type;
int         collide = 0;

struct Point{
    float   x;
    float   y;
};

struct Vector{
    float   x;
    float   y;
};

struct Color{
    float   r;
    float   g;
    float   b;
};

struct Circle{
    Point   center;
    float   radius;
    float   mass;
    Color   color;
    Vector  velocity;
};

Circle      *circle;
```

```
Point           *vertex;

float           delta_x, delta_y;
float           fix_radius;

Point           Window_Center;
float           Window_radius;

void Init (void) {
    glClearColor (1.0, 0.3, 0.3, 0.0) ; // Set display-window color to white.
    glMatrixMode (GL_PROJECTION); // Set projection parameters.
    gluOrtho2D (0, width, 0, height);
    draw_type = GL_LINES;

    vertex = new Point[num];
    delta = 2.0 * PI / num;
    for ( int i = 0; i < num; i++ ) {
        vertex[i].x = cos(delta * i);
        vertex[i].y = sin(delta * i);
    }

    Window_radius = width/3.0;
    Window_Center.x = width/2.0;
    Window_Center.y = height/2.0;

    delta = 2.0 * PI / c_num;
    circle = new Circle[c_num];
    for ( i = 0; i < c_num; i++ ) {
        circle[i].radius = radius * (1.5 + sin(i));
        circle[i].center.x = Window_Center.x + Window_radius * cos(delta * i);
        circle[i].center.y = Window_Center.y + Window_radius * sin(delta * i);
        circle[i].velocity.x = -1.0*(circle[i].center.x - Window_Center.x) / Window_radius;
        circle[i].velocity.y = -1.0*(circle[i].center.y - Window_Center.y) / Window_radius;
        circle[i].mass = 1.5 + sin(i);
        circle[i].color.r = (float)(1.0 -  (i%3)/2);
        circle[i].color.g = (float)(1.0 - (c_num - i)/c_num);
        circle[i].color.b = (float)(1.0 * (i%10)/9);
    }

}

void Draw_Circle(int    index) {
    float     x, y, radius;

    x = circle[index].center.x;
    y = circle[index].center.y;
    radius = circle[index].radius;
    glColor3f(circle[index].color.r, circle[index].color.g, circle[index].color.b);
    glBegin(GL_POLYGON);
    for ( int i = 0; i < num; i++ )
```

```
        glVertex3f( x + radius * vertex[i].x, y + radius * vertex[i].y, 0.0);

    glEnd();
}

void MyReshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D (0, width, 0, height);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void  Check_Collision_Ball_Wall(int index) {
 if (circle[index].center.x+circle[index].radius>width||circle[index].center.x-circle[index].radius<0)
    circle[index].velocity.x *= (-1.0);
 if (circle[index].center.y+circle[index].radius>height||circle[index].center.y-circle[index].radius<0)
    circle[index].velocity.y *= (-1.0);
}


void  Check_Collision_Ball_Ball(int i, int j) {
    float    distance;
    Vector   VA, VB, R_AB, normal ;
    float    inner_value;

    normal.x = circle[j].center.x - circle[i].center.x;
    normal.y = circle[j].center.y - circle[i].center.y;

    distance = sqrt( normal.x * normal.x + normal.y * normal.y );

    if ( distance < circle[j].radius + circle[i].radius ) {
            distance = sqrt( normal.x * normal.x + normal.y * normal.y );
            normal.x /= distance;
            normal.y /= distance;

            VA.x = circle[i].velocity.x;
            VA.y = circle[i].velocity.y;

            VB.x = circle[j].velocity.x;
            VB.y = circle[j].velocity.y;

            R_AB.x = VA.x - VB.x;
            R_AB.y = VA.y - VB.y;

            inner_value = R_AB.x * normal.x + R_AB.y * normal.y;
            if ( inner_value > 0.0 ) {
                circle[i].velocity.x=VA.x-(1+epsilon)*circle[j].mass*
                                inner_value*normal.x/(circle[i].mass+circle[j].mass);
```

```
                    circle[i].velocity.y=VA.y-(1+epsilon)*circle[j].mass*
                                        inner_value*normal.y/(circle[i].mass+circle[j].mass);

                    circle[j].velocity.x=VB.x+(1+epsilon)*circle[i].mass*
                                        inner_value*normal.x/(circle[i].mass+circle[j].mass);
                    circle[j].velocity.y=VB.y+(1+epsilon)*circle[i].mass*
                                        inner_value*normal.y/(circle[i].mass+circle[j].mass);

                    collide = 1;
                }
        }

}

void  Update_Position(void) {
        for ( int i = 0; i < c_num; i++ ) {
                circle[i].center.x += circle[i].velocity.x;
                circle[i].center.y += circle[i].velocity.y;
        }
}

void RenderScene (void) {
        int                 i;

        glClear (GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 0.0, 0.0);

        for ( i = 0; i < c_num; i++ )
            Check_Collision_Ball_Wall(i);
        Update_Position();

        for ( i = 0; i < c_num; i++ ) {
                for ( int j = i+1; j < c_num; j++ ) {
                        if ( i != j )
                            Check_Collision_Ball_Ball(i, j);
                }
        }
        Update_Position();

        for ( i = 0; i < c_num; i++ )
                Draw_Circle(i);

        glFlush ();
        glutSwapBuffers();
}

void SpecialKey( int key, int x, int y) {
        int                 i;

        switch (key) {
```

```
        case GLUT_KEY_LEFT :
                for ( i = 0; i < c_num; i++ ) {
                        circle[i].velocity.x -= 0.1;
                        circle[i].velocity.y -= 0.1;
                }
                break;
        case GLUT_KEY_RIGHT :
                for ( i = 0; i < c_num; i++ ) {
                        circle[i].velocity.x += 1.5;
                        circle[i].velocity.y += 1.5;
                }
                break;
        default :                               break;
        }
}

void IdleFunction (void) {
        RenderScene();
        glutPostRedisplay();
}

void main (int argc, char** argv) {
        glutInit (&argc, argv);
        glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(width, height);
        glutCreateWindow ("Ball_Ball_Collision_Response");
        Init();
        glutDisplayFunc (RenderScene);
        glutReshapeFunc(MyReshape);
        glutSpecialFunc(SpecialKey);
        glutIdleFunc(IdleFunction);
        glutMainLoop();
}
```