Новосибирский государственный технический университет

Кафедра вычислительной техники

Пояснительная записка к Расчётно-графической работе.

«Использование графической библиотеки “Graphics.h”»

Группа: АВТ-909
Студент: Платонов А.В.

Новосибирск
2009.

1. Постановка задачи

Задание: визуализировать движение частиц, порождаемых другими частицами. Изначально на экране есть одна «частица» - круг, постепенно увеличиваясь в размерах, он «лопается» и на его месте появляется несколько новых, которые тоже в свою очередь способны породить новые круги. Новые круги, разлетаясь от центра взрыва сталкиваются с себе подобными или со стенами и отталкиваются от них, не противореча законам физики.

2.Основная идея

Для решения поставленной задачи необходимо обдумать несколько проблем:

- Движение массива частиц относительно экрана и их постепенное увеличение в размерах.
- «Взрыв» частицы с последующим появлением новых.
- Необходимо реализовать модель столкновения шарик-шарик и шарик-стена.

Геометрическая модель:

В процессе решения задачи задействовано 2 системы координат.

Первая система координат:

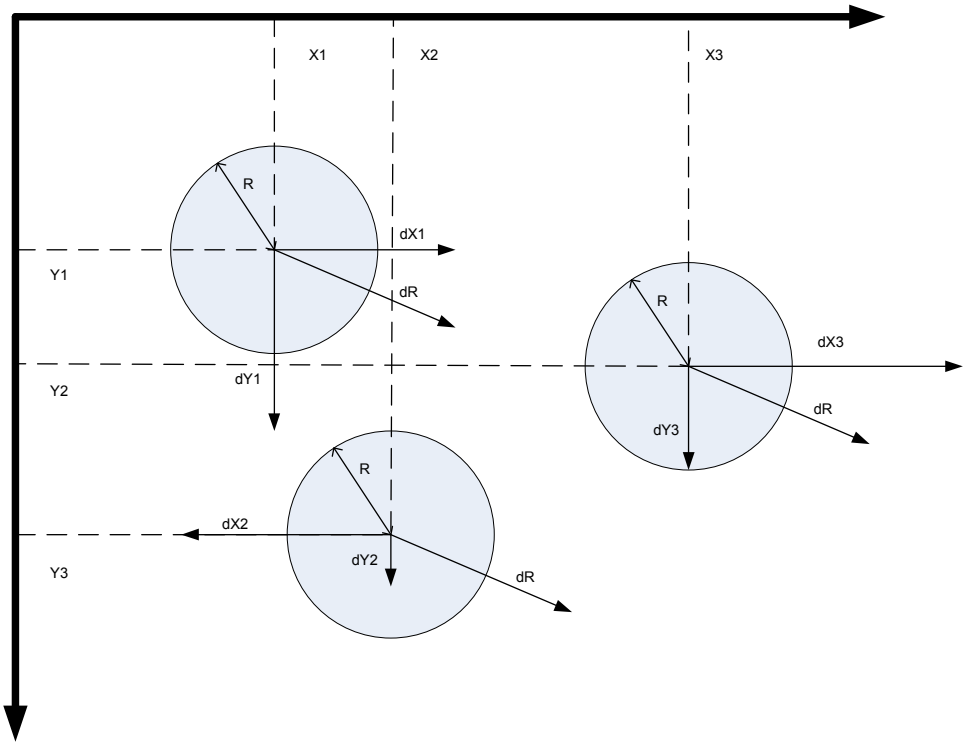

Рис. 1. Модель координатной оси

***Пояснение*:** в этой системе координат происходит движение тела и увеличение его размеров. (По определённым законам движения)

Вторая система координат:

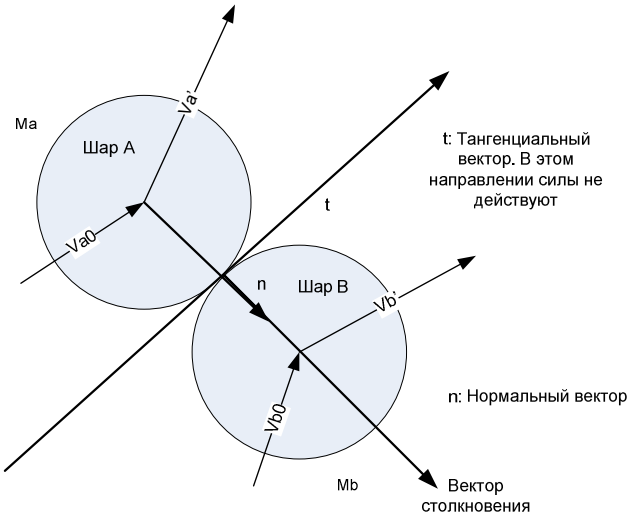

Рис. 2. Модель столкновения

Пояснение: в этой системе координат происходит расчет скоростей шаров после столкновения.

3.Основные методы и решения

В соответствии с приведённой выше совокупностью систем координат реализуется выполнение поставленной задачи:

1) ***Движение массива тел в системе координат относительно дисплея.*** Первая система координат является базовой, в ней происходит определение принципов движения: вычисление координат текущего положения объектов, вычисление направления движения, вычисление «физических» размеров тел.

2) ***«Взрыв» частицы с последующим появлением новых.*** Когда круг достигает определенных размеров, то происходит иллюзия взрыва – он резко уменьшается в размерах, случайным образом мгновенно перемещается в радиусе «взрыва» и принимает вектор скорости, с направлением от центра «взрыва». Таким же образом в радиусе «взрыва» появляется несколько новых кругов, взятых из неиспользуемой части массива, хранящего в себе данные о всех кругах.

3) ***Модель столкновения.***

   Шарик-стена: все просто, во время работы основного цикла, нужно перебирать координаты всех шариков на предмет вылезания за пределы экрана, если это происходит, то следует просто обратить скорость, перпендикулярную к стене.

Шарик-шарик:

Скорость шаров, после соударения претерпевает значительные изменения. Для упрощения модели следует принять систему не теряющую кинетическую энергию со временем. Что бы скорость шаров на момент соударения изменилась, силы действующие на момент соударения должны быть мгновенными. Силы этого типа есть силы импульсные. Импульсы тел можно вычислить исходя из закона сохранения импульса (1) и сохранения энергии (2):

$$m_a v_{a0} + m_b v_{b0} = m_a v_{a'} + m_b v_{b'}; \quad (1)$$

$$\frac{m_a v_{a0}^2}{2} + \frac{m_b v_{b0}^2}{2} = \frac{m_a v_{a'}^2}{2} + \frac{m_b v_{b'}^2}{2}; \quad (2)$$

Из этих формул можно вычислить скорости тел A (3) и B (4) после столкновения:

$$v_{a'} = \frac{2m_b v_{b0} + (m_a - m_b) v_{a0}}{m_a + m_b}; \quad (3)$$

$$v_{b'} = \frac{2m_a v_{a0} + (m_b - m_a) v_{b0}}{m_a + m_b}; \quad (4)$$

Однако эти формулы справедливы для одномерного столкновения. Главное, что можно рассчитать из них – это коэффициент восстановления ε (5), показывающий тип столкновения, от абсолютно упругого, ε = 1, до абсолютно неупругого, ε = 0:

$$\varepsilon = \frac{v_{a'} - v_{b'}}{v_{a0} - v_{b0}}; \quad (5)$$

Используя этот коэффициент и закон сохранения импульса можно получить следующие выражения скоростей после столкновения (6,7):

$$v_{a'} = \frac{(1+\varepsilon) m_b v_{b0} + (m_a - \varepsilon m_b) v_{a0}}{m_a + m_b}; \quad (6)$$

$$v_{b'} = \frac{(1+\varepsilon) m_a v_{a0} + (m_b - \varepsilon m_a) v_{b0}}{m_a + m_b}; \quad (7)$$

Если посмотреть на рис. 2. то можно обратить внимание на оси координат помеченные как **n** и **t.** Ось **n** (нормальная) идет в направлении линии столкновения, а ось **t** (тангенциальная) перпендикулярна ей. Первые уравнения (8, 9), которые можно написать связаны с тангенциальными составляющими скоростей до и после столкновения. Поскольку сил трения нет и импульсные силы вдоль тангенциальной оси не действуют, проекции импульсов (и скоростей) шаров вдоль тангенциальной оси остаются неизменными:

$$m_a v_{a0} t = m_a v_{a'} t; \quad (8)$$

$$m_b v_{b0} t = m_b v_{b'} t; \quad (9)$$

Поскольку массы шаров после столкновения не меняются, то из уравнении их можно исключить.

Т.к. сил трения нет, то можно записать закон сохранения импульса (10) вдоль нормальной оси и значение коэффициента восстановления (11):

$$m_a v_{a0} n + m_b v_{b0} n = m_a v_{a'} n + m_b v_{b'} n; \quad (10)$$

$$\varepsilon = \frac{v_{a'} n - v_{b'} n}{v_{a0} n - v_{b0} n}; \quad (11)$$

Комбинируя уравнения 10 и 11 можно получить нормальные составляющие скоростей (12, 13):

$$v_{a'} = \frac{(1+\varepsilon) m_b v_{b0} n + (m_a - \varepsilon m_b) v_{a0} n}{m_a + m_b}; \quad (12)$$

$$v_{b'} = \frac{(1+\varepsilon) m_a v_{a0} n + (m_b - \varepsilon m_a) v_{b0} n}{m_a + m_b}; \quad (13)$$

4) ***Система координат n-t:***

Для получения вектора **n**, достаточно заметить, что это вектор единичной длины, направленный вдоль линии, соединяющие центры шаров $A(x_{a0}; y_{a0})$ и $B(x_{b0}; y_{b0})$. От сюда найдем этот вектор **n:**

$$N = B - A = (x_{b0} - x_{a0}; y_{b0} - y_{a0}); \quad (14)$$

$$n = N / |N|;$$

Чтобы получить вектор **t**, нужно повернуть **n** на угол π/2 против часовой стрелки, т.е.:

$$t = (-n_y; n_x); \quad (15)$$

Теперь можно находить n и t компоненты вектора начальной скорости:

$$v_{a0} n = (x_{va0}; y_{va0})(n_x; n_y) = x_{va0} \cdot n_x + y_{va0} \cdot n_y; \quad (16)$$

$$v_{a0} t = (x_{va0}; y_{va0})(t_x; t_y) = x_{va0} \cdot t_x + y_{va0} \cdot t_y; \quad (17)$$

$$v_{b0} n = (x_{vb0}; y_{vb0})(n_x; n_y) = x_{vb0} \cdot n_x + y_{vb0} \cdot n_y; \quad (18)$$

$$v_{b0} t = (x_{vb0}; y_{vb0})(t_x; t_y) = x_{vb0} \cdot t_x + y_{vb0} \cdot t_y; \quad (20)$$

5) ***Теперь можно расписать решение задачи о столкновениях:***
   a. Вычислить **n** и **t** в соответствии с формулой (14);
   b. Найти все нормальные и тангенциальные компоненты начальных скоростей, используя формулы (16-20);
   c. Подставить полученные значения в уравнения (12-13);
   d. Преобразовать полученные результаты из системы координат **n-t** в систему координат **x-y**;

6) ***Преобразование полученных координат в систему x-y***

Можно записать скорости шаров после столкновения в координатах **n-t**:

$$v_{a'} = (v_{a'}n; v_{a'}t); \quad (21)$$

$$v_{b'} = (v_{b'}n; v_{b'}t); \quad (22)$$

Вектор скорости можно представить как сумму проекций на оси **n** и **t** , для получения проекций на оси x-y необходимо суммировать соответствующие проекции n- и t-компонентов исходного вектора. Все проекции можно найти при помощи скалярного произведения векторов (23):

$$v_{a'} = (v_{a'}n)n + (v_{a'}t) = (v_{a'}n)(n_x; n_y) + (v_{a'}t)(t_x; t_y); \quad (23)$$

Откуда:

$$x_{a'} = v_{a'}(1;0) = (v_{a'}n)n_x + (v_{ax}t)t_x; \quad (24)$$

$$y_{a'} = v_{a'}(0;1) = (v_{a'}n)n_y + (v_{ax}t)t_y; \quad (25)$$

4. Принцип работы

Прежде всего нужно определить переменные, структуры, константы и их значения в процессе работы программы:

```
#define max 40          //Максимальное количество кругов
#define cof_E 1         //Коэффициент восстановления, сейчас удар
                        //абсолютно упругий

typedef struct circ     //Структура, хранящая в себе данные об
                        окружности
{
 double x,y;            //Координаты окружности
 double r;              //Радиус
 double dx,dy,dr;       //Скорость по Ox, Oy, Or
 int color;             //Цвет окружности
};
```

```
int p;                  //Переменная-счетчик для таймера
int j;                  //Количество «существующих» шариков
int xold, yold;         //Координаты взорвавшегося шарика, для расчета
                        //векторов скоростей после взрыва
double rold;            //Радиус взорвавшегося шарика
int n;                  //Переменная для нормализации вектора скорости
```

Программа состоит из главного цикла в void main() , функции Collision() и DOT_PRODUCT():

- *Collision ()* – функция просчета скоростей шариков после столкновения;
- *DOT_PRODUCT ()* – функция скалярного умножения векторов.

Главный цикл

В теле главного цикла выполняется: вычисление новых значений переменных, вычисление новых координат окружностей, вызов функций *Collision()* , «взрыв» и «генерация» новых шариков, запуск таймера.

```
void main()
{
 //Объявление и инициализация переменных
 int i,j=1,k=0,q=0,p;
 int xold, yold;
 double rold;
 int n;

 int gdriver = DETECT, gmode;
 initgraph(&gdriver, &gmode, "c:\\bc\\bgi");

 //Создание массива кругов
 circ circles[max];

//Сбрасываем счетчик случайных чисел и заполняем поля структур
                        //случайными значениями
 randomize();
 for(i=0; i<max; i++)
  {
   circles[i].r = 3;
   circles[i].color = rand()%255+1;
   circles[i].dr = 0.5;
  }

//Задаем параметры для шарика-прародителя
 circles[0].x = (getmaxx()-1)/2;
 circles[0].y = (getmaxy()-1)/2;
 circles[0].dy = 0;
 circles[0].dx = 0;
 circles[0].dr = 0.5;
```

```
 while(!kbhit())
 {
  delay(25);

  setfillstyle(1, BLACK);
  bar(0,0, getmaxx(), getmaxy());

//Цикл последовательной прорисовки окружностей
  for(i=0; i<j; i++)
   {
//Обработка столкновений с другими шариками
   Collision(circles, j);

//Приращение координат к очередному шарику
   circles[i].x += circles[i].dx;
   circles[i].y += circles[i].dy;
   circles[i].r += circles[i].dr;

//Рисование очередного шарика
   setcolor(circles[i].color);
   circle((int)circles[i].x,(int)circles[i].y, (int)circles[i].r);

//Проверка столкновения с границей экрана
   if(circles[i].y + circles[i].r >= getmaxy()-1)
      {circles[i].dy =- circles[i].dy; circles[i].y -= 5;}
   if(circles[i].y - circles[i].r <= 1)
      {circles[i].dy =- circles[i].dy; circles[i].y += 5;}
   if(circles[i].x + circles[i].r >= getmaxx()-1)
      {circles[i].dx =- circles[i].dx; circles[i].x -= 5;}
   if(circles[i].x - circles[i].r <= 1)
      {circles[i].dx =- circles[i].dx; circles[i].x += 5;}

//Если очередной шарик достиг «критической массы»
   if(circles[i].r >= 60)
   {
      //То мы «взрываем» его - уменьшаем радиус
                    //и создаем из него «безопасный осколок»
      circles[i].r = 10; circles[i].dr = 0;
      //Если количесво шариков на экране превышает максимальное
                    //то все шарики «разминировать и включить
                    //таймер
      if(j>=max-1)
      {
           for(k=0; k<max; k++)
                {circles[k].dr = 0;circles[k].r = 20; q=1;}
           continue;}
```

```
    else
    {
     q=0;

     xold = circles[i].x;
     yold = circles[i].y;

     //иначе для взорвавшегося шарика определяем новые координаты
                 //такие, что бы шарик не пересекался с другими
     for(p=0;p<j+2;p++)
     {
       circles[i].x = xold + rand()%100 - 50;
       circles[i].y = yold + rand()%100 - 50;
       if((circles[i].x < circles[p].x-circles[p].r) ||
            (circles[i].x > (circles[p].x+circles[p].r)) ||
            (circles[i].y < circles[p].y-circles[p].r) ||
            (circles[i].y > (circles[p].x+circles[p].r))) break;
       if(p>=j+2)p=0;
     }
     //и так для других двух «новых» шариков
     for(p=0;p<j+2;p++)
     {
        circles[j].x = xold + rand()%100 - 50;
        circles[j].y = yold + rand()%100 - 50;
        if((circles[j].x < circles[p].x-circles[p].r) ||
             (circles[j].x > (circles[p].x+circles[p].r)) ||
             (circles[j].y < circles[p].y-circles[p].r) ||
             (circles[j].y > (circles[p].x+circles[p].r))) break;
       if(p>=j+2)p=0;
     }

     for(p=0;p<j+2;p++)
     {
        circles[j+1].x = xold + rand()%100 - 50;
        circles[j+1].y = yold + rand()%100 - 50;
        if((circles[j+1].x < circles[p].x-circles[p].r) ||
              (circles[j+1].x > (circles[p].x+circles[p].r)) ||
              (circles[j+1].y < circles[p].y-circles[p].r) ||
              (circles[j+1].y > (circles[p].x+circles[p].r)))
                     break;
        if(p>=j+2)p=0;
     }

//Расчет направления вектора скорости для нового шарика
      if(circles[i].x < xold) circles[i].dx = -rand()%3-1;
         else circles[i].dx = rand()%3+1;
```

```
      if(circles[j].x < xold) circles[j].dx = -rand()%3-1;
         else circles[j].dx = rand()%3+1;
      if(circles[j+1].x < xold) circles[j+1].dx = -rand()%3-1;
         else circles[j+1].dx = rand()%3+1;

      if(circles[i].y < yold) circles[i].dy = -rand()%3-1;
         else circles[i].dy = rand()%3+1;
      if(circles[j].y < yold) circles[j].dy = -rand()%3-1;
         else circles[j].dy = rand()%3+1;
      if(circles[j+1].y < yold) circles[j+1].dy = -rand()%3-1;
         else circles[j+1].dy = rand()%3+1;

//Нормализуем этот вектор, ведь сила взрыва для всех шариков
                     //одинакова -> и скорости будут одинаковы
      n = hypot(circles[i].dx, circles[i].dy);
      circles[i].dx = circles[i].dx/n;
      circles[i].dy = circles[i].dy/n;

      n = hypot(circles[j].dx, circles[j].dy);
      circles[j].dx = circles[j].dx/n;
      circles[j].dy = circles[j].dy/n;

      n = hypot(circles[j+1].dx, circles[j+1].dy);

      circles[j+1].dx = circles[j+1].dx/n;
      circles[j+1].dy = circles[j+1].dx/n;
//Теперь шариков на два больше!
      j=j+2;
     }
   }
 }
//Если количество шариков равно максимально возможному и был
                     //включен таймер, то собственно таймер мы и
                     //запускаем (один тик = delay)
 if(q!=0)q++;

//Через 25 секунд начинаем сценку заново
 if(q==1000)
   {
      q=0,j=1;
      for(i=0; i<max; i++)
      {
       circles[i].r = 1;
       circles[i].color = rand()%255+1;
       circles[i].dr = 0.5;
       circles[i].x = rand()%(getmaxx()+20)-10;
```

```
            circles[i].y = rand()%(getmaxy()+20)-10;
            circles[i].dy = rand()%6-3;
            circles[i].dx = rand()%6-5;
           }
       }
      }
      closegraph();
     }
```

Функция скалярного умножения векторов (DOT_PRODUCT).

В качестве параметров функция получает координаты двух векторов.

```
double DOT_PRODUCT(double x1, double y1, double x2, double y2)
{return(x1*x2+y1*y2);}
```

Функция расчета столкновений (Collision).

На вход функции подается массив шариков и размерность этого массива

```
void Collision(circ circles[], int n)
{

     int i,j;
     double nx, ny, l; //Нормальный вектор
     double tx, ty; //Тангенциальный вектор
     double vait, vain, vbit, vbin; //Начальные скорости двух шариков
          //(тангенциальные и нормальные)
     double vaft, vbft; //Тангенциальные компоненты конечных скоростей
     double ma,mb; //массы шариков
     double xfa, yfa, xfb, yfb; //координаты для конечных векторов
          //скоростей в X-Y с.к.
     double vafn, vbfn; //Нормальные составляющие конечных скоростей

     for(i=0; i<n; i++)
     {
      for(j=i+1; j<n; j++)
      {
       if(i==j) continue;
     //Вычисляем нормальный вектор a->b
          nx = circles[j].x - circles[i].x;
          ny = circles[j].y - circles[i].y;
          l = sqrt(nx*nx+ny*ny);



     //Есть ли столкновение?
          if(l <= (circles[i].r+circles[j].r)*1.1)
           {
     //Шарики столкнулись
```

```
        //Нормализуем вектор нормали и вычисляем тангенциальный вектор
               nx/=l;
               ny/=l;
               tx=-ny;
               ty = nx;
        //Вычисление начальных скоростей относительно тангенциального и
             //нормального векторов
               vait = DOT_PRODUCT(circles[i].dx, circles[i].dy, tx, ty);
               vain = DOT_PRODUCT(circles[i].dx, circles[i].dy, nx, ny);
               vbit = DOT_PRODUCT(circles[j].dx, circles[j].dy, tx, ty);
               vbin = DOT_PRODUCT(circles[j].dx, circles[j].dy, nx, ny);
        //Вычисляем массу шариков (в данной задаче она пропорциональна
             //радиусу)
               ma = 4/3*pi*pow(circles[i].r, 3);
               mb = 4/3*pi*pow(circles[j].r, 3);
        //Вычисляем нормальные составляющие скорости после столкновения
               vafn = (mb*vbin*(cof_E+1)+vain*(ma-cof_E*mb))/(ma+mb);
               vbfn = (ma*vain*(cof_E+1)-vbin*(ma-cof_E*mb))/(ma+mb);
        //Тангенциальный компонент не меняется
               vaft = vait;
               vbft = vbit;
        //Так как мы находимся в другой системе координат, то нужно
             //выполнить преобразование в новую с.к.
               xfa = vafn*nx+vaft*tx;
               yfa = vafn*ny+vaft*ty;
               xfb = vbfn*nx+vbft*tx;
               yfb = vbfn*ny+vbft*ty;
        //Запоминаем результат
               circles[i].dx = xfa;
               circles[i].dy = yfa;

               circles[j].dx = xfb;
               circles[j].dy = yfb;
        //Делаем приращение координат (уменьшаем шанс зацепления шариков)
               circles[i].x += circles[i].dx;
               circles[i].y += circles[i].dy;
               circles[j].x += circles[j].dx;
               circles[j].y += circles[j].dy;

             }
           }
        }
}
```