Mathl. Comput. Modelling Vol. 16, No. 12, pp. 83–98, 1992
Printed in Great Britain.

0895-7177/92 $5.00 + 0.00

A COMPUTER METHOD FOR ESTIMATING VOLUMES AND SURFACE AREAS OF COMPLEX STRUCTURES CONSISTING OF OVERLAPPING SPHERES

ROLF OTT, JELLE BIJMA, AND CHRISTOPH HEMLEBEN
Institut für Geologie und Paläontologie
Universität Tübingen, Sigwartstrasse 10, 7400 Tübingen, West Germany
MIGUEL SIGNES
Departamento de Geología, Facultad de Ciencias Biológicas
Universidad de Valencia, 46100-Burjassot, Spain

(Received February 1992)

Abstract—A PASCAL program which calculates volumes and surface areas of structures consisting of overlapping spheres is designed. The calculation is done by modelling the structure in the memory of a computer and then scanning the memory bit- or bytewise. A brief discussion of the error is presented, and an example for testing the algorithm is provided.

INTRODUCTION

Modern computers can easily calculate the volume and surface area of simple geometric structures by means of fundamental geometric formulas. Nevertheless, such calculations apply only where geometric structures are not arranged in a three-dimensional space and do not interpenetrate. Even in the case of simple spheres arranged in a coil, the calculation of volume and surface area is very difficult. We encountered this problem in the course of our research on the morphology of shells of foraminifers. In order to solve this problem, an algorithm was developed which can be used for certain species whose shells can be approximated by an arrangement of overlapping spheres.

METHODS

The following cases (i, ii, iii) outline our approach in estimating the volume and surface area of these coiled shells:

(i) One Single Sphere. This is the trivial case; volume and surface are calculated using the well-known formulas $V = \frac{4}{3}\pi r^3$ and $S = 4\pi r^2$.

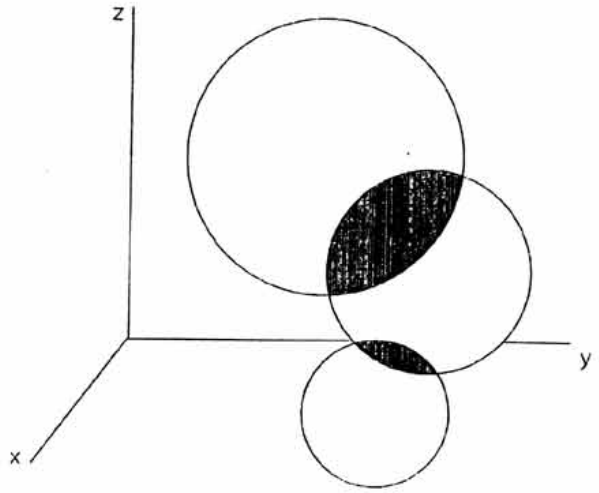
(ii) Two Overlapping Spheres. This calculation is easily managed by means of elementary geometry. In order to compute the volume and surface of the whole solid, we can use

$$V = V_1 + V_2 - V_L \quad \text{and} \quad S = S_1 + S_2 - S_L,$$

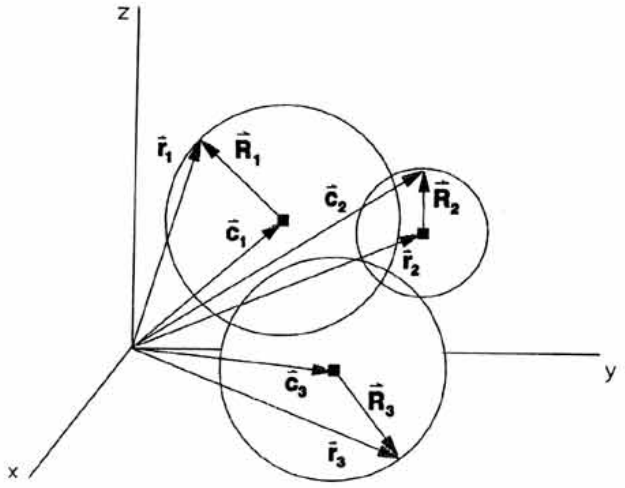
where V_1, V_2, S_1, S_2 are the volumes and surfaces of the spheres, V_L, S_L the volume and surface of the "lens" common to both spheres (see Figure 1a).

(iii) Three or More Overlapping Spheres. If the spheres are arranged in some kind of chain (see Figure 1a), we can successively use the formulas of case (ii). However, where several spheres overlap each other arbitrarily, more sophisticated mathematics are necessary. We

We are grateful to R. Olsson for critically reading the manuscript and for English corrections. The DFG (He 697/3) and University of Tübingen (Sondermittel) are acknowledged for financial support.



(a) A structure composed of three spheres. Two spheres build a common "lens."



(b) Three (or more) spheres overlapping arbitrarily. The calculation of volume and surface area becomes difficult.

Figure 1.

shall here outline a path to the solution of this problem: Two overlapping spheres result in a spacial intersection curve which is, of course, a circle, but three overlapping spheres intersect in only two points. The determination of these two points is the first step in calculating the volume. Therefore, the system

$$(\vec{r}_1 - \vec{c}_1)2 = R_1^2, \qquad (\vec{r}_2 - \vec{c}_2)2 = R_2^2, \qquad (\vec{r}_3 - \vec{c}_3)2 = R_3^2,$$

where each equation denotes a sphere, $\vec{r}$ points to a point on the surface, $\vec{c}$ to the center, and R is the radius of the sphere (Figure 1b) must be solved. The functional determinant (f_n is the equation of sphere n) of this quadratic simultaneous equation system cannot equal zero in order to solve independent solutions [1]:

$$\begin{vmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{vmatrix} \neq 0.$$

This solution can be done in an algebraic manner or by using an iteration algorithm in the computer.

As a second step, the space common to the three spheres must be integrated, using the intersection points and the intersection surface as integration limits. Then the volume is calculated as the sum of the three sphere's volume minus the volume of the intersection solid. At this point it is obvious that the calculation of a structure consisting of more than three spheres is almost impossible, using the tools of algebra and analysis.

What Can We Learn from Archimedes?

Archimedes probably would have modelled a geometric structure with plaster and immersed it in a container of water. He could then determine, more or less exactly, the structure's volume by measuring the volume of the displaced water mass. If, in a similar manner, only a close approximation of the volume of a geometric structure is required, then Archimedes' method can be simulated in a computer. A given quantum of computer memory can be substituted for the volume of water and, instead of a geometric solid, the structure is constructed in the computer in terms of information units. Thus, a solid whose dimension is 1000 length units (e.g., 1000 μm) can be placed in a cube with an edge length of 1000 information units. Information units can be chosen as bytes or bits. Bytes are easy to handle in a computer program and the resulting code of such a program will be fast; the use of bits will demand more sophisticated programming techniques, resulting in a slower code but saving considerable memory. An estimation of the memory consumption gives the following results: By using bytes, a memory size of 954 MB will be needed; the use of bits will consume "only" 120 MB. But if the cube is divided into 1000 layers of 1 length unit thickness each, only one layer is required in memory, and the need of computer memory is only 1 MB for a cube of bytes and 123 KB for a cube of bits. These are hardware requirements which should be available in most computers. Our first algorithm was implemented on an IBM compatible PC(80286) with 640 KB of RAM, hard disk, and math coprocessor. The programming work was done with Borland's TURBO PASCAL [2]. The edge length of the bit-cube was reduced to a modest 160 bits; more modern PCs, equipped with 4 or more MB of extended memory, allow larger cubes and, therefore, higher resolution of a structure in question. Subsequently, when more precision in the estimation of the volume and the surface area of structures is desired, a second algorithm using a cube of bytes was developed on a CONVEX II machine using Standard PASCAL [3] as programming language. As this computer is equipped with a memory of 128 Mbytes, the edge length of the cube can be extended to 1000 if needed.

Structures under study are fitted into the cube by either scaling them down or up, thereby gaining a maximum resolution and minimal calculation error. For this procedure, a special scaling algorithm was developed which calculated down- (up-) scaled radii of spheres and new centers of spheres. This establishes a scaling factor which is used to recalculate the volume and surface area to the original coordinate system.

The Algorithm

After reading the center coordinates and the radii of the spheres from standard input (or an external data file), the spheres are optimally fitted into the bit (or byte) cube using the procedures `ReadData` and `ScaleDown`. Instead of a cube, a prism with the edge lengths `maxx`, `maxy`, and `maxz` can be used in order to obtain an optimal fitting. The whole structure then is divided into `maxz` layers which are parallel to the x-y plane. The circles obtained for all spheres intersecting with a distinct layer are "filled out" either with "TRUE" values in a bytes cube or by setting a bit to "1" when using a bit cube [4]. Procedure `CreatePlane` performs this task. According to image processing conventions, a single bit or byte is considered as a pixel [5]. A single layer consisting of "0" or "1" (FALSE or TRUE)-pixels is scanned by the procedure `Trace`. If in the scanning of a pixel a value of "1" is found, an amount of one is added to the volume of the structure. For each pixel, the eight neighbor pixels are scanned counter-clockwise in order to detect changes in pixels from "0" to "1" (or the reverse). In the case where a surface pixel is encountered, the surface of the structure is increased by one (Figure 2). Finally, after scanning the entire structure, the sums of the volume and the surface area are retrieved by standard output.

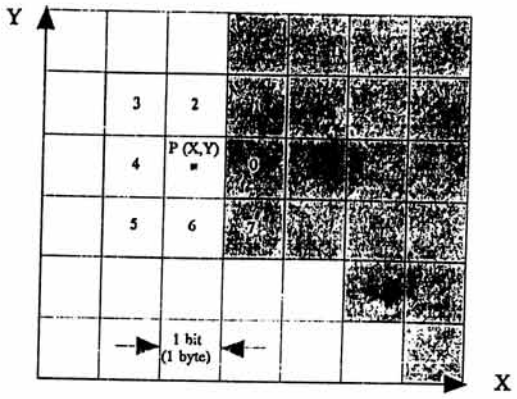
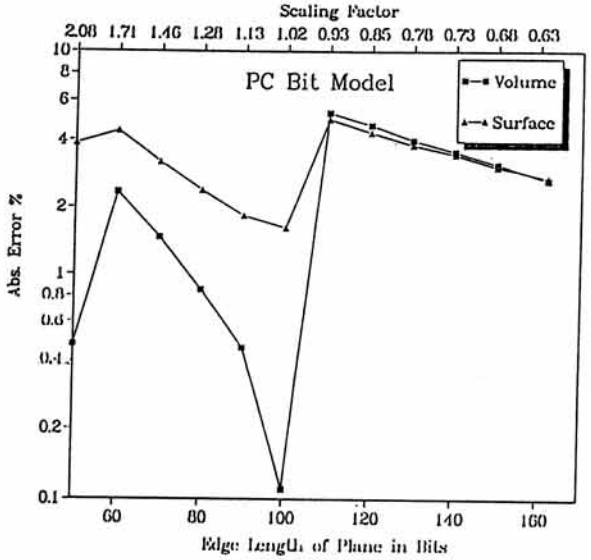
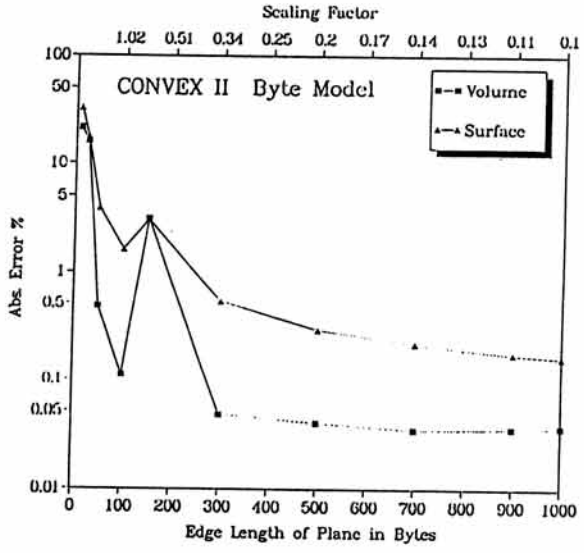


Figure 2. The neighbor pixels (numbered by 0 to 7) of $P(X,Y)$ are scanned anticlockwise starting with 0, in order to detect pixels belonging to the investigated structure.

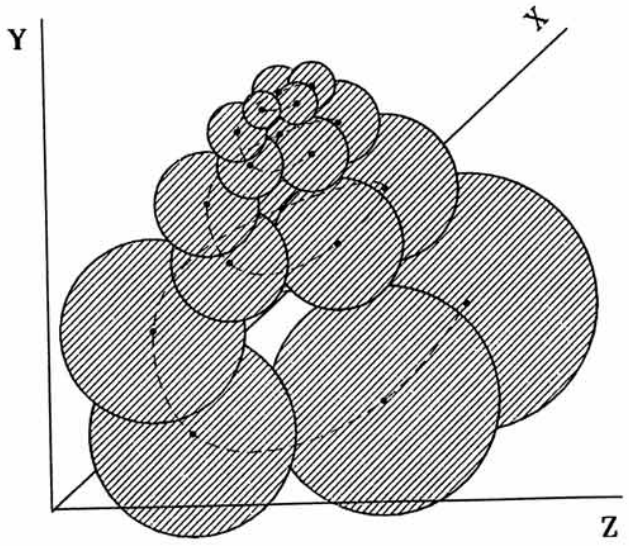


(a) A single sphere with a radius of 50 units as a test case. An edge length of the bit cube equal to the diameter of the sphere (i.e., scaling factor = 1) results in minimal computation errors.

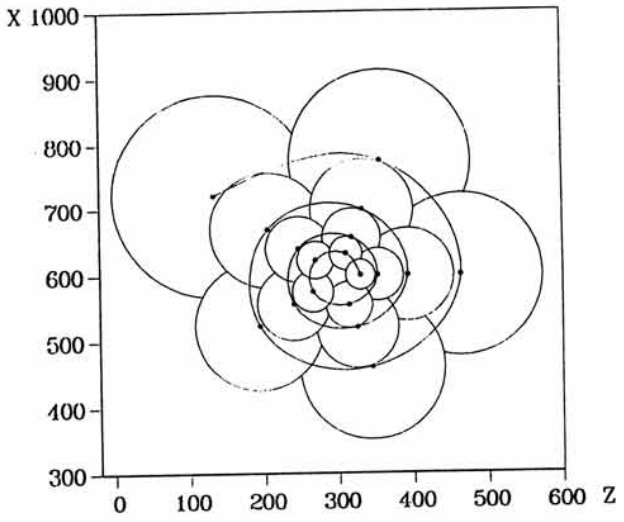


(b) For large edge lengths of the byte cube, the resulting errors become nearly constant.

Figure 3.



(a) A set of spheres whose centers are situated on a three-dimensional logarithmic spiral. Only two spheres are overlapping each other; so volume and surface area of the structure can be calculated by means of simple geometry.



(b) A projection into the x-z plane of the structure shown in Figure 4a.

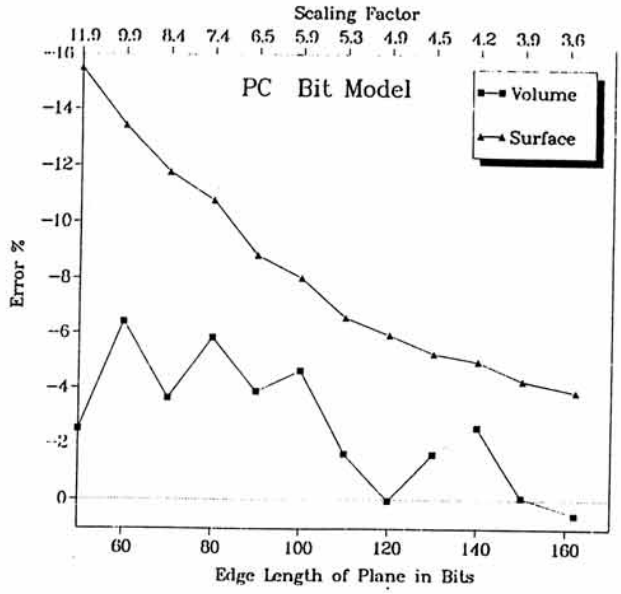
Figure 4.

RESULTS AND DISCUSSION

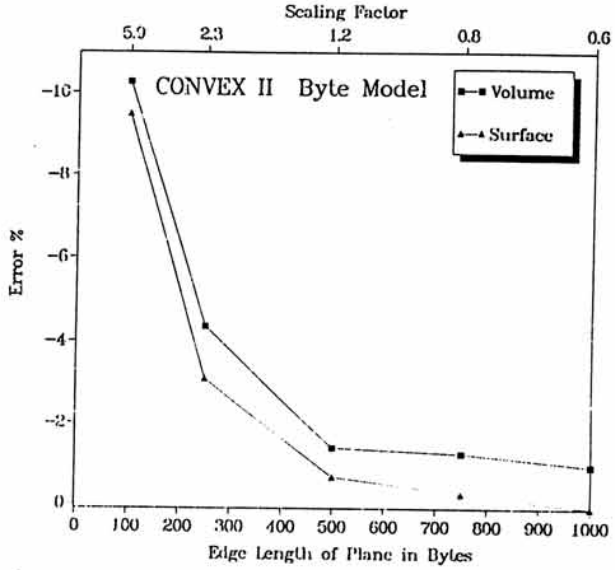
Testing the Algorithm

Testing with a single sphere

In order to compare calculated values with theoretical values, a sphere of a radius of 50 units was chosen for testing. As expected, the error of the volume and surface area is a function of the cube's edge length. If the sphere fills the entire bit- or byte space, respectively, which results in a scaling factor of 1, the error shows up at a minimum. Curiously, somewhat greater edge lengths give relatively high error values but with very large edge lengths the error remains almost a constant (Figures 3a and b). In cases where an error of about 0.5% is acceptable, an edge length of 300 units is sufficient. The scaling factor at this edge length computes to 0.34, which means that the structure is enlarged three times its original size.



(a) As the maximum extension of the test structure amounts to ca. 600 units, a personal computer will deliver results with considerable errors.



(b) A very fast computer equipped with a large memory should be used in order to achieve more exact results.

Figure 5.

Testing with a more complex structure

The second test structure consists of an arrangement of 18 spheres with increasing radii arranged along a logarithmic spiral, which is defined by the following set of equations in Cartesian coordinates:

$$x = x_0 + a\,e^{b\theta}, \qquad y = y_0 - c\,(e^{b\theta} - 1), \qquad z = z_0 + a\,e^{b\theta}.$$

The size of the spheres is computed according to

$$R_n = k\,R_{n-1}, \qquad n = 1, 2, \ldots, 18.$$

Table 1 provides a data set calculated by means of the above formulas. Only two spheres penetrate mutually, so the volume and the surface of the whole structure can be determined by elementary

geometry (Figures 4a and b). As Figures 5a and b show, an edge length of 300 units will cause an error of almost 4%. In cases where more precision is desired, a larger edge length (ca. 700 units) is required which eliminates the use of a personal computer. For example, the CONVEX II computer operated for many hours to master this task.

Table 1. Input data set representing overlapping spheres on a logarithmic spiral calculated with the following parameters: $x_0 = 330$, $y_0 = 710$, $z_0 = 600$, $a = 30$, $b = 0.09$, $c = 66$, $k = 1.12$. θ specifies the angle argument; x, y, z are the center coordinates of the spheres, and R is the radius of the sphere.

n	θ	x	y	z	R
1	0.0000	330.0	710.0	600.0	20.0
2	1.2566	310.4	702.1	631.9	22.4
3	2.5133	269.6	693.2	622.1	25.1
4	3.7699	265.9	683.3	575.2	28.1
5	5.0265	314.6	672.2	555.1	31.5
6	6.2832	352.8	659.8	600.0	35.2
7	7.5398	318.3	645.9	656.2	39.5
8	8.7965	246.4	630.3	638.9	44.2
9	10.0531	240.0	612.9	556.4	49.5
10	11.3097	325.7	593.4	521.0	55.5
11	12.5664	393.0	571.5	600.0	62.1
12	13.8230	332.2	547.0	699.0	69.6
13	15.0796	205.7	519.6	668.5	77.9
14	16.3363	194.4	488.9	523.3	87.3
15	17.5929	345.2	454.5	461.0	97.7
16	18.8496	463.6	416.0	600.0	109.5
17	20.1062	356.6	372.9	774.3	122.6
18	21.3628	134.0	324.6	720.6	137.3

CONCLUSION

A technique is designed for estimating the volume and surface area of an arrangement of spheres. By substituting procedure `CreateSphere` with a similar procedure (e.g., `CreatePrism`) or by adding other structure creating procedures to the program, even more complex structures can be calculated.

REFERENCES

1. L.N. Bronstein and K.A. Semendjajew, *Taschenbuch der Mathematik*, 840p., Harri Deutsch, Frankfurt/M., (1988).
2. TURBO-PASCAL, *Reference Manual*, 636p., Borland International, Scotts Valley, California, (1989).
3. K. Jensen and N. Wirth, *PASCAL, User Manual and Report*, Springer-Verlag, New York, (1979).
4. T. Pavlidis, *Algorithms for Graphics and Image Processing*, 416p., Springer-Verlag, Berlin-Heidelberg, (1982).
5. V.A. Kovalevsky, *Image Pattern Recognition*, 241p., Springer-Verlag, New York, (1980).

APPENDIX
PROGRAM LISTING

```
program ABACUSByte(input,output);

{Author:   Rolf Ott
Purpose:   Calculation of volume and surface of structures built of inter-
           secting spheres. The program is written in Standard Pascal and
           was tested on a CONVEX II machine under the operating system
           UNIX. There are no references to external files, so the
           input/output files have to be redirected: ABACUSBYTE >
           output_file < input_file. The length unit of the cube contain-
           ing the structure is 1 BYTE}

const
  maxsph        =   30;                          {The maximum number of spheres}
  maxdim        = 1000;                           {maximum edge length in bytes}

type
  plane         = array[0..maxdim, 0..maxdim] of boolean;
  SpheresArray  = array[1..maxsph] of real;

var
  N                          :integer;                          {number of spheres}
  Scale,Scale2,Scale3        :real;    {scaling factor: scale, scale**2, scale**3}
  xyplane                    :plane;                    {a layer of the structure}
  radii,xc,yc,zc             :array[1..maxsph] of integer;
                                        {scaled radii and centers of the spheres}
  xx,yy,zz,rr                :SpheresArray;
                                        {original radii and centers of the spheres}
  Xmin,Ymin,Zmin,
  Xmax,Ymax,Zmax             :real;
                                {minimum and maximum coordinates of the structure}
  MaxX,MaxY,MaxZ             :integer;    {maximum extension of the prism (cube).}
  dx,dy                      :array[0..7] of integer;

{=============================================================================}

 procedure ReadData(var N: integer; var f: text);       {Reads data input file}
   var
   i,kk   :integer;


   begin                                                              {ReadData}
     readln(f,MaxX); MaxY:=MaxX; MaxZ:=MaxX;          {read extension of cube}
     readln(f,N);                                     {read number of spheres}
     for i:=1 to N do
         readln(f,kk,xx[i],yy[i],zz[i],rr[i]);        {read centers and radii}
   end;                                                               {ReadData}

{=============================================================================}
procedure ByteSpace (N :integer);                                  {Main procedure}
var
    i,j,x,y,z          :integer;
    Vol,Surf,Ratio     :real;
    iVol,iSurf         :integer;

 procedure ScaleDown(N: integer; var Scale: real);
                        {Calculates scaled centers and radii of the spheres,
                                                  and issues a scaling factor.}
   var
     i :integer;

   procedure MiniMax;
     var
       i :integer;
```

```
begin                                                          {MiniMax}
  Xmin:=1.0E30; Xmax:=-Xmin;
  Ymin:=Xmin;   Ymax:=-Xmin;
  Zmin:=Xmin;   Zmax:=-Xmin;
  for i:=1 to N do
  begin
    if xx[i]+rr[i] > Xmax then Xmax:=xx[i]+rr[i];
    if yy[i]+rr[i] > Ymax then Ymax:=yy[i]+rr[i];
    if zz[i]+rr[i] > Zmax then Zmax:=zz[i]+rr[i];
    if xx[i]-rr[i] < Xmin then Xmin:=xx[i]-rr[i];
    if yy[i]-rr[i] < Ymin then Ymin:=yy[i]-rr[i];
    if zz[i]-rr[i] < Zmin then Zmin:=zz[i]-rr[i];
  end;
   writeln('Minima:',Xmin:10:3,Ymin:10:3,Zmin:10:3);
   writeln('Maxima:',Xmax:10:3,Ymax:10:3,Zmax:10:3);
 end;


 procedure DoScaling;
   type
     ScalingFactor = record
                       value :real;
                       class :1..5;
                     end;
    var
      SF                          :array[0..4] of ScalingFactor;
      temp                        :ScalingFactor;
      ordered, fitted, flipXZ     :boolean;
      i,j,delta,x2,y2,z2          :integer;
                                                               {DoScaling}
    begin
      delta:=0;  fitted:=false;
      repeat
      SF[0].value:=(Xmax-Xmin+delta)/(MaxX-1);
      SF[1].value:=(Ymax-Ymin+delta)/(MaxY-1);
      SF[2].value:=(Zmax-Zmin+delta)/(MaxZ-1);
      SF[3].value:=(Xmax-Xmin+delta)/(MaxZ-1);
      SF[4].value:=(Zmax-Zmin+delta)/(MaxX-1);
      for i:=0 to 4 do SF[i].class:=i+1;
      repeat
        ordered:=true;
        for i:=4 downto 1 do
          if SF[i-1].value > SF[i].value then
            begin
              temp:=SF[i];
              SF[i]:=SF[i-1];
              SF[i-1]:=temp;
              ordered:=false;
            end;
       until ordered;
       j:=-1;
```

```
            repeat
              j:=j+1;
              Scale:=SF[j].value;
              y2:=round((Ymax-Ymin)/Scale);
              if SF[j].class < 4 then
                 begin
                   x2:=round((Xmax-Xmin)/Scale);
                   z2:=round((Zmax-Zmin)/Scale);
                 end
              else { <-- class > 4 }
                 begin
                   x2:=round((Zmax-Zmin)/Scale);
                   z2:=round((Xmax-Xmin)/Scale);
                 end;
             if (x2 < (MaxX-1)) then
                 if (y2 < (MaxY-1)) then
                    if (z2 < (MaxZ-1)) then fitted:=true;
            until fitted or (j=4);
            delta:=delta+1;
          until fitted;
          flipXZ:=SF[j].class > 4;
          if flipXZ then
             for i:=1 to N do
               begin
                 xc[i]:=trunc((zz[i]-Zmin)/Scale)+1;
                 zc[i]:=trunc((xx[i]-Xmin)/Scale)+1;
                 yc[i]:=trunc((yy[i]-Ymin)/Scale)+1;
                 radii[i]:=trunc(rr[i]/Scale);
               end
            else
              for i:=1 to N do
                begin
                  xc[i]:=trunc((xx[i]-Xmin)/Scale)+1;
                  zc[i]:=trunc((zz[i]-Zmin)/Scale)+1;
                  yc[i]:=trunc((yy[i]-Ymin)/Scale)+1;
                  radii[i]:=trunc(rr[i]/Scale);
                end;
       end;                                                    {DoScaling}
    begin                                                      { Scaledown}
      MiniMax;
      DoScaling;
      for i:=1 to N do
      writeln(i:3,xx[i]:10:3,yy[i]:10:3,zz[i]:10:3,rr[i]:10:3,
              xc[i]:5,yc[i]:5,zc[i]:5,radii[i]:5);
    end;                                                       { of ScaleDown}
{-------------------------------------------------------------------------------}
    procedure CreatePlane(zi: integer);      {Create a layer of the structure.}
      var
        rcirc,h,rh      :real;
        rcirc2          :real;
        i               :integer;

      procedure CreateCircle(i:integer);
                          {Create a circle and fill it  with ''TRUE" values.}
        var
          x,w           :integer;
          y1,y2,ix1,iy  :integer;
          d             :real;
```

```
      begin                                                    {CreateSphere}
        for x:=round(xc[i]-rcirc) to xc[i] do
          begin
            d:=rcirc2 - Sqr(x-xc[i]);
            if d > 0.0 then
               begin
                 w:=trunc(Sqrt(d));
                 y1:=yc[i]+w; y2:=yc[i]-w; ix1:=xc[i]+xc[i] - x;
                 for iy:=y2 to y1 do
                   begin
                     XYPlane[x,iy]:=true; XYPlane[ix1,iy]:=true;
                     XYPlane[x,iy]:=true; XYPlane[ix1,iy]:=true;
                   end;
               end;
          end;
       end;                                                    {CreateSphere}

   begin                                                       {CreatePlane}
     for i:=1 to N do
       begin
         h:=zi - zc[i]; rh:= Radii[i];
         rcirc2:=Sqr(Rh) - Sqr(h);
         if rcirc2 > 0 then
            begin
              rcirc:=Sqrt(rcirc2);
              CreateCircle(i);
            end;
       end;
   end;

{-------------------------------------------------------------------------}

   procedure Trace(var Vol,Surf: integer);
                                        {Perform scanning of a single layer.}

   label 99;
   var
     x,y,z,i,n        :integer;

   begin                                                             {Trace}
     Vol:=0; Surf:=0;
     for y:= 1 to MaxY-1 do
     for x:= 1 to MaxX-1 do
       begin
         if XYPlane[x,y] then
            begin
              Vol:=Vol+1;
              for i:=0 to 7 do
                if (not XYPlane[x+dx[i],y+dy[i]]) then
                   begin
                     Surf:=Surf+1;
                     goto 99;
                   end;
            end;
 99:   end;
   end;                                                            {Trace}
{-------------------------------------------------------------------------}
```

```
  procedure ResetPlane;          {Fills a single layer with ''FALSE" values.}
    var
      x,y           :integer;

    begin                                                          {ResetPlane}
      for x:=0 to MaxX do
      for y:=0 to MaxY do
        xyplane[x,y]:=false;
    end;                                                           {ResetPlane}

begin                                                              {ByteSpace}
  write(chr(27),'[2J');                  {Clear screen (ANSI terminals only)}
  Scaledown(n,Scale);
  Scale2:=Sqr(Scale); Scale3:=Scale2*Scale;
  Vol:=0; Surf:=0;
  {writeln('     j     Vol    Surf');}
  for j:= 0 to MaxZ do
    begin
      ResetPlane;
      CreatePlane(j);
      Trace(iVol,iSurf);
      Vol:=Vol+iVol;
      Surf:=Surf+iSurf;
      {writeln(j:10,Vol*Scale3:10:1,Surf*Scale2:10:1);}
    end;
  Vol:=Vol*Scale3;
  Surf:=Surf*Scale2;
  Ratio:=Vol/Surf;
  writeln('  Scale        Vol    Surf     V/S');
  writeln(Scale:10:7,Vol:12:1,Surf:10:1,Ratio:10:2);
  writeln('------- END BYTESPACE -------');
end;                                                               {ByteSpace}

begin                                                              {ABACUSByte}
  dx[0]:= 1; dx[1]:=-1; dx[2]:=-1; dx[3]:= 1;
  dx[4]:= 1; dx[5]:= 0; dx[6]:=-1; dx[7]:= 0;
  dy[0]:= 1; dy[1]:= 1; dy[2]:=-1; dy[3]:=-1;
  dy[4]:= 0; dy[5]:= 1; dy[6]:= 0; dy[7]:=-1;
  ReadData(N,input);
  ByteSpace(N);
end.
```

```
{$A+,B-,D-,E-,F-,I-,L-,N+,O-,R-,S-,V-}
{$M 6000,0,655360}
program ABACUSBit;
{Author:     Rolf Ott
Purpose:     Calculation of volume and surface of structures built
             of intersecting spheres. The program is written in
             TURBO Pascal Version 5.5 and was tested on a 80386 PC
             under the operating MS-DOS V. 3.3. The length unit of
             the cube containing the structure is 1 BIT}

const
  maxsph   =  30;
  maxdim   = 400;
  maxyword = maxdim div 16 + 1;


type
  plane    = array[0..maxdim] of ^vector;
  vector   = array[0..maxyword] of word;
  real     = single;
  str12    = string[12];
  sarray   = array[1..maxsph] of real;
var
  maxx,maxy,maxz,code          :integer;
  fname,fout                   :str12;
  count,i,n                    :byte;
  scale,scale2,scale3          :real;
  xyplane                      :plane;
  f,inf                        :text;
  radii,xc,yc,zc               :array[1..maxsph] of integer;
  xx,yy,zz,rr                  :sarray;
  xmin,ymin,zmin,xmax,
  ymax,zmax                    :real;

procedure scaledown(n:integer; var scale: real);
  begin                              {the same code as in program ABACUSByte.}
  end;                                                                {ScaleDown}

procedure readdata(var n:byte);
  var i  :byte;
  begin
    assign(inf,fname); reset(inf);
    readln(inf,n);
    for i:=1 to n do readln(inf,kk,xx[i],yy[i],zz[i],rr[i]);
    close(inf);
  end;
```

```
procedure bitspace (n  :byte;  fname,fout   :str12);

  var
    i,j,x,y,z         :integer;
    startz,endz       :integer;
    vol,surf,ratcomp  :real;
    ivol, isurf       :longint;

{------------------------- Bit Handling Routines -------------------------}

procedure putbit(x,y: word);
  var
    nword,nbit       :word;
  begin
    nword := y div 16;
    nbit := y mod 16;
    XYPlane[x]^[nword] := XYPlane[x]^[nword]  OR (1 SHL nBit);
  end;


function getbit(x,y: word):boolean;
  var
    nword,nbit       :word;
  begin
    nword := y div 16;
    nbit := y mod 16;
    getbit := ((XYPlane[x]^[nword] SHR nBit) AND 1) = 1;
  end;

procedure createplane(zi: integer);
 var
   rcirc,rcirc2,h,rh   :real;
   i                   :integer;
 procedure createcirc(i:integer);
   const
     delta   =   1;
   var
     x,w,y1,y2,ix1,iy  :integer;
     d                 :real;
   begin
     for x:=round(xc[i]-rcirc) to xc[i] do
       begin
         d:=rcirc2 - sqr(x-xc[i]);
         if d > 0.0 then
            begin
              w:=trunc(sqrt(d));
              y1:=yc[i]+w; y2:=yc[i]-w; ix1:=xc[i]+xc[i] - x;
              for iy:=y2 to y1 do
                 begin
                   putbit(x,iy); putbit(ix1,iy);
                   putbit(x,iy); putbit(ix1,iy);
                 end;
            end;
       end
    end;
```

```
begin                                                          {createplane}
  for i:=1 to n do
    begin
      h:=zi - zc[i]; rh:= Radii[i];
      rcirc2:=sqr(Rh) - sqr(h);
      if rcirc2 > 0 then
         begin
           rcirc:=Sqrt(rcirc2);
           createcirc(i);
         end;
    end;
end;                                                           {CreatePlane}

procedure trace(var vol,surf: longint);
  label continue;

  const
    dx: array[0..7] of shortint = ( 1,-1,-1, 1, 1, 0,-1, 0);
    dy: array[0..7] of shortint = ( 1, 1,-1,-1, 0, 1, 0,-1);
  var
    x,y,z,i,n              :integer;
    xchange               :boolean;

  begin
    vol:=0; surf:=0;
    for y:= 1 to maxy-1 do
    for x:= 1 to maxx-1 do
       begin
         if getbit(x,y) then
            begin
              inc(vol);
              for i:=0 to 7 do
                 if (not getbit(x+dx[i],y+dy[i])) then
                    begin
                      Inc(surf);
                      goto continue
                    end;
            end;
  continue:
         end;
    end;                                                       {trace}

  procedure ResetPlane;
    var
      x,y  :integer;

    begin
      for x:=0 to maxx do
      for y:=0 to maxyword do
         xyplane[x]^[y]:=0;
    end;
```

```
begin                                                          {of BitSpace}
  for i:=0 to maxx do new(xyplane[i]);
  write(#27,'[2J');                                                {clrscr;}
  writeln('memory =  ',MemAvail:12,' bytes','      file: ',fname);
  writeln;

  assign(f,fout); rewrite(f);
  writeln(f,'  INPUT FILE: ',fname,'   OUTPUT FILE: ',fout); writeln(f);
  scaledown(n,scale);
  scale2:=sqr(scale); scale3:=scale2*scale;
  writeln('      j       Vol      Surf');
  for j:= 0 to maxz do
    begin
      ResetPlane;
      createplane(j);
      trace(ivol,isurf);
      vol:=vol+ivol;
      surf:=surf+isurf;
      writeln(j:10,vol*scale3:10:0,surf*scale2:10:0);
    end;
  vol:=vol*scale3;
  surf:=surf*scale2;
  ratcomp:=vol/surf;
  writeln(f,'  Scale       Vol      Surf       V/S');
  writeln(f,scale:10:7,vol:10:0,surf:10:0,ratcomp:10:2);
end;                                                             {BitSpace}

begin                                                          {ABACUSBit}
  Fname:=ParamStr(1); Fout:=ParamStr(2); Val(ParamStr(3),maxx,code);
  if code <> 0 then begin writeln('Illegal parameter'); halt(1); end;
  maxy:=maxx; maxz:=maxx;
  readdata(n);
  BitSpace(n,Fname,Fout);
  close(f);
end.
```