Kissing circles: A French romance in METAPOST

Denis Roegel

Abstract

When circles meet, they kiss. If three of them kiss, others can try to join and kiss all of them at once. In this article, we look at this problem from the METAPOST point of view, and we try to tell circles how to kiss, no matter their position and size. Recursive kissing will also be attempted.

1 Introduction

Apollonius of Perga (3rd century BC) was a Greek geometer, author of, among other things, a treatise on conical sections. He is credited of having coined the terms *ellipse*, *parabola* and *hyberbola*. His book *Tangencies*, cited by Pappus, defines the tangents problem as the problem of finding a circle tangent to three other objects being any combination of points, lines or circles. Apollonius showed how to solve this problem with a compass and straightedge, and it is now known as Apollonius' problem. When the three objects are circles, there are in general eight different solutions (Gisch and Ribando, 2004).

However, when the three circles are externally tangent to each other, these solutions reduce to only two non-trivial ones, namely the external and internal tangent circles, known as outer and inner Soddy circles (figure 1).

Descartes found a simple analytic solution. The curvatures $e_1 = 1/r_1$, $e_2 = 1/r_2$, $e_3 = 1/r_3$ of three circles kissing each other are related to the curvature e_4 of a Soddy circle through the equation

$$2(e_1^2 + e_2^2 + e_3^2 + e_4^2) = (e_1 + e_2 + e_3 + e_4)^2 \quad (1)$$

In this equation, e_1, e_2, and e_3 being given, e_4 has two solutions. The positive solution corresponds to the inner Soddy circle and the negative solution to the outer Soddy circle (whose radius is then $-1/e_4$). The analytic solution can be used to iterate the construction, but one has to be careful to avoid overflows. The outer Soddy circle will always appear at the border, hence a small curvature value, whereas smaller and smaller circles will be packed on the border, hence larger and larger values for the other curvatures. The METAPOST language is not very well suited to handling very small or very large values, and a geometric construction with no calculations is better suited to this problem.

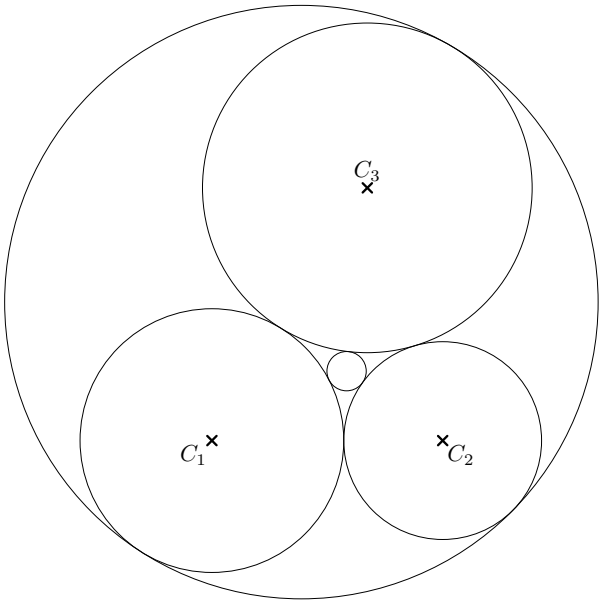


Figure 1: Inner and outer Soddy circles of circles C_1, C_2 and C_3.

2 David Eppstein's construction

In 2001, David Eppstein published a new construction for the inner and outer Soddy circles (Eppstein, 2001). Our purpose will not be to prove that this construction is indeed correct, but to see how best it can be implemented, and in particular in the most general way.

Eppstein's construction goes as follows. Given three tangent circles C_1, C_2 and C_3 (figure 2), a triangle connecting their centers is drawn. From each center, a perpendicular line is dropped to the opposite side of the triangle. The intersections between the perpendiculars and the triangle sides are marked with discs having small holes. The perpendiculars intersect their originating circle at two points, marked with discs and circles. Now, each disc can be connected to the tangency point (vertical cross) of the two other circles. This line meets the first circle at another point than the one with a disc, and we mark it with a filled square. The three points with filled squares are the tangency points of the inner Soddy circle.

The same procedure applied to the circle points yields the square points which are the points of tangency of the outer Soddy circle.

This construction can be used to find the circles internally tangent to the outer Soddy circle and circles C_1 and C_3 for instance, and the procedure can be used to build the Apollonian gasket (figure 3).

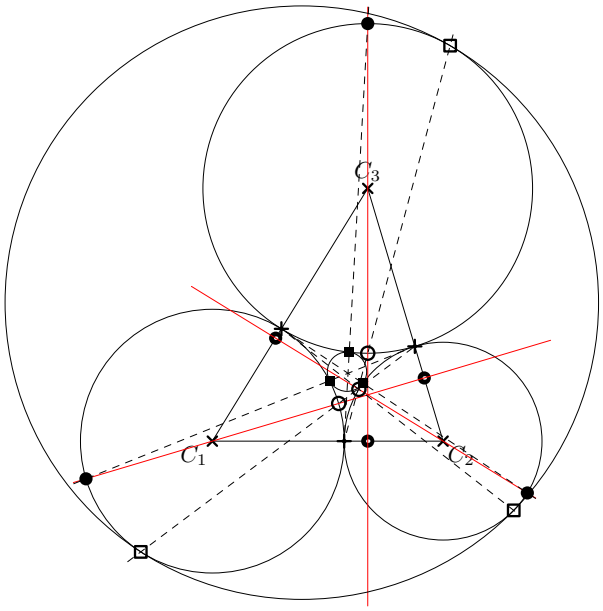


Figure 2: Eppstein's construction.

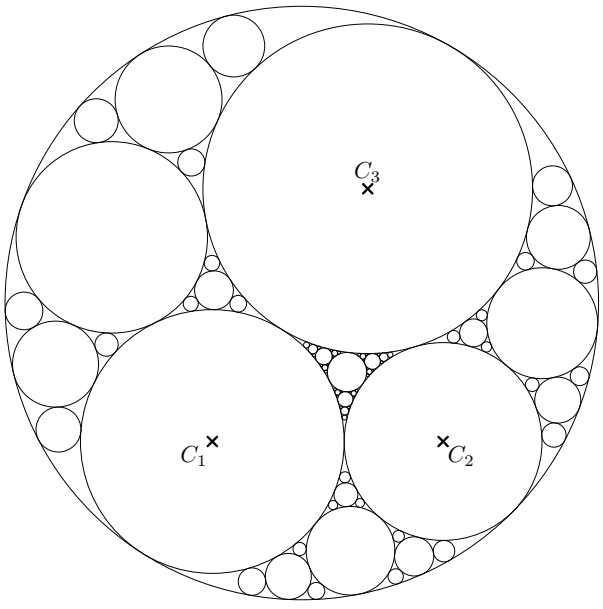


Figure 3: An Apollonian gasket of depth 3, with 83 circles.

3 Construction problems and METAPOST preliminaries

Finding the Soddy circles is rather straightforward, although special cases arise already at this stage. But the real problems are met when the construction is iterated, as the configurations of the circles change and there are several cases. The main source of difficulty is the duplicity of the circles. Eppstein's construction gives six points, but we must take great care in grouping these six points into two sets, and we have to find out which set corresponds to the circle we want to draw.

We will start by building a number of robust macros for common tasks. Some of these macros can easily be reused in other applications.

3.1 Height of a triangle

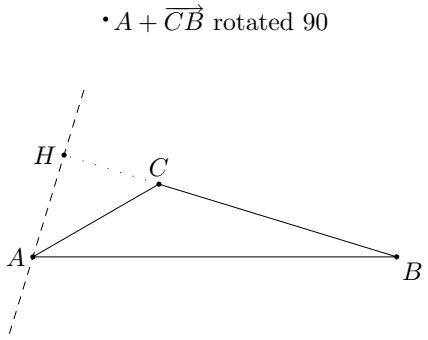
Our first macro (figure 4) finds the height of a triangle ABC starting at A. The height may not actually intersect the opposite side of the triangle, hence it is a good idea to use the `whatever` construction. Our macro states that the intersection H is *somewhere* on $[B, C]$ and *somewhere* on $[A, D]$ where D is a point constructed using $\overrightarrow{BC}$. The macro doesn't return the intersection H, but a path going beyond A and H by at least r, which is a value provided to the macro.

The idea is that we will use these paths to find their intersection with the originating circles, hence they need to be extended by at least the radius of the circle, and actually a bit more, in order to be sure that there is an intersection. Two paths, having an intersection coinciding with the start of one of the paths, may actually have no intersection for METAPOST due to rounding errors.

3.2 Circles

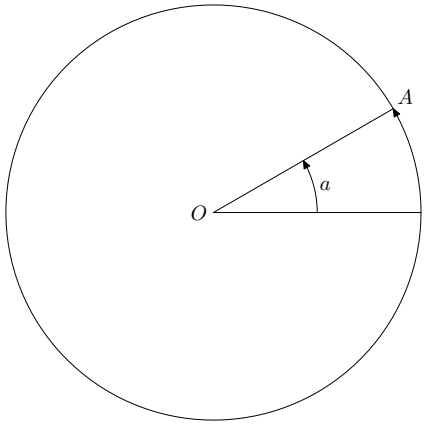
Circles can be obtained with the `fullcircle` macro, and this macro will be used to find intersections. However, circles are often a problem in that their intersection with a line is usually not unique, and if only one intersection is wanted, it is necessary to ensure that we get the right one. Another related problem is the discontinuity within a circle. Although not visible, a circle has a beginning and an end for METAPOST. It is often a good idea to avoid the discontinuity, which is a source of problems.

One convenient way of influencing the intersection returned by using the `intersectionpoint` function with a circle is to *rotate* the circle around its center (figure 5). This may seem of no use, but actually the intersections are computed in such a




```
vardef triangle_height(expr A,B,C,r)=
  save H,d;
  hide(
    pair H,d;
    H=whatever[B,C]
     =whatever[A,A+((B-C) rotated 90)];
    d=unitvector(H-A);
    )
  ((A-1.1r*d)--(H+r*d)) % height
enddef;
```

Figure 4: Finding a triangle height.




```
def circle(expr c,r,a)=
  (fullcircle scaled 2r
              rotated a shifted c)
enddef;
```

Figure 5: A circle rotated by a degrees. A is the origin and end of the circle path of center O.

way that the parameters of the paths are minimized. So, if we can make sure that the circle is in a position such that the intersection with the minimal parameter is the one we want, we will be rewarded.

We therefore define a macro for a circle of radius r, centered at c and rotated by an angle a.

3.3 Tangencies

We define a macro for the tangency point between two circles which are known to be tangent (figure 6). This macro needs to take care of the case where one circle is inside the other. This is the case when the distance between the two centers is smaller than one of the radii. Then, the circle with the smallest radius lies within the other. The tangency is obtained by extending the line connecting the circle centers. For instance, if C_a lies inside C_b, the tangency point is $C_a + r_a \times \dfrac{\overrightarrow{C_aC_b}}{\|\overrightarrow{C_aC_b}\|}$.

3.4 Angle between two lines

The angle between two lines is obtained using `angle` and is brought within the interval $[0, 180°[$ (figure 7).

3.5 Circle-line intersection

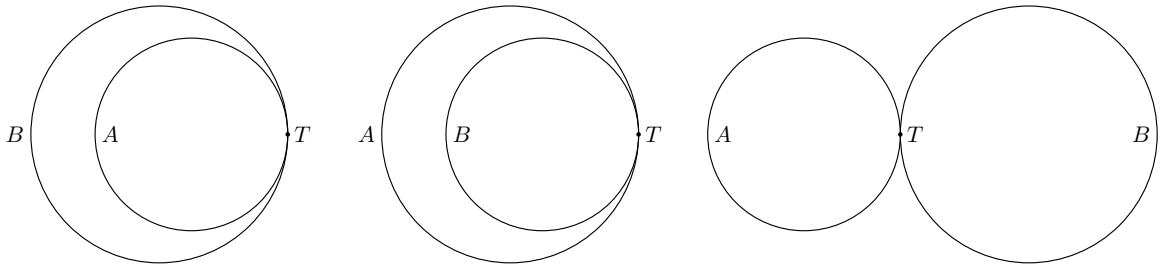
Eppstein's construction makes it necessary to find an intersection between a line and a circle, other than a given point. Devising a macro for that purpose which works in all cases is not trivial. One case that has to be taken into account is the case where the line is tangent to the circle, and possibly has no intersection at all with the circle due to rounding errors.

When writing such a macro (figure 8), we also have to ensure that it will not fail when the two intersections are too close. Given a point A on the circle, and another point B, our solution is to first compute the angle between the radius at A and the vector $\overrightarrow{AB}$. If this angle is between 89 and 91 degrees, we assume that the line (AB) is tangent to the circle. Even if it is not exactly tangent, the two intersections will be very close and can be confounded. In that case, we return A.

If not, we find the second intersection by comparing the distance to the center of the circle of two points chosen in opposite directions from A on the line. If B' is such that $\overrightarrow{AB} + \overrightarrow{AB'} = \overrightarrow{0}$, then the second intersection is on $[A, E]$, where E is the closest of B and B' from the center of the circle. We slightly rotate the circle counterclockwise in order to avoid A being found by the intersection. In this way, A corresponds to a very large parameter and will be avoided in the computation.

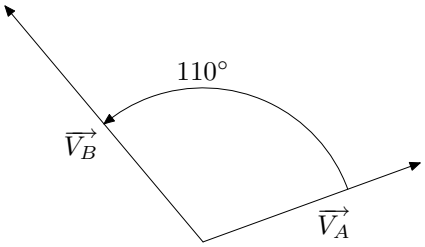
3.6 Circle going through three points

The final operation in Eppstein's construction takes three points and finds the circle going through these points (figure 9). We first obtain the center of this circle as the intersection of two medians. The macro `whatevermedian` returns an undefined point on a




```
def tangency(expr Ca,ra,Cb,rb)=
 (if (arclength(Ca--Cb)<rb) % a inside b
      or (arclength(Ca--Cb)<ra): % or b inside a
      if ra<rb: % a inside b
        Ca+ra*unitvector(Ca-Cb)
      else: % b inside a
        Cb+rb*unitvector(Cb-Ca)
      fi
  else: circle(Ca,ra,0) intersectionpoint (Ca--Cb)
  fi)
enddef;
```

Figure 6: The three cases of tangencies. Notice that this macro will not fail if the circles have no intersection due to rounding errors.




```
vardef angleof(expr Va,Vb)=
  save a;
  hide(
    a=angle(Vb)-angle(Va);
    forever:
      if a>=180:a:=a-180;fi;
      if a<0:a:=a+180;fi;
      exitif ((a<180) and (a>=0));
    endfor;
  )
  a % angle
enddef;
```

Figure 7: Angle between two lines defined by vectors $\overrightarrow{V_A}$ and $\overrightarrow{V_B}$, brought on the interval $[0, 180°[$.

line and is used in the `circle_through` macro. This macro defines the center and the radius of the circle.

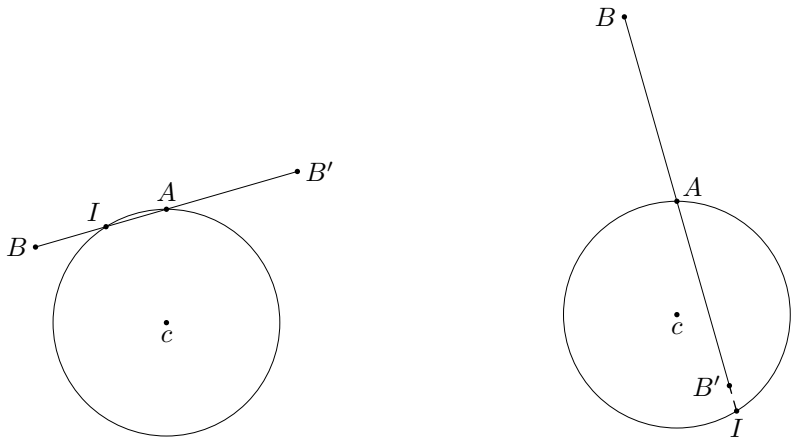
3.7 Existence of a segment intersection

It will be useful to have a macro telling when two segments (and not lines) intersect, and when they do not. An easy way to achieve this is to provide a boolean version of the `intersectionpoint` macro and have it return `true` instead of the intersection and `false` instead of an error message, as follows:

```
secondarydef p intersectionpoint_b q =
 begingroup save x_,y_;
   (x_,y_)=p intersectiontimes q;
   if x_<0:
     false
   else: true
   fi
  endgroup
enddef;
```

3.8 Innerness and outerness

Next, we define the following macro which will take four circle centers. Circles B and C are tangent externally and circle D is internally tangent to both. A is a circle externally tangent to B and C, but internally tangent to D. In other words, D is the outer Soddy circle of A, B and C.




```
vardef intersection_circle_line(expr c,r,A,B)=
  save a,BP,I,uv;
  hide(
    pair BP,I,uv;
    a=abs(angleof(B-A,A-c)-90);
    if a<1: I=A;
    else:
      BP=A+(A-B);
      if arclength(c--B)<arclength(c--BP):
        uv=unitvector(A-B);
        I=circle(c,r,angle(A-c)+1) intersectionpoint ((B-2.1r*uv)--(1.1[B,A]));
      else:
        uv=unitvector(A-BP);
        I=circle(c,r,angle(A-c)+1) intersectionpoint ((BP-2.1r*uv)--(1.1[BP,A]));
      fi;
    fi;)
  I % point returned
enddef;
```

Figure 8: Intersection between a line and a circle. The circle, its point A and B are given. We search the other intersection I. The shortest of cB and cB' indicates on which side of the line (BB') is I, with respect to A.

Conversely, given B, C and D, Eppstein's construction gives two circles, one of them being A. It turns out that which points lead to which circle depends on the existence of an intersection between $[C_a, C_d]$ and $[C_b, C_c]$, where C_a, C_b, C_c and C_d are the circle centers. We call the two possible circles tangent externally to B and C, but internally to D, *inner* when the aforementioned intersection exists and *outer* when it doesn't. This is consistent with the outer Soddy circle center being such that there is no intersection for the previous segments.

In practice, of the two circles, the inner one will be the smaller of the two.

```
def is_inner(expr Ca,Cb,Cc,Cd)=
  ((Cb--Cc) intersectionpoint_b (Ca--Cd))
enddef;
```

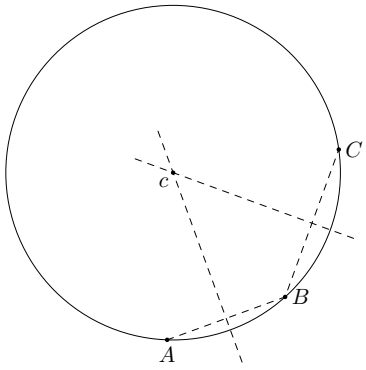
3.9 Slope

Given a segment, the `slope` macro gives its slope as an angle. This will be a convenient macro when we need to rotate a circle in order to favor a certain intersection.

```
def slope(expr p)=
  angle((point 1 of p)-(point 0 of p))
enddef;
```

4 The main macro

The main macro takes four circle centers, as well as four radii, and an integer for the recursion depth. Initially, the first three circles are three tangent circles for which we find the Soddy circles. The fourth circle will be non-meaningful and assigned a negative radius.




```
def whatevermedian(expr A,B)=
  whatever[.5[A,B],
           .5[A,B]+(B-A) rotated 90]
enddef;

def circle_through
      (expr A,B,C)(text c)(text r)=
  c=whatevermedian(A,B)
   =whatevermedian(B,C);
  r=arclength(A--c);
  draw fullcircle scaled 2r shifted c;
enddef;
```

Figure 9: Circle going through three points A, B and C.

Later, when iterating the construction, there will be two cases. In the first case the three first circles are tangent externally, and an inner Soddy circle is to be found. In that case, the fourth circle is also non-meaningful.

The second case is a border case, where the first circle is the outer Soddy circle. The second and third circles are externally tangent, but internally tangent to the Soddy circle. And then the fourth circle is another circle externally tangent to the second and third, and internally tangent to the first. This fourth circle has already been drawn, but it is necessary to find out which one is the other circle not yet drawn at the border.

4.1 Computing the tangencies and triangle heights

The tangencies and triangle heights are obtained straightforwardly, given the previous definitions.

```
vardef tangent_circles
     (expr Ca,Cb,Cc,Co,
           ra,rb,rc,ro,n)=
  save C,T,r,ht,Ihc,Ilc;
  pair C[]; % centers
  pair T[][]; % tangencies
  numeric r[]; % radii
  path ht[]; % heights
  pair Ihc[][]; % intersections between
                % heights and circles
  pair Ilc[]; % final intersections
  C1=Ca;C2=Cb;C3=Cc;
  r1=ra;r2=rb;r3=rc;
  T[1][2]=tangency(C1,r1,C2,r2);
  T[2][3]=tangency(C2,r2,C3,r3);
  T[3][1]=tangency(C3,r3,C1,r1);
  ht1=triangle_height(C1,C2,C3,r1);
  ht2=triangle_height(C2,C1,C3,r2);
  ht3=triangle_height(C3,C1,C2,r3);
```

In order to simplify certain expressions, we also define

```
def next(expr i)=
  (if i+1<4:i+1 else: 1 fi)
enddef;
```

4.2 Diameters and other intersections

The intersections between the heights and the circles are easy to obtain, but the key to success is to group them correctly. In our case, we first group the intersections which go towards the feet of the heights, by slightly rotating the circles clockwise. This gives the points `Ihc[i][1]`, which are marked with small circles.

The second group of points is the opposite one and they are marked with small discs.

Then, the circles and discs are joined to the opposite tangencies, yielding the squares and filled squares.

```
for i:=1 upto 3:
  % circle
  Ihc[i][1]
     =circle(C[i],r[i],slope(ht[i])-5)
      intersectionpoint ht[i];
  % disc
  Ihc[i][2]-C[i]=C[i]-Ihc[i][1];
  % square
  Ilc[i]=intersection_circle_line(
        C[i],r[i],
        Ihc[i][1],
        T[next(i)][next(next(i))]);
  % filled square
  Ilc[3+i]=intersection_circle_line(
        C[i],r[i],
        Ihc[i][2],
        T[next(i)][next(next(i))]);
endfor;
```

4.3 The final step

In the final step (figure 10), we have to distinguish whether this is the first time the macro is called. When it is called for the first time (case 1), only the outer and inner Soddy circles need to be drawn.

```
if firststep: % case 1
  firststep:=false;
  % outer Soddy
  circle_through(Ilc1,Ilc2,Ilc3)
                (C4)(r4);
  % inner Soddy
  circle_through(Ilc4,Ilc5,Ilc6)
                (C5)(r5);
  % we recurse
  if n>0: % border cases
    tangent_circles(C4,C1,C2,C3,
                    r4,r1,r2,r3,n-1);
    tangent_circles(C4,C2,C3,C1,
                    r4,r2,r3,r1,n-1);
    tangent_circles(C4,C3,C1,C2,
                    r4,r3,r1,r2,n-1);
  fi;
else:
  if ro<0: % case 2
    circle_through(Ilc4,Ilc5,Ilc6)
                  (C5)(r5)
  else: % case 3
    if is_inner(Co,C2,C3,C1):
      circle_through(Ilc1,Ilc2,Ilc3)
                    (C4)(r4);
    else:
      circle_through(Ilc4,Ilc5,Ilc6)
                    (C4)(r4);
    fi;
  fi;
fi;

% we recurse
if n>0: % case 4
  if ro>0: % case 5
    tangent_circles(C1,C2,C4,C3,
                    r1,r2,r4,r3,n-1);
    tangent_circles(C1,C3,C4,C2,
                    r1,r3,r4,r2,n-1);
    tangent_circles(C2,C3,C4,origin,
                    r2,r3,r4,-1,n-1);
  else: % case 6
    tangent_circles(C1,C2,C5,origin,
                    r1,r2,r5,-1,n-1);
    tangent_circles(C1,C3,C5,origin,
                    r1,r3,r5,-1,n-1);
    tangent_circles(C2,C3,C5,origin,
                    r2,r3,r5,-1,n-1);
  fi;
fi;
```

Figure 10: The high-level structure of the recursion.

```
pair C[]; % centers
numeric r[];
numeric n; % depth
n=7;
C1=origin;
r1=4cm;
r2=3cm;
r3=6cm;
% we find the center C2:
C2-C1=(r1+r2,0);
% there are now two possibilities for C3,
% we keep only one
C3=circle(C1,r1+r3,0)
    intersectionpoint circle(C2,r2+r3,0);
draw circle(C1,r1,0);
draw circle(C2,r2,0);
draw circle(C3,r3,0);
firststep:=true;
tangent_circles(C1,C2,C3,origin,
                r1,r2,r3,-1,n);
```

Figure 11: The driver of the construction. Three initial circles are defined and the general macro is called.

Once the outer Soddy circle C_4 has been obtained, the macro is again called on each border. In each case, we provide the outer Soddy circle as the first parameter, then two of the three inner circles, then the third inner circle. During the next call of the macro, the "case 3" part will be in effect, and the new circle to be squeezed between the two inner circles and the outer Soddy circle is found out using the innerness criterion mentioned above.

"Case 2" applies when the inner Soddy circle of three externally tangent circles has to be found.

But no matter what, when "case 4" is reached, we have found a circle and two new cases apply: either we have a border case with the original outer Soddy circle C_1 (case 5), or the circle is the inner Soddy circle of three externally tangent circles (case 6). Splitting case 5 leads to two new border cases and one inner case (the usual inner Soddy circle). Splitting case 6 leads to three new inner cases (the usual inner Soddy circles). Note that the parameter `origin` is irrelevant when the radius is negative.

The macro is started as shown in figure 11.

5 Conclusion

Our little journey through kissing circles has presented every detail of the METAPOST construction. We have followed Eppstein's construction closely.

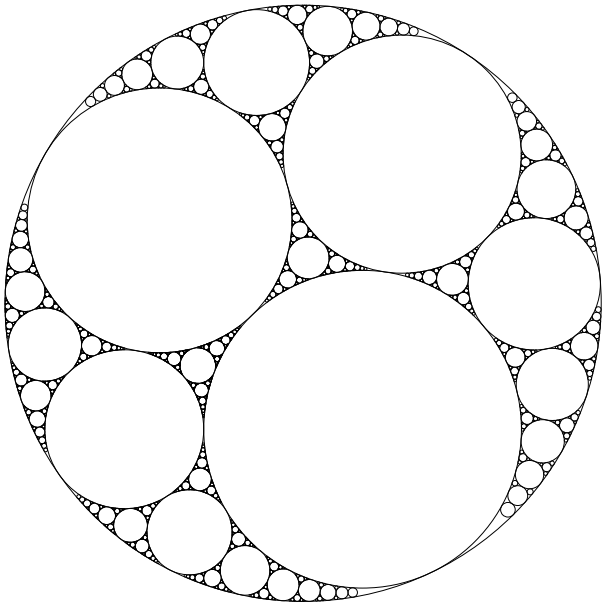


Figure 12: An Apollonian gasket of depth 7, with 6563 circles. The total number of circles is $5+3\times(3^n-1)$ where n is the depth. For $n=0$, we have five circles, which are the three base circles and the two Soddy circles.

The resulting code is simple, but this simplicity was not achieved right away. It is supported by the robustness of each macro which tries to be as general as possible. The code produced is not tied to a particular set of circle radii and should work in all cases, provided METAPOST's capacities are not overflowed. We conclude with an Apollonian gasket of depth 7, built with our code (figure 12).

References

Eppstein, David. "Tangent Spheres and Triangle Centers". *American Mathematical Monthly* **108**, 63–66, 2001.

Gisch, David and J. M. Ribando. "Apollonius' Problem: A Study of Solutions and Their Connections". *American Journal of Undergraduate Research* **3**(1), 15–25, 2004.

◇ Denis Roegel
LORIA
BP 239
54506 Vandœuvre-lès-Nancy
France
roegel@loria.fr
http://www.loria.fr/~roegel