A Generalized Malfatti Problem

Ching-Shoei Chiang,[1] Christoph M. Hoffmann,[2] and Paul Rosen[3]
Soochow University, Taipei, Taiwan, R.O.C.
Purdue University, West Lafayette, IN, USA

Abstract

Malfatti's problem, first published in 1803, is commonly understood to ask fitting three circles into a given triangle such that they are tangent to each other, externally, and such that each circle is tangent to a pair of the triangle's sides. There are many solutions based on geometric constructions, as well as generalizations in which the triangle sides are assumed to be circle arcs. A generalization that asks to fit six circles into the triangle, tangent to each other and to the triangle sides, has been considered a good example of a problem that requires sophisticated numerical iteration to solve by computer. We analyze this problem and show how to solve it quickly.

Keywords: Malfatti's problem, circle packing, geometric constraint solving, GPU programming.

1. Introduction

Geometric constraint solving plays a pivotal role in computer-aided design (CAD). Practical solvers include graph-theoretic solvers in which the constraint problem is decomposed into sub problems of known structure, those sub problems subsequently are solved, and then the solution fragments are assembled into a solution of the original problem; e.g., [1,11,12,14,16]. Such constraint sub problems often involve constructing circles with unknown radius and center. In the literature on this subject, such circles are determined, one at a time, either sequentially, for instance when required to be tangent to three geometric elements [5], or simultaneously with other elements, for example when tangent to four geometric elements, not all fixed in relation to each other; e.g., [6]. In this paper we consider a third class of problems in which several circles are to be determined simultaneously, such that they are tangent to each other and to the sides of a given container, in this case a triangle. We consider this problem with three and with six circles. The problem thus has more similarity to circle and sphere packing problems, although the radii are not required to be equal.

Malfatti's problem, as originally stated in 1803, was to carve three circular columns from a prismatic slab of marble so as to minimize waste. Equivalently, the problem asked to fit three circles into a given triangle such that the sum of their areas is maximum. Malfatti erroneously believed that the solution was provided by three inscribed circles that are tangent to each other and tangent to the sides of the triangle. An instance of this problem is shown in Figure 1. Since then, this second problem, which Malfatti believed to maximize the area, has become understood to be *Malfatti's Problem* in the literature. There are several known solutions of Malfatti's problem. The interested reader is referred to [2,17,18] for a history of the problem and an explanation of various solutions and generalizations.

[1] Soochow University, Taipei, Taiwan, R.O.C.. Email: chiang@scu.edu.tw; Tel/Fax: +886-2-2311-1531 ext. 3801
[2] Purdue University, West Lafayette, IN 47907, USA. E-mail: cmh@cs.purdue.edu; Tel: 1-765-494-6185
[3] Purdue University, West Lafayette, IN 47907, USA. E-mail: rosen@purdue.edu

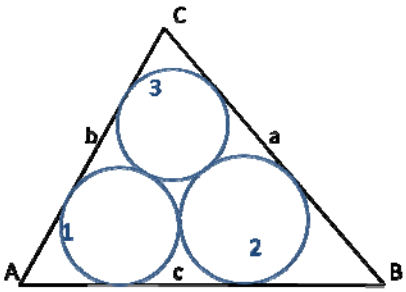

Figure 1: Malfatti's Problem

Our interest in Malfatti's problem comes via a generalization, popularized in the geometric constraint solving community by Lamure and Michelucci in [15]. In this generalization, we seek to fit six circles into a given triangle such that the circles are tangent to each other and to the sides of the triangle. An example of this generalization is shown in Figure 2. Lamure and Michelucci cited this generalization as a prime example of a constraint problem whose algebraic solution is difficult and is best solved by homotopy continuation, an advanced and expensive numerical method; see also [7].

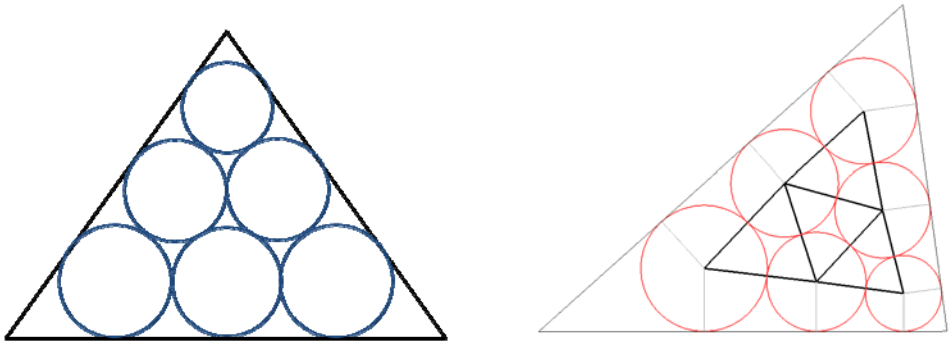

Figure 2: The Generalized Malfatti Problem

In this paper, we revisit these two problems and solve them efficiently – without the need for homotopy continuation – but not without iteration. Section 2 reviews some definitions and concepts, Section 3 deals with the classical case of fitting three circles. Two direct solutions are given, one using the GPU to solve equation systems, the other using classical formulae. We develop an inverse approach as well and introduce an angle-clamping method that allows us to change two angles of the triangle while keeping the third one fixed. Section 4 considers the six-circle generalization. Here, we derive a direct solution for isosceles triangles, requiring no iteration first. Then, we develop an inverse method that solves the six-circle problem efficiently. Section 5 discusses the implementation of the various methods and the speeds that we have measured. Section 6 concludes with some thoughts on the problems and on GPU-based approaches to solving them.

Our problem is related to the circle-packing problem [7] that asks, in its basic form, to place a number of circles (externally) tangent to each other such that a particular topology is realized. The topology is expressed by a graph whose vertices represent the circles and whose edges represent tangencies between them. Variations of the problem include that circles could overlap and that a cycle of tangent circles winds multiply around an interior circle. A solution of this circle-packing problem is a radius assignment to the graph vertices such that the tangencies are as required and the vertices of the resulting graph

embedding are centers of circles with the assigned radii. The problem can be considered in non-Euclidean geometry as well.

In this paper, we require no multiple coverings and consider only Euclidean geometry. The graph *K* of [7] is then the central net considered in Section 4.2. Stephenson's algorithm iteratively solves the circle packing problem for arbitrary graphs using iteration. The conditions that express whether a particular radius assignment solves the packing problem are formulated by angle constraints on the triangles in the embedded graph. This leads to a general iterative method for the packing problem that has been demonstrated to pack large numbers of circles. The performance, however, has only been reported as estimate of the number of floating point operations. Such estimates do not account for latency and memory transfers that can adversely impact performance, especially when considering GPU implementations.

The angle conditions underlying Stephenson's algorithm elegantly solve the packing problem in multiple geometries. We believe that they can be modified, in principle, so as to express tangency conditions to the sides of a given triangle. See also Figure 13. It remains to be seen whether the algorithm has a suitable adaptation to packing circles into a polygonal domain at competitive speeds with acceptable accuracy, and whether a GPU implementation then performs at interactive speeds. Also, since the construction depends on angle conditions, we believe that Stephenson's algorithm needs a scaling step that fits the circle complex into the given enclosure.

2. Definitions

We name vertices, sides and angles of triangles in the traditional manner, with angle α at vertex *A*, side *a* opposite vertex *A*, and so on. All tangencies between two circles are understood to be exterior, that is, neither circle contains the other.

The *bisector*, or *medial axis*, of a circle and a line is a parabola. If the circle with radius *r* is centered at (0,*r*) and the line is the *x*-axis, then the bisector equation is $4ry = x^2$. The bisector of two circles is a hyperbolic arc. Recalling the definition that a hyperbola is the locus of points whose distance from the foci differs by a constant, we note that the hyperbolic arc has the circle centers as foci and the distance difference from the foci is the radius difference. So, if the radii are equal, the bisector is a straight line.

We recall that the angle over a secant of a circle is constant. If the secant is a diameter, then the angle is $\pi/2$. A circle that is tangent to two sides of a triangle, on the inside, is centered on the (interior) angle bisector of those sides. The incircle is the inscribed circle tangent to all sides and centered that the incenter, the intersection of the angle bisectors. Formulae for the incenter and incircle radius are found in the standard textbooks on plane geometry.

3. The Classical Malfatti Problem

3.1. Three Circles

We are given a triangle $\Delta(A,B,C)$ and are to fit three circles such that each circle touches two sides of the triangle and the three circles touch each other. An example is shown in Figure 3.

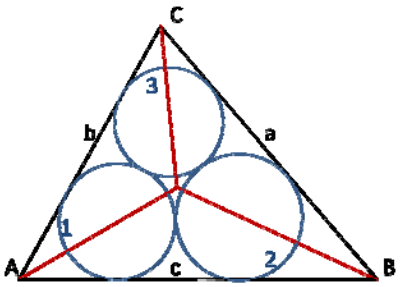

Figure 3: Fitting three circles into a triangle. Side lengths are $a = \sqrt{85}$, $b = \sqrt{65}$, $c = 10$. The computed radii are $r_1 = 1.6227$, $r_2 = 1.7591$, $r_3 = 1.5070$.

The 3-circle Malfatti problem translated into algebra yields three quadratic equations, so we expect eight solutions in all. The variables are the square roots u_k of the three radii. A second solution is shown in Figure 4 and gives insight into the geometry of the other solutions. Here, the circle 1, with a negative u_1, is tangent to the sides *b* and *c*, as required, but its center lies outside the triangle. Circle 3 is tangent to sides *b* and *a*, but it intersects the side c. We conclude that the solution(s) of interest require positive u_k .

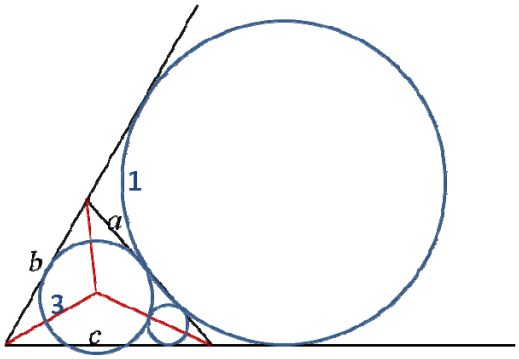

Figure 4: A solution in which u_1 is negative and the other variables are positive. Circle 1 is tangent to the (extended) sides *b* and *c*. The intersection of circle 3 with side *c* of the triangle is not an error, since the equations only require tangency with sides *a* and *b*. Similarly, circle 1 intersects side *a*.

The derivation of the three quadratic equations describing the eight solutions is best understood from Figure 5; see also [18]. The circles with radius r_1 and r_2 touch side *c* and touch each other. The length of side *c* is the sum $l_1 + l_{12} + l_2$, which can be expressed as $c = r_1 \cot(\alpha/2) + 2\sqrt{r_1 r_2} + r_2 \cot(\beta/2)$. We substitute $u_k = \sqrt{r_k}$, where $k = 1, 2, 3$, and obtain the equation system

$$a = u_2^2 \cot\left(\frac{\beta}{2}\right) + 2u_2u_3 + u_3^2 \cot\left(\frac{\gamma}{2}\right)$$

$$b = u_3^2 \cot\left(\frac{\gamma}{2}\right) + 2u_1u_3 + u_1^2 \cot\left(\frac{\alpha}{2}\right)$$

$$c = u_1^2 \cot\left(\frac{\alpha}{2}\right) + 2u_2u_1 + u_2^2 \cot\left(\frac{\beta}{2}\right)$$

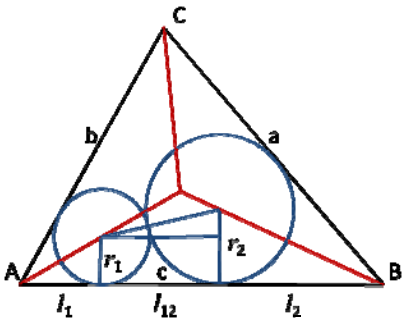

Figure 5: Length equation for side *c*. The circle centers lie on the (red) angle bisectors.

Since all angles in a proper triangle are less than 180°, the half angles cannot exceed 90° and therefore the coefficients of the quadratic terms are all positive. The type of the conic depends on the discriminant of the quadratic terms which is, for the third equation, $D = 4 - 4\cot(\alpha/2)\cot(\beta/2)$. The half angles have to be large for the cotangent to be small, but the sum of the two half angles is less than 90° in a proper triangle. Since $\cot(\alpha)\cot(\beta) < 1$ if $\alpha + \beta < \pi/2$, the conic is an ellipse.

We approach solving the system geometrically and render three elliptic cylinders in 3D, so obtaining the solutions using the graphics hardware. Because of the form of the equations, the three cylinder axes intersect perpendicularly in the $u_1\ u_2\ u_3$–space. The conic base curves are easily parameterized and extruded. Only the intersection in the positive octant is of interest. The three surfaces are rendered on a raster of size 1000 by 1000, using the depth buffer to extract the intersection in the positive octant. See also [13,6].

3.2. Algebra Vs. Geometry

The equation system for the classical Malfatti problem is well-structured. Each equation describes an elliptic cylinder in a principal direction, rotated by a certain angle owing to the $2u_iu_k$-term. The main advantage of the sampling approach using the GPU is that surfaces that are moderately complicated to tessellate can be intersected quickly because of the highly parallel nature of the problem and the large number of processors in the GPU. So, when the algebraic nature of the system is complex, the GPU approach can work very well, because it reduces the complexity of solving the system algebraically and ideally avoids sorting out which roots are of relevance to the problem; see also the problems in [5,6]. However, there are many papers on the Malfatti problem and the solution we seek can be described very simply [17]. That paper gives the original formulae of Malfatti for computing the radii of the three circles. The formulae are very well suited to computation and are simpler than solving the algebraic system given before.

Let I be the incenter of the triangle and r the incircle radius. With r_k the radius of the k^{th} Malfatti circle and $d(I,A)$ the distance between the incenter I to vertex A, and $s = (a+b+c)/2$, we have [17]:

$$r_1 = \frac{r}{2(s-a)}\Big(s - r - \big(d(I,B) + d(I,C) - d(I,A)\big)\Big)$$
$$r_2 = \frac{r}{2(s-b)}\Big(s - r - \big(d(I,C) + d(I,A) - d(I,B)\big)\Big)$$
$$r_3 = \frac{r}{2(s-c)}\Big(s - r - \big(d(I,A) + d(I,B) - d(I,C)\big)\Big)$$

The circle centers lie on the angle bisectors and can be computed using distance proportionality:

$$\frac{d(I,A)}{r} = \frac{d(A,O_1)}{r_1}$$

Here O_1 denotes the center of the circle C_1. Thus, so determining the Malfatti circles is easy. Note that this construction bypasses the algebra of the quadratic equations. As [17] points out, the radius formulae are due to Malfatti.

3.3. Inverse Construction

We investigate the inverse construction in which a triangle is fit to three given circles. Call the containing triangle in the Malfatti problem the *outer triangle* and the triangle spanned by the three centers of the inscribed circles the *central triangle*. Let the corners of the central triangle be C_k, k=1, 2, 3, in the usual enumeration, and let the r_k be the corresponding radii. The side lengths of the central triangle are

$$\begin{aligned} a' &= r_2 + r_3 \\ b' &= r_3 + r_1 \\ c' &= r_1 + r_2 \end{aligned}$$

Equivalently, we have:

$$\begin{aligned} 2r_1 &= b' + c' - a' \\ 2r_2 &= c' + a' - b' \\ 2r_3 &= a' + b' - c' \end{aligned}$$

Thus, the central triangle uniquely determines three circles that are pairwise tangent from the outside. Given the central triangle, we can construct the outer tangents, so defining the containing triangle. The angle φ between the common exterior tangent and the center line is given by

$$\sin(\varphi) = \frac{r_2 - r_1}{r_2 + r_1}$$

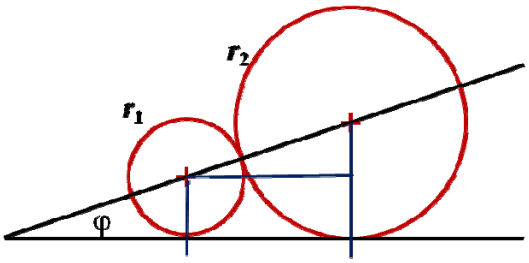

Figure 6: Angle between tangent and center line.

φ is positive when $r_2 > r_1$. See Figure 6.

To construct the exterior tangent from the radii partitioning the sides of the central triangle, erect the segments with lengths r_1 and r_2 at the angle 90−φ, tilting away from the larger radius vertex. The endpoints of those segments then define the outer triangle sides, Figure 7. The details are routine.

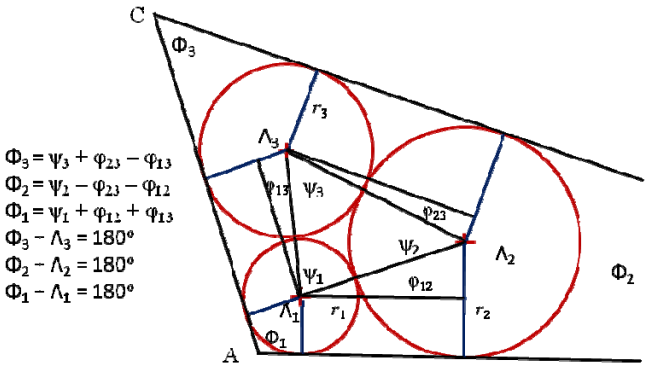

Figure 7: Constructing the triangle from the central triangle.

The implementation of the inverse three-circle construction is simple. The central triangle can be manipulated interactively and the problem solution, including rendering, requires less than 5 μsec.

3.4. Angle Locking

Recall that a solution for a similar triangle can be scaled trivially to the required size. If we fix one of the angles in the triangle, say γ, and wish to vary the other two angles, then *C* must be constrained to move on the perimeter of the circum circle while fixing the side *AB*. We use the direct solution of Subsection 3.2 here. See also Figure 8. We will show later how to use this observation for the six-circle problem.

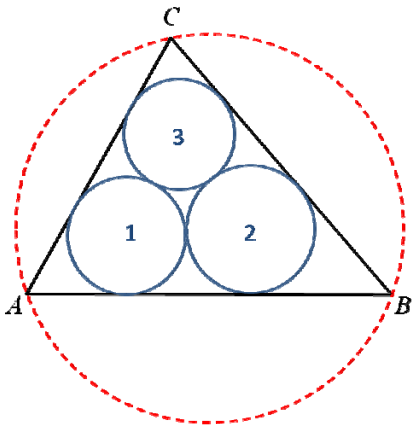

Figure 8: Angle γ remains unchanged while moving *C* on the perimeter of the red arc above line *AB*. Vertices *A* and *B* do not move.

4. Six Circles

The direct six-circle problem has a complex system of algebraic equations in six unknowns. Rather than tackle it head-on, we will develop an inverse construction. However, the isosceles case reduces to a system with only three unknowns that is tractable and can be implemented using the GPU. In this section, we enumerate the six circles as shown in Figure 9.

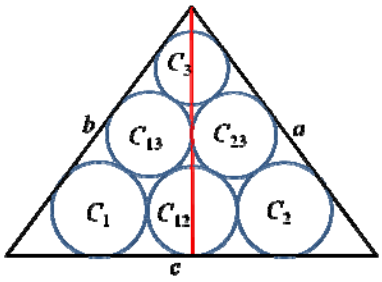

Figure 9: Naming conventions.

4.1. Six-Circle Isosceles Case, Direct Solution

Here, we exploit the symmetry of the triangle and leverage the solution of the three-circle problem. We proceed as follows:

1. Given the triangle $\Delta(A,B,C)$, solve the (3-circle) Malfatti problem. By symmetry, circles C_1 and C_2 have the same radius.
2. Scale the triangle such that the circles C_1 and C_2 have radius 1 and choose the coordinate system such that the circles are placed with their centers at (−1,0) and (+1,0), respectively. Rename C_1 to C_{13} and C_2 to C_{23}.
3. Solve for the three remaining circles, C_1 and C_{12}, extending the sides a and b of the triangle as needed.
4. Scale the resulting triangle to the size of the original triangle.

Let the angles of the triangle be α and γ and note that C_1 and C_2 are congruent and symmetrically placed.

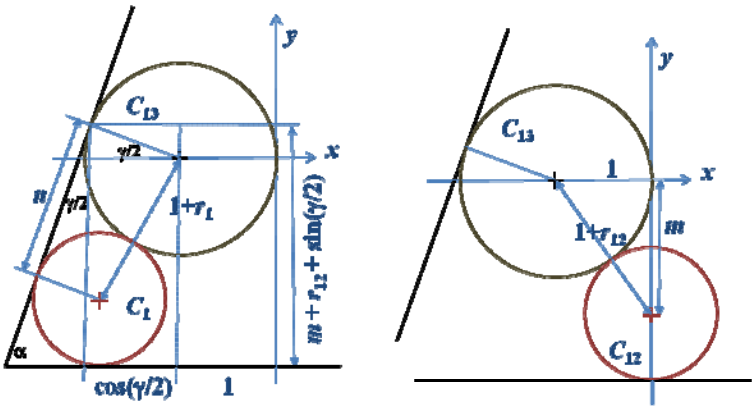

Figure 10: Isosceles case – construction of the lower three circles.

Two equations for the circle C_{12} express that the circle center is on the y-axis, at $(0,-m)$, and that it is tangent to circle C_{13}, and to a line parallel to the x-axis at distance $m+r_{12}$ below:

$$1 + m^2 = (r_{12} + 1)^2$$
$$-y = m + r_{12}$$

The equations for C_1 have to account for the fact that the sides are not parallel to the y-axis, so that the length of the scaled triangle side c is unknown. The quantity n is the distance between the contact

points of C_1 and C_{13} on (the extended) side b. Thus, the line parallel to the x-axis and tangent to C_1 from below is

$$-y = \left(n + r_1 \cot\left(\frac{\alpha}{2}\right)\right)\cos\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\gamma}{2}\right)$$

The length of the new side c is unknown. It is determined by

$$\frac{c}{2} = \left(m + r_{12} + \sin\left(\frac{\gamma}{2}\right)\right)\tan\left(\frac{\gamma}{2}\right) + \cos\left(\frac{\gamma}{2}\right) + 1$$

and the tangency between C_1 and C_{12} is stipulated by

$$r_1 \cot\left(\frac{\alpha}{2}\right) + 2\sqrt{r_1 r_{12}} = \frac{c}{2}$$

The final equation system is then

$$\begin{aligned} 1 + m^2 &= (r_{12} + 1)^2 \\ n &= 2\sqrt{r_1} \\ m + r_{12} &= \left(n + r_1 \cot\left(\frac{\alpha}{2}\right)\right)\cos\left(\frac{\gamma}{2}\right) - \sin\left(\frac{\gamma}{2}\right) \\ r_1 \cot\left(\frac{\alpha}{2}\right) + 2\sqrt{r_1 r_{12}} &= 1 + \cos\left(\frac{\gamma}{2}\right) + \left(n + r_1 \cot\left(\frac{\alpha}{2}\right)\right)\sin\left(\frac{\gamma}{2}\right) \end{aligned}$$

We eliminate n and obtain 3 equations in the variables $m,\ u_1 = \sqrt{r_1},\ u_{12} = \sqrt{r_{12}}$:

$$\begin{aligned} m^2 - 2u_{12}^2 - u_{12}^4 &= 0 \\ u_1^2 \cot\left(\frac{\alpha}{2}\right)\cos\left(\frac{\gamma}{2}\right) + 2u_1 \cos\left(\frac{\gamma}{2}\right) - u_{12}^2 - m - \sin\left(\frac{\gamma}{2}\right) &= 0 \\ u_1^2 \cot\left(\frac{\alpha}{2}\right)\left(1 - \sin\left(\frac{\gamma}{2}\right)\right) + 2u_1 u_{12} - 2u_1 \sin\left(\frac{\gamma}{2}\right) - 1 - \cos\left(\frac{\gamma}{2}\right) &= 0 \end{aligned}$$

The first and third equations denote cylinders and are easy to facet. The second surface is more complex and can be tessellated by building piecewise linear approximations to the section conics m=const and connecting successive layers. We have not implemented this case.

4.2. Six-Circle Inverse Constructions

We noted before that the inverse construction of the three-circle problem is trivial. In the six-circle case this is no longer true, but the simplicity of interactively constructing six circles tangent to each other in the required pattern simplifies our task considerably. We work with the *central net*, the triangle net spanned by the centers of the six circle; see Figure 11. The net is composed of three triangles, Δ_1, Δ_2, Δ_3, and the circle centers C_1, C_2, and C_3 are its *outer vertices*. Because of the linear relationship between side lengths and radii, the net can be easily manipulated interactively and the circles rendered.

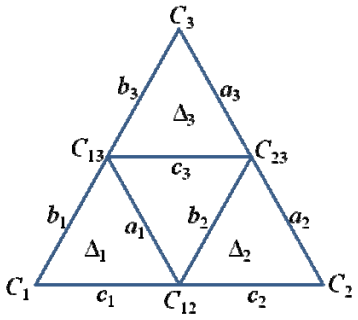

Figure 11: Central net; C_k and C_{ik} denote both circles and their centers.

We consider the operation of dragging the vertices C_k. Moving C_k alters the lengths a_k and b_k so the radii associated with the three vertices must be recomputed from the triangle sides. Two of the new radii are then used to adjust the other two triangles Δ_j, $j \neq k$, thereby maintaining tangency of the circles with each other. An example is shown in Figure 12. Note that the outer circles need not have a common tangent. In addition, we allow adjusting the radii individually as an interaction mode. By trial and error the radii could be so adjusted to achieve a triangle enclosure.

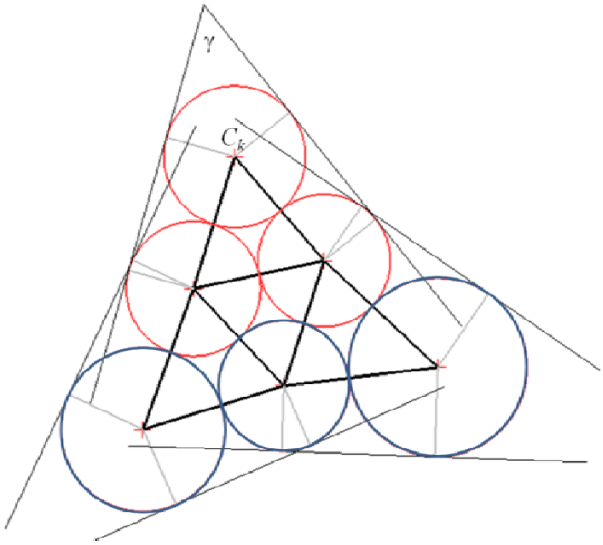

Figure 12: Inverse construction, unconstrained.

In these unconstrained operations, interactive manipulation does not determine an enclosing triangle directly. To accomplish that, when dragging vertex C_k, we will adjust the three circles that are only indirectly affected and are shown in blue in Figure 12. The idea is to vary the radii of those circles, while holding the red circles fixed, so that the angles between the corresponding external tangents become zero. Note that, by fixing the red circles, we have fixed the angle γ. The other two angles of the enclosing triangle depend on the outcome of the radius adjustments.

A brute-force iteration to obtain the enclosing triangle would be to sample the three blue radii and measure the angle discrepancies. For the three circles touching side c of the outer triangle the problem is illustrated by Figure 13. Note that the distances u and v are given by $u = \sqrt{r_1 r_{12}}$ and $v = \sqrt{r_{12} r_2}$. Moreover, for the tangents to match, the distance between U and V must be $d(U,V) = \sqrt{r_1 r_{12}} + \sqrt{r_{12} r_2}$.

Thus, for an interactive operation we can maintain an enclosure by sampling the blue radii to pixel accuracy and testing whether the tangents match. The condition can be put in terms of angles:

$$2\arctan\left(\sqrt{\frac{r_1}{r_{12}}}\right)+2\arctan\left(\sqrt{\frac{r_2}{r_{12}}}\right)=\delta$$

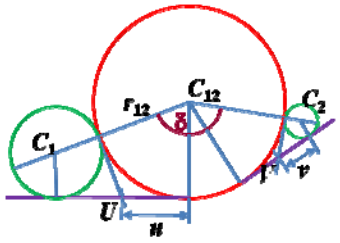

Figure 13: Common tangent condition.

where δ is the angle between the two lines connecting the circle centers. Alternatively, we could measure the distance $d(U,V)$ and require it to be equal to $\sqrt{r_1 r_{12}}+\sqrt{r_{12} r_2}$.

We reduce the search space utilizing the bisector between circle and tangent. In the following, assume that k=3, and consider adjusting the radii. By fixing the three red circles we fix two sides of the enclosing triangle we seek. If the circle C_1 is to be tangent to the red circle C_{13} and to the side b of the enclosing triangle, then its center must lie on the bisector, a parabola. The parametric representation of the parabola, see Figure 14, is given by

$$C' = K + t\,\vec{u} + \frac{t^2}{4r}\vec{v}, \quad |u| = |v| = 1$$

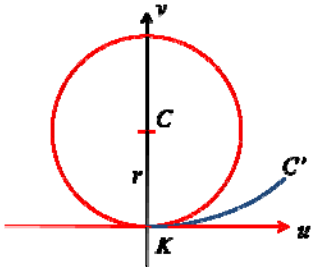

Figure 14: Bisector of circle, radius r, and its tangent.

We proceed as follows:

1. Assume that vertex C_3 is being dragged.
2. Vary the radius r_{12}. For each value of r_{12}, determine the radii r_1 and r_2 as follows.
3. Using the parametric representation of the bisector of C_{13} and the tangent to C_3, pick a center C'=C_1 and check whether the lengths of sides c_1 and b_1 of the central net are consistent with the radius r_1. If not, decrease t if $b_1 - r_{13} > c_1 - r_{12}$ and increase t if $b_1 - r_{13} < c_1 - r_{12}$. Determine r_2 in like manner.
4. Continue until the tangents to C_{12} match: if $2\arctan\left(\sqrt{r_1/r_{12}}\right)+2\arctan\left(\sqrt{r_2/r_{12}}\right)<\delta$, then decrease r_{12}, and if δ is smaller, increase r_{12}.

This iteration achieves excellent performance. Angle locking has also been implemented for C_1 and C_2.

4.3. Angle Locking

We wish to solve the six-circle Malfatti problem using the inverse construction. To do so, we need an interaction mode in which one of the angles can be locked to a prescribed value. We proceed as follows:

1. Interact in the constrained mode until the angle γ has the prescribed value. This can be done beginning with an equilateral configuration and moving vertex C_3 up or down until γ has the required value.
2. Lock the angle value and reposition C_3, thereby changing the angles α and β to the values prescribed in the original problem.
3. Scale the resulting configuration to the dimensions of the given triangle.

Note that this process is easily automated.

The angle lock is accomplished by adjusting the radii r_{13}, r_{23}, and r_3 as if dealing with the three-circle configuration. The angle-locking routine of Subsection 3.4 is used for this purpose on the top three circles. At any position of C_3 we adjust the remaining radii, r_1, r_{12}, r_2, using the iteration of Subsection 4.2 to obtain a triangle enclosure as already described. See also Figure 15.

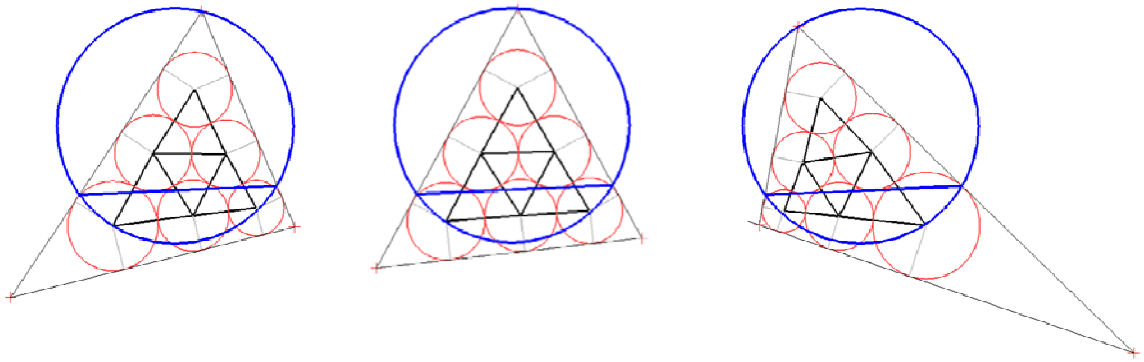

Figure 15: Angle locking for the top three circles. In this example $\gamma = 57^o$.

As illustrated, we construct the tangent to C_{13} and C_{23}, shown in blue. The intersection with the tangent to C_{13} and C_3, and with the tangent to C_{23} and C_3 defines two of the vertices of the triangle in which to solve the three-circle problem. The tangent to C_{13} and C_{23} defines a secant of the circum circle of the triangle, as shown. Vertex C_3 is then clamped to this circle. Vertex C_3 is repositioned on the arc. Using the direct solution of the three-circle problem, we determine the three radii r_3, r_{13}, r_{23}, from which the top triangle in the central net is constructed. Then, the inverse method for the six-circle configuration is used to complete the larger triangle.

When moving C_3, say, the angle variation of the six-circle enclosure, that is, of the angles α and β, is greater than it would be if we could clamp the base line c of the six-circle enclosure. To keep the length of side c of the larger triangle fixed, a scaling step has to be added. The details are routine.

5. Implementation and Results

The performance measurements quoted here were obtained on a PC running Windows Vista (32-bit) with the following configuration: Intel Xeon X5460 CPU at 3.16GHz, 4GB main memory, and an nVidia GeForce GTX 285 graphics card driving a display with 2560x1600 pixels. The program was implemented in C++ and was run in release mode alongside routine applications such as Outlook and Word. The performance is summarized in the table using frames-per-second (fps) as performance measure. Measurements are taken by averaging the actual fps over the period of about 1 second.

Classical Malfatti – Three Circles			
GPU method	direct	Section 3.1	225-230 fps
CPU only	direct	Section 3.2	2300-2400 fps
Angle lock	direct	Section 3.4	1150-1250 fps
Generalized Malfatti – Six Circles			
Isosceles case	direct	Section 4.1	200+ fps (est.)
Inverse constrained	inverse	Section 4.2	2300-2400 fps
Angle lock	hybrid	Section 4.3	1150-1250 fps

Summary of performance

The program has two windows. One window displays the triangle and the circles and allows the user to interact with these objects using the mouse. For the direct methods, the user may move the vertices of the enclosing triangle, and for the inverse method the user may move the vertices (O_1, O_2, O_3) of the central triangles. The second window renders the elliptic cylinders of Section 3.1 in the three-circle case. In the six-circle case, the second window displays controls allowing mode switching and displaying angle values. All performance measurements assume that the second window is minimized. If it is not, performance may drop.

For the three-circle problem, the classical Malfatti problem, using the GPU-rendered surfaces to solve the equations, we achieve a speed of 225-230 frames per second (fps). This means that, starting with the coordinates of the (enclosing) triangle vertices, the surface creation by tessellation, the extraction of the intersections, and the final display of the circles inside the triangle can all be done in less than five milli-seconds (ms) using a raster of size 1000x1000 for rendering the surfaces and finding their intersection. The interaction window is of comparable size. Performance increases an order of magnitude when using the classical formulae as discussed in Section 3.2, to 2300-2400 fps. Again, the second window is minimized in this case. Angle-locked manipulation also uses the formulae and achieves a frame rate of 1150-1250 fps when only the interaction window is open.

The six-circle problem does not involve the GPU and renders no surfaces. It uses the formulae of the three-circle case. We have not implemented the direct solution of the isosceles case. Estimating conservatively, we would expect a performance of 200 fps or better. We reason this estimate as follows: Two of the intersecting surfaces are cylinders and can be tessellated in time equivalent to the three circle case. The third surface is a quadric and its tessellation will require many more facets in order to achieve appropriate accuracy. In [6], we tessellated surfaces including conic sections rotated about a skew axis and achieved a frame rate of 180-200 fps when such surfaces were involved. Since the quadric surface is less complex, we believe that 200 fps is a conservative estimate for the time needed to do the GPU computation of the isosceles case.

The inverse (constrained) six-circle problem maintains the triangle enclosure iteratively and is very fast. The adjustment to the blue radii (circles C_3, C_{23} and C_2 when moving vertex A), is done by binary search and the frame rates achieved are comparable to the three-circle CPU method, indicating that the iteration is almost free. The speeds achieved for angle-locked manipulation, in the six-circle case, range between 1150 and 1250 fps. This speed is comparable to the three-circle case, again confirming that the time needed for the subsequent iteration adjusting the blue circle radii is negligible.

6. Discussion

We began our investigation with the intention to further explore the utility of GPU implementations in constraint solving, the subject of [5,6]. There, GPU implementations have simplified enormously the previously known algebraic methods for variable-radius circle constructions, both in sequential as well as in simultaneous settings. Key to that success has been the simplicity with which the graph of the Euclidean distance function of various shape elements can be constructed. More complicated, the problems of [6] require configuration space surfaces that have considerable algebraic complexity, yet their geometry permits simple algorithms for faceting them.

Our conclusion is that the utility of the GPU approach depends in large measure on the number and structure of the equations. Roughly speaking, the ideal case is characterized by three equations that define surfaces that are easy to tessellate and to render. Conic cylinders derived for the classical Malfatti problem are especially simple to facet since the tessellation is essentially two-dimensional. Another requirement, for an attractive GPU-based solution, is that the solutions of interest, represented by the intersection of the surfaces, are exposed from above or below, thus allowing their identification using the depth buffer hardware setting. This is indeed the case for the method of Subsection 3.1, since only the positive octant needs to be rendered. Moreover, considering the high-degree univariate polynomial proposed in [18], whose roots would be needed to solve the equations articulated there algebraically, we would strongly advocate the GPU approach over algebraic root finding. However, the geometric solutions in the literature, such as Malfatti's original formulae given in [17], have such simplicity that a GPU approach for the classical Malfatti problem is rather cumbersome. So, the classical, three-circle problem is not a good argument for a GPU-based implementation.

The six-circle generalization is another matter. Only for the isosceles case do we have a GPU-ready system of equations at this time. The general case has not yielded a comparable system so far. The value of our solution with the inverse method is that, in principle, the concept of a two stage solution in which

1) three circles are determined using the classical problem and
2) the configuration is completed by determining the remaining radii separately,

should permit an equation system for the second step that lends itself to a GPU-based implementation. Short of having found such a system, we chose an iteration instead to complete the solution. Even with GPU-ready equations, ours would still remain an inverse solution since the circles determine the triangle, not the other way around.

The angle-locking computation justifies our claim to have a complete solution of the six-circle problem. As the chosen corner is moved on the clamped arc, the other two angles of the enclosing triangle can be read out in real time, and since they monotonically depend on the direction of motion, we

can easily find the solution of the generalized problem after scaling. All this can be done very fast with our implementation.

As mentioned before, Stephenson's algorithm is more general than our work and can solve problems with hundreds of circles to be packed with prescribed topology, but not at the speeds we have been able to achieve. It remains to be seen whether Stephenson's algorithm can be sped up to comparable performance, in the case we have considered here, and whether a GPU implementation is appropriate.

Acknowledgements

This work has been supported in part by NSC Grants NSC 97-2212-E-031-002, NSC-98-2918-I-031-004, by NSF Grant CPATH CCF-0722210, by DOE award DE-FG52-06NA26290, and by a gift from Intel Corp.

References

1. W. Bouma, I. Fudos, C. Hoffmann, J. Cai, R. Paige. "A Geometric Constraint Solver," *CAD 27*, 1995, 487-501.
2. O. Bottema, "The Malfatti Problem," *Forum Geometricorum 1*, 43-50, 2000.
3. C.-S. Chiang and Robert Joan-Arinyo, "Revisiting Variable Radius Circles in Constructive Geometric Constraint Solving", Computer-Aided Geometric Design 21, pages 371-399, 2004/04.
4. C.-S. Chiang and C. Hoffmann, "Apollonius meets Pythagoras," Technical Report.
5. C.-S. Chiang, C. Hoffmann and Paul Rosen, "Hardware Assist for Constrained Circle Constructions I: Sequential Problems," *CAD&A 7(1),* 17−33, 2010.
6. C.-S. Chiang, C. Hoffmann, and P. Rosen, "Hardware Assist for Constrained Circle Constructions II: Cluster Merging Problems," *CAD&A 7(1)*, 33−44, 2010.
7. C. Collins and K. Stephenson, "A Circle Packing Algorithm," *Comp. Geometry: Thy and Appl. 25*, 2003, 233−256.
8. C. B. Durand. *Symbolic and numerical techniques for constraint solving*. PhD thesis, Computer Science Department, Purdue University, 1998
9. I. Fudos and C. M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.
10. X.-S. Gao, Q. Lin, and G.-F. Zhang. A c-tree decomposition algorithm for 2d and 3d geometric constraint solving,. *Computer-Aided Design*, 38(1):1–13, 2005.
11. C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems, Part I: Performance measures for CAD. *Journal of Symbolic Computation*, 31(4):367–408, 2001.
12. C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems, Part II: New algorithms. *Journal of Symbolic Computation*, 31(4):409–427, Apr. 2001.
13. K. Hoff III, T. Culver, J. Keyser, M. Lin, D. Manocha. "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware," Siggraph '99, Los Angeles, CA, 277-286.
14. C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraints systems: A survey. *International Journal of Computational Geometry and Applications*, 23(7):1–35, 2006.
15. H. Lamure and D. Michelucci. "Solving geometric constraints by homotopy." *Proc. Symp. on Solid Modeling and Applications*, ACM Press 1995: 263-269.
16. J. Owen. Algebraic solution for geometry from dimensional constraints. In *SMA'91: Proceedings of the first ACM symposium on Solid modeling Foundations and CAD/CAM applications*, 397–407, New York 1991. ACM, ACM Press.

17. M. Stefanović. “Triangel centers associated with the Malfatti circles.” *Forum Geometricorum 3*, 83-93, 2003.
18. Wolfram MathWorld. “Malfatti Circles,” http://mathworld.wolfram.com/MalfattiCircles.html