Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Note:

A note on circle packing*

Young Joon AHN[1], Christoph M. HOFFMANN[2], Paul ROSEN[3]

([1]*Department of Mathematics Education, Chosun University, Gwangju 501-759, Korea*)

([2]*Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA*)

([3]*Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT 84112, USA*)

E-mail: ahn@chosun.ac.kr; cmh@cs.purdue.edu; prosen@sci.utah.edu

Received Jan. 13, 2012; Revision accepted June 21, 2012; Crosschecked July 6, 2012

Abstract: The problem of packing circles into a domain of prescribed topology is considered. The circles need not have equal radii. The Collins-Stephenson algorithm computes such a circle packing. This algorithm is parallelized in two different ways and its performance is reported for a triangular, planar domain test case. The implementation uses the highly parallel graphics processing unit (GPU) on commodity hardware. The speedups so achieved are discussed based on a number of experiments.

Key words: Circle packing, Algorithm performance, Parallel computation, Graphics processing unit (GPU)
doi:10.1631/jzus.C1200010 **Document code:** A **CLC number:** TP391

1 Introduction

Circle packings, such as the one shown in Fig. 1, have been investigated in the mathematical literature not only for their geometric appeal, but also for their connection to the theory of analytic functions (Rodin and Sullivan, 1987; Willams, 2001; Stephenson, 2005). The problem may be conceptualized as follows. Given a graph K whose vertices and edges represent circles and external tangents respectively, the problem is to assign radii to the circles such that the implied configuration is realized (Bern and Eppstein, 2000). The problem can be considered in Euclidean and non-Euclidean geometries (Wang *et al.*, 2002). We refer the reader to Stephenson (2003) for an exposition of circle packing with a view towards mathematical applications.

Our interest in the matter originates from geometric constraint solving, e.g., Chiang *et al.* (2010). Briefly, constraints on circles can be expressed in terms of tangencies to other geometric entities, and circle packings are a specific instance of such constraint problems; see also Lamure and Michelucci (1995), Hoffmann and Joan-Arinyo (2002), Liu *et al.* (2009), and López and Beasley (2011). Another area in which circle packings play a role is computer graphics where a closely related circle arrangement pattern is associated with a given mesh surface in 3D. The circle pattern can then be used to define conformal maps, which is useful in devising quality texture maps for the mesh (Kharevych *et al.*, 2005).

A good sequential algorithm for circle packing has been reported by Collins and Stephenson (2003) and is the subject of our note. In Chiang *et al.* (2012), we have considered the special case of packing six circles into a triangle such that the circles are tangential to each other and to the sides of the triangle. On the one hand, this problem is related to Malfatti's problem (Bottema, 2000; Andreatta *et al.*, 2011); on the other hand, it is related to constraint solving, as explained in this paper. We gave a GPU-based algorithm that solves the generalized Malfatti problem in less than 1 ms. From this work, the question arose whether the general circle packing problem can be solved by a computation of comparable

* Project supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) by the Ministry of Education, Science and Technology (No. 2012-0002715), NSF Grants CPATH (Nos. CCF-0722210 and CCF-0938999), DOE award (No. DE-FG52-06NA26290), and by a gift from the Intel Corporation

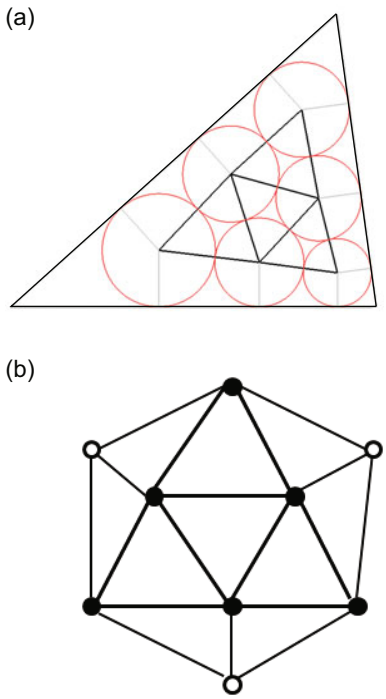

Fig. 1 Six circles packed into a triangular domain (a) and the graph K specifying tangencies (b). Boundary circles are represented by the three vertices with light interior

speed. For this question we report an affirmative answer. Specifically, we explain how we implemented the Collins-Stephenson algorithm efficiently, for the Euclidean case, and we develop a variant suitable for parallel computation. We then present the performance of the sequential algorithm and of the GPU-based parallel variant. We found that the parallel version is extremely efficient and capable of packing hundreds of circles in a few seconds with a double-digit speedup over the sequential version.

2 Notations and preliminaries

We use the notations of Collins and Stephenson (2003) but restrict them to circle packings in the Euclidean plane.

Given a triangulated planar graph K, we seek an arrangement of circles such that there is a circle C_v for every vertex $v \in K$ and such that two circles C_u and C_v are externally tangential to each other iff there is an edge $(u, v) \in K$. An assignment $R(K)$: $v \to (0, \infty)$ of positive numbers to the vertices of K is a solution of the packing problem if K can be embedded into the Euclidean plane such that the center of the circle C_v, represented by v, is at the position of v of the embedding; its radius is $R(v)$, and two circles are tangential iff there is a corresponding edge in K.

The flower of a vertex v is v and its star of adjacent vertices. Equivalently, it is the circle C_v represented by v and the circles tangential to it. Whether we mean the graph structure or the circle assembly will be clear from the context.

Let v, u, and w be three graph vertices forming a triangle, and consider the circles they represent, with radii r_v, r_u, and r_w. The angle $\psi(v; u, w)$ at v, of the triangle formed by the circle centers, can be determined as follows:

$$\psi(v; u, w) = 2 \arcsin \sqrt{\frac{r_u r_w}{(r_v + r_u)(r_v + r_w)}}. \quad (1)$$

The algorithm of Collins and Stephenson (2003) iterates over the nonboundary circle vertices of K, adjusting the radii based on the angle sums spanned by the triangles in the flower. At vertex v, consider the adjacent vertices $w_1, w_2, \cdots, w_k$ in cyclic order. The angle sum $\theta(v)$ is computed from the angles at v of the triangles $\triangle(v; w_j, w_{j+1})$ where $w_{k+1} = w_1$,

$$\theta(v) = \sum_{j=1}^{k} \psi(v; w_j, w_{j+1}).$$

Then, a radius $\hat{r}$ is determined such that, if all circles tangential to C_v were of that radius, the same angle sum would be obtained at v. Finally, the radius of the circle at v is adjusted from $\hat{r}$ and the desired angle sum, $A(v)$. If the petals of the flower wrap once around C_v, then $A(v) = 2\pi$ is the appropriate angle sum target.

3 Main idea for packing circles into a triangle

We consider several different packing algorithm implementations and will illustrate their performance using a triangle enclosure. Given any triangle and number n, the object is to pack $N = n(n+1)/2$ circles into the triangle. See Fig. 1a for the case $n = 3$. For $n = 2$ we obtain the Malfatti problem, and for $n = 1$ the incircle of the triangle.

We adapt the method of Collins and Stephenson (2003) to solve the problem of packing circles into a triangle. The main deviation of our method from that of Collins and Stephenson (2003), is that each side of triangle is considered as a circle of infinite

radius. An example is given in Fig. 1b. The edge between center v and a side of the triangle domain is obtained by the orthogonal projection of v onto the triangle side. For the graph vertices v, u, and w, if one of u and w is of infinite radius, then the angle is computed using Eq. (1). If u or w is a vertex of infinite radius, then the angle is determined by

$$\psi(v;u,w') = \frac{\pi}{2} - \arcsin\frac{r_v - r_u}{r_v + r_u}, \tag{2}$$

as shown in Fig. 2, and

$$\psi(v;u',w) = \psi(v;w,u'), \tag{3}$$

where w' and u' indicate that the respective circles have infinite radii; i.e., they are straight lines. Finally, if both u and w are the vertices of infinite radius, then

$$\psi(v;u',w') = \pi - \alpha, \tag{4}$$

where α is the angle enclosed by the lines u' and w'.

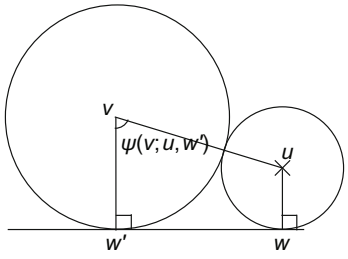

Fig. 2 The angle $\psi(v;u,w')$ between two graph edges (v,u) and (v,w') when one circle has infinite radius

4 Algorithms

4.1 Sequential algorithm

In this subsection we consider the algorithm presented by Collins and Stephenson (2003). As previewed in Section 3, it iterates over all the circle vertices of K, adjusting the radius of each circle so that it is tangential to its flower circles. Hence, at each vertex v, using uniform radii of all petal circles, $\hat{r}$, the adjusted radius r'_v of the circle at v is obtained. This iteration is performed until the error e is less than the error bound ϵ.

Input: triangle $\triangle ABC$, integer n, error bound ϵ.

Output: radii r_i and centers c_i, $i = 1, 2, \cdots, N$.

1. $N := n(n+1)/2$;
2. Initial radii $r := r_i^{(0)}$, $i = 1, 2, \cdots, N$;
3. For each interior vertex v, $i = 1, 2, \cdots, N$
   Compute the angle sum $\theta(v)$;
   $b := \sin(\theta(v)/(2k))$;
   $\hat{r} := r_v b/(1-b)$;
   $d := \sin(A(v)/(2k))$;
   $r'_v := \hat{r}(1-d)/d$;
   End For
4. Calculate error $e := \max_v |\theta(v) - 2\pi|$;
5. If $e > \epsilon$, goto 3.

As Collins and Stephenson (2003) showed, the iteration converges and is insensitive to the order in which the interior vertices are processed. The algorithm is straightforward to implement and stable. Note that a solution can be scaled uniformly, regardless of the domain into which the circles are packed. Hence, packing circles into a triangle, as we do in this work, entails a final scaling step to accomplish the actual fit.

4.2 First parallel version

To parallelize the Collins-Stephenson (CS) algorithm, observe that the radius update of two circles that are not tangential is independent. Thus, we can partition the set of interior vertices of K into subsets where each subset consists of vertices whose associated circles are not tangential. That is, each subset contains vertices no two of which are adjacent in K. Finding such a partition is the well-studied graph vertex coloring problem. Since we consider only planar graphs K, at most four colors suffice. Thus, we can partition the vertices of K into at most four subsets, each composed of nodes that can be processed independently.

In the case of packing $N = n(n+1)/2$ circles into a triangle, only three colors are needed (Fig. 3). Thus, up to a third of the interior circles can be processed concurrently, in three consecutive phases, hence accomplishing the work of one iteration over all circles.

4.3 Second parallel version

We can also restructure the algorithm to first compute the angle sum in parallel for each interior circle and then compute $\hat{r}$ and the new radius. This simplifies the concurrent computation. However, we find that this method requires more iterations to reach the same error bound as the first parallel

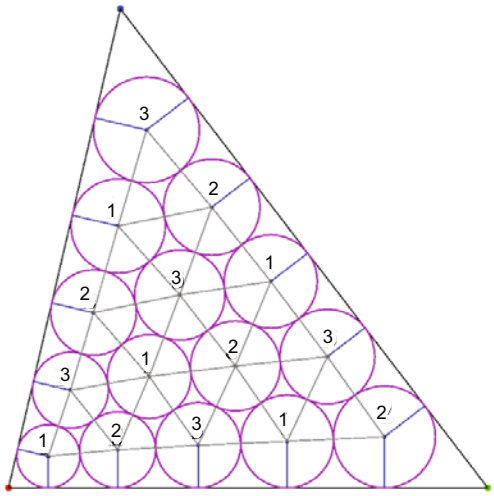

Fig. 3 Circle partition into three subsets such that the circles in each subset can be processed independently and concurrently

version. In the case of packing circles into a given triangle, this second version requires almost twice the number of iterations to reach the same accuracy.

5 Interactive updates

To test the overall performance of the algorithms, we consider packing a fixed number of circles into an enclosing domain that is changed by user interaction. In the case of a triangular domain, for instance, the user may drag a triangle vertex with the mouse and expect the circles to change as needed to fit. This can be done by running, for each mouse position, as many iterations as needed to fit into the new enclosure. Since the changes are incremental, fewer iterations will be necessary each time the display is refreshed due to a changed mouse position.

Now the display refresh takes a certain amount of time that comprises the circle packing and the frame buffer redrawing. While the latter is roughly constant, the former varies with the number of iterations needed to reach the required accuracy. Accordingly, a good performance measure is the number of frames per second (frames/s) achieved for a fixed accuracy, as discussed further below. If more than 20 frames are achieved per second, the experience is a smooth interactive one. Note that 20 frames/s means a running time of at most 0.05 s for the screen update. An update rate of 10 frames/s (0.1 s running time) is acceptable but the update delays are quite noticeable.

6 Performance

The Collins-Stephenson algorithm has been implemented in both single and double precision. The basic performance has been measured empirically for the triangle containment configuration, for a variety of numbers of circles and error specifications. The platform is a PC running Windows Vista (32-bit) with the following configuration: Intel® Xeon® X5460 CPU at 3.16 GHz, 4 GB main memory, and an nVidia® GeForce® GTX 285 graphics card driving a display with 2560×1600 pixels. The program was implemented in C++ and was run in release mode.

6.1 Sequential complexity

The first set of experiments determines empirically the performance of the sequential implementation of the Collins-Stephenson algorithm (Table 1). The test configuration is the triangle packing illustrated in Fig. 3. The triangle vertices are situated at $(-0.75, -0.75)$, $(0.75, -0.75)$, and $(-0.4, 0.75)$. The number of circles packed is N, and n shows the number of circles tangential to each side of the triangle. Results are shown for two error bounds, $\epsilon = 10^{-5}$

Table 1 Performance of the sequential implementation with two error bounds using double-precision floating-point numbers

n	N	Total number of iterations		Number of iterations per cycle		Time (s)	
		$\epsilon = 10^{-5}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-8}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-8}$
5	15	50	79	3.3	5.3	0.0012	0.0017
10	55	166	262	3.0	4.8	0.0056	0.0087
25	325	931	1465	2.9	4.5	0.15	0.23
50	1275	3587	5638	2.8	4.4	2.18	3.43
75	2850	7968	12 522	2.8	4.4	10.68	16.80
100	5050	14 076	22 116	2.8	4.4	32.33	52.42

n: number of circles tangential to each side of the triangle; N: number of circles packed; ϵ: error bound

and $\epsilon = 10^{-8}$. To achieve such accuracy for moderately large graphs K, the implementation has to use double precision.

Empirically, after fixing the accuracy to be achieved, the algorithm complexity is quadratic in the number of circles packed: for larger N, the algorithm complexity, measured by the number of iterations, is linear in the number of circles packed, requiring close to 3 iterations per circle for $\epsilon = 10^{-5}$ and 4.4 iterations per circle for the tighter error bound. Each iteration is linear in the number of circles, since each circle radius is adjusted once in each iteration and each circle has at most six adjacent circles. Moreover, the observed running time, divided by N, grows linearly with N.

6.2 Parallel speedup

We implemented the parallel versions of the algorithm on the GPU. Since our GPU can compute only single-precision floating-point numbers, a fair comparison requires timing a single-precision sequential implementation to establish a reference speed. Accordingly, we have measured the performance of a single-precision, sequential implementation using an error of 10^{-3} and compared it to the simple GPU implementation where all angle sums are computed in an iteration before radius adjustments are calculated (Parallel 2 version). We have also implemented the Parallel 1 version in which three or four independent subsets of circles are processed in parallel.

The results are shown in Table 2. Even though the required number of iterations for the Parallel 2 version is roughly twice the needed number of iterations for the sequential implementation, the GPU performance excels for larger problem sizes. In fact, for $n = 100$ ($N = 5050$), using an error bound of 10^{-2}, the parallel GPU implementation is faster by a factor of almost 45 (Parallel 1). The break-even problem size is $n = 91$. The timing of the parallel computations shows separately the time spent by the GPU, as kernel time, in addition to the overall time which includes the CPU-GPU communication.

How large a problem can be solved with an acceptable interactive performance? We can estimate the number of frames per second by linear interpolation of the total running time in Table 2. For the accuracy of 10^{-3} and 20 frames/s the sequential implementation can pack about 220 circles, whereas the parallel versions 1 and 2 can pack about 280 and 430 circles, respectively. For 10 frames/s, with the same accuracy, the number of circles packed is approximately 300 for the sequential version, 1060 for Parallel 1, and 985 for Parallel 2.

7 Summary and conclusions

In this note we present an algorithm for packing circles into a triangle. Our algorithm is based on the Collins-Stephenson algorithm (Collins and Stephenson, 2003) and it treats the sides of the triangle as circles of infinite radius. We also develop two suitable algorithms for parallel computation and measure their performance. Implementing the parallel versions on the GPU results in significant speedup over the sequential version.

Although Collins and Stephenson (2003) asserted that the sequence of radius updates in an iteration makes little difference, we find that posting

Table 2 Performance of the sequential implementation using single-precision floating-point numbers and comparison with two parallel implementations

n	N	Number of iterations*	Time* (s)	Kernel time† (s)	Total time† (s)	Speedup†	Number of iterations◇	Kernel time◇ (s)	Total time◇ (s)	Speedup◇
20	210	374	0.043	0.043	0.046	0.9	660	0.022	0.025	1.7
30	465	817	0.204	0.056	0.060	3.4	1455	0.050	0.054	3.8
40	820	1437	0.621	0.067	0.070	8.9	2571	0.084	0.089	7.0
50	1275	2237	1.490	0.122	0.127	11.7	4052	0.128	0.134	11.1
60	1830	3323	3.149	0.159	0.164	19.2	6055	0.195	0.200	15.7
70	2485	4844	6.204	0.258	0.268	23.1	9977	0.324	0.330	18.8
100	5050	6037	15.93		0.363	43.9	10 409		0.355	44.9
200	20 100	26 335	279.43		4.855	57.6	46 900		3.622	77.1

An error bound of 10^{-2} is used when $n = 100$ and 200, and 10^{-3} otherwise. Note that for the Parallel 2 version a larger number of parallel iterations is needed to achieve the same tolerance. * Sequential implementation; † Parallel 1; ◇ Parallel 2

the radius updates throughout the course of each iteration significantly reduces the number of required iterations. Nonetheless, when the number of circles to be packed is sufficiently large, a parallel implementation is much faster, even when the radius adjustments are not posted and there are almost twice as many iterations.

Table 2 seems to suggest that the Parallel 1 version is faster than the Parallel 2 version. However, for $n = 100$, packing $N = 5050$ circles, the speedups are nearly identical, 44.9 vs. 44.2 times. We do not have a good explanation for that.

References

Andreatta, M., Bezdek, A., Boroński, J.P., 2011. The problem of Malfatti: two centuries of debate. *Math. Intell.*, **33**(1):72-76. [doi:10.1007/s00283-010-9154-7]

Bern, M., Eppstein, D., 2000. Quadrilateral meshing by circle packing. *Int. J. Comput. Geom. Appl.*, **10**(4):347-360. [doi:10.1142/S0218195900000206]

Bottema, O., 2000. The Malfatti problem. *Forum Geom.*, **1**:43-50.

Chiang, C.S., Hoffmann, C.M., Rosen, P., 2010. Hardware assistance for constrained circle constructions I: sequential problems. *Comput. Aid. Des. Appl.*, **7**(1):17-32. [doi:10.3722/cadaps.2010.17-32]

Chiang, C.S., Hoffmann, C.M., Rosen, P., 2012. A generalized Malfatti problem. *Comput. Geom.*, **45**(8):425-435. [doi:10.1016/j.comgeo.2010.06.005]

Collins, C.R., Stephenson, K., 2003. A circle packing algorithm. *Comput. Geom.*, **25**(3):233-256. [doi:10.1016/S0925-7721(02)00099-8]

Hoffmann, C.M., Joan-Arinyo, R., 2002. Parametric Modeling. *In*: Farin, G., Hoschek, J., Kim, M.S. (Eds.), Handbook of Computer Aided Geometric Design. Elsevier North Holland, Amsterdam.

Kharevych, L., Springborn, B., Schröder, P., 2005. Discrete Conformal Mappings via Circle Patterns. ACM SIGGRAPH Courses, p.6. [doi:10.1145/1198555.1198665]

Lamure, H., Michelucci, D., 1995. Solving Geometric Constraints by Homotopy. Proc. Symp. on Solid Modeling and Applications, p.263-269.

Liu, J., Xue, S., Liu, Z., Xu, D., 2009. An improved energy landscape paving algorithm for the problem of packing circles into a larger containing circle. *Comput. Ind. Eng.*, **57**(3):1144-1149. [doi:10.1016/j.cie.2009.05.010]

López, C.O., Beasley, J.E., 2011. A heuristic for the circle packing problem with a variety of containers. *Eur. J. Oper. Res.*, **214**(3):512-525. [doi:10.1016/j.ejor.2011.04.024]

Rodin, B., Sullivan, D., 1987. The convergence of circle packings to the Riemann mapping. *J. Differ. Geom.*, **26**:349-360.

Stephenson, K., 2003. Circle packing: a mathematical tale. *Notices Amer. Math. Soc.*, **50**(11):1376-1388.

Stephenson, K., 2005. Introduction to Circle Packing: the Theory of Discrete Analytic Functions. Cambridge University Press, New York.

Wang, H., Huang, W., Zhang, Q., Xu, D., 2002. An improved algorithm for the packing of unequal circles within a larger containing circle. *Eur. J. Oper. Res.*, **141**(2):440-453. [doi:10.1016/S0377-2217(01)00241-7]

Willams, G.B., 2001. Approximation of quasisymmetries using circle packings. *Discr. Comput. Geom.*, **25**:103-124.