Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 2381–2399
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Number-theoretic interpretation and construction of a digital circle

Partha Bhowmick[a,*], Bhargab B. Bhattacharya[b]

[a] *Computer Science and Technology Department, Bengal Engineering and Science University, Shibpur, Howrah, India*
[b] *Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India*

Received 29 March 2006; received in revised form 15 October 2007; accepted 23 October 2007
Available online 26 December 2007

Abstract

This paper presents a new interpretation of a digital circle in terms of the distribution of square numbers in discrete intervals. The number-theoretic analysis that leads to many important properties of a digital circle succinctly captures the original perspectives of digital calculus and digital geometry for its visualization and characterization. To demonstrate the capability and efficacy of the proposed method, two simple algorithms for the construction of digital circles, based on simple number-theoretic concepts, have been reported. Both the algorithms require only a few primitive operations and are completely devoid of any floating-point computation. To speed up the computation, especially for circular arcs of high radii, a hybridized version of these two algorithms has been given. Experimental results have been furnished to elucidate the analytical power and algorithmic efficiency of the proposed approach. It has been also shown, how and why, for sufficiently high radius, the number-theoretic technique can expedite a circle construction algorithm.

Keywords: Digital circle; Digital geometry; Number theory; Bresenham's algorithm; Computer graphics

1. Introduction

Characterization and construction of the simple yet prevalent geometric primitives, especially straight lines and circles, in the discrete domain, hail long back from their successful realization in graphics display [1,20]. Today, with the evolution of new in-theory digital paradigms, such as digital calculus [36], digital geometry [28], theory of words and numbers in the digital perspective [29,34], etc., the analytical studies and theoretical interpretations of these geometric primitives have become inevitable in order to augment different inter-theoretical relations and applications, and to actuate and observe the comeback of classical theories into the formation and augmentation of emerging theories.

Digital circles, abiding with their weird and challenging nature in the discrete domain, have drawn immense research interest since the early adoption of the scan-conversion technique [2,6,12,14,17,24,30,38,42] for their efficient approximation from the continuous domain to the digital domain. Subsequent improvements of these algorithms meant for generation of circular arcs were achieved by different researchers in the later periods, which may be seen in [5,7,26,33,43,46–48]. Further, apart from the circle-generation algorithms, since the properties, parameterization, characterization, and recognition of digital circles and circular arcs shape up a very engrossing area of research, several other interesting works on digital circles and related problems have also appeared from time to time, some of which are as follows:

* Corresponding author. Tel.: +91 33 26686151.
E-mail addresses: partha@cs.becs.ac.in, bhowmick@gmail.com (P. Bhowmick), bhargab@isical.ac.in (B.B. Bhattacharya).

0166-218X/$ - see front matter
doi:10.1016/j.dam.2007.10.022

Table 1
A comparative study of some existing approaches with the proposed one

Algorithm	Features	Scan conversion[a]	Readiness for hybridization	Interval searching[b]
1. Incremental algorithm [6]	Decision function based on minimum-residual criterion	Yes	No	No
2. Optimized midpoint algorithm ([20], *DCB*: Fig. 7)	First- and second-order differences to expedite the computation of decision function	Yes	No	No
3. Short run algorithm [26]	Selection of minimum-distance pixels	No	No	No
4. Hybrid run length slice algorithm [48][c]	(i) uses run length properties	No[e]	Yes	No
	(ii) uses decision variable (d)[d]			
5. Number-theoretic algorithm [proposed]	(i) uses run length properties	No	Yes	Yes
	(ii) run lengths are binary searched in integer intervals using the predicted upper and lower bounds of runs			

[a] pixels are generated one in each iteration in accordance with the decision function derived from some chosen criterion.
[b] instead of pixels, runs are generated by searching the largest square number in an integer interval.
[c] an improvement and hybridization of the algorithm by [26].
[d] computation of d is aided by double-step and quadruple-step forward differencing for non-singular runs.
[e] scans the "runs of pixels" instead of scanning the "pixels of runs".

– *polygonal approximation of digital circles*: [3,25];
– *characterization of digital circles*: [4,18,22,35,45];
– *detection/segmentation of circular arcs/objects in digital images*: [9–11,13,15,23,27,31,36,39,40];
– *parameterization of circular arcs*: [8,16,44,49];
– *anti-aliasing solutions for digital circles*: [19]; etc.

It may be mentioned that there exist various composite techniques [5,26,46–48], designed to expedite and contend the procedure of Bresenham's Circle Drawing algorithm [6]. However, these algorithms do not have considerably large gains over Bresenham's method. In fact, Bresenham's algorithm cannot be overtaken by a great margin by some other algorithm, which has been verified by the other algorithms developed in later years. For instance, for radii from 1 to 128, the most efficient ones (please see, for example, the algorithms and results by Hsu et al. [26], and by Yao and Roken [48]) of these algorithms are 0.62 times to 1.53 times faster on the average with respect to Bresenham's algorithm. This is quite expected for the inherent brevity and simplicity of Bresenham's algorithm owing to the judicious usage of primitive arithmetic operations, which have been worked out sensibly and skillfully from the fundamentals of digital calculus.

However, the essence of all these algorithms is that, although a circle drawing algorithm primarily originates from the naive concept of intelligent digital applications of first-order and second-order derivatives (differences) as in Bresenham's, a digital circle is endowed with some other interesting properties contributed by the classical theories in the discrete domain. It has been gradually made apparent from these works that digital circles, similar to digital straight lines [29], possess some striking characteristics, which, if interpreted rightly and exploited properly, may produce interesting results and scope for subsequent potential applications in the discrete domain.

This paper reveals the relation between perfect squares (square numbers) in discrete (integer) intervals and few interesting and useful properties of a digital circle. Based on these number-theoretic properties, the problem of constructing a digital circle or a circular arc maps to the new domain of number theory. However, the construction of a digital circle using these properties should not be treated only in terms of its performance compared to Bresenham's algorithm or any other similar algorithm; rather, these number-theoretic properties enrich the understanding of digital circles from a perspective that is different from the customary aspects of digital calculus. It is also discussed in this paper how these intervals can be obtained for the given radius of a circle, and what effects these intervals have on the construction of a digital circle. For a brief overview, in Table 1, we have presented a comparison of our method with some existing methods.

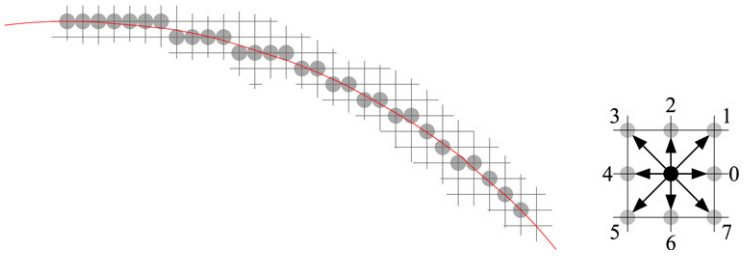


Fig. 1. The digital circular arc $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, 41)$ for a real circle (shown in red) of radius 41. The chain codes of the eight neighbors of the grid point (i, j) are shown aside, with the convention of the right point $(i, j + 1)$ as '0', followed by the other points in counterclockwise direction. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

An instance of the digital circle for radius 41 is shown in Fig. 1. It may be seen that the corresponding *chain code* [21] for this part of the circle, starting from the point (0, 41), is given by $0^6 7 0^3 7 0^3 7 0 7 0 7 0 7^3 0 7^3$, whence it is easy to observe that the runs of 0 "usually" decrease or continue to be same in length in the first octant as long as 7 appears as a singular character in the chain code. Alternatively, as shown in this paper, the topmost run length (no. of grid points with ordinate $= 41$) is given by the number of perfect squares in the (closed) interval $[0, 40]$, the next run length (no. of grid points with ordinate $= 40$) given by that in $[41, 120]$, the next one (ordinate $= 39$) by that in $[121, 198]$, and so on. To elaborate, if $s[v, w]$ denotes the number of perfect squares in the closed interval $[v, w]$, then we get $s[0, 40] = 7$ (since 7 perfect squares, i.e., $0, 1, 4, \ldots, 36$, lie in $[0, 40]$), $s[41, 120] = 4$ (since 49, 64, 81, 100 lie in $[41, 120]$), $s[121, 198] = 4$ (since 121, 144, 169, 196 lie in $[121, 198]$), etc. Hence, the *square numeric code* of the above circle in the first octant is given by $\langle 7, 4, 4, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1\rangle$, which can be shortened to $\langle 7, 4^2, 2^4, 1^2, 2, 1^3\rangle$, whence it is again apparent that the numbers of perfect squares are either non-increasing (predominant) or increasing by at most unity (occasional).

The brief outline of the paper is as follows. In Section 2, we start with some fundamental properties of a digital circle, followed by the basic principle for its generation in Section 2.1. In Section 2.2, we have explored few properties on the distribution of grid points for a digital circle based on the square numbers in discrete intervals, which may be used to design an algorithm (*DCS*) on generation of the circle using few primitive integer operations only, given in Section 2.3, along with its analysis in Section 2.4 and comparison with Bresenham's algorithm (*DCB*) in Section 2.5. Some other excellent properties have been presented in Section 3, which have been derived using the rudimentary number-theoretic concepts once again. These properties improve the former algorithm (*DCS*) to actualize another algorithm (*DCR*), given in Section 3.1, using a modified binary search technique and without using any floating-point operation at any stage, which has been shown to be very effective for circles of higher radii, as evident from its analysis given in Section 3.2. In Section 3.3, we have given a hybridized version (*DCH*) to ensure the optimization of computations in the adopted scheme. Section 4 exhibits some test results that demonstrate the analytical strength and algorithmic applications of the novel approach proposed in this paper. In Section 5, procedures for construction, segmentation, and characterization of digital circles/circular arcs by judicious and intelligent deployment of such number-theoretic properties, have been discussed.

2. Properties of digital circles

Let $\mathcal{C}^{\mathbb{Z}}(p, r)$ be the digital circle with center at $p \in \mathbb{Z}^2$ and radius $r \in \mathbb{Z}^+$. For simplicity of notations, $\mathcal{C}^{\mathbb{Z}}(p, r)$ is also used to denote the set of grid points (pixels) constituting the digital circle with center at p and radius r in an appropriate context. Further, if $q \in \mathbb{Z}^2$ be such that $q \in$ (set) $\mathcal{C}^{\mathbb{Z}}(p, r)$, then q is said to be "lying on" (circle) $\mathcal{C}^{\mathbb{Z}}(p, r)$.

To start with, we observe a simple and very useful representation of a digital circle, which is required for construction of the digital circle about any grid point p as its center. The most convenient and customary representation of any digital curve (and a digital circle, thereof) is by means of Freeman's chain code [21], where the locally represented enumeration of the digital curve is mapped to the global coordinate system using a defined (global) point of reference, customized to specific requirements. For the digital circle $\mathcal{C}^{\mathbb{Z}}(p, r)$, if we consider the center p as the origin (point of reference) of the local coordinate system in $\mathbb{Z}^2$, then it can be shown that the set of grid points, enumerated in chain coded form w.r.t. p, representing the circle $\mathcal{C}^{\mathbb{Z}}(p, r)$, will be always independent of p and will be

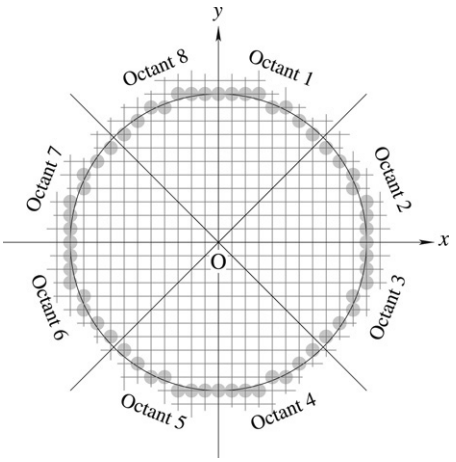


Fig. 2. A real circle, $\mathcal{C}^{\mathbb{R}}(O, 11)$, and the eight octants of the corresponding digital circle, $\mathcal{C}^{\mathbb{Z}}(O, 11)$.

depending only on its radius r. Hence, we can draw a digital circle $\mathcal{C}^{\mathbb{Z}}(p, r)$ centered at any point $p \in \mathbb{Z}^2$, provided $\mathcal{C}^{\mathbb{Z}}(O, r)$ is known, where, $O = (0, 0)$.

It may be noted that, the conversion of a digital circle $\mathcal{C}^{\mathbb{Z}}(O, r)$ from the corresponding real circle $\mathcal{C}^{\mathbb{R}}(O, r)$ is done using the property of 8-axes symmetry of digital circles (see [6,20]). A sample digital circle $\mathcal{C}^{\mathbb{Z}}(O, 11)$ along with its eight octants, and the corresponding real circle $\mathcal{C}^{\mathbb{R}}(O, 11)$ are shown in Fig. 2 for a ready reference. In order to obtain the complete circle $\mathcal{C}^{\mathbb{Z}}(O, r)$, therefore, generation of the first octant, $\mathcal{C}^{\mathbb{Z},I}(O, r)$, (pixels or grid points in the closed interval $[0^0, 45^0]$, measured w.r.t. $+y$-axis in clockwise direction) of $\mathcal{C}^{\mathbb{Z}}(O, r)$ suffices.

2.1. Generation of a digital circle

While generating $\mathcal{C}^{\mathbb{Z},I}(O, r)$, decision is taken to select between east pixel ($E := (i + 1, j)$) or south-east pixel ($SE := (i + 1, j - 1)$), standing at the current pixel ($P := (i, j)$), depending on which one between E and SE is closer to the point of intersection of the next ordinate line (i.e., $x = i + 1$) with the real circle $\mathcal{C}^{\mathbb{R}}(O, r)$. In the case of ties, any one between E and SE may be selected. It is, however, interesting to note that such a tie is possible only if there is any computation error, the proof being as follows.

Suppose a tie occurs when the real circle $\mathcal{C}^{\mathbb{R}}(O, r)$ has $(i, j + 1/2)$ as the corresponding point of intersection in the first octant with the vertical grid line $x = i$. Since $(i, j + 1/2)$ lies on $\mathcal{C}^{\mathbb{R}}(O, r)$, we have

$$i^2 + (j + 1/2)^2 = r^2,$$
$$\text{or,} \quad r^2 - (i^2 + j^2 + j) = 1/4,$$

which is impossible, since $r^2 \in \mathbb{Z}$, and $(i^2 + j^2 + j) \in \mathbb{Z}$. Hence we observe the following lemma.

Lemma 2.1. *A tie for selecting one of the two candidate pixels can never occur.*

Remark. Ideally, a tie as described above is impossible. Due to errors in finite-precision floating-point computation, a tie may apparently occur. However, since most of the algorithms on circle construction work in the integer domain only, the question of a tie does not arise. We have used the "impossibility of a tie" in establishing some strict inequality in Section 2.2.

2.2. Relation of digital circles with square numbers

If $P(i, j)$ be a grid point that lies in $\mathcal{C}^{\mathbb{Z},I}(O, r)$, then the point of intersection Q of $\mathcal{C}^{\mathbb{R}}(O, r)$ with the vertical grid line $x = i$ should have distance less than $\frac{1}{2}$ from P, as shown in Fig. 3. Hence, if $(i, j - \delta)$ are the coordinates of Q, then we have $-\frac{1}{2} < \delta < \frac{1}{2}$, where it may be noted that $|\delta|$ can never be equal to $\frac{1}{2}$, as evident from Lemma 2.1.

Now, since $Q(i, j - \delta)$ lies on the real circle $\mathcal{C}^{\mathbb{R}}(O, r)$, we get

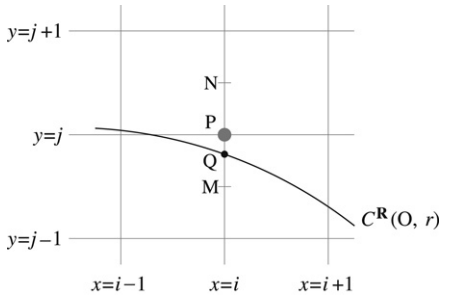


Fig. 3. P(i, j) lies on $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ provided the point of intersection Q of $\mathcal{C}^{\mathbb{R}}(\mathrm{O}, r)$ with the vertical line $x = i$ lies between M $(i, j - \frac{1}{2})$ and N $(i, j + \frac{1}{2})$.

$$
\begin{aligned}
& i^2 + (j - \delta)^2 = r^2, \\
& \text{or,} \quad \delta^2 - 2j\delta + (i^2 + j^2 - r^2) = 0, \\
& \text{or,} \quad \delta = \frac{1}{2}\left(2j \pm \sqrt{4j^2 - 4(i^2 + j^2 - r^2)}\right), \\
& \text{or,} \quad \delta = j - \sqrt{r^2 - i^2}, \quad \text{since } -\frac{1}{2} < \delta < \frac{1}{2}. \qquad (1)
\end{aligned}
$$

Therefore, extending Eq. (1) and using the fact that $-\frac{1}{2} < \delta < \frac{1}{2}$, we get

$$
\begin{aligned}
& -\frac{1}{2} < j - \sqrt{r^2 - i^2} < \frac{1}{2}, \\
& \text{or,} \quad -\frac{1}{2} - j < -\sqrt{r^2 - i^2} < \frac{1}{2} - j, \\
& \text{or,} \quad \frac{1}{2} + j > \sqrt{r^2 - i^2} > j - \frac{1}{2}, \\
& \text{or,} \quad \left(j - \frac{1}{2}\right)^2 < r^2 - i^2 < \left(j + \frac{1}{2}\right)^2, \\
& \text{or,} \quad j^2 - j + \frac{1}{4} < r^2 - i^2 < j^2 + j + \frac{1}{4}. \qquad (2)
\end{aligned}
$$

In the above equation, the terms $j^2 - j$, $r^2 - r$, and $j^2 + j$ are in $\mathbb{Z}$, as $i, j, r \in \mathbb{Z}$. Hence, $j^2 - j + \frac{1}{4} < r^2 - i^2$ implies $(j^2 - j) - (r^2 - i^2) < -\frac{1}{4} < 0$, or, $j^2 - j < r^2 - i^2$. Similarly, the expression $(r^2 - i^2) - (j^2 + j)$, being in $\mathbb{Z}$ and less than $\frac{1}{4}$, is never positive. Thus,

$$
\begin{aligned}
& j^2 - j < r^2 - i^2 \leqslant j^2 + j, \\
& \text{or,} \quad -j^2 - j \leqslant i^2 - r^2 < -j^2 + j, \\
& \text{or,} \quad r^2 - j^2 - j \leqslant i^2 < r^2 - j^2 + j. \qquad (3)
\end{aligned}
$$

Eq. (3) reveals the pattern of the grid points that would represent the digital circle $\mathcal{C}^{\mathbb{Z}}(\mathrm{O}, r)$ in the first octant. Since the first grid point in the first octant is always $(0, r)$ (considering the clockwise enumeration), Eq. (3) for the topmost grid points (i.e. $j = r$) becomes $0 \leqslant i^2 < r^2 - r^2 + r = r$, or, $0 \leqslant i^2 \leqslant r - 1$. This implies that, in the first octant, the grid points, having their ordinates as r, will have the squares of their abscissae in the (closed) interval $[0, r - 1]$. Let us denote the interval $[0, r - 1]$ by I_0 (zeroth interval).

Let I_1 be called the first interval, which is obtained by substituting $j = r - 1$ in Eq. (3). So, $I_1 = [r^2 - (r - 1)^2 - (r - 1), r^2 - (r - 1)^2 + (r - 1) - 1] = [r, 3r - 3]$, which contains the squares of abscissae of all the grid points (in the first octant) whose ordinates are $r - 1$. Thus, in general, if I_k denotes the kth $(k \geqslant 1)$ interval, given by

$$
I_k = \left[r^2 - (r - k)^2 - (r - k), r^2 - (r - k)^2 + (r - k) - 1\right]
$$

$$= [(2k-1)r - k(k-1), (2k+1)r - k(k+1) - 1], \tag{4}$$

which is obtained by substituting $j = r - k$ in Eq. (3), then, proceeding in this way, for a digital circle with radius $r \geqslant 1$ and center at $(0, 0)$, we get the following lemma.

Lemma 2.2. *The interval* $I_k = [(2k-1)r - k(k-1), (2k+1)r - k(k+1) - 1]$ *contains the squares of abscissae of the grid points of* $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ *whose ordinates are* $r - k$, *for* $k \geqslant 1$.

The length l_k of the interval I_k ($k \geqslant 1$) is, therefore, given by

$$\begin{aligned} l_k &= ((2k+1)r - k(k+1) - 1) - ((2k-1)r - k(k-1)) + 1 \\ &= 2r - 2k. \end{aligned} \tag{5}$$

Using Eq. (5), the length l_{k+1} for interval I_{k+1} is given by $l_{k+1} = 2r - 2(k+1) = l_k - 2$, whence we have the following lemma.

Lemma 2.3. *The lengths of the intervals containing the squares of equi-ordinate abscissae of the grid points in* $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ *decrease constantly by* 2, *starting from* I_1.

2.3. Algorithm DCS: Digital circle using square numbers

An algorithm for construction of digital circles can be designed, therefore, based on (numbers of) square numbers in the intervals

$$\begin{aligned} I_0 &= [0, r-1], \\ I_1 &= [r, 3r-3], \\ I_2 &= [3r-2, 5r-7], \ldots, \\ I_k &= [(2k-1)r - k(k-1), (2k+1)r - k(k+1) - 1], \ldots, \end{aligned}$$

as shown in Section 2.2. The length of the first interval I_1 is greater than that of I_0 by $r - 2$, as given by Eq. (5). More interestingly, for $k \geqslant 1$, the length of each interval I_{k+1} is less than that of I_k by 2, which is a constant. Hence, in the algorithm *DCS*, using square numbers, we search for the number of perfect squares in each interval $I_k, k \geqslant 0$, which, in turn, gives the number of grid points with ordinate $r - k$. The following theorem contains the above facts and findings in a concise way.

Theorem 2.4. *The squares of abscissae of grid points, lying on* $\mathcal{C}^{\mathbb{Z},\mathbb{I}}(\mathrm{O}, r)$ *and having ordinate* $r-k$, *lie in the interval* $[u_k, v_k := u_k + l_k - 1]$, *where* u_k *and* l_k *are given as follows.*

$$u_k = \begin{cases} u_{k-1} + l_{k-1} & \text{if } k \geqslant 1 \\ 0 & \text{if } k = 0 \end{cases} \tag{6}$$

$$l_k = \begin{cases} l_{k-1} - 2 & \text{if } k \geqslant 2 \\ 2r - 2 & \text{if } k = 1 \\ r & \text{if } k = 0. \end{cases} \tag{7}$$

This follows easily from Lemmas 2.2 and 2.3.

It is worth mentioning that, since $j - \frac{1}{2} < \sqrt{r^2 - i^2} < j + \frac{1}{2}$ (as shown in Section 2.2), we get $j = \mathrm{NI}$ (the nearest integer to) $(r^2 - i^2)^{\frac{1}{2}}$; or, as mentioned in [38], we get $i = \mathrm{NI}\ (r^2 - j^2)^{\frac{1}{2}}$ on interchanging i and j. This helps in searching for integer points near a circle of radius r, starting from the point $(r, 0)$. As a further sophistication, we can generate the parabolic arcs $y^2 = 2x + 1, 4x + 4, 6x + 9$, etc., from a direct algorithm [37]. This is analogous to the (change in) interval lengths stated in Theorem 2.4.

Exerting Theorem 2.4, therefore, the algorithm *DCS* may be designed, as shown in Fig. 5. It may be noted that, the $(i+1)$th square number $S_{i+1} = (i+1)^2$ can be obtained easily (without using any multiplication) from the previous square number $S_i = i^2$, since $S_{i+1} = (i+1)^2 = S_i + 2i + 1$, which is equivalent to adding a "gnomon" [41], as shown in Fig. 4 (courtesy, http://mathworld.wolfram.com). This is incorporated in the algorithm (steps 5–7), shown

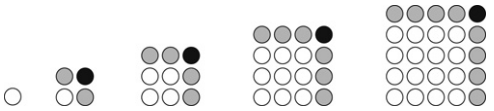


Fig. 4. Generation of a square number S_{i+1} by adding a "gnomon" to S_i. All circles in the previous square number S_i are shown by unfilled circles in S_{i+1}, whereas the gray ones ($2i$ numbers) and the black one denote the gnomon added to S_i to get S_{i+1}.

```
Algorithm DCS (int r) {
1.  int i = 0, j = r, s = 0, w = r − 1;
2.  int l = w << 1;
3.  while (j ⩾ i) {
4.        do { include_8_sym_points (i, j);
5.             s = s + i;
6.             i++;
7.             s = s + i; } while (s ⩽ w);
8.        w = w + l;
9.        l = l − 2;
10.       j−−; }}
```

Fig. 5. Algorithm *DCS*.

1.	$i = 0, j = 11, s = 0, w = 10$								
2.	$l = 20$								
3.	**while** $(j = 11 \geqslant 0 = i)$				**10** $\geqslant$ **4**		**9** $\geqslant$ **6**	**8** $\geqslant$ **7**	
4.	$(0, 11)$	$(1, 11)$	$(2, 11)$	$(3, 11)$	$(4, 10)$	$(5, 10)$	$(6, 9)$	$(7, 8)$	$(8, 8)$
5.	$s = 0$	$s = 2$	$s = 6$	$s = 12$	$s = 20$	$s = 30$	$s = 42$	$s = 56$	$s = 72$
6.	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$	$i = 9$
7.	$s = 1 \leqslant w$	$4 \leqslant w$	$9 \leqslant w$	$16 \not\leqslant w$	$25 \leqslant w$	$36 \not\leqslant w$	$49 \not\leqslant w$	$64 \leqslant w$	$81 \not\leqslant w$
8.				$w = 30$		$w = 48$	$w = 64$		$w = 78$
9.				$l = 18$		$l = 16$	$l = 14$		$l = 12$
10.				$j = 10$		$j = 9$	$j = 8$	$j = 7(\not\geqslant i = 9)$	

Fig. 6. A demonstration of *DCS* for $r = 11$.

in Fig. 5, where the gnomon addition of $2i + 1$ is realized by adding i with the previous square s, followed by adding an incremented value $(i + +)$ of i, in order to optimize the primitive arithmetic operations. It is evident that, in step 4, the procedure *include_8_sym_points* (i, j) includes the set of eight symmetric grid points, namely $\{(\pm x, \pm y) : \{x\} \cup \{y\} = \{i, j\}\}$, in $\mathcal{C}^{\mathbb{Z}}(\mathrm{O}, r)$. A demonstration of *DCS* for $r = 11$ is given in Fig. 6, the corresponding digital circle having been already shown in Fig. 2.

It is worthwhile mentioning here that, although, from the two recurrence equations for u_k and l_k in Theorem 2.4, the run length $\lambda(r-k)$ (and the "square numeric code", thereof) at ordinate $r-k$ in the first octant can be obtained by

$$\lambda(r-k) = \left\lfloor \sqrt{u_k + l_k - 1} \right\rfloor - \left\lfloor \sqrt{u_k - 1} \right\rfloor. \tag{8}$$

The realization of Eq. (8) needs square root operations, invoking unwanted floating-point computation, corresponding to each run. The algorithm *DCS*, given in Fig. 5, is a circumvention of Eq. (8), using few primitive operations in the integer domain.

2.4. Analysis of algorithm DCS

We discuss here the number of elementary operations, namely comparison, addition (subtraction), and increment (decrement), required over all the iterations in the algorithm *DCS*. The inside do–while loop (steps 4–7) will find out

```
Algorithm DCB (int r) {
1.  int i = 0, j = r, h = 1 − r, dE = 3, dSE = −2r + 5;
2.  include_8_sym_points (i, j);
3.  while (j > i) {
4.      if (h < 0) {
5.          h = h + dE;
6.          dE = dE + 2;
7.          dSE = dSE + 2;
8.          i + +; }
9.      else {
10.         h = h + dSE;
11.         dE = dE + 2;
12.         dSE = dSE + 4;
13.         i + +; j + +; }
14.     include_8_sym_points (i, j); }}
```

Fig. 7. Bresenham's algorithm.

the consecutive equi-ordinate (ordinate $= j$) grid points (to be precise, the squares of their abscissae) in $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ at each iteration of the outside **while** loop (step 3).

Now, each iteration of the outside **while** loop corresponds to a particular ordinate j, which gets decremented by unity (step 10) for the next iteration corresponding to the next ordinate, i.e. $j - 1$, for which, it may be observed, the upper limit w and the interval length l of the corresponding interval are being updated (steps 8 and 9) in accordance with Theorem 2.4. Hence, the total number of iterations of the outside **while** loop is given by the number of south-east (SE) transitions in the digital arc representing $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$. The total number of comparisons required for checking the condition ($j \geqslant i$) is, therefore, given by $|\mathrm{SE}|+1$, where, $|\mathrm{SE}|$ denotes the total number of SE transitions in $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$. The total number of comparisons for checking the condition ($s \leqslant w$) involved in the do–while loop, would be given by the total number of grid points in $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$, i.e. $|\mathrm{E}|+|\mathrm{SE}|$, where, $|\mathrm{E}|$ denotes the total number of east (E) transitions in $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$. Also, for each transition (whether E or SE), two additions (step 5 and step 7) and one increment (step 6) are mandatory. In the case of each SE transition, two extra additions (step 8 and step 9) and one extra increment (step 10) are required. Hence, for the entire algorithm, the total numbers of comparisons (n_c), additions (n_a), and increments (n_i) are given by

$$\begin{aligned} n_c &= |\mathrm{E}| + 2|\mathrm{SE}| + 1, \\ n_a &= 2|\mathrm{E}| + 4|\mathrm{SE}|, \\ n_i &= |\mathrm{E}| + 2|\mathrm{SE}|. \end{aligned} \tag{9}$$

2.5. Comparison with Bresenham's algorithm

Similar to the analysis given in Section 2.4, for Bresenham's algorithm [6], given in Fig. 7, the total numbers of comparisons (nb_c), additions (nb_a), and increments (nb_i) would be given by

$$\begin{aligned} nb_c &= 2|\mathrm{E}| + 2|\mathrm{SE}| + 1, \\ nb_a &= 3|\mathrm{E}| + 3|\mathrm{SE}|, \\ nb_i &= |\mathrm{E}| + 2|\mathrm{SE}|. \end{aligned} \tag{10}$$

Hence, it is easy to observe that the algorithm *DCS* and Bresenham's algorithm are very much similar in their efficiency and ease in implementation. However, the algorithm *DCS* can be further endowed with some other (run length) properties of digital circles, which are discussed in the following section.

3. Run length properties of digital circles

Lemma 3.1. *The number of perfect squares in a closed interval* $[v, w]$ *is at most one more than the number of perfect squares in the preceding closed interval* $[u, v-1]$ *of equal length, where the intervals are taken from the non-negative integer axis.*

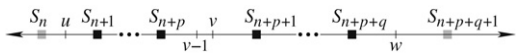


Fig. 8. Square numbers on the number axis.

Let p and q denote the numbers of perfect squares in the closed intervals $[u, v-1]$ and $[v, w]$ respectively. Let $S_n = n^2$ be the largest square number less than u. Then, the square numbers lying in $[u, v-1]$ are $S_{n+1}, S_{n+2}, \ldots, S_{n+p}$, and those lying in $[v, w]$ are $S_{n+p+1}, S_{n+p+2}, \ldots, S_{n+p+q}$, as shown in Fig. 8. Now, using the relation $S_{m+1} = (m+1)^2 = S_m + 2m + 1$ and applying the method of induction, we get

$$S_{n+k} = S_n + 2kn + k^2. \tag{11}$$

Considering the fact that the difference between the highest and the lowest square numbers in $[v, w]$ is at most the difference between its upper and lower bounds, that is $w - v$ (see Fig. 8), we have

$$S_{n+p+q} - S_{n+p+1} \leqslant w - v, \tag{12}$$

where, it may be noted, the equality holds true when both v and w are perfect squares and, therefore, become equal to S_{n+p+1} and S_{n+p+q} respectively. Similarly, considering the square number (S_{n+p+1}) just greater than the largest square in $[u, v-1]$ and the square number (S_n) just less than the smallest square in $[u, v-1]$ (S_n would be equal to 0 for the degeneracy when $u = 0$), we get

$$S_{n+p+1} - S_n > v - u. \tag{13}$$

Hence, combining Eqs. (12) and (13) with our consideration that $w - v = v - 1 - u$ (i.e., $w - v < v - u$), we get

$$\begin{aligned}
&S_{n+p+q} - S_{n+p+1} \leqslant w - v < v - u < S_{n+p+1} - S_n,\\
&\text{or,}\quad S_{n+p+q} - S_{n+p+1} < S_{n+p+1} - S_n,\\
&\text{or,}\quad S_n + 2(p+q)n + (p+q)^2 - S_n - 2(p+1)n - (p+1)^2 < S_n + 2(p+1)n + (p+1)^2 - S_n,\\
&\text{or,}\quad 2qn + (p+q)^2 - 2n - (p+1)^2 < 2(p+1)n + (p+1)^2,\\
&\text{or,}\quad 2(q-1)n + (2p+q+1)(q-1) < 2(p+1)n + (p+1)^2,\\
&\text{or,}\quad (q-1)(2n+2p+q+1) < (p+1)(2n+p+1).
\end{aligned} \tag{14}$$

Since the second factor in the left-hand side of Eq. (14) never falls short of the second factor in the right-hand side, i.e., $2n + 2p + q + 1 \geqslant 2n + p + 1$ for all cases, including the degenerate cases when $p = 0$ or/and $q = 0$ or/and $n = 0$, the validity of Eq. (14) implies that the first factor in its left-hand side is strictly less than the first factor in its right-hand side. Hence, $q - 1 < p + 1$, or, $q \leqslant p + 1$.

Theorem 3.2. *The run length of grid points of* $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ *with ordinate* $j-1$ *never exceeds one more than the run length of its grid points with ordinate* j.

From Lemma 2.3, it is obvious that the length of the interval, containing the squares of abscissae (of grid points of $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$) with ordinate $j-1$, is 2 less than that corresponding to ordinate j. Even if the interval corresponding to ordinate $j-1$ had been equal in length to the preceding interval corresponding to ordinate j, then from Lemma 3.1, the maximum number of grid points with ordinate $j-1$ would not have exceeded one more than the number of grid points with ordinate j. Hence the proof.

To uphold the significance of Theorem 3.2 apropos the state of the art, it should be mentioned here that, in [48], a similar theorem has been stated and proved. The proof in [48] is, however, different from the proof of this paper; there the proof has been done – for the analogous theorem as well as for others – by using *floor function* (over the difference of two *square roots*); whereas, the proofs in this paper are based on the distribution of square numbers in integer intervals.

It is interesting to note that Theorem 3.2 provides a good and useful upper bound on the number of grid points with ordinate j w.r.t. that corresponding to ordinate $j-1$. The derivation of lower bound that we have obtained is, however, not so straightforward, as evident in the following discussion.

If p and q denote the numbers of perfect squares in the closed intervals $[u, v-1]$ and $[v, w]$ (of equal length, i.e., $v - 1 - u = w - v$) (Fig. 8) respectively, and $S_n = n^2$ is the largest square number less than u, as discussed earlier,

then we have

$$S_{n+p+q+1} > w \quad \text{and} \quad -S_{n+p} > -v, \quad \text{which imply } S_{n+p+q+1} - S_{n+p} > w - v;$$

and

$$S_{n+p} \leqslant v - 1 \quad \text{and} \quad -S_{n+1} \leqslant -u, \quad \text{which imply } S_{n+p} - S_{n+1} \leqslant v - 1 - u \ (=w - v).$$

Hence,

$$\begin{aligned} & S_{n+p+q+1} - S_{n+p} > S_{n+p} - S_{n+1}, \\ \text{or,} \quad & 2n(p+q+1) + (p+q+1)^2 - 2np - p^2 > 2np + p^2 - 2n - 1, \\ \text{or,} \quad & 2n(q+1) + 2p(q+1) + (q+1)^2 > 2n(p-1) + p^2 - 1. \end{aligned} \tag{15}$$

Now, since our concern is to find a lower bound of q in terms of p, we consider $q + 1 \leqslant p - 1$, i.e., $2n(q+1) \leqslant 2n(p-1)$, whence Eq. (15) produces the following relation.

$$\begin{aligned} & 2p(q+1) + (q+1)^2 > p^2 - 1, \\ \text{or,} \quad & (p+q+1)^2 > 2p^2 - 1, \\ \text{or,} \quad & (p+q+1)^2 \geqslant 2p^2, \\ \text{or,} \quad & q \geqslant (\sqrt{2}-1)p - 1, \\ \text{or,} \quad & q \geqslant \left\lceil (\sqrt{2}-1)p - 1 \right\rceil, \\ \text{or,} \quad & q \geqslant \left\lfloor (\sqrt{2}-1)p \right\rfloor, \end{aligned} \tag{16}$$

since p and q are integers and $(\sqrt{2}-1)p - 1$ is irrational.

The bound for q in Eq. (16) has two weaknesses. The first one is that we get a loose lower bound of q that would cause some redundant operations while deciding the run length of grid points at ordinate $j - 1$ from that in the preceding ordinate j. Secondly, and more importantly, computation of the lower bound of q in accordance with Eq. (16) requires a square root operation followed by a floor/truncation operation, which is computationally expensive and not desirable in the run length finding procedure of a circle construction algorithm.

In order to circumvent the aforesaid problems, therefore, we turn around in a different way from Eq. (15) to obtain a nicer bound for q as follows.

$$\begin{aligned} & (q+1)(2n+2p+q+1) > (p-1)(2n+p+1), \\ \text{or,} \quad & (q+1)(p+q) > (p-q-2)(2n+p+1). \end{aligned} \tag{17}$$

From Eq. (17), it may be observed that, if the interval $[u, v-1]$ containing p perfect squares is the interval I_k (Section 2.3), $k \geqslant 1$, and S_n is the largest square number less than u, then by Theorem 3.2, $p < n + 2$. Since $n + 2 \leqslant 2n$ for $n \geqslant 2$, we get $p < 2n$ for $n \geqslant 2$,[1] or, $p < 2n$ for $r \geqslant 2$.[2]

Now, it is evident that Eq. (17) is true for *any two* consecutive intervals $[u, v-1]$ and $[v, w]$ of equal length; whereas, $p < 2n$ for $n \geqslant 2$ is true when the interval $[u, v-1]$ is identical with the interval I_k that contains (the squares of abscissae of) the grid points whose ordinates are $r - k$, for $k \geqslant 1$ (Lemma 2.2). Since the length of I_{k+1} is smaller than that of I_k by 2 but $[v, w]$ is of same length as $[u, v-1]$ in Eq. (17), we cannot consider $[v, w]$ to be identical with I_{k+1} if we consider I_k as $[u, v-1]$. However, from Theorem 3.2 (and its preceding paragraph), we always have $q \leqslant p + 1$, whether or not $[u, v-1]$ is identical with I_k.

[1] If $k = 1$, then I_0 contains first n perfect squares. If $k > 1$, then $I_0 \cup I_1 \cup \ldots I_{k-1}$ contains first n perfect squares. Hence, the relation $p < 2n$ loses its tightness as k goes on increasing. In other words, p falls shorter and shorter of $2n$ as k increases.

[2] Since $k \geqslant 1$ is not valid for $r = 1$ but from $r = 2$ on, we consider $r \geqslant 2$.
$r = 1$: only I_0 exists, and therefore, $k = 0$.
$r = 2$: I_0 and I_1 exist ($k = 0, 1$); for $k = 1$, we have $n = 1$, $p = 0$.
$r = 3$: I_0, I_1, and I_2 exist ($k = 0, 1, 2$); for $k = 1$, we have $n = 1$, $p = 1$; for $k = 2$, we have $n = 2$, $p = 0$.
etc.

Thus, if we consider $[u, v-1]$ to be I_k, then we have $q \leqslant p+1$ and $p < 2n$ for $n \geqslant 2$, which, in combination, gives $q < 2n+1$, or, $p+q < p+2n+1$. Hence, if $[u, v-1] = I_k$ $(k \geqslant 1)$ and $w - v = v - 1 - u$, then from Eq. (17), we get

$$\begin{aligned} & q+1 > p-q-2, \\ \text{or,} \quad & 2q > p-3, \\ \text{or,} \quad & q > \frac{p-1}{2} - 1, \\ \text{or,} \quad & q \geqslant \frac{p-1}{2} \geqslant \left\lfloor \frac{p-1}{2} \right\rfloor. \end{aligned} \tag{18}$$

From the result obtained in Eq. (18), we can, therefore, make the following observation.

Lemma 3.3. *If $[u, v-1]$ is the interval $I_k, k \geqslant 1$, and $[v, w]$ is the interval of same length as $[u, v-1]$, then the number of perfect squares in $[v, w]$ is at least (floor of) half the number of perfect squares less one in $[u, v-1]$.*

Putting together the above findings, therefore, we get the following theorem that captures the run length properties of a digital circle in a precise form.

Theorem 3.4. *If $\lambda(j)$ be the run length of grid points of $\mathcal{C}^{\mathbb{Z},I}(\mathrm{O}, r)$ with ordinate j, then the run length of grid points with ordinate $j-1$ for $j \leqslant r-1$ and $r \geqslant 2$, is given by*

$$\lambda(j-1) \geqslant \left\lfloor \frac{\lambda(j)-1}{2} \right\rfloor - 1.$$

From Lemma 2.3, the length of the interval containing the squares of abscissae of grid points with ordinate $j-1$ is 2 less than that corresponding to ordinate j for $j \leqslant r-1$.[3] Had the interval corresponding to ordinate $j-1$ been equal in length to the preceding interval (ordinate j), then from Lemma 3.3, the number of grid points with ordinate $j-1$ would have been at least $\lfloor(\lambda(j)-1)/2\rfloor$ for $r \geqslant 2$. In case the next integer or the next-to-next integer immediately after the interval corresponding to ordinate $j-1$ is a perfect square, the minimum possible value of $\lambda(j-1)$ would be further less by unity. Note that, for $r = 1$, we have only one run in the first octant corresponding to $j = 1$, and hence the case of $\lambda(j-1)$ is inapplicable. This is also justified by the fact that Lemma 3.3 is for $k \geqslant 1$, which implies $r \geqslant 2$ as explained earlier.

3.1. Algorithm DCR

Combining Theorems 3.2 and 3.4, therefore, we obtain Eq. (19), which can be used to derive the horizontal run of grid points with ordinate $j-1$, from the previous run with ordinate j, for $j \leqslant r$.

$$\left\lfloor \frac{\lambda(j)-1}{2} \right\rfloor - 1 \leqslant \lambda(j-1) \leqslant \lambda(j) + 1. \tag{19}$$

The algorithm *DCR* that incorporates the relation between the consecutive runs, as captured in Eq. (19), is given in Fig. 9. It may be noted that, in Fig. 9, only the upper limit $(\lambda(j)+1)$ of the term $\lambda(j-1)$ has been considered for simplicity. The lower limit $(\lfloor\frac{1}{2}(\lambda(j)-1)\rfloor - 1)$ of $\lambda(j-1)$ may be considered in a similar fashion. Since we search for $\lambda(j-1)$ in an integer interval whose upper and lower limits are computed every time using its preceding run length, $\lambda(j)$, resorting to both the limits in the run-finding procedure is computationally effective for a sufficiently large value of $\lambda(j-1)$. The computation of the lower (or upper) limit involves a fixed number of operations, whereas the operations in a binary search is logarithmic on the length of the corresponding interval. As r increases sufficiently, the top runs also increase in length, whose interval searching would, therefore, be improved by constraining the concerned intervals using both the upper and the lower limits. The algorithm presented in Fig. 9 and the related results presented in this paper are, however, based on usage of the upper limit only.

[3] While deriving Eq. (18) from Eq. (17), the consideration of $[u, v-1]$ to be I_k in which $k \geqslant 1$, implies that $j \leqslant r-1$.

```
Algorithm DCR (int r) {
1.  int i = 0, j = r, s = 0, w = r − 1, t = r, m;
2.  int l = w << 1;
3.  while (j ⩾ i) {
4.        while (s < t) {
5.              m = s + t;
6.              m = m >> 1;
7.              if (w ⩽ square[m])
8.                    t = m;
9.              else
10.                   s = m + 1; }
11.       if (w < square[s])
12.             s − −;
13.       s + +;
14.       include_run (i, s − i, j);
15.       t = s + s − i + 1;
16.       i = s;
17.       w = w + l;
18.       l = l − 2;
19.       j − −; }}
```

Fig. 9. Algorithm *DCR* using run length properties in part (see the text for explanation).

In order to find the exact value of $\lambda(j-1)$, binary search has been used. The binary search assumes the minimum value (lower limit) as 1 and the maximum value (upper limit) as $\lambda(j)+1$ to start with, for finding $\lambda(j-1)$. The binary search is performed on the Look-Up Table, implemented in the form of a one-dimensional array, namely *square*[ ], that contains the square S_n of each integer $n = 0, 1, 2, \ldots, N$, where N^2 is the largest square not exceeding the maximum value R of radius r. For example, for $R = 1000$, N (and the size of the Look-Up Table, thereof) equals 31.

It may be noted that, the binary search procedure incorporated in Fig. 9 is a modified one, based on our requirements, from the conventional one found in the literature [32]. In accordance with the conventional procedure, one has to check at first whether the middle element (m in our figure) equals the search key, failing which, one of two other checks (smaller or greater) is performed. We have modified this (steps 7–10) to reduce the number of comparisons in each iteration of the inner **while** loop.

A demonstration of the algorithm *DCR* for $r = 106$ has been graphically shown in Fig. 10 until a run of unit length is found. For each row, the binary search is illustrated by circular dots, where each dot corresponds to the middle element (m) of the respective sub-array (*square*[$s..t$]). It may be noticed that, m in Fig. 9 (step 6) denotes the abscissa (vertical) line $x = m - 1$ in Fig. 10. As the binary search, associated with a particular row, proceeds and converges to produce the final run of the corresponding row, the respective dots have been gradually darkened to depict the impact of the run length finding procedure. The end of the run at each row is emphasized by highlighted abscissa lines passing through the end point of that run for visual clarity. For instance, for the topmost row, m starts with 53, followed by 26, 13, and so on, until $s = t = 11$, whence m finally becomes 11, thereby yielding the run length equal to 11. Similarly, for the next row, since the start values of s and t are 11 and 23 respectively, the subsequent values of m are 17, 20, 19, and (finally) 18, which makes the corresponding run length equal to $18 - 11 = 7$. Thus, the run length of a particular row ($y = j$) is given by the cumulative run length up to that row minus the preceding cumulative run length, which is realized by the function *include_run* $(i, s - i, j)$ in step 6 of the algorithm. The "square numeric code" of a circle with radius 106 in the first octant, therefore, turns out to be $\langle 11, 7, 5, 5, 3, 3, 3, 3, 2, 2, 2, 3, 1, 2, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1\rangle$, which can be compressed to $\langle 11, 7, 5^2, 3^4, 2^3, 3, 1, 2^3, 1, 2, 1, 2, 1^2, 2, 1^3, 2, 1^5\rangle$, which, in turn, can be used to easily derive the chain code representation of the circular arc, as mentioned in Section 1 for Fig. 1.

It is worth mentioning here that, implementation of the Look-Up Table in the form, mentioned above, avoids floating-point operations of any sort at any stage, such as square root operation to find out the run for $j = r$ in the algorithm proposed in [48]. Also, the above algorithm will not be so efficient for circles (1st octant) of small radii, or to find a run length of small value, such as when $j/i \leqslant 4$ or so (see Section 4) since the operations (comparisons etc.) needed in the binary search for small run lengths would raise the overall time of the algorithm. Hence, the algorithm *DCR* may be used in some model of hybridization, whose one possible realization is given in Section 3.3.

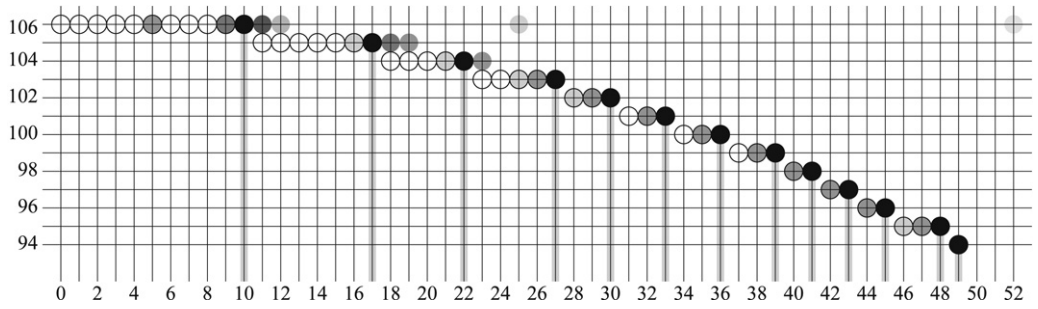


Fig. 10. Demonstration of algorithm *DCR* for radius 106 (see the text for explanation).

3.2. Analysis of algorithm DCR

We analyze the algorithm *DCR* for showing its fitness to hybridize a circle construction algorithm, e.g., *DCS* given in Fig. 5. We assume that some of the top runs (i.e., $j = r, r-1, r-2, \ldots$) are generated by the algorithm *DCR*, followed by the generation of the remaining runs by a simpler algorithm like *DCS*. The problem is, therefore, for the generation of how many runs (starting from the topmost one) should the algorithm *DCR* be used, so that the overall time of the resultant hybrid algorithm is minimized.

The analysis of the algorithm *DCS*, as put in Section 2.4, shows that the total number of operations (comparisons, additions, and increments) for generation of $k+1$ runs is given by $4|\mathrm{E}| + 8|\mathrm{SE}| + 1$, from which, by dropping '+1', the lower bound is given by $4|\mathrm{E}| + 8|\mathrm{SE}|$. Now, observe that $|\mathrm{SE}|$ is the number of south-east transitions, which is simply k, and $|\mathrm{E}|$ is the number of east transitions, given by

$$\begin{aligned} |\mathrm{E}| &= \left\lfloor \sqrt{kr + r(k+1) - k(k+1) - 1} \right\rfloor - k \quad \text{[from Eq. (4)]} \\ &> \left\lfloor \sqrt{kr} \right\rfloor - k \quad \text{[since } r > k\text{]}. \end{aligned}$$

For producing the top $k+1$ runs (i.e., for $j = r$, $j = r-1, \ldots, j = r-k$) by the algorithm *DCS*, the total number of primitive operations is given by

$$n_1^{(k+1)} > 4\left(\left\lfloor \sqrt{kr} \right\rfloor - k\right) + 8k = 4\left\lfloor \sqrt{kr} \right\rfloor + 4k. \tag{20}$$

For the algorithm *DCR*, in accordance with the simple implementation, as shown in Fig. 9, for the topmost row (i.e., $j = r$), the number of iterations of the inner **while** loop (step 4), meant for binary search as discussed in Section 3.1, is given by $\lceil \log r \rceil$, the base of $\log(\cdot)$ being 2 in this discussion. For the binary search on finding out each run length at $j < r$, the number of iterations depends on the length of the search interval (i.e., initial values of s and t for the corresponding j, see Fig. 9), which depends on the preceding run length due to the inherent recursive nature of Eq. (19) used in the algorithm. This poses a problem in deriving the number of primitive operations in the algorithm for the individual runs. However, it is easy to observe that for each subsequent value of j ($<r$), the inner **while** loop iterates for not more than $\lceil \log(\sqrt{r}) \rceil = \left\lceil \frac{1}{2} \log r \right\rceil$ times, since the run length for $j \leqslant r-1$ would be always less than that for $j = r$ (which is $\lfloor \sqrt{r} \rfloor + 1$); in fact, for $j \leqslant r-1$, the value $\lfloor \sqrt{r} \rfloor$ turns out to be a loose upper bound for its run length, where the actual run length for $j = r-k$ ($k > 0$) falls shorter and shorter of $\lfloor \sqrt{r} \rfloor$ as k goes higher and higher.

Now, for each iteration of the inner **while** loop, including the check ($s < t$) in step 4, and assuming that the logical expression ($w \leqslant square[m]$) in step 7 is always false (worst case), thereby requiring one addition in step 10 at each iteration, the total number of operations (comparisons, additions, right shifts, and increments) per iteration turns out to be 5 in the worst case.

Similarly, for each iteration of the outer **while** loop, the total number of all primitive operations can be shown to be at most 11. Further, for the outer **while** loop, the total number of iterations is simply the number of SE transitions, which becomes k, if $k+1$ runs are produced by the algorithm *DCR*.

Thus, for producing $k+1$ runs, the total number of primitive operations (worst case) required by the algorithm *DCR* is given by

$$n_2^{(k+1)} < 5\,(\lceil k/2 \rceil + 1)\,\lceil \log r \rceil + 11k. \tag{21}$$

Hence, from Eqs. (20) and (21), the algorithm *DCR* would be faster than *DCS* for generating the top $k+1$ runs, provided

$$5\,(\lceil k/2 \rceil + 1)\,\lceil \log r \rceil + 11k < 4\left\lfloor \sqrt{kr} \right\rfloor + 4k,$$

$$\text{or,}\quad 5\,(\lceil k/2 \rceil + 1)\,\lceil \log r \rceil + 7k < 4\left\lfloor \sqrt{kr} \right\rfloor. \tag{22}$$

It may be noted that, while deriving the lower bound for $n_1^{(k+1)}$ and upper bound for $n_2^{(k+1)}$ in Eqs. (20) and (21) respectively, we have resorted to some rough approximations. The relation shown in Eq. (22), therefore, does not effectuate a tight restriction on the number of runs (i.e., $k+1$) that should be produced by the algorithm *DCR*. However, from Eq. (22), we get, for a given value of radius r, the least number of runs that should be generated by *DCR*, followed by the remaining runs by the algorithm *DCS* (or Bresenham's algorithm) for an efficient hybridization.

3.3. Hybrid algorithm DCH

As explained in Section 3.2, the algorithm *DCR* is computationally attractive in the case of production of long horizontal runs, which occurs in the upper part of octant 1 (i.e., for $j = r, r-1, \ldots$). In particular, for circles with high radii, we get quite many long runs for which *DCR* works with higher efficiency compared to *DCS*. However, towards the end of octant 1 (i.e., as j approaches i), the runs go on decreasing in length, whereby the efficiency of *DCR* decreases. To generate a small run length, therefore, it is wiser to use the algorithm *DCS* (or Bresenham's algorithm) instead of the algorithm *DCR*. Hence, a proper hybridization of these two algorithms is a solution to ensure the efficiency of a circle construction algorithm.

The hybrid algorithm, namely *DCH*, designed and experimented by us, is shown in Fig. 11. In the algorithm *DCH*, the argument p acts as the *hybridizing parameter*, which decides the minimum run length generated by the run length finding procedure (*DCH*: steps 3–21). A lower value of p makes the algorithm *DCH* more dependent on *DCR* than on *DCS* (steps 22–34), whereas a higher value of p makes *DCH* more dependent on *DCS* than on *DCR*. For example, if we make $p = 5$, then only the first five top runs ($j = 106, 105, \ldots, 102$) of the circle with radius $r = 106$ (Fig. 10) will be generated by steps 3–21 of *DCH*, and the remaining twenty eight runs/rows (each of length less than 5) will be produced by steps 22–34. Similarly, if $p = 4$, then steps 3–21 produce the top five runs; if $p = 3$, then top nine runs, and so on. As an illustration, the number of top runs generated by *DCH* for some typical circles corresponding to few different values of p, are shown in the Appendix.

4. Test results

Implementations of the two algorithms *DCS* and *DCR*, whose pseudocodes are given in Figs. 5 and 9, are as simple as that of Bresenham's. As mentioned in Section 1, since the construction of a digital circle using the number-theoretic properties, discussed in this paper, should not be considered as contending Bresenham's algorithm, but supplementing it in theoretical perspectives, we have not produced here the results on (CPU) run times; rather, few results on the number of primitive operations required for the algorithms are presented below. It may be noted that, for a given radius r, the number of primitive operations to generate (first octant of) the corresponding digital circle is always fixed (i.e., deterministic) for any one of the algorithms on circle construction. Furthermore, since the CPU times are dependent on the machine configuration and code optimization by the associated compiler, CPU times not necessarily reflect the true picture of the speed and efficiency of an algorithm.

In Fig. 12, the major primitive operations, namely comparisons, additions, and increments, have been shown for the algorithm *DCS* in $\log_{10}$ scale for the values of radius from 1 to 1000. It is evident from these plots, which are just experimental authentications of the theoretical values of the number of operations, given in Eq. (9), that the number-theoretic algorithm *DCS* is no less efficient and no less dependable than Bresenham's algorithm.

In Fig. 13, the total number of primitive operations that include comparisons, additions, right shifts, and increments, has been shown for the algorithm *DCR*. The plots in this figure are for the generation of some initial parts (till $j \geqslant ti, t = 1, 2, 4, 16$) of the first octant, starting from the topmost row (i.e., $j = r$). As discussed in Sections 3 and 3.2, proper hybridization techniques may be designed that would first generate the runs (using a binary search

```
Algorithm DCH (int r, int p) {
1.  int i = 0, j = r, s = 0, w = r − 1, t = r, m;
2.  int l = w << 1;
3.  while (j ⩾ i) {
4.      while (s < t) {
5.          m = s + t;
6.          m = m >> 1;
7.          if (w ⩽ square[m])
8.              t = m;
9.          else
10.             s = m + 1; }
11.     if (w < square[s])
12.         s − −;
13.     s + +;
14.     include_run (i, s − i, j);
15.     if (s − i < p)
16.         break;
17.     t = s + s − i + 1;
18.     i = s;
19.     w = w + l;
20.     l = l − 2;
21.     j − −; }
22. i = s − 1;
23. s = square[s];
24. w = w + l;
25. l = l − 2;
26. j − −;
27. while (j ⩾ i) {
28.     do { include_8_sym_points (i, j);
29.         s = s + i;
30.         i + +;
31.         s = s + i; } while (s ⩽ w);
32.     w = w + l;
33.     l = l − 2;
34.     j − −; }}
```

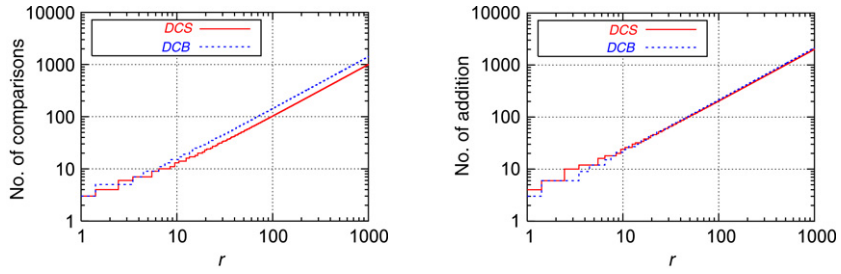
Fig. 11. The hybrid algorithm *DCH* for construction of digital circles.

technique, as discussed earlier, and an example illustrated in Fig. 10), starting from the topmost row, and then use *DCS* (or Bresenham's algorithm, or any similar algorithm) for the remaining part of the first octant. The top runs would be generated by a number-theoretic algorithm, such as the algorithm *DCR*, as long as the run-generation procedure is appreciably faster than the other, as explained in Sections 3.2 and 3.3.

That the trade-off in the hybridization is a critical issue has been exhibited by the plots in Fig. 13. From these plots, it is apparent that for radius exceeding 100 or so, the number-theoretic algorithm (*DCR*) has appreciable margin over Bresenham's algorithm. Further, as evident from the plots, the gain for the run length finding approach using the number-theoretic technique goes on increasing as the radius increases. For a very large radius, say exceeding 1000, the number-theoretic technique would contribute substantial improvements to a circle-generation procedure.

To demonstrate the need of hybridization, especially for generating circles with high radii, some experimental results have also been given in this paper. The plots shown in Fig. 14 give a comparative idea of the total number of operations used in the algorithm *DCH* versus those used in *DCS* and in *DCB*, for radius $r = 1$–$10\,000$ (in $\log_{10}$ scale) corresponding to the hybridizing parameter $p = 1$–50. The radius and the hybridizing parameter have been sampled selectively at regular intervals for visual clarity of the two pair of plots. It is evident from these plots that, although the algorithm *DCS* behaves marginally better than *DCB*, the performance of the algorithm *DCH* (hybridized between *DCS* and *DCR*) is, however, appreciably better than that of *DCB*, especially for circles with high radii.

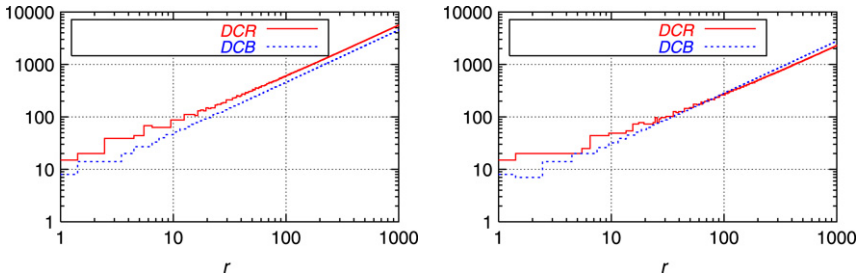
For example, for $r = 10\,000$, the number of operations in *DCB* is higher than that in *DCH* by about 40%, provided p lies in the range 10–50. For p in the range 1–5, however, *DCH* is not computationally attractive compared to *DCB*, since a low value of p makes *DCH* too much dependent on *DCR* than on *DCS* even for producing smaller runs, thereby bringing down its efficiency, as explained in Section 3.3.



(a) Number of comparisons. (b) Number of additions.

(c) Number of increments. (d) Total number of operations.

Fig. 12. Primitive operations in *DCS* algorithm and Bresenham's algorithm for radius r from 1 to 1000. The plots have been given in $\log_{10}$ scale.



(a) $j \geqslant i$. (b) $j \geqslant 2i$.

(c) $j \geqslant 4i$. (d) $j \geqslant 8i$.

Fig. 13. Total number of operations (given in $\log_{10}$ scale) in *DCR* algorithm (given in Fig. 9) and Bresenham's algorithm for different starting parts ($j \geqslant i$, $j \geqslant 2i, \ldots$) of the 1st octant.

5. Conclusion

We have shown how the number-theoretic method can be used to establish some fundamental and interesting properties of a digital circle, which are of immense significance for its interpretation and construction. As mentioned

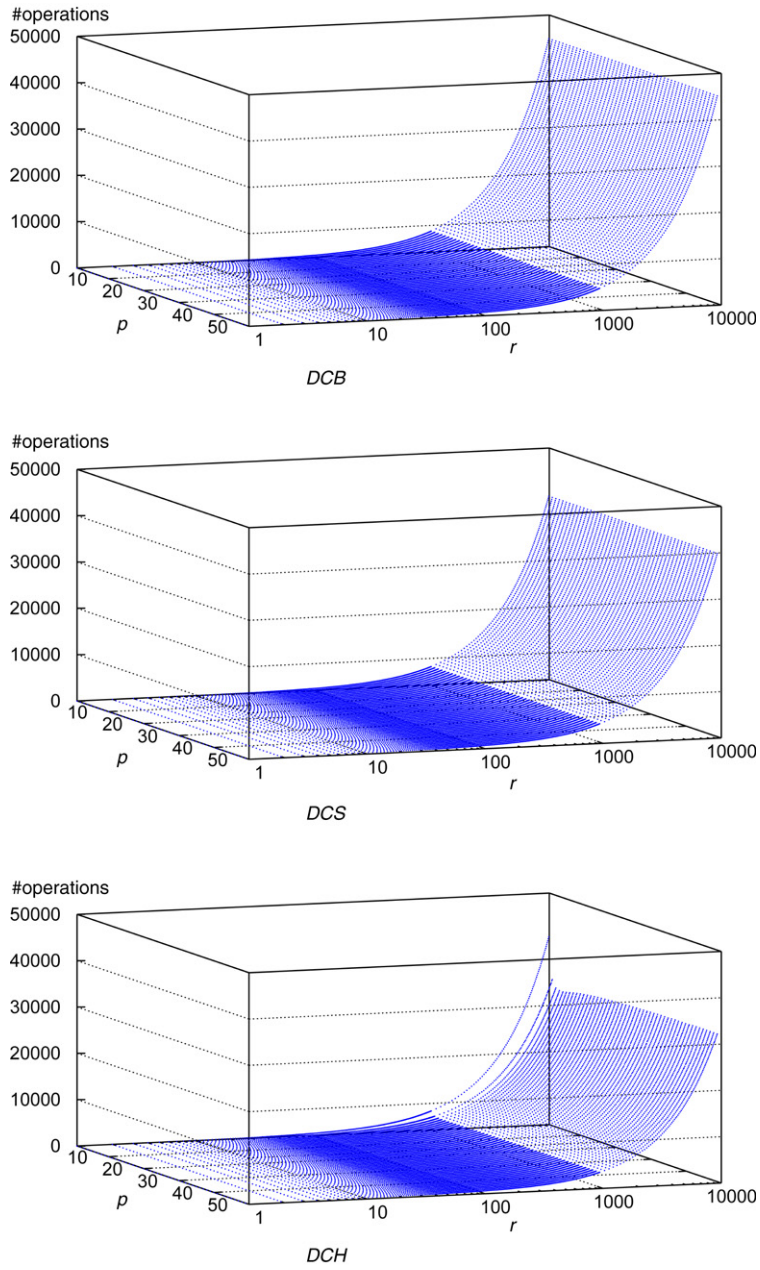


Fig. 14. Plots showing how the number of operations used by three algorithms (*DCB*, *DCS*, and *DCH*) varies with radius r of the digital circle and with the hybridizing parameter p.

earlier, these properties not only conform to, but also supplement and enrich the properties underlying the conventional concepts of digital calculus, such as Bresenham's circle construction algorithm.

Although we have discussed in this paper about the scope and merits of designing circle construction algorithms (regular as well as hybridized) using number-theoretic properties, the improvements of these algorithms still remain an open problem. For instance, to explore the possibility of finding all the run lengths in the first octant for a given radius r, without using the information of the other (preceding) run length(s) is really an engrossing problem that may be worked upon, which, if realized, would bring down the present linear time complexity for circle construction to a sub-linear one, and would also facilitate the parallelization of a circle construction algorithm.

Further, although there exist several schemes for segmentation and characterization of digital circles and circular arcs using geometric properties, as indicated in Section 1, an intricate problem lies in discovering some advanced number-theoretic properties that may aid and expedite the process of segmentation/characterization of circular arcs in a digital image. For instance, given the chain code representation (the "square numeric code", thereof) of a digital curve, how one can determine whether or not the corresponding curve is an arc of a digital circle. In order to design an efficient algorithm, using the primitive operations in the integer domain only, several other necessary (and desirably, sufficient) number-theoretic properties have to be discovered. The segmentation of circular arcs in a digital image is a challenging problem, the solution of which would have numerous applications in image processing and computer vision.

Appendix

Number of runs ($m_r^{(p)}$, generated by steps 1–21 of *DCH*) versus radius (r) and the hybridizing parameter ($p \geqslant 3$).

r	p	$m_r^{(p)}$
10	3	2
	4	2
	5	1
20	3	3
	4	2
	5	2
	7	1
40	3	5
	4	3
	5	2
	7	2
	9	1
80	3	7
	4	5
	5	3
	7	3
	9	2
	12	1
160	3	12
	4	8
	5	5
	7	4
	9	3
	12	2
	15	1
320	3	20
	4	13
	5	8
	7	5
	9	4
	12	3
	15	2
	19	1
640	3	38
	4	24
	5	16
	7	9
	9	6
	12	4
	15	3
	19	2
	24	2
	30	1
1280	3	71
	4	44
	5	28
	7	16
	9	10
	12	6
	15	5
	19	3
	24	3
	30	2
	38	1
2560	3	145
	4	81
	5	57
	7	29
	9	18
	12	11
	15	8
	19	5
	24	4
	30	3
	38	2
	48	2
	60	1
5120	3	275
	4	161
	5	107
	7	56
	9	33
	12	21
	15	14
	19	9
	24	6
	30	5
	38	3
	48	3
	60	2
	75	1
10240	3	539
	4	320
	5	208
	7	108
	9	68
	12	40
	15	25
	19	17
	24	10
	30	7
	38	5
	48	4
	60	3
	75	2
	94	2
	118	1
20480	3	1087
	4	623
	5	410
	7	218
	9	133
	12	76
	15	49
	19	30
	24	20
	30	13
	38	9
	48	6
	60	4
	75	3
	94	3
	118	2
	148	1
40960	3	2135
	4	1246
	5	811
	7	423
	9	254
	12	148
	15	95
	19	60
	24	39
	30	25
	38	16
	48	11
	60	7
	75	5
	94	4
	118	3
	148	3
	185	2
	232	1
81920	3	4243
	4	2491
	5	1607
	7	838
	9	514
	12	290
	15	185
	19	119
	24	75
	30	49
	38	31
	48	19
	60	13
	75	9
	94	6
	118	5
	148	3
	185	3
	232	2
	290	1

References

[1] J.R.V. Aken, M. Novak, Curve-drawing algorithms for raster display, ACM Transactions on Graphics 4 (2) (1985) 147–169.
[2] N.I. Badler, Disk generators for a raster display device, Computer Graphics and Image Processing 6 (1977) 589–593.
[3] P. Bhowmick, B.B. Bhattacharya, Approximation of digital circles by regular polygons, in: Proc. Intl. Conf. Advances in Pattern Recognition, ICAPR, in: LNCS, vol. 3686, Springer, Berlin, 2005, pp. 257–267.
[4] S.N. Biswas, B.B. Chaudhuri, On the generation of discrete circular objects and their properties, Computer Vision, Graphics, and Image Processing 32 (2) (1985) 158–170.
[5] J.F. Blinn, How many ways can you draw a circle? IEEE Computer Graphics and Applications 7 (8) (1987) 39–44.
[6] J.E. Bresenham, A linear algorithm for incremental digital display of circular arcs, Communications of the ACM 20 (2) (1977) 100–106.
[7] J.E. Bresenham, Run length slice algorithm for incremental lines, in: R.A. Earnshaw (Ed.), Fundamental Algorithms for Computer Graphics, in: NATO ASI Series, vol. F17, Springer-Verlag, New York, 1985, pp. 59–104.
[8] Y.T. Chan, S.M. Thomas, Cramer–Rao lower bounds for estimation of a circular arc center and its radius, Graphical Models and Image Processing 57 (6) (1995) 527–532.
[9] S. Chattopadhyay, P.P. Das, D. Ghosh-Dastidar, Reconstruction of a digital circle, Pattern Recognition 27 (12) (1994) 1663–1676.
[10] T.C. Chen, K.L. Chung, An efficient randomized algorithm for detecting circles, Computer Vision and Image Understanding 83 (2) (2001) 172–191.
[11] S.H. Chiu, J.J. Liaw, An effective voting method for circle detection, Pattern Recognition Letters 26 (2) (2005) 121–133.
[12] W.L. Chung, On circle generation algorithms, Computer Graphics and Image Processing 6 (1977) 196–198.
[13] D. Coeurjolly, Y. Gérard, J.-P. Reveillès, L. Tougne, An elementary algorithm for digital arc segmentation, Discrete Applied Mathematics 139 (2004) 31–50.
[14] P.E. Danielsson, Comments on circle generator for display devices, Computer Graphics and Image Processing 7 (2) (1978) 300–301.
[15] E.R. Davies, A high speed algorithm for circular object detection, Pattern Recognition Letters 6 (1987) 323–333.
[16] E.R. Davies, A hybrid sequential-parallel approach to accurate circle centre location, Pattern Recognition Letters 7 (1988) 279–290.
[17] M. Doros, Algorithms for generation of discrete circles, rings, and disks, Computer Graphics and Image Processing 10 (1979) 366–371.
[18] M. Doros, On some properties of the generation of discrete circular arcs on a square grid, Computer Vision, Graphics, and Image Processing 28 (3) (1984) 377–383.
[19] D. Field, Algorithms for drawing anti-aliased circles and ellipses, Computer Vision, Graphics, and Image Processing 33 (1) (1986) 1–15.
[20] J.D. Foley, A.v. Dam, S.K. Feiner, J.F. Hughes, Computer Graphics — Principles and Practice, Addison-Wesley, Reading, Mass., 1993.
[21] H. Freeman, On the encoding of arbitrary geometric configurations, IRE Transactions on Electronic Computers EC-10 (1961) 260–268.
[22] R.M. Haralick, A measure for circularity of digital figures, IEEE Transactions on Systems, Man & Cybernetics 4 (1974) 394–396.
[23] C.T. Ho, L.H. Chen, A fast ellipse/circle detector using geometric symmetry, Pattern Recognition 28 (1) (1995) 117–124.
[24] B.K.P. Horn, Circle generators for display devices, Computer Graphics and Image Processing 5 (2) (1976) 280–288.
[25] P.I. Hosur, K.-K. Ma, A novel scheme for progressive polygon approximation of shape contours, in: Proc. IEEE 3rd Workshop on Multimedia Signal Processing, 1999, pp. 309–314.
[26] S.Y. Hsu, L.R. Chow, C.H. Liu, A new approach for the generation of circles, Computer Graphics Forum 12 (2) (1993) 105–109.
[27] H.S. Kim, J.H. Kim, A two-step circle detection algorithm from the intersecting chords, Pattern Recognition Letters 22 (6–7) (2001) 787–798.
[28] R. Klette, A. Rosenfeld, Digital Geometry: Geometric Methods for Digital Picture Analysis, in: Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, Morgan Kaufmann, San Francisco, 2004.
[29] R. Klette, A. Rosenfeld, Digital straightness: A review, Discrete Applied Mathematics 139 (1–3) (2004) 197–230.
[30] Z. Kulpa, A note on "circle generator for display devices", Computer Graphics and Image Processing 9 (January) (1979) 102–103.
[31] Z. Kulpa, B. Kruse, Algorithms for circular propagation in discrete images, Computer Vision, Graphics, and Image Processing 24 (3) (1983) 305–328.
[32] Y. Langsam, M.J. Augenstein, A.M. Tenenbaum, Data Structures Using C and C++, Prentice-Hall of India, New Delhi, 2000.
[33] M.D. Mcllroy, Best approximate circles on integer grids, ACM Transactions on Graphics 2 (4) (1983) 237–263.
[34] F. Mignosi, On the number of factors of Sturmian words, Theoretical Computer Science 82 (1) (1991) 71–84.
[35] B. Nagy, Characterization of digital circles in triangular grid, Pattern Recognition Letters 25 (11) (2004) 1231–1242.
[36] A. Nakamura, K. Aizawa, Digital circles, Computer Vision, Graphics, and Image Processing 26 (2) (1984) 242–255.
[37] M.L.V. Pitteway, Algorithm for drawing ellipses or hyperbolae with a digital plotter, The Computer Journal 10 (3) (1967) 282–289.
[38] M.L.V. Pitteway, Integer circles, etc. — some further thoughts, Computer Graphics and Image Processing 3 (1974) 262–265.
[39] F. Pla, Recognition of partial circular shapes from segmented contours, Computer Vision and Image Understanding 63 (2) (1996) 334–343.
[40] P.L. Rosin, G.A.W. West, Detection of circular arcs in images, in: Proc. 4th. Alvey Vision Conf., Manchester, 1988, pp. 259–263.
[41] D. Shanks, Solved and Unsolved Problems in Number Theory, AMS Chelsea Publishing, New York, 1993.
[42] K. Shimizu, Algorithm for generating a digital circle on a triangular grid, Computer Graphics and Image Processing 15 (4) (1981) 401–402.
[43] Y. Suenaga, T. Kamae, T. Kobayashi, A high speed algorithm for the generation of straight lines and circular arcs, IEEE Transactions on Computers 28 (1979) 728–736.
[44] S.M. Thomas, Y.T. Chan, A simple approach for the estimation of circular arc center and its radius, Computer Vision, Graphics, and Image Processing 45 (3) (1989) 362–370.
[45] M. Worring, A.W.M. Smeulders, Digitized circular arcs: Characterization and parameter estimation, IEEE Transactions on PAMI 17 (6) (1995) 587–598.
[46] W.E. Wright, Parallelization of Bresenham's line and circle algorithms, IEEE Computer Graphics and Applications 10 (5) (1990) 60–67.
[47] X. Wu, J.G. Rokne, Double-step incremental generation of lines and circles, Computer Vision, Graphics, and Image Processing 37 (3) (1987) 331–344.
[48] C. Yao, J.G. Rokne, Hybrid scan-conversion of circles, IEEE Transactions on Visualization and Computer Graphics 1 (4) (1995) 311–318.
[49] P.C. Yuen, G.C. Feng, A novel method for parameter estimation of digital arc, Pattern Recognition Letters 17 (9) (1996) 929–938.