I N F O R M A T I K

Rational Points on Circles

Christoph Burnikel

MPI–I–98–1–023 October 1998

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Im Stadtwald 66123 Saarbrücken Germany

Author's Address

Max-Planck-Institut für Informatik Im Stadtwald D-66123 Saarbrücken
email: burnikel@mpi-sb.mpg.de

Acknowledgements

This work was supported in part by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141(ALCOM II) and the BMFT(Förderungskennzeichen ITS 9103).

Abstract

We solve the following problem. For a given rational circle C passing through the rational points p, q, r and a given angle α, compute a rational point on C whose angle at C differs from α by a value of at most ϵ. In addition, try to minimize the bit length of the computed point.

This document contains the C++ program *rational_points_on_circle.c*. We use the literate programming tool *noweb* by Norman Ramsey.

Keywords

Exact arithmetic, rational sines, rational cosines, literate programming, computational geometry

1 Introduction

In recent years the call for reliable implementations of geometric alogrithms has been voiced. Such implementations should handle even degenerate and numerically difficult input correctly. It has turned out that *exact computation* is the safest method to produce reliable implementations, meaning that all results of geometric primitives used in the algorithm are computed exactly. Typically, exact implementations of geometric algorithms can process input given by integers and rationals of arbitrary sizes.

As we said before, we want to have our programs compute the correct results even for degenerate inputs. One variation of degeneracy is particularly important for this work, namely instances of four or more cocircular points that arise e.g. in the computation of planar Voronoi diagrams and Delaunay triangulations. Although cocircluar points are difficult to deal with, being a notorious source of errors, they are in some sense *rare* inputs. If we want to test our algorithms for correctness, we want a tool to create common points on a given circle. The pupose of this paper is to provide such a tool that produces exactly cocircular rational points near a specified location on a circle.

More specifically, we are given a set of three rational points p, q, r defining a circle C and an angle α. We are looking for a rational point s on C whose angle $\angle(s)$ at C satisfies $|\angle(s) - \alpha| < \epsilon$ for some given error bound $\epsilon > 0$.

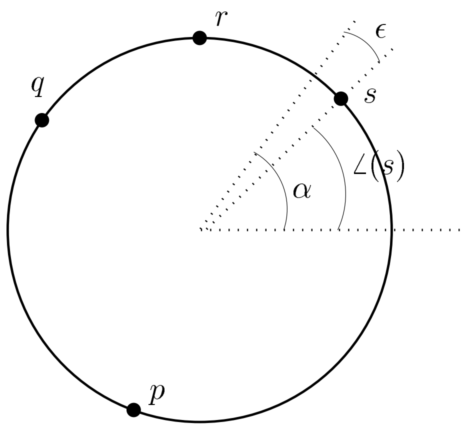

Figure 1: The problem: Find a rational point s on the circle C near the angle α.

In our paper we generalize some results given by John Canny et al. [CDR92]: *A Rational Rotation Method for Robust Geometric Algorithms.* In Canny's paper it is shown how to construct rational points on the unit circle. Note that the present problem cannot be reduced to the problem considered in Canny's paper since the rational circle C has in general nonrational radius. In particular, scaling C to radius 1 does not help. However, most methods used in Canny's paper generalize to our problem.

1.1 The structure of the program

Before we describe our method, let us formulate the programming goal in terms of a procedure declaration. Here we use *rat_points*, *rat_circles* and *rationals* from LEDA to represent rational points, rational circles (circles given by three rational points) and rational numbers, respectively. Our main procedure is

rat_point_on_circle(*const rat_circle*& C, *double alpha*, *double epsilon*).

A call of this function returns a rational point on the circle C whose angle has a distance less than *epsilon* from *alpha*. The meaning of the following procedures will be explained later.

2 ⟨*rational_point_on_circle.c* 2⟩≡

```
// This file contains the program rational_points_on_circle.c
// We implement the procedure rat_point_on_circle that computes
// for a circle C given by rational points p, q, r and an angle
// alpha a rational point s on C such that the angle of s
// at C differs from alpha by at most a given delta

#include <math.h>
#include <LEDA/rat_point.h>
#include <LEDA/rat_circle.h>
#include <LEDA/rational.h>

rat_point rat_point_on_circle
  (const rat_circle& C, double alpha, double epsilon);
```

⟨*procedure for the computation of phi_C(rt)* 4⟩
⟨*procedure for rational approximations* 9b⟩
⟨*procedure for the computation of t(alpha)* 5⟩

⟨*main procedure* 6⟩

⟨*test program* 10⟩

Uses `rat_point_on_circle` 6.

2 A Rational Parametrization $\phi(t)$ of the Circle

As Canny did for his specialized problem, we define a *rational parametrization* of the circle C. A rational parametrization of C is a parametrization that maps rational parameters to rational points on C. Here we use a *pencil of lines* as explained in Hoffmann's book on solid modeling [Hof90]. The general idea is the following. We fix a rational point p on our curve, draw a line with rational slope t through this point and compute the other point s of intersection of the line with the curve. It is well-known that for all algebraic curves of degree 2, s is always a rational point. In fact *every* rational point s on C is such an intersection point as the slope of the line passing through p and s is rational.

First we assume for simplicity that p is the origin. Let the circle C have center c with integral homogeneous coordinates (c_x, c_y, c_w). Recall that the homogeneous coordinates (c_x, c_y, c_w) correspond to the Euclidean point $(c_x/c_w, c_y/c_w)$. The pencil of lines passing through point $p = (0,0)$ on C is given by

$$l_t : l_t(x) = t \cdot x, t \in R - \{0\}.$$

Each l_t non-tangent to C intersects C in p and another point $\phi(t)$ that we compute now. If l_t is tangent to C at p we set $\phi(t) = p$. The homogeneous coordinates of $\phi(t)$ are denoted by $(\varphi_x(t), \varphi_y(t), \varphi_w(t))$ and the cartesian coordinates of $\phi(t)$ are denoted by $(\phi_x(t), \phi_y(t))$.

The point $\phi(t)$ is found by inserting $y = l_t(x) = t \cdot x$ into the equation

$$E_C : (x - c_x/c_w)^2 + (y - c_y/c_w)^2 = r_C^2$$

of the circle C where r_C is the radius of C. Setting $y = tx$ we get

$$\begin{aligned} 0 &= (x - c_x/c_w)^2 + (tx - c_y/c_w)^2 - r_C^2 \\ &= x^2 - 2xc_x/c_w + c_x^2/c_w^2 + t^2x^2 - 2txc_y/c_w + c_y^2/c_w^2 - r_C^2 \\ &= x[(1+t^2)x - 2(c_x + tc_y)/c_w] \end{aligned}$$

and hence the Euclidean x-coordinate of $\phi(t)$ is

$$\phi_x(t) = \frac{2(c_x + tc_y)}{c_w(1+t^2)} =: \frac{\varphi_x(t)}{\varphi_w(t)}.$$

Note that the solution is rational because the constant terms cancel out. To compute the Euclidean y-coordinate of $\phi(t)$ we set $x = y/t$ and similarly get

$$\begin{aligned} 0 &= t^2[(y/t - c_x/c_w)^2 + (y - c_y/c_w)^2 - r_C^2] \\ &= y^2 - 2ytc_x/c_w + t^2c_x^2/c_w^2 + t^2y^2 - 2yt^2c_y/c_w + t^2c_y^2/c_w^2 - t^2r_C^2 \\ &= y[(1+t^2)y - 2(tc_x + t^2c_y)/c_w], \end{aligned}$$

hence

$$\phi_y(t) = t\phi_x(t) = \frac{2(tc_x + t^2c_y)}{c_w(1+t^2)} =: \frac{\varphi_y(t)}{\varphi_w(t)}.$$

We now consider the transformation $\phi_C(t)$ of the curve $\phi(t)$ into the coordinate system where the center of C is the origin. The homogeneous coordinates (p_x, p_y, p_w) of the point p in the new coordinate system is expressed by the coordinates of the center in the original coordinate system by $(p_x, p_y, p_w) = (-c_x, -c_y, c_w)$. Hence we get

$$\begin{aligned} \phi_C(t) &= \begin{pmatrix} \varphi_x(t) \\ \varphi_y(t) \\ \varphi_w(t) \end{pmatrix} - \begin{pmatrix} c_x(t) \\ c_y(t) \\ c_w(t) \end{pmatrix} = \begin{pmatrix} \varphi_x(t) - (1+t^2)c_x \\ \varphi_y(t) - (1+t^2)c_y \\ \varphi_w(t) \end{pmatrix} \\ &= \begin{pmatrix} (1-t^2)\, c_x + 2tc_y \\ (t^2-1)\, c_y + 2tc_x \\ (1+t^2)\, c_w \end{pmatrix} = \begin{pmatrix} (t^2-1)\, p_x - 2tp_y \\ (1-t^2)\, p_y - 2tp_x \\ (1+t^2)\, p_w \end{pmatrix}. \end{aligned}$$

Taking the unit circle for C with center $(0, 0, 1)$ and reference point $p = (-1, 0, 1)$ this turns out to be the well-known parametrization

$$\frac{1}{1+t^2} \begin{pmatrix} 1-t^2 \\ 2t \end{pmatrix}$$

that is used in [CDR92]. For a rational $t = m/n$ with integral m, n we have

$$\phi_C(m/n) = \begin{pmatrix} (m^2 - n^2)\, p_x - 2mn\, p_y \\ (n^2 - m^2)\, p_y - 2mn\, p_x \\ (m^2 + n^2)\, p_w \end{pmatrix}.$$

In the following code chunk we use this formula to compute $\phi_C(rt)$ for a *rational* rt.

4 ⟨*procedure for the computation of phi_C(rt)* 4⟩≡ (2)

```
void compute_phi_C(
  const integer& m,  const integer& n,
  const rat_point& p,
  integer& phi_x, integer& phi_y, integer& phi_w)
{
  // C is a circle with center in the origin that passes through
  // the point p.
  // Then every rational point on C has the form
  //        phi(n/m) = (phi_x/phi_w,phi_y/phi_w)
  // for integeral l, m where (phi_x, phi_y, phi_w) are as below

  phi_x = (m*m-n*n) * p.X() - 2*m*n * p.Y();
  phi_y = (n*n-m*m) * p.Y() - 2*m*n * p.X();
  phi_w = (m*m+n*n) * p.W();
}
```

Defines:
`compute_phi_C`, used in chunk 6.

3 Solving $\phi_C(t)$ for a given α

For a given angle α, we compute the parameter value $t = t(\alpha)$ such that $\phi_C(t)$ has angle α at C. We call this point p_α. In the coordinate system where the center of C is the origin, p_α is given by $(r_C \cos(\alpha), r_C \sin(\alpha))$. Comparing y-coordinates we see that $r_C \sin(\alpha) = \varphi_y(t)/\varphi_w(t)$ and hence $t = t(\alpha)$ satisfies the following quadratic equation.

$$\begin{aligned}
0 &= (1+t^2)p_w[r_C \sin(\alpha) - \varphi_y(t)/\varphi_w(t)] \\
&= (1+t^2)p_w r_C \sin(\alpha) + (2tp_x + (t^2-1)p_y) \\
&= (1+t^2)(p_x^2+p_y^2)^{1/2}\sin(\alpha) + 2tp_x + (t^2-1)p_y \\
&= t^2(A+p_y) + 2tp_x + (A-p_y)
\end{aligned}$$

where $A = (p_x^2+p_y^2)^{1/2}\sin(\alpha)$. Hence for $A = \pm p_y$ this equation degenerates to a line which is the case if and only if $p_\alpha = p$ or p_α is the opposite of p at C. If $A \neq \pm p_y$ we can divide by $A + p_y$ such that we find

$$\begin{aligned}
0 &= t^2 + 2t\frac{p_x}{A+p_y} + \frac{A-p_y}{A+p_y} \\
&= \left(t + \frac{p_x}{A+p_y}\right)^2 + \frac{(A+p_y)(A-p_y) - p_x^2}{(A+p_y)^2} \\
&= \left(t + \frac{p_x}{A+p_y}\right)^2 - \frac{(p_x^2+p_y^2)\cos^2(\alpha)}{(A+p_y)^2}.
\end{aligned}$$

Hence we get two solutions

$$t(\alpha) = \frac{-p_x \pm \cos(\alpha)(p_x^2+p_y^2)^{1/2}}{\sin(\alpha)(p_x^2+p_y^2)^{1/2} + p_y} = \frac{-\overline{p_x} \pm \cos(\alpha)}{\sin(\alpha) + \overline{p_y}} \tag{1}$$

where $(\overline{p_x}, \overline{p_y}) = (p_x^2 + p_y^2)^{-1/2}(p_x, p_y)$. Note that $(\overline{p_x}, \overline{p_y})$ is equal to $(\cos(\alpha_p), \sin(\alpha_p))$ for the angle α_p of p at C. We find the correct sign of $\cos(\alpha)$ in Equation 1 by taking the limit

$$\lim_{\alpha \to \alpha_p} t(\alpha) = \lim_{\alpha \to \alpha_p} \frac{\pm\cos(\alpha) - \cos(\alpha_p)}{\sin(\alpha) + \sin(\alpha_p)} = \frac{\pm\cos(\alpha_p) - \cos(\alpha_p)}{2\sin(\alpha_p)}$$

which is 0 for the $+$ sign and $-\cot(\alpha_p)$ for the $-$ sign. It is easy to verify that $-\cot(\alpha_p)$ is in fact the slope of the tangent of C at p, hence we have to take the $-$ sign in Equation 1.

Expanding $t(\alpha)$ with the term $\cos(\alpha) - \cos(\alpha_p)$ we get the alternative representation

$$\begin{aligned} t(\alpha) &= -\frac{\cos(\alpha) + \cos(\alpha_p)}{\sin(\alpha) + \sin(\alpha_p)} = \frac{\cos^2(\alpha_p) - \cos^2(\alpha)}{(\sin(\alpha) + \sin(\alpha_p))(\cos(\alpha) - \cos(\alpha_p))} \\ &= \frac{1 - \sin^2(\alpha_p) - \cos^2(\alpha)}{(\sin(\alpha) + \sin(\alpha_p))(\cos(\alpha) - \cos(\alpha_p))} = \frac{\sin^2(\alpha) - \sin^2(\alpha_p)}{(\sin(\alpha) + \sin(\alpha_p))(\cos(\alpha) - \cos(\alpha_p))} \\ &= \frac{\sin(\alpha) - \sin(\alpha_p)}{\cos(\alpha) - \cos(\alpha_p)} = \frac{\sin(\alpha) - \overline{p_y}}{\cos(\alpha) - \overline{p_x}} \qquad (2) \end{aligned}$$

We can assume without loss of generality that $\alpha, \alpha_p - \pi \in [0, \pi/4]$; in this case $t \in [0, 1]$. Note that in general $t(\alpha)$ takes arbitrary real values. This assumption also guarantees that the floating point evaluation of the representation of $t(\alpha)$ given by Equation 2 is robust because every subterm is positive.

In the next code chunk we do the computation of $t(\alpha)$ under the precondition that $\alpha \in [0, \pi/2]$ and that $\alpha_p \in [\pi, 5/4\pi]$. If necessary, we take $\alpha' = \pi/2 - \alpha$ by switching $\cos(\alpha)$ and $\sin(\alpha)$. Note that the condition $\alpha_p \in [\pi, 5/4\pi]$ is equivalent to $0 \geq p_y \geq p_x$.

5 ⟨*procedure for the computation of t(alpha)* 5⟩≡ (2)

```
// For given alpha we determine a value t = t(alpha) such that
//       angle(phi(t)) = alpha.
// The solution is
//       t = (r*sin(alpha)-px) / (r*cos(alpha)-py)
// where
//       r = sqrt(px^2+py^2)

double compute_t(double alpha, const rat_point& p)
{
  // preconditions: 0 <= alpha <= pi/2,
  //                 0 >= p.Y() >= p.X()

  double si = sin(alpha), co = cos(alpha);
  double sinalpha, cosalpha;
  if (co < si) {  sinalpha = co; cosalpha = si; }
  else         {  sinalpha = si; cosalpha = co; }

  double px = p.X().todouble();
  double py = p.Y().todouble();

  integer ipx = p.X(), ipy = p.Y();
  integer rsq = ipx*ipx+ipy*ipy;
  double r = sqrt(rsq.todouble());
  double t = (r*sinalpha-py) / (r*cosalpha-px);
  return(t);
}
```

Defines:
`compute_t`, used in chunk 6.

4 The main procedure

We describe now the main procedure *rat_point_on_circle*. In this procedure the input consists of a circle C passing through three rational points p, q, r and an angle α in degrees together with an error bound ϵ for α. The procedure computes a rational point ϕ_α on C such that $|\angle(\phi_\alpha) - \alpha| < \epsilon$ and the bit length of ϕ_α is as small as possible.

Remember that $t(\alpha) \in [0,1]$ only for $\alpha \in [0, \pi/4]$ and $\angle(p) \in [\pi, 5/4\pi]$. Hence we transform α to an angle $\alpha' \in [0, \pi/4]$ and p to a point p' such that $\angle(p) - \pi \in [0, \pi/4]$. The idea is now to compute a *double* approximation t' of $t(\alpha)$ together with an error bound $\delta > 0$ such that any parameter in $I_t = [t' - \delta, t' + \delta]$ is mapped by ϕ_C to a point whose angle is in $[\alpha - \epsilon, \alpha + \epsilon]$. Here there is a restriction that $\epsilon \geq 2^{-48}$, as we will see later. Then we compute a rational r_t in I_t whose bit size is as small as possible and insert this parameter into our parametrization ϕ_C, giving the desired result. The outline of our procedure is as follows.

1. Compute the center c of C.
2. Convert α from degrees to radians and transform α to an angle α' such that $\alpha' \in [0, \pi/4]$.
3. Transform p into the coordinate system with center c and next to a point p' such that $\angle(p') - \pi \in [0, \pi/4]$.
4. Compute a parameter $t' = t(\alpha')$ such that $\angle(\phi_C(t')) \approx \alpha'$ by double–precision floating point approximation.
5. Compute an error bound δ for t' as described above.
6. Compute a rational r_t with smallest possible bit length such that $t' - \delta < r_t < t' + \delta$.
7. Compute $\phi_C(r_t)$.
8. Compute ϕ_α from $\phi_C(r_t)$ by back transformation to the original coordinate system.

6 ⟨*main procedure* 6⟩≡ (2)

```
// The following main procedure computes a rat_point phi_alpha
// on the circle C whose angle at C satisfies
//     |angle(phi_alpha)-alpha| < epsilon
// alpha and the corresponding error bound epsilon are in degrees

rat_point rat_point_on_circle(
  const rat_circle& C, double alpha, double epsilon)
{
  // precondition: epsilon >= 2^{-48}

  rat_point phi_alpha;
  rat_point c = C.center();

  double alpha_radians, alpha_trans;
  ⟨compute alpha_radians and alpha_trans 7a⟩
  double sinalpha = sin(alpha_radians);
  double cosalpha = cos(alpha_radians);

  rat_point p_trans;
  ⟨transform p to p_trans 7b⟩

  double t = compute_t(alpha_trans,p_trans);
```

```
    double delta;
    ⟨compute delta from t and epsilon 8a⟩
    rational rt_low = t - delta, rt_high = t + delta;
    if (rt_high > 1) rt_high = 1;
    if (rt_low  < 0) rt_low  = 0;
    rational rt = small_rational_between(rt_low, rt_high);
    integer m = rt.numerator(), n = rt.denominator();
    integer phi_x, phi_y, phi_w;
    compute_phi_C(m,n,p_trans,phi_x,phi_y,phi_w);
    ⟨transform phi_C back to phi_alpha 8b⟩
    ⟨check the result phi_alpha 9a⟩
    return phi_alpha;
  }
```

Defines:
    `rat_point_on_circle`, used in chunks 2 and 8–10.
Uses `compute_phi_C` 4, `compute_t` 5, `main` 10, and `small_rational_between` 9b.

In the next sections we fill out the details in the previous outline in successive order. The implementation of Step 2 is straightforward.

7a ⟨*compute alpha_radians and alpha_trans* 7a⟩≡ (6)

```
  {
    int sector_360 = (int) floor(alpha/360);
    alpha -= sector_360 * 360;
    int sector_90 = (int) floor(alpha/90);
    double alpha_90 = alpha - sector_90 * 90;
    alpha_radians = alpha*asin(1)/90;
    alpha_trans = alpha_90*asin(1)/90;
  }
```

In step 3 we first shift the origin into c, translating p by $-c$. Here we can avoid to duplicate a factor $p.W(\,)$ that is already contained in $c.W()$ since p is one of the three points that define c. Next we achieve $\angle(p) - \pi \in [0, \pi/4]$ by taking dx and dy as the negative of the maximum and the minimum of $|p_x|$ and $|p_y|$, respectively.

7b ⟨*transform p to p_trans* 7b⟩≡ (6)

```
  {
    rat_point p = C.point1();
    integer pfactor = c.W()/p.W();
    // p.W() is a divisor of c.W()
    integer dx = pfactor*p.X()-c.X();
    integer dy = pfactor*p.Y()-c.Y();
    if (abs(dx) >= abs(dy))
       p_trans = rat_point(-abs(dx),-abs(dy),c.W());
    else
       p_trans = rat_point(-abs(dy),-abs(dx),c.W());
  }
```

For Step 5 we first analyze the error in the floating point computation of t in function *compute_t*. Remember that there we basically evaluated the fraction $t = (\sin(\alpha)r - p_y)/(\cos(\alpha)r - p_x)$ for $r = (p_x^2 + p_y^2)^{1/2}$. We ignore the error in the conversion of α from degrees to radians, that is, we consider the computed value of α in radians as exact.

We follow the method for error estimation described in [Bur96], pages 61-62, which uses the notion of the *index* $\mathrm{ind}(E)$ of an arithmetic expression E. Note that all subexpressions of our formula for t are nonnegative since p_x and p_y are guaranteed to be negative, hence the error in the computation of t can be estimeted off-line. In particular, the error in the evaluation $\widetilde{t}$ of t is expressed by the relative precision eps of each operation (in our case $\mathrm{eps} = 2^{-53}$) and the index of t:

$$|t - \widetilde{t}| \le |\widetilde{t}| \ \mathrm{ind}(E) \ \mathrm{eps}.$$

We have $\mathrm{ind}(\sin\alpha) = \mathrm{ind}(\cos\alpha) = 1$ since the IEEE standard guarantees that the results of both functions are exactly rounded to the used precision. It is easy to see that the numerator and denominator have index 6 each. Ignoring higher-order terms we get $\mathrm{ind}(t) = 13$, hence the error of $\widetilde{t}$ is bounded by $13|\,\widetilde{t}|\ 2^{-53}$.

The error in $\alpha(t)$ that results from the error in t is not difficult to estimate. Remember that

$$t(\alpha) = \frac{\sin(\alpha) - \sin(\alpha_p)}{\cos(\alpha) - \cos(\alpha_p)} = -\frac{2\cos\frac{\alpha+\alpha_p}{2}\sin\frac{\alpha-\alpha_p}{2}}{2\sin\frac{\alpha+\alpha_p}{2}\sin\frac{\alpha-\alpha_p}{2}} = -\cot\frac{\alpha+\alpha_p}{2}.$$

This implies that

$$\alpha(t) = 2\mathrm{arccot}(-t) - \alpha_p = 2\arctan(t) + \pi - \alpha_p$$

and hence the derivation of $\alpha(t)$ is given by $\alpha'(t) = 2/(1+t^2)$ wich is at most 2.

If we take any parameter t' in $[\widetilde{t} - \delta, \widetilde{t} + \delta]$ for some $\delta > 0$, the error of the angle $\alpha(t')$ of the point $\phi_C(t')$ is by the mean value theorem bounded by $2(13\ |\widetilde{t}|\ \mathrm{eps} + \delta)$ which is smaller than ϵ if and only if

$$\delta \le (\epsilon - 26\ |\widetilde{t}|\ \mathrm{eps})/2,$$

provided that the right-hand side is positive. This is certainly true if $\epsilon \ge 2^{-48}$, because $|\widetilde{t}| \le 1$. We exit with an error message if the computed δ is negative.

8a ⟨*compute delta from t and epsilon* 8a⟩≡ (6)

```
{
  double epsilon_radians = epsilon * asin(1) /90;
  delta = epsilon_radians - 26*t*ldexp(1,-53);
  delta = ldexp(delta,-1);
  if (delta <= 0)
    error_handler(1,
    "rat_point_on_circle called with epsilon_degree too small");
}
```

Uses `rat_point_on_circle` 6.

In this section we compute the final result ϕ_α by transforming our intermediate result $\phi_C(r_t)$.

8b ⟨*transform phi_C back to phi_alpha* 8b⟩≡ (6)

```
{
  integer xresult, yresult, wresult;
  if (fabs(cosalpha) >= fabs(sinalpha))
    { xresult = phi_x; yresult = phi_y; }
  else
    { xresult = phi_y; yresult = phi_x; }
  wresult = phi_w;
  if (cosalpha < 0) xresult = -xresult;
  if (sinalpha < 0) yresult = -yresult;
  phi_alpha = rat_point(
    xresult+(m*m+n*n)*c.X(),yresult+(m*m+n*n)*c.Y(),wresult);
}
```

We first check that *phi_alpha* indeed lies on the given circle C and second we check the accuracy of sine and cosine of the resulting angle *alpha* using *double*s.

9a ⟨*check the result phi_alpha* 9a⟩≡ (6)

```
{
  if (!C.contains(phi_alpha))
    error_handler(1,"rat_point_on_circle: result not on circle");
  rat_point phi_C = phi_alpha.translate(-c.X(),-c.Y(),c.W());
  double x = phi_C.xcoordD(), y = phi_C.ycoordD();
  double radius = sqrt(x*x+y*y);
  if (fabs(x/radius - cosalpha) > epsilon)
     error_handler(1,"rat_point_on_circle: phi_x/phi_w not correct");
  if (fabs(y/radius - sinalpha) > epsilon)
     error_handler(1,"rat_point_on_circle: phi_y/phi_w not correct");
}
```

Uses `rat_point_on_circle` 6.

We still have to implement the procedure *small_rational_between* for Step 6. The function has two parameters x, y such that $0 \leq x \leq y \leq 1$. The function returns a rational z between x and y where the bit size of z is small possible. It is implemented according to Figures 6 and 7 from [CDR92].

9b ⟨*procedure for rational approximations* 9b⟩≡ (2)

```
rational small_rational_between(const rational& x, const rational& y)
{
  // precondition: 0 <= x <= y <= 1

  integer p0 = 0, q0 = 1, p1 = 1, q1 = 1, r;
  rational z;

  if ( (y > 1) || (x < 0) || (y < x) )
    error_handler(1,
      "small_rational_between: precondition violated");
  if (x == 0) return x;
  if (y == 1) return y;

  while(1)
  {
```

```
  // we have the invariant (I):
  //        0 <= p0/q0 < x <= y < p1/q1 <= 1

     if (rational(p0,q0) >= x)
       error_handler(1,
         "find_small_rational_between: invariant violated");
     r = floor( (p1-q1*x)/(q0*x-p0) );
     p1 = r*p0+p1; q1 = r*q0+q1;
     z = rational(p1,q1);
     if (z <= y) return z;

  // Invariant (I) from above holds again

     if (rational(p1,q1) <= y)
        error_handler(1,
          "find_small_rational_between: invariant violated");
     r = floor( (q0*y-p0)/(p1-q1*y) );
     p0 = p0+r*p1; q0 = q0+r*q1;
     z = rational(p0,q0);
     if (z >= x) return z;
  }
}
```

Defines:
small_rational_between, used in chunk 6.

5 A Test Program

The following test program constructs an arbitrary number n of rational points on a circle such that the points have approximate angles $2\pi i/n, i = 1, \ldots, n$. The program asks the user for the number n of points and the error bound ϵ and then reads three points defining a rational circle in a LEDA window.

10 ⟨*test program* 10⟩≡ (2)

```
#include <LEDA/window.h>
#include <LEDA/circle.h>

main()
{
  cout << "How many rational points shall be constructed? ";
  int n;
  cin >> n;
  cout << "Give an upper bound bigger than 1E-12 for the error!\n";
  double epsilon;
  cin >> epsilon;

  window W;
  W.display();
  cout << "Specify the circle by three points! \n";
  point p_dbl, q_dbl, r_dbl;
  W.read_mouse(p_dbl);
  W.read_mouse(q_dbl);
  W.read_mouse(r_dbl);
  rat_point p((int) p_dbl.xcoord(), (int) p_dbl.ycoord());
```

```
    rat_point q((int) q_dbl.xcoord(), (int) q_dbl.ycoord());
    rat_point r((int) r_dbl.xcoord(), (int) r_dbl.ycoord());

    rat_circle C(p,q,r);
    W.draw_circle(C.to_circle());
    double alpha_deg;
    rat_point result;
    for (int i = 1; i<= n; i++)
    {
      alpha_deg = 360*i/n;
      result = rat_point_on_circle(C,alpha_deg,epsilon);
      W.draw_point(result.xcoordD(),result.ycoordD());
    }
    cout << "type return to continue\n";
    getchar();
  }
```

Defines:
`main`, used in chunk 6.
Uses `rat_point_on_circle` 6.

References

[Bur96] Christoph Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections.* PhD thesis, Universitaet des Saarlandes, 1996.

[CDR92] J. Canny, B.R. Donald, and E.K. Ressler. A rational rotation method for robust geometric algorithms. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 251–260, 1992.

[Hof90] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufman, San Francisco, 1990. Chapter 4.

6 Code chunks

7 Identifiers

I N F O R M A T I K

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid *E*-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves

MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees
MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Sch'onherr	On the Design of CGAL, the Computational Geometry Algorithms Library
MPI-I-98-1-006	K. Jansen	A new characterization for parity graphs and a coloring problem with costs
MPI-I-98-1-005	K. Jansen	The mutual exclusion scheduling problem for permutation and comparability graphs
MPI-I-98-1-004	S. Schirra	Robustness and Precision Issues in Geometric Computation
MPI-I-98-1-003	S. Schirra	Parameterized Implementations of Classical Planar Convex Hull Algorithms and Extreme Point Compuations
MPI-I-98-1-002	G.S. Brodal, M.C. Pinotti	Comparator Networks for Binary Heap Construction
MPI-I-98-1-001	T. Hagerup	Simpler and Faster Static AC^0 Dictionaries
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values
MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda \rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-004	W. Charatonik, A. Podelski	Solving set constraints for greatest models
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics
MPI-I-97-2-001	D.A. Basin, S. Matthews, L. Viganò	Labelled modal logics: quantifiers
MPI-I-97-1-028	M. Lermen, K. Reinert	The Practical Use of the $\mathcal{A}^*$ Algorithm for Exact Multiple Sequence Alignment
MPI-I-97-1-027	N. Garg, G. Konjevod, R. Ravi	A polylogarithmic approximation algorithm for group Steiner tree problem
MPI-I-97-1-026	A. Fiat, S. Leonardi	On-line Network Routing - A Survey
MPI-I-97-1-025	N. Garg, J. Könemann	Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems
MPI-I-97-1-024	S. Albers, N. Garg, S. Leonardi	Minimizing Stall Time in Single and Parallel Disk Systems
MPI-I-97-1-023	S.A. Leonardi, A.P. Marchetti-Spaccamela	Randomized on-line call control revisited
MPI-I-97-1-022	E. Althaus, K. Mehlhorn	Maximum Network Flow with Floating Point Arithmetic
MPI-I-97-1-021	J.F. Sibeyn	From Parallel to External List Ranking
MPI-I-97-1-020	G.S. Brodal	Finger Search Trees with Constant Insertion Time
MPI-I-97-1-019	D. Alberts, C. Gutwenger, P. Mutzel, S. Näher	AGD-Library: A Library of Algorithms for Graph Drawing
MPI-I-97-1-018	R. Fleischer	On the Bahncard Problem