Journal of Graphics Tools

ISSN: 1086-7651 (Print) (Online) Journal homepage: http://www.tandfonline.com/loi/ujgt19

Fast and Accurate Circle-Circle and Circle-Line 3D Distance Computation

David Vranek

To cite this article: David Vranek (2002) Fast and Accurate Circle-Circle and Circle-Line 3D Distance Computation, Journal of Graphics Tools, 7:1, 23-31, DOI: 10.1080/10867651.2002.10487552

To link to this article: http://dx.doi.org/10.1080/10867651.2002.10487552

Vol. 7, No. 1: 23–32

Fast and Accurate Circle-Circle and Circle-Line 3D Distance Computation

David Vranek
Technical University of Brno

Abstract. This paper presents a novel high-precision algorithm for the calculation of the minimal distance between two circles in three dimensions. Contrary to the common approach that mainly utilizes a solution of an eighth-degree polynomial equation, the proposed algorithm is based on directly using the distance function and its derivatives. As the theory is not too complicated, the algorithm is very easy to implement. The paper also suggests a small modification of the circle-circle distance algorithm for the circle-line distance calculation. Results of computational simulation compared with Eberly's Wild Magic library are given and summarized in detail at the end of the paper. They show that the proposed algorithm always correctly finds the global minimum with much higher accuracy at the same speed in the circle-circle distance calculation case, whereas circle-line calculation is approximately four times faster. The source code is available online at the web site given at the end of this paper.

1. Introduction

Finding the minimal distance between two circles in three-dimensions is a common geometrical operation. It has a simple theoretical solution based on an inference of a distance equation and its conversion to an eighth-degree polynomial equation, which is then solved by a standard polynomial root finder. But what is easy in mathematical theory can be difficult in computational practice. If the numerical stability and accuracy are the main goal,

1086-7651/02 $0.50 per page

then the solution of the relatively high-order polynomial equation need not be the best to meet this goal. Let us imagine that we are searching for a possible collision of two cylinders moving along a defined path. Since distance calculation is one of the collision detection approaches [Lin, Gottschalk 98], we have to find the minimal distance between the cylinders for each point on the path. This means that we evaluate the minimal distance between surface- surface, surface-base and base-base pairs and then take the shortest one. The calculation of the minimal distance between parts of the cylinders can be transferred to line-line, circle-line and circle-circle distance subtasks.[1] As the number of evaluated path points is usually large, we can readily face a problem of evaluating hundreds of thousands of possible cases of the line-line, circle-line, circle-circle distance cases. It is apparent that a routine for the minimal distance calculation should never fail.

This paper proposes a novel approach for a fast and accurate circle-circle, circle-line distance calculation, which does not have the shortcomings of the polynomial equation solution and always provides a correct result. Computational results based on a simulation are also given at the end of the paper.

2. Circle-Circle Distance Calculation

In order to calculate the minimal distance d between two circles, we first need to know the minimal distance between a point and the circle. Let us consider a circle a in 3D , which is represented by a center $\vec{C}_a$, radius R_a and a plane $\rho : \vec{N}_a \cdot (\vec{X} - \vec{C}_a) = 0$ containing the circle, where $\vec{N}_a$ is a vector of a unit length (see Figure 1). Derivation of point-circle minimal distance is given in Eberly's book [Eberly 00, p. 69] and can be simplified into the formula given in Equation 1:

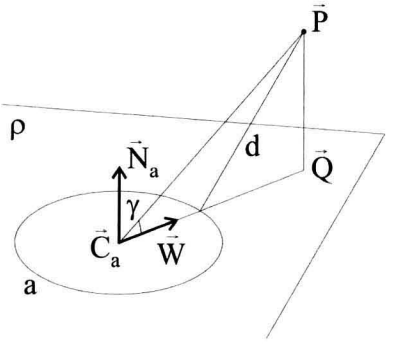

Figure 1. Point-Circle distance calculation.

[1] Appendices are available online at http://www.acm.org/jgt/papers/Vranek02/

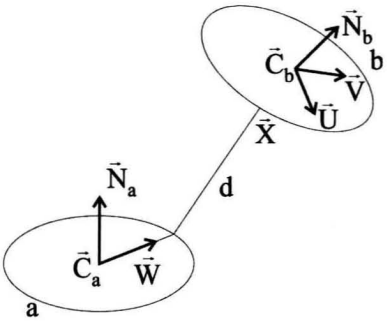

Figure 2. Circle-Circle distance calculation.

$$d^2 = R_a^2 + \left|\vec{C_a} - \vec{P}\right|^2 - 2R_a \left|\vec{P} - \vec{C_a} - \left(\vec{N_a} \cdot (\vec{P} - \vec{C_a})\right) \cdot \vec{N_a}\right|. \quad (1)$$

Let us introduce another circle b, which is represented by a center $\vec{C_b}$, radius R_b and a normal vector $\vec{N_b}$ perpendicular to the plane containing the circle (see Figure 2). The circle equation can be expressed as

$$\vec{X}(t) = \vec{C_b} + R_b(\cos(t)\vec{U} + \sin(t)\vec{V}), \quad (2)$$

where $\vec{U}$ and $\vec{V}$ are unit length vectors forming a right-handed orthonormal coordinate system together with vector $\vec{N_b}$. By substituting $\vec{P} = \vec{X}(t)$ from Equation (2) into Equation (1), we obtain a function giving the squared distance between the point $\vec{X}(t)$ on the circle a and the circle b which can be converted to

$$f(t) = a_9 \cos t + a_8 \sin t + a_7 + a_6\sqrt{a_5 \cos^2 t + a_4 \sin^2 t + a_3 \cos t + a_2 \sin t + a_1 \cos t \sin t + a_0}. \quad (3)$$

This formula will be called the *distance function.* The coefficients are as follows

$$\begin{aligned}
a_9 &= 2R_b\vec{C_b} \cdot \vec{U} - 2R_b\vec{C_a} \cdot \vec{U},\\
a_8 &= 2R_b\vec{C_b} \cdot \vec{V} - 2R_b\vec{C_a} \cdot \vec{V},\\
a_7 &= R_a^2 + R_b^2 + \vec{C_a} \cdot \vec{C_a} + \vec{C_b} \cdot \vec{C_b} - 2\vec{C_a} \cdot \vec{C_b},\\
a_6 &= -2R_a,\\
a_5 &= -(R_b\vec{N_a} \cdot \vec{U})^2,\\
a_4 &= -(R_b\vec{N_a} \cdot \vec{V})^2,\\
a_3 &= 2R_b\vec{C_b} \cdot \vec{U} - 2R_b\vec{C_a} \cdot \vec{U} - 2R_b(\vec{N_a} \cdot \vec{C_b} - \vec{N_a} \cdot \vec{C_a})\vec{N_a} \cdot U,\\
a_2 &= 2R_b\vec{C_b} \cdot \vec{V} - 2R_b\vec{C_a} \cdot \vec{V} - 2R_b(\vec{N_a} \cdot \vec{C_b} - \vec{N_a} \cdot \vec{C_a})\vec{N_a} \cdot \vec{V},\\
a_1 &= -2R_b^2\vec{N_a} \cdot \vec{U}\vec{N_a} \cdot \vec{V},\\
a_0 &= R_b^2 + \vec{C_b} \cdot \vec{C_b} + \vec{C_a} \cdot \vec{C_a} - 2\vec{C_a} \cdot \vec{C_b} - (\vec{N_a} \cdot \vec{C_b} - \vec{N_a} \cdot \vec{C_a})^2.
\end{aligned}$$

To get the minimal squared distance, we need to minimize $f(t)$, which can be symbolically written as $f(t) = \min$. There are two ways to do this minimization. The first one is based on finding the roots of a polynomial equation, and the second one, proposed by this paper, utilizes numerical methods working directly with $f(t)$ and its derivatives.

2.1. *Standard Solution*

The standard way to find the minimum is to differentiate $f(t)$ with respect to t and to set $f'(t) = 0$. Since $f'(t) = 0$ can be easily converted to a trigonometric equation, the solution is based on a conversion of the trigonometric equation to a polynomial equation of the eighth degree with unknown $\cos t$ or $\sin t$ [Eberly 00].

Unfortunately, such a solution usually requires finding all eight roots numerically. Furthermore, double squaring of $f'(t) = 0$, required for the conversion to the polynomial equation, introduces relatively significant numerical inaccuracy into the root-finding procedure. In addition, some multiple or closely related roots can emerge, which are always difficult to find. To avoid these issues, a special numerical technique for the minimization of $f(t)$ is discussed in the following section.

2.2. *Numerical Minimization of the Distance Function*

The idea of numerical minimization is relatively simple. It can be seen from the distance equation (3), that solving $f(t) = 0$ instead of $f(t) = \min$ leads to a polynomial equation of the fourth degree, $P_4(\cos t) = 0$. Even if the equation does not provide the minimum, the fact that it is just a fourth-degree polynomial can significantly help in the minimum search.

Providing that we localize one minimum, we can shift $f(t)$ down to touch the t-axis at this minimum (see Figure 3). And as the minimum of the shifted $f(t)$ creates a double root, the deflation of $P_4(\cos t)$ by this double root gives only a polynomial equation of the second degree! The solution of $P_2(\cos t) = 0$ can provide a new interval bracketing the global minimum or detect that the global minimum has already been found!

This is the basic idea that we will now elaborate on further. First, a brief analysis of $f(t)$ is required. The function is smooth (continuous and differentiable) on the interval $-\pi \leq t < \pi$ and periodic with period 2π. As the solution of $f(t) = 0$ can have at most four roots, it is possible to prove that the function has either one minimum and one maximum or two minima and two maxima on the interval $-\pi \leq t < \pi$. Knowing these basic facts about $f(t)$, we can start with the search for the first minimum. When searching for the minimum of $f(t)$, any numerical minimum search method can be employed. As

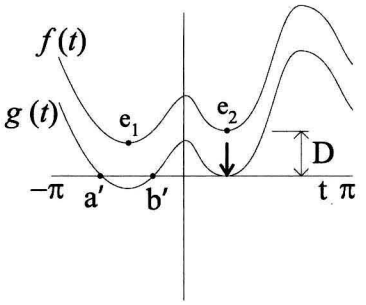

Figure 3. Offsetting of the distance function.

$f(t)$ is a smooth function, there are a number of numerical methods which will always succeed. One such method is a modified Brent's method for minimum search with first derivatives [Press et al. 86].

All methods of minimum search usually require a starting bracketing triplet of points such, that for $a < b < c$, $f(b)$ is less than both $f(a)$ and $f(c)$. This condition ensures that the function has a minimum in the interval (a, c). Since the function $f(t)$ is periodic, it is sufficient to select any three points (say $a = -2\pi/3$, $b = 0$, $c = 2\pi/3$) and assuming that $f(a) \neq f(b) \neq f(c)$ ($f(t)$ is periodic), we can always reorder the points a, b, c to form the bracketing triplet (see Figure 4). By satisfying the starting condition for the numerical minimum search, we can use Brent's algorithm to find the minimum.

With the first minimum localised, the following two cases can arise:

1. The global minimum (point e_1) has already been found. This case occurs when the function $f(t)$ has either one minimum or the global one was found by chance.

2. Only the local minimum (point e_2) was found so we need to search for the global one.

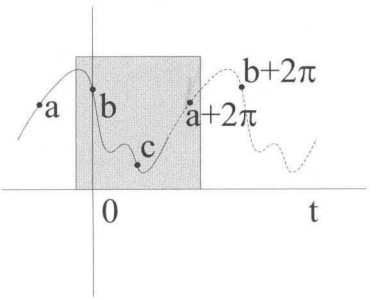

Figure 4. Bracketing triplet.

In both cases, we still do not know which minimum was found so we need to continue with further computations.

Assuming that we localized the local minimum e_2 for the parameter $t = t_2$, we can shift the distance function $f(t)$ by offset $D = f(t_2)$ to get a new function,

$$g(t) = f(t) - D.$$

The next step is to solve $g(t) = 0$; we obtain the solution by conversion to the polynomial equation of the fourth degree. Using the substitution, $z = \cos t$, the conversion can be written as

$$g(t) = 0 \rightarrow P_4(z) = 0;$$

we need to solve

$$b_4 z^4 + b_3 z^3 + b_2 z^2 + b_1 z + b_0 = 0,$$

where

$$\begin{aligned} b_4 &= c_6^2 + c_5^2, \\ b_3 &= 2c_1 c_5 + 2c_4 c_6, \\ b_2 &= c_1^2 - c_5^2 + 2c_3 c_6 + c_4^2, \\ b_1 &= -2c_1 c_5 + 2c_3 c_4, \\ b_0 &= c_3^2 - c_1^2, \end{aligned}$$

and

$$\begin{aligned} c_1 &= 2a_8(a_7 - D) - a_6^2 a_2, \\ c_2 &= a_6^2 a_4, \\ c_3 &= (a_7 - D)^2 - a_6^2 a_0 + a_8^2 - a_6^2 a_4, \\ c_4 &= 2a_9(a_7 - D) - a_6^2 a_3, \\ c_5 &= 2a_9 a_8 - a_6^2 a_1, \\ c_6 &= -a_8^2 + a_6^2 a_4 + a_9^2 - a_6^2 a_5. \end{aligned}$$

Since the offsetting made the root t_2 a double root of both $g(t)$ and $P_4(z)$, deflating $P_4(z)$ by the double root $z_2 = \cos(t_2)$ gives only a second degree polynomial

$$P_2(z) = P_4(z)/(z - z_2)^2$$

which has a well known analytical solution. Now, if P_2 has no solution, we can be certain that the root t_1 gives the global minimum. Otherwise points a' and b' bracketing the global minimum are obtained. Brent's algorithm for searching for the root of $f'(t) = 0$ can be employed in this search.

3. Circle-Line Distance Calculation

Let us introduce a line $\vec{L}(t) = \vec{B} + t\vec{M}$. By substituting $\vec{P} = \vec{L}(t)$ into Equation (1) we obtain a function which gives the squared distance between the point on the line and the circle, so

$$f(t) = a_6 t^2 + a_5 t + a_4 + a_3\sqrt{a_2 t^2 + a_1 t + a_0}, \tag{4}$$

where

$$\begin{aligned} a_6 &= \vec{M} \cdot \vec{M}, \\ a_5 &= 2\vec{D} \cdot \vec{M}, \\ a_4 &= \vec{D} \cdot \vec{D} + R_a^2, \\ a_3 &= -2R_a, \\ a_2 &= \vec{E} \cdot \vec{E}, \\ a_1 &= 2\vec{E} \cdot \vec{F} \\ a_0 &= \vec{F} \cdot \vec{F}, \end{aligned}$$

and

$$\begin{aligned} \vec{D} &= \vec{B} - \vec{C_a}, \\ \vec{E} &= \vec{M} - (\vec{N_a} \cdot \vec{M})\vec{N_a}, \\ \vec{F} &= \vec{D} - (\vec{N_a} \cdot \vec{D})\vec{N_a}. \end{aligned}$$

Equation (4) can be minimized by setting its derivative to zero and converting it to a fourth-degree polynomial equation that is then solved. But since the fourth-degree polynomial equation does not have any stable analytical solution, a numerical technique for finding roots of the polynomial has to be employed. Therefore, it is possible to use an alternative approach similar to the circle-circle distance calculation technique that has been already discussed. The main benefit over the standard polynomial solution is that $f(t)$ usually has only one minimum, which can be found very quickly. In this case, it is sufficient to solve $f'(t) = 0$ which has only one root. In the case of two minima, a technique of $f(t)$ shift will provide the global minimum.

4. Computational Results

The performance of the proposed algorithm was verified on a PC with 256 MB memory and an Intel Pentium III processor at 650 MHz, using Windows 2000. The code was written in Microsoft Visual C++ 6.0 and was built up on the Magic Library core [Eberly 00]. The purpose of the code was to evaluate computational speed and numerical accuracy of the algorithm.

	Circle-Circle		Circle-Line	
	double	float	double	float
Dvr speed [n/s]	9140	9660	140100	177800
Mgc speed [n/s]	9130	9650	33900	37900
Dvr error rate	0%	0%	0%	0%
Mgc error rate	5.08%	4.11%	16.3%	2.99%
Mgc max error (ϵ)	$9.85 \cdot 10^{1}$	$3.40 \cdot 10^{38}$	$2.28 \cdot 10^{-3}$	$7.54 \cdot 10^{13}$
Dvr max error (ϵ)	$1.42 \cdot 10^{-14}$	$7.63 \cdot 10^{-6}$	$1.78 \cdot 10^{-15}$	$2.10 \cdot 10^{-5}$

Table 1. Computational evaluation of the circle-circle, circle-line distance calculation. The row speed gives the number of computations per second; row error rate shows percentage of wrongly-calculated minimal distance with respect to Equation (5).

The code was run on a number of test cases ($N > 10^4$) varying position and orientation of the tested objects. Two versions of the testing code were built—one for double precision numbers (8 bytes) and one for float precision numbers (4 bytes). The overall execution time of the code was measured and its numerical accuracy checked. The following criterion was used for accuracy control and error rate computation:

$$|f(t_{\min}) - DST| < \epsilon, \tag{5}$$

where $t_{\min}$ is a numerically-calculated parameter giving the global minimum, DST is a known minimal squared distance and ϵ is the required calculation accuracy value: $1 \cdot 10^{-7}$ for double and $1 \cdot 10^{-4}$ for float code precision version. The error rate value was calculated as a percentage of the unsatisfied criterion (5). As in the case of the proposed algorithm, the precision was always much greater than that required by the accuracy criterion applied only to Eberly's code. The results of the computer simulation compared with Eberly's library are given in Table 1.

The results of the test show that the proposed algorithm always correctly found the global minimum for both precision code versions with much greater accuracy. Circle-circle distance calculation was comparably fast, whereas circle-line calculation was approximately four times faster.

References

[Eberly 00] D. H. Eberly. *3D Game Engine Design.* San Francisco, CA: Morgan Kaufmann Publishers, 2000.

[Lin, Gottschalk 98] M. C. Lin and S. Gottschalk. "Collision detection between geometric models: a survey," in *IMA Conference on Mathematics of Surfaces San Diego, CA*, vol. 1, pp. 602–608, May 1998.

[Press et al. 86] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling. *Numerical Recipes in C*, Cambridge,UK: Cambridge University Press, 1986.

Web Information:

C++ source code for the circle-circle and circle-line tests is available online at http://www.acm.org/jgt/papers/Vranek02/

David Vranek, Technical University of Brno, Faculty of Civil Engineering, Veveri 95, 662 37 Brno, Czech Republic. (vranek@hotmail.com)

Received May 29, 2001; accepted in revised form March 16, 2002.