# Fast distance computation between a point and cylinders, cones, line swept spheres and cone-spheres

Aurélien Barbier and Eric Galin
LIRIS – CNRS
Université Claude Bernard Lyon 1
69622 Villeurbanne Cedex, France
e-mail: eric.galin@liris.cnrs.fr

## Abstract

This paper presents algorithms for computing the minimal distance between a point and a cylinder, a cone, a line swept sphere and a cone-sphere. Some optimizations are provided when queries are performed along a line which may be useful for voxelization applications. Source code is available online.

## 1 Introduction

Cones, cylinders as well as cone-spheres and line swept spheres [3], which will be refered to as tubular primitives, can be considered fundamental primitives as they can be assembled to model generalized cylinders that can produce a variety of organic shapes such as trees [2].

Distance computations may be used to generate level set models as well as adaptive distance fields models [1], or to create implicit surfaces from skeletal elements [5]. Figure 1 shows some implicit surface models created from cylinders and cone-spheres skeletal elements.

Figure 1: Skeletal implicit surface characters created with cone, cylinder and cone-spheres skeletons

Several accelerated distance computation algorithms have been proposed for a variety of primitives such as line segments, circles, triangles or polyhedra: [4] present an overview as well as an implementation of those algorithms.

This paper presents some efficient algorithms to compute the distance between a point and a cylinder, a cone, a line swept sphere or a cone-sphere, which are missing in existing textbooks. Optimizations are provided when queries are performed along a line, which is useful for voxelization applications. C++ code is available on the web site listed at the end if this paper.

## 2 Notations

Cylinders, cones and cylinder spheres and cone-spheres are revolution primitives whose axis will be characterized by two supporting vertices denoted as $\mathbf{a}$ and $\mathbf{b}$. Let $\mathbf{u}$ denote the unit axis vector $\mathbf{u} = (\mathbf{b} - \mathbf{a})/\|\mathbf{b} - \mathbf{a}\|$.

Since we are dealing with surfaces of revolution, the computation of the distance from a point $\mathbf{p}$ in space to a primitive will be performed in the plane of $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{p}$. The point $\mathbf{h}$ will denote the orthogonal projection of $\mathbf{p}$ onto the axis. $l = \|\mathbf{b} - \mathbf{a}\|$ will refer to the length of the axis.

## 3 Cylinder

Cylinders are characterized as follows: let $\mathbf{a}$ and $\mathbf{b}$ denote the end vertices of the axis, let $r$ denote the radius of the cylinder and rounded cylinder. The point $\mathbf{c}$ will denote the center of the segment $[\mathbf{a}, \mathbf{b}]$. Computing the distance between a point $\mathbf{p}$ in space and a cylinder requires the classification of this point in the Voronoï regions of the cylinder. The signed distance along the axis will be denoted as $x = (\mathbf{c} - \mathbf{p}) \cdot \mathbf{u}$. The squared distance to vertex $\mathbf{c}$ will be denoted as $n^2 = (\mathbf{c} - \mathbf{p})^2$. The squared distance to the axis will be denoted as $y^2 = \|\mathbf{p} - \mathbf{h}\|^2 = n^2 - x^2$. Figure 2 represents a cut view of the Voronoï regions of the cylinder.

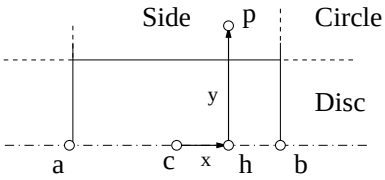


Figure 2: Voronoï regions of a cylinder

Within the plane defined by the three vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{p}$, we can identify six different regions. Processing may be simplified by taking advantage of the symetry properties of the cylinder, which limits the classification of a point $\mathbf{p}$ in the Voronoï regions of the cylinder to four cases. The algorithm proceeds as follows:

1. If $|x| < l/2$, then $\mathbf{h}$ is on the line segment $[\mathbf{ab}]$.

   1.1 If $y^2 < r^2$ then $p$ is inside the cylinder and the distance is 0.

1.2 Otherwise the squared distance to the cylinder is $(y-r)^2$.

2. Otherwise, $|x| > l/2$ then we need to compute the distance to a disc.

   2.1 If $y^2 < r^2$ then $\mathbf{p}$ projects onto the disc and the squared distance is simply $(|x| - l/2)^2$.

   2.2 Otherwise $\mathbf{p}$ projects onto a circle and the squared distance is $(y-r)^2 + (|x| - l/2)^2$.

The order of tests have an influence over the overall performance of the algorithm. Experiments demonstrate that it is better to perform tests over the $x$ coordinate before testing if $y^2 < r^2$. The computational cost in the worst case as well as timings are reported in Table 1.

## 4 Line swept sphere

Line swept spheres are specific cylinders with half sphere ends with radius $r$. Figure 3 represents a cut view of the Voronoï regions of the line swept sphere primitive.

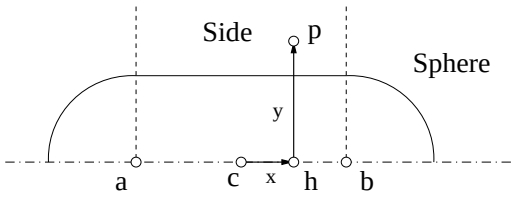


Figure 3: Voronoï regions of a line swept sphere

In this case, the algorithm may be optimized so that it should resemble the cylinder's in many aspects. If the projected point $\mathbf{h}$ lies on the line segment [$\mathbf{ab}$], the computations are the same as in step 1. Otherwise, we need to compute the distance to a sphere as follows:

2. If $|x| > l/2$, then we need to compute the distance to a sphere. Let $n^2 = y^2 + (|x| - l/2)^2$ denote the squared distance to the end vertex of the cylinder. If $n^2 < r^2$ then $\mathbf{p}$ is inside the sphere, otherwise the squared distance is $(n-r)^2$.

## 5 Cone

Let $\mathbf{a}$ and $\mathbf{b}$ denote the end vertices of the axis, let $r_\mathbf{a}$ and $r_\mathbf{b}$ denote the corresponding radii. Without loss of generality, we will assume that $r_\mathbf{a} > r_\mathbf{b}$, and $\delta = r_\mathbf{a} - r_\mathbf{b}$ will refer to the difference between the two radii. The length of the side of the cone will be denoted as $s = (l^2 + \delta^2)^{1/2}$.

As for cylinders and line swept spheres, we need to compute the coordinates of the point $\mathbf{p}$ in a frame attached to the primitive. The signed distance along the axis will be denoted as $x = (\mathbf{a} - \mathbf{p}) \cdot \mathbf{u}$. The squared distance to vertex $\mathbf{a}$ will be denoted as $n^2 = (\mathbf{a} - \mathbf{p})^2$. The squared distance to the axis will be denoted as $y^2 = \|\mathbf{p} - \mathbf{h}\|^2 = n^2 - x^2$.

Figure 4 represents a cut view of the Voronoï regions of the cone which are more complex than the cylinder's. The classification of a point in space $\mathbf{p}$ is performed as follows:

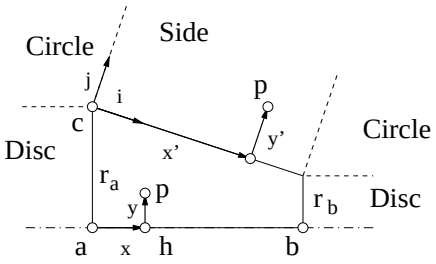


Figure 4: Voronoï regions of a cone

1. If $x < 0$ then two cases may occur:

   1.1 if $y^2 < r_\mathbf{a}^2$ then $\mathbf{p}$ projects on the large cap of the cone and the squared distance is $x^2$.

   1.2 Otherwise we compute the squared distance to the circle is $(y - r_\mathbf{a})^2 + x^2$.

2. If $y^2 < r_\mathbf{b}^2$, then there are two cases:

   2.1 if $x > l$ then $\mathbf{p}$ projects onto the small disc and the squared distance is $(x-l)^2$.

   2.2 Otherwise, $\mathbf{p}$ is inside the cone and the distance is 0.

Further computations require computing the coordinates of $\mathbf{p}$ in a new frame system whose origin is located at point $\mathbf{c}$ and whose orthonormal vectors, denoted as $(\mathbf{i}, \mathbf{j})$, are computed as follows (Figure 4):

$$\mathbf{i} = \left(\frac{\delta}{s}, \frac{l}{s}\right) \qquad \mathbf{j} = \left(\frac{l}{s}, -\frac{\delta}{s}\right)$$

Let $y'$ and $x'$ denote the coordinates of point $\mathbf{p}$ in the new frame. The algorithm proceeds as follows:

3. Compute the coordinates denoted as $y'$ and $x'$ of the point $\mathbf{p}$ in the new frame.

   3.1 If $x' < 0$, then $\mathbf{p}$ projects onto the circle of the large cap and the squared distance is $(y - r_\mathbf{a})^2 + x^2$.

   3.2 Otherwise, if $x' > s$, then $\mathbf{p}$ projects onto the circle of the small cap and the squared distance is $y'^2 + (x' - s)^2$.

   3.3 Otherwise, $\mathbf{p}$ projects onto the side of the cone and the result is $y'^2$.

As for the cylinder, the order of the tests has an influence over the overall performance of the algorithm as well. Although the overall complexity remains the same, timings may change according to the shape of the primitive and to the location of distance queries. It is worth pointing out that some cases overlap. In the second step of this algorithm, we do not need to check the case $x < 0$ since it has already been processed in the first step. Our experiments suggest that it is better to perform tests of the $x$ coordinate to check the caps before computing the $y'$ and $x'$ coordinates which requires expensive square roots evaluations. The computational cost in the worst case as well as timings are reported in Table 1.

## 6 Cone-sphere

Cone-spheres deserve a special case as the computation of the squared distance differs significantly from cones. Let $\mathbf{a}$ and $\mathbf{b}$ denote the centers of two spheres with radii denoted as $r_\mathbf{a}$ and $r_\mathbf{b}$. Without loss of generality, we will assume that $r_\mathbf{a} > r_\mathbf{b}$ and $\delta$ will refer to the difference $r_\mathbf{a} - r_\mathbf{b}$.

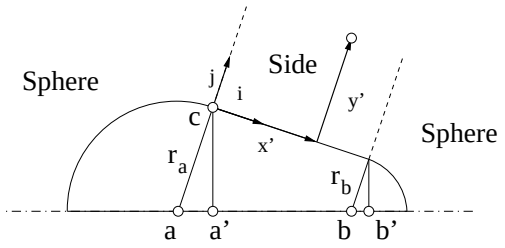


Figure 5: Voronoï regions of a cone-sphere

A cone smoothly joins the two spheres if and only if $l > \delta$. In that case, the length of the side of the cone will be denoted as $s = (l^2 - \delta^2)^{1/2}$. The cone is characterized by two end vertices denoted as $\mathbf{a}'$ and $\mathbf{b}'$ with the corresponding radii $r'_\mathbf{a}$ and $r'_\mathbf{b}$. Let $h_\mathbf{a}$ and $h_\mathbf{b}$ denote the distance between the vertices $\mathbf{a}$ and $\mathbf{a}'$, and $\mathbf{b}$ and $\mathbf{b}'$ respectively. We have:

$$h_\mathbf{a} = \frac{r_\mathbf{a}\delta}{l} \qquad h_\mathbf{b} = \frac{r_\mathbf{b}\delta}{l}$$

$$r'_\mathbf{a} = \frac{r_\mathbf{a}s}{l} \qquad r'_\mathbf{b} = \frac{r_\mathbf{b}s}{l}$$

Figure 5 represents a cut view of the three Voronoï regions of the cone-sphere. Distance computations require computing the coordinates of $\mathbf{p}$ in a rotated frame system whose origin is located at point $\mathbf{c}$ and whose orthonormal vectors $\mathbf{j} = (-\delta/l, s/l)$ and $\mathbf{i} = (s/l, \delta/l)$ are set as depicted in Figure 5. Let $y'$ and $x'$ denote the coordinates of point $\mathbf{p}$ in the new frame. The algorithm proceeds as follows:

1. If $x' < 0$, then we need to compute the distance to a sphere: let $n^2$ denote the distance to vertex $\mathbf{a}$, if $n^2 > r_\mathbf{a}^2$, then the squared distance is $(n - r_\mathbf{a})^2$, otherwise $\mathbf{p}$ is inside the cone-sphere and the distance is 0.

2. If $x' > s$, then we need to compute the distance to the other sphere: let $n^2$ denote the distance to vertex $\mathbf{b}$, if $n^2 > r_\mathbf{b}^2$, then the squared distance is $(n - r_\mathbf{b})^2$, otherwise $\mathbf{p}$ is inside the cone-sphere and the distance is 0.

3. If $y' > 0$, the point $\mathbf{p}$ projects onto the side of the cone-sphere and the result is $y'^2$, otherwise $\mathbf{p}$ is inside the cone-sphere.

As is, the algorithm always involves the computation of $y'$ and $x'$ which is computationally demanding. It may be greatly optimized by adding the following big sphere and small cylinder pre-processing steps (Figure 6) that only require the computation of $y$ and $x$:

1. If $x < h_\mathbf{a}$, then we need to compute the distance to a sphere: let $n^2$ denote te distance to vertex $\mathbf{a}$, if $n^2 > r_\mathbf{a}^2$, then the squared distance is $(n - r_\mathbf{a})^2$, otherwise $\mathbf{p}$ is inside the cone-sphere and the distance is 0.

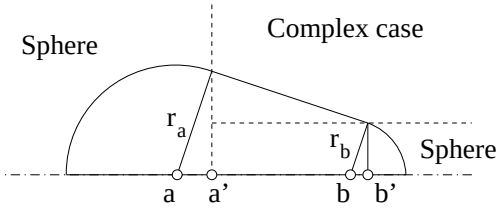


Figure 6: Regions of influence of a cone-sphere that can be directly processed with $y$ and $x$ coordinates

2. If $y^2 < r'^2_\mathbf{b}$, then two cases arise:

   2.1 If $x > l + h_\mathbf{b}$, then we compute the distance to the other sphere: let $n^2 = y^2 + (x - l - h_\mathbf{b})^2$ denote the distance to vertex $\mathbf{b}$, if $n^2 > r_\mathbf{b}^2$, then the squared distance is $(n - r_\mathbf{b})^2$.

   2.2 Otherwise, $\mathbf{p}$ is inside the cone-sphere and the distance is 0.

Although the overall computational cost in the worst case increases by two tests, timings are considerably reduced as reported in Table 1. Without optimizations, timings yield 23.38 seconds for $10^8$ random queries, whereas the optimized algorithm completed in only 13.02 seconds.

## 7 Accelerations

The distance computations may be accelerated so as to take advantage of spatial coherence whenever queries are performed along a line, which is often the case in voxelization applications.

All the previous algorithms are based on the evaluation of $y^2$, $x$ and $x^2$ which are computed as follows: first we evaluate the vector $\mathbf{n} = \mathbf{p} - \mathbf{c}$ between a point in space $\mathbf{p}$ and a vector of the axis $\mathbf{c}$. Then, we compute $x = \mathbf{n} \cdot \mathbf{u}$ and $y$ is defined as $y^2 = \mathbf{n}^2 - x^2$. The overall number of operations involved is 7 multiplies and 8 additions.

Let $\mathbf{p}(t) = \mathbf{o} + \mathbf{d}\,t$ denote the parameterization of the line, with $\|\mathbf{d}\| = 1$. The distance along the axis may be written as a linear polynomial denoted as $x(t)$ and the squared distance to the axis $y^2(t)$ is a quadric polynomial in $t$. Let $\mathbf{n} = \mathbf{o} - \mathbf{c}$, let $\alpha = \mathbf{d} \cdot \mathbf{u}$, $\beta = \mathbf{n} \cdot \mathbf{u}$ and $\gamma = \mathbf{d} \cdot \mathbf{n}$, we have:

$$y^2(t) = (1 - \alpha^2)\,t^2 + 2\,(\gamma - \alpha\,\beta)\,t + \mathbf{n}^2 - \beta^2$$

$$x(t) = \alpha\,t + \beta$$

Those polynomials may be computed during a pre-processing step. The cost of the pre-processing step is only 14 multiplies and 12 additions, which becomes negligible if more than 4 distance queries are performed. The evaluation of $y^2$, $x$ and $x^2$ for a given point on a line only requires 4 multiplies and 3 additions.

## 8 Implementation and performance

The algorithms have been implemented in C++ as member functions of cylinder, cone, line swept spheres and cone-sphere classes. We tried not only to reduce the overall number of operations in the worst case scenario, but also to identify specific cases that can be processed very efficiently.

The algorithms are robust and do not involve divisions by small numbers which could produce floating point overflows. Robustness problems may arise when cone and cone-spheres have almost the same end radii. To handle these cases in a reasonable way, it is preferable to check radii and create cylinders or line swept spheres if $r_{\mathbf{a}} - r_{\mathbf{b}} < \varepsilon$ where $\varepsilon$ is a user defined threshold.

| Primitive | + | × | $\sqrt{}$ | \|\| | ? | Time |
|---|---|---|---|---|---|---|
| Cylinder | 11 | 9 | 1 | 1 | 3 | 3.74 |
| Cone | 14 | 14 | 1 | 0 | 4 | 9.36 |
| Cone-sphere | 12 | 11 | 2 | 0 | 5 | 13.02 |
| Line swept sphere | 11 | 9 | 1 | 1 | 2 | 3.67 |

Table 1: Number of operations involved in the distance computation in the worst case and timings (in seconds) for performing $10^8$ random queries

Timings were measured for several scenarios by generating $10^4$ random shapes in a unit cube and performing $10^4$ random distance queries for every case. Table 1 reports the number of operations performed in the worst case situation as well as timings on a Pentium IV 2.4GHz workstation using `g++ -O -s` as compiler command and options.

| Primitive | + | × | $\sqrt{}$ | \|\| | ? | Time |
|---|---|---|---|---|---|---|
| Cylinder | 7 | 6 | 1 | 1 | 3 | 2.92 |
| Cone | 9 | 11 | 1 | 0 | 4 | 6.14 |
| Cone-sphere | 7 | 8 | 2 | 0 | 5 | 8.19 |
| Line swept spheres | 6 | 6 | 1 | 1 | 2 | 3.07 |

Table 2: Number of operations involved in the distance computation in the worst case and timings (in seconds) for performing $10^8$ random queries using the optimized algorithms

Table 2 reports the corresponding number of operations and timings for the optimized algorithms. Figures do not take into account the pre-processing step. Timings were measured for several scenarios by generating $10^4$ random shapes in a unit cube and performing $10^4$ random distance queries along a random ray for every case.

Our optimized algorithms were used in an implementation of the BlobTree [5] modeling and animation system. Ray tracing skeletal implicit surface models involves hundreds of thousands distance queries for every image and the proposed optimizations proved to speed up renderings significantly compared to the non-optimized variant.

## Web information

A $C++$ implementation of the algorithm and tests are available online at:

`http://www.acm.org/jgt/papers/GalinBarbier04`.

## References


[1] S. Frisken, R. Perry, A. Rockwood, Jones. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. *Siggraph Proceedings*, 249–254, 2000.

[2] J. Hart and B. Baker. Implicit modeling of tree surfaces. *Proceedings of Implicit Surfaces'96*, 143–152, 1996.

[3] N. Max. Cone-spheres. *Siggraph Proceedings*, 59–62, 1990.

[4] P. Schneider and D. Eberly. Geometric Tools for Computer Graphics. *Morgan Kaufmann*, 2003.

[5] B. Wyvill, A. Guy and E. Galin. Extending the CSG Tree (Warping, Blending and Boolean Operations in an Implicit Surface Modeling System). *Computer Graphics Forum*, **18**(2), 149–158, 1999.