COMPUTATION OF THE ARC LENGTH FROM THE SHADOW BOUNDARY OF A CAD OBJECT

Master of Science in Electrical Engineering with Specialization in Signal Processing & Wave Propagation.
Supervisor: Sven-Erik Sandström

Author: Amoateng Eric
Date: 2012-12-14
Subject: Electrical Engineering
Level: Master
Course code: 5ED06E
Abstract

CAD objects are geometrical descriptions of physical scenes from the real world. Ray tracing is used to project the objects onto a pixel screen. A lit and a shadow zone are formed according to the direction of the incoming field (light) and the orientation of the pixel screen. The arc length along the surface of the object, from the shadow boundary to a point in the lit zone, is computed by means of numerical integration. The arc lengths corresponding to two orthogonal directions that are aligned with the pixels on the pixel screen are computed and used for interpolation to obtain the arc length for all directions. A number of simulations for various CAD geometries are made using a ray-tracer implemented in FORTRAN 90.

Keywords: Ray tracing, arc length, shadow boundary, lit zone, CAD object
Acknowledgement

The author would like to acknowledge the supervisor Sven-Erik Sandström for his many helpful comments that led to a substantial improvement of this report. A special thanks also to my family for their unfailing support during the period of study and development of this thesis and my colleagues in the Electrical Engineering program for their motivations.
## Contents

1 Introduction 3

2 Generating CAD objects for the ray tracing 5
  2.1 The program Polgen ................................. 5

3 Implementation of ray tracing 7
  3.1 Intersection calculation ............................ 8
  3.2 Acceleration structures for ray tracing .............. 9
    3.2.1 The grid partitioning method .................... 9
    3.2.2 The binary space partitioning method ............ 10
  3.3 The ray tracing outputs ............................. 11

4 Shadow boundary computation 15
  4.1 Shadow boundary of a sphere ......................... 15
  4.2 Shadow boundary for an oblate spheroid .............. 17

5 The arc length 19
  5.1 The arc length on a sphere ........................ 19
  5.2 Results of the arc length computation ............... 23
  5.3 Arc length for the oblate spheroid .................. 25

6 Conclusion 29
  6.1 Conclusion ....................................... 29
  6.2 Further work .................................... 30

A The program POLGEN which generates the objects for ray tracing 34
B A script in Mathematica on interpolation.
Chapter 1

Introduction

The growing demand for wireless communication systems has necessitated the use of both statistical and deterministic methods to model radio wave propagation. Hence, the concept of ray tracing has become important in the context of wave propagation models. It provides a fast method for simulating wave propagation, primarily in indoor and urban settings, and can be combined with other methods for radio propagation predictions [1][2].

This thesis presents a pixel screen method for computing the arc length from the shadow boundary to a point in the lit zone of a smooth CAD object. Such computations can be used in the study of current distributions in scattering problems where the scatterer is modeled as a CAD object. The model consists of polygonal plane facets that could represent buildings and terrains, typically in micro cells and pico cells [3]. For a given object, one uses the ray tracing method to project the object onto a pixel screen [4]. The image produced on the pixel screen is in principle the status of a set of pixels. The resolution of the image is improved by increasing the number of pixels. Based on the direction of the incoming light (field) in relation to the object, zones of shadow and light are formed.

There are four chapters in this thesis. In chapter 2, the generation of CAD objects by means of the program Polgen is considered. The program produces a ".dxf file" and a ".pol file" that is used for viewing purposes and ray tracing, respectively. Then follows the implementation of the ray tracing, which produces a ".ppm image" of the object.
on the pixel screen. The ray tracing is needed since the calculation of the arc length is based on a pixel facet map and especially identification of rays that hit the object close to the shadow boundary is important. Since the number of ray-object hits increases with object complexity, one examines ray tracing acceleration techniques in order to reduce the computational time. This is discussed in chapter 3. Once the ray tracing is done, one considers pixel based computation of the arc length from the shadow boundary into the lit zone. The shadow boundary is computed as a first step toward finding the arc length, as described in chapter 4, and the arc length computation is considered in chapter 5. Appendix A lists the FORTRAN code for the program Polgen which generates the objects. A Mathematica script on the several versions of arc length interpolation with varying accuracy is given in appendix B.
Chapter 2

Generating CAD objects for the ray tracing

AutoCAD .dxf objects have been used to represent building features and terrain profiles [5] since such features are readily represented in the CAD format. CAD primarily gives a description of the object in terms of triangles (facets). There are also CAD programs that use higher order surfaces (NURBS) or curved triangles.

In this thesis one uses FORTRAN codes to generate the CAD object for the ray tracing. The object in polygon form serves as the input file to the ray tracer and the essence is to project the object onto a pixel screen.

2.1 The program Polgen

The program Polgen is used to generate the object to be ray traced. It is essentially written in old f77. Two file formats are generated. The first is a polygon file with only flat polygons and this goes to the ray tracer. The other is a dxf file with the same geometrical information. The dxf is an ASCII format as opposed to a binary format. This file is generated for viewing purposes since software for this is available [6].

The type of objects that are generated are spheres, spheroids (prolate and oblate), conespheres and a type of cylinders. The type of object is selected by setting logical variables
The geometry information essential to generate the object is specified by defining the entities, layers and facets as shown in the Appendix A [7]. The `ut.pol` file produced from the code serves as the input for the ray tracer. It consists of polygons with no further information. The polygons are later split into triangles. Rendering is illustrated in Fig 2.1 where a conesphere is viewed via the dxf format.

![Figure 2.1: Rendering of a conesphere created with the code Polgen and viewed via the dxf format.](image)

Figure 2.2 shows a sphere generated and viewed in the same fashion.

![Figure 2.2: A similar rendering of a sphere.](image)
Chapter 3

Implementation of ray tracing

Ray tracing is a technique for synthesizing images. The process involves sending rays from an observer location through pixels in an image plane and proceeding to see if the ray intersects with an object behind the screen. Each ray is followed from the pixel towards the object and if there is an intersection, the pixel will be associated with the lit part of the object [8][9].

Fig 3.1 illustrates the ray tracing process by referring to a camera. The rays are sent from a source point through the pixels and onto the object behind the screen. However, the ray tracer implemented uses parallel rays. This corresponds to an incident plane wave or a source that is very far away.

Figure 3.1: A simple illustration of the ray tracing process.
3.1 Intersection calculation

The fundamental problem is to find the point on the 3D object that the ray intersects. A ray starting from a pixel must be tested for intersection with some part of the object or geometry, to find which is the nearest intersection. Once the intersection has been determined for all the pixels, the direction of the incoming light and the material properties of the object can be used to calculate the color of the pixel.

To find the point on the surface that the ray hits, the ray is represented in parametric form and the surface could be given in implicit form [10]. With a spherical object, this calculation is simple.

A ray is defined in equation (3.1) by the origin $O$, a direction vector $d$ and a scalar variable $t$,

$$ r(t) = o + td, $$  \hspace{1cm} (3.1)

where $t$ generates different points on the incoming ray. A value of $t$ smaller than zero is said to describe a point behind the origin and a positive value describes a point in front. A sphere, with center $c = (C_x, C_y, C_z)$ and radius $R$, is described by the surface equation,

$$ (x - C_x)^2 + (y - C_y)^2 + (z - C_z)^2 - R^2 = 0 $$  \hspace{1cm} (3.2)

This could also be written in the vector form as [4],

$$ (r - c) \cdot (r - c) - R^2 = 0 $$  \hspace{1cm} (3.3)

where $r$ is any point on the sphere that satisfies the equation. One then combines equations (3.1) and (3.3) and computes the length of the ray which intersects with the sphere. This is used to find the nearest intersection point of the ray and the surface.

$$ (o + td - c) \cdot (o + td - c) - R^2 = 0 $$  \hspace{1cm} (3.4)
Solving equation (3.4) produces,

$$(d \cdot d)t^2 + 2d \cdot (o - c)t + (o - c) \cdot (o - c) - R^2 = 0$$  
(3.5)

This is a quadratic equation of the form,

$$At^2 + Bt + C = 0,$$  
(3.6)

with the solution,

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}.$$  
(3.7)

The discriminant indicates the number of solutions. If there are two solutions, the ray enters and leaves the sphere. A single solution corresponds to a ray that grazes the sphere at one point. If there is no solution the ray misses the sphere [4]. Since all entities are defined, the value of $t$ can be computed.

3.2 Acceleration structures for ray tracing

High efficiency in the ray tracing requires the use of hierarchical data structures for subdivision of the geometry. Whitted [11] showed that about 75 to 95 percent of the total time spent on ray tracing is spent on intersection calculations, depending on the complexity of the structure. The time spent on the intersection calculations is directly dependent on the number of facets in the object and its complexity [12][13]. Ray tracing without the use of any acceleration mechanism is very slow. Very many ray-object intersection testings are carried out which prolongs the computational time. The remedy is to use spatial subdivision schemes like binary space partitioning and grid partitioning.

3.2.1 The grid partitioning method

This method was introduced by Fujimoto et al. The method subdivides the volume containing the object into boxes that contain a suitable number of triangles. The grid
links the triangles to boxes and provides a decomposition of the object. Starting from a bounding box around the object, the volume is easily divided and this leads to a drastic speed up of the ray-object intersection calculation. The grid consists of rectangular volumes called cells. The triangles are contained within the cells of the grid. The grid partitioning method organizes the large number of triangles. By using this method, one can ray trace complex geometries [14] [15].

To find the intersection of a ray and a triangle, the ray first finds intersection with the nearest cell and then finds intersection with the triangles in the cell. By this arrangement, a ray does not have to find intersection with all the triangles but only the triangle facets in the cell pierced by the test ray [16] [17]. Each time a ray hits a particular cell (voxel) of the grid, all the triangles in that cell are tested for intersection with the ray. If the closest intersection is found then the ray is terminated and a pixel facet map is established. The color is determined later. This reduces the number of intersections and improves on the efficiency of the ray tracer [2].

3.2.2 The binary space partitioning method

BSP stands for binary space partitioning and is another subdivision structure. This method subdivides the space occupied by the object into boxes but these boxes are arranged into a binary tree. Each box has two sub-boxes (left/right) unless it is a leaf on the tree. A leaf contains a number of facets that is suitably small for finding the ray-facet intersection [18].

The criterion is typically the number of triangles in a leaf and the boxes are split in two in all three dimensions until a box with a suitable number of triangles is found. For a given object, a plane is chosen as the splitting plane. The plane is used to split the bounding box of the object along one of the coordinate axis, starting with the x-axis for example. This produces two identical boxes with half the volume of the previous bounding box called the parent. These boxes are further subdivided with another splitting plane in the y coordinate direction to produce four offspring of the parent bounding volume. The process can be continued, dividing smaller volumes containing fewer triangles, until the leaf condition for the number of triangles in a box is reached. In this way, there are fewer triangles to test for intersection with a ray [19] [18]. This process makes BSP an
acceleration device for the ray tracing. A simple illustration of the BSP tree is shown in Figure 3.2.

![BSP Tree Illustration](image.png)

**Figure 3.2:** An illustration of Binary Space Partitioning in three dimensions.

Table 3.1 compares the computational time to render the objects on the pixel screen, using the different acceleration methods. There is a pixel dimension of 1000 by 1000 and approximately 3480 triangular facets in the polygon file. However, this can vary depending on the object. A sphere, conesphere and spheroid are studied.

### 3.3 The ray tracing outputs

The ray tracer is coded in FORTRAN 90. One could use a high resolution of the pixel screen in order to obtain a high quality representation of the object. This is specified by the programmer in addition to information on the aspect of the object that is chosen. One could either rotate the object or the screen in order to change the aspect [19].
Table 3.1: Simulation Results for Grid and BSP

<table>
<thead>
<tr>
<th>Object</th>
<th>Method</th>
<th>Cpu Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphere</td>
<td>Grid Method</td>
<td>24.917</td>
</tr>
<tr>
<td></td>
<td>BSP Method</td>
<td>34.169</td>
</tr>
<tr>
<td>spheriod</td>
<td>Grid Method</td>
<td>11.388</td>
</tr>
<tr>
<td></td>
<td>BSP Method</td>
<td>15.388</td>
</tr>
<tr>
<td>conesphere</td>
<td>Grid Method</td>
<td>30.078</td>
</tr>
<tr>
<td></td>
<td>BSP Method</td>
<td>35.103</td>
</tr>
</tbody>
</table>

With ray tracing and rendering, complex lighting effects are achieved. This is seen in Figure 3.3 where a lit zone and a shadow zone are formed. The lit zone has pixels coded in RBG color components. In this crude model the shadow is very distinct since the object is convex. The image that is generated depends on the position of the pixel screen and the direction of the incoming light.

The coloring of the object is obtained by mean of a scalar product. The angle between the unit normal of the surface and the direction of the illumination is calculated and used to concoct an RGB color. The normal for a triangular facet is easily obtained from the vertices. A triangle is declared to be in shadow if the dot product between the triangle normal and the direction vector of the incident light is greater than zero. Otherwise, the color is computed schematically from the value of the dot product. The generated image consists of a flat screen of colored pixels. Figure 3.3 shows the image generated for a top view of a sphere.
Figure 3.3: Illustration of a rendering in terms of colors of a sphere illuminated from the left.

Subsequent computations are based on the status of the pixels for the same direction of the incoming field. The following rendering shows the pixel status for the same direction of the incoming field. The pixel screen is rotated down to $\theta = 45^\circ$ with $\phi = 0^\circ$. Figure 3.4 shows a conesphere seen from the side.
Figure 3.4: A rendering of a conesphere seen from an angle.
Chapter 4

Shadow boundary computation

Depending on the angle of incidence of the illumination, ray tracing will produce a lit region and a shadow region [20]. First, the shadow boundary is computed as a starting point for the computation of the arc length. The shadow boundaries of a spherical geometry and some other CAD objects are considered. For a perfect sphere, it is trivial to determine the shadow boundary. When the sphere is faceted, however, this becomes a major problem since the hit point can fluctuate a great deal when the ray is nearly grazing the approximate surface.

4.1 Shadow boundary of a sphere

Various methods have been used to estimate the shadow boundary for ray traced objects [21] [22]. In this thesis, the method of averaging of the ray hit points is studied.

The simplest way to compute the shadow boundary (SB) is to use the hit points directly (CORRECTION = False.). If averaging is used, the hit points for an interval are stored and used to compute an average that is representative for that interval. One could also shift this average, based on heuristic statistics, to improve the approximation. This is done when the variable CORRECTION is true. When SPHERE is true, it is assumed that the shadow boundary corresponds to a zero longitudinal coordinate for a sphere at the origin.
In Figure 4.1, the longitudinal (z) coordinate of the hit point is shown as a function of the counter that records hits along half the boundary. The counter is incremented based on recognition of a pattern of nearest neighbors at the SB. In this example, SB is set to zero since SPHERE is true.

Figure 4.1: The z-coordinates of the hit points for a faceted sphere are shown in dashed line for half the pixel screen. The heavy line shows the SB that is known \textit{a priori} for the sphere.

The shadow boundary is computed numerically when SPHERE is false. This is shown in Figure 4.2. The fluctuation of the hit points is the same but the averaging procedure leads to a staircase pattern for the SB.

Figure 4.2: The z-coordinates for the hit points and the computed SB. The staircase pattern shown in heavy line shows the estimate of the SB obtained by averaging the hit points over an interval.
Figure 4.3 shows the computed shadow boundary for a sphere in three dimensions. The fluctuation around the point $z=0$ appears in a manner corresponding to Figure 4.2. Due to the one-dimensional nature of the shadow boundary, some numerical arrangements are made to enhance the presentation.

**Figure 4.3:** A 3D illustration of the computed shadow boundary for a sphere with the fluctuation around the exact value $z=0$ shown.

### 4.2 Shadow boundary for an oblate spheroid

The shadow boundary for an oblate spheroid could be computed in the same way. The hit points representing an interval are stored and averaged, but in addition there is now also a variation in the exact shadow boundary. The hit points are shown in thin line in Figure 4.4 and follow the contour of one half of the object. The irregularity that depends on the pixel density and the facet density of the object appears. Averaging produces a staircase pattern that now includes the variation of the exact shadow boundary. The oblate spheroid has its center at the aiming point, a vertical polar axis, and the angle of observation (incidence) corresponds to a polar angle $\theta = 60^\circ$. The semiaxes of the spheroid are $a=1.3$ and $b=0.7$ and this requires a pixel screen of 3 by 3 meters.
Figure 4.4: The z-coordinates for the hit points and the computed SB for an oblate spheroid at 60° incidence.

The shadow boundary for the same oblate spheroid is also plotted in 3D in Figure 4.5.

Figure 4.5: The computed SB for an oblate spheroid observed at the polar angle $\theta=60^\circ$. 
Chapter 5

The arc length

The arc lengths from the shadow boundary to a point in the lit zone for fairly general CAD objects are computed. The objects are modeled using rectangular facets that are split into triangles in the actual calculation since triangular facets have geometrical advantages [13][23]. In general, the arc length must be computed numerically, but for a sphere this can be done analytically as an illustration of the procedure.

5.1 The arc length on a sphere

In general, the arc lengths $L_x$ and $L_y$, referring to the $x$ and $y$ directions on the screen, are computed. An analytical derivation is made for the unit sphere and this approach is also used in appendix B.

One introduces three points $(x_0, y_0), (x_1, y_0), (x_0, y_1)$ in the $z=0$ plane. They are related according to,

$$1 = x_1^2 + y_0^2,$$

(5.1)

$$1 = x_0^2 + y_1^2,$$

(5.2)
where \((x_1, y_0)\) and \((x_0, y_1)\) are points on the SB and \((x_0, y_0)\) is an interior point. In general one has,

\[
z = \sqrt{1 - x^2 - y^2}. \tag{5.3}
\]

The two functions \(z_x\) and \(z_y\) prepare for the arc lengths \(L_x\) and \(L_y\).

\[
z_x = \sqrt{1 - y_0^2 - x^2} \tag{5.4}
\]

\[
z_y = \sqrt{1 - x_0^2 - y^2} \tag{5.5}
\]

\(Z_x\) and \(z_y\) are z-coordinates for a unit sphere with one coordinate fixed, since the plan is to evaluate the arc length in two orthogonal directions. The arc lengths \(L_x\) and \(L_y\) are obtained by integration from the SB point to the lit point \((x_0, y_0)\).

\[
L_x(x_0, y_0) = -\int_{x_1}^{x_0} \sqrt{(z_x'^2 + 1)} \, dx \tag{5.6}
\]

\[
L_y(x_0, y_0) = -\int_{y_1}^{y_0} \sqrt{(z_y'^2 + 1)} \, dy \tag{5.7}
\]

With \(a_x = \sqrt{1 - x_0^2}\) and \(a_y = \sqrt{1 - y_0^2}\) one obtains,

\[
L_x = a_x \left[ x_1 \right]_{x_0} \arcsin \frac{x}{a_x} \tag{5.8}
\]

\[
L_y = a_y \left[ y_1 \right]_{y_0} \arcsin \frac{y}{a_y} \tag{5.9}
\]

For every point \((x_0, y_0)\) there are now two known arc lengths in the reference directions and the arc length for intermediate directions is obtained by angular interpolation. This is elaborated upon in appendix B where also additional reference directions are used to improve the interpolation accuracy.
Only the case with two reference directions is implemented for the pixel screen. The numerical quadrature is easily implemented since the two reference directions conform to rows and columns on the pixel screen. Adding more reference directions is a considerable complication.

A difficulty with the numerical quadrature is that the derivative in Equation 5.6 and 5.7 varies rapidly at the shadow boundary. This can be handled by increasing the pixel density and thereby increasing the density of quadrature points. A reasonable accuracy can be obtained but the main problem lies elsewhere. For faceted objects it is rather difficult to estimate the position of the shadow boundary and this is the main source of error in this type of calculation.

With two reference directions there is an angular interpolation interval of $90^\circ$. This is too long to provide high accuracy and in Appendix B higher order interpolation, based on diagonals, is implemented for comparison.

The computation is based on the z-coordinates (cf. Eqs. 5.6 and 5.7) and for faceted objects there is an error in the z value. For spheres this facet noise can be reduced by means of Eq. 5.3 since the radius is known. As a simple illustration, Figures 5.1 and 5.2 show the radius as a function of transverse coordinates.

![Figure 5.1: Plot of the radius of a sphere as a function of the transverse coordinate with $r=0$ outside the object.](image)

Figure 5.1: Plot of the radius of a sphere as a function of the transverse coordinate with $r=0$ outside the object.
In Figure 5.2 the radius of the sphere is also plotted in 3 dimensions in terms of the transverse coordinates.

![Figure 5.2: A 3D representation of the radius of a unit sphere at the origin, in terms of the transverse coordinates.](image)

The derivative of $z$ with respect to the transverse coordinates of the sphere is shown in Figure 5.3. The point $\rho = 1$ corresponds to the SB. The derivative diverges at the shadow boundary since the surface of the sphere is normal to $\rho$. The derivative tends to zero at $\rho = 0$ since the sphere is tangential to $\rho$. Since the square root evaluation of $z$ is used, the derivative has no facet noise.

![Figure 5.3: Plot of the derivative of $z$ showing a divergence at the SB.](image)
5.2 Results of the arc length computation

With the procedure for the computation of the shadow boundary established, the arc length can be computed. Figure 5.4 shows the arc length from the SB radially toward the center, for a unit sphere. Since the sphere is symmetrical the arc length is well defined in the entire lit zone.

![Figure 5.4: The computed arc length, from the SB to a lit point, for a sphere.](image)

The error with respect to the exact solution is of interest. For a sphere, with KORR= True and SPHERE= True, the error is smooth and small, provided that the pixel density is sufficient (1000 by 1000 pixels).

![Figure 5.5: The error relative to the exact arc length for a sphere (SPHERE= True, KORR= True).](image)
Figures 5.6 and 5.7 show the difference (error) between the computed arc length and the exact arc length. Figure 5.6 corresponds to using the hit points directly while Figure 5.7 shows the effect of using the shifted average method. The error is smaller for the shifted average and this method can also be applied to a general geometry where there is no a priori information about the SB.

Figure 5.6: The error relative to the exact arc length for a sphere, when the ray tracing hit points are used to compute the shadow boundary.

Figure 5.7: The corresponding error when the averaging method is used to compute the shadow boundary.

These results could be compared to the results obtained by means of higher order interpolation (Appendix B). Figure 5.8 shows the error resulting from a quadratic 3 point interpolation and similarly Figure 5.9 presents a quadratic interpolation obtained with 5 points on the 90° interval. The error is about 1% of that obtained with the pixel
screen and 2 point interpolation. Higher order interpolation is accurate but difficult to implement with pixels.

Figure 5.8: The error in the computed arc length for a sphere, shown in what corresponds to the first quadrant of the pixel screen. A three point interpolation is compared to the exact result.

Figure 5.9: The error in the computed arc length for the first quadrant of the pixel screen. A five point interpolation is compared to the exact result.

5.3 Arc length for the oblate spheroid

The arc length is computed in the same fashion based on an SB extracted directly from the hit points. One could also use the average method to compute the SB for the oblate spheroid.
Figure 5.10: An oblate spheroid with semiaxes $a = 1.3$ and $b = 0.7$.

Figure 5.11 shows the radius for the oblate spheroid in relation to the transverse coordinate. The spheroid is viewed from above. The minimum corresponding to the shorter semiaxis appears in the graph.

Figure 5.11: The radius of an oblate spheroid viewed at an angle $\theta = 60^\circ$, as a function of the transverse coordinate.

The arc length is computed in the same way for the spheroid and Figure 5.12 shows the arc length for the spheroid viewed at $\theta = 0^\circ$. This results is almost symmetrical since the oblate spheroid is viewed from above. A small interpolation error appears.
Figure 5.12: Arc length computed from the SB point to a point in the lit zone for an oblate spheroid viewed at $\theta = 0^\circ$.

If the arc length is instead computed for the angle $\theta = 90^\circ$, the object is no longer symmetrical and the arc length at the center is in principle multivalued. Figure 5.13 illustrates this non-uniqueness. The radial arc length depends on where the arc starts at the shadow boundary.

Figure 5.13: Arc length for the $(1.3, 0.7)$ spheroid for the observation angle $\theta = 90^\circ$. 
This oblate spheroid is now viewed from the side and the asymmetry becomes obvious also in the arc length. This is a concern if the arc length is to be used for computational purposes such as describing current distributions on the object.
Chapter 6

Conclusion

6.1 Conclusion

Ray tracing techniques were used to obtain a mapping between a pixel screen and a CAD object. This mapping was then used to compute the arc length from the shadow boundary to points in the lit zone of the object. The computation involves numerical quadrature in terms of the pixels. The resolution of the pixel screen must therefore be sufficiently high. However, in terms of accuracy, the critical calculation is the estimation of the shadow boundary for the object. Averaging methods were used to improve the results.

CAD based computation, like the arc length calculation studied here, could be used in schemes to compute or approximate the current on scatterers. One then assumes that the object is smooth and convex and not too unlike a sphere.

A problem with unsymmetrical objects is that although the arc length is well defined close to the shadow boundary, there is the difficulty that the arc length is not uniquely defined at the center of the lit zone. There are many paths of different length leading out to the shadow boundary.
6.2 Further work

For future work one could consider higher order interpolation based on diagonals on the pixel screen. For this to be meaningful, a better method to estimate the position of the shadow boundary seems essential. This is not so easy to obtain without resorting to more elaborate CAD descriptions of the object such as NURBS surfaces. The problem of unsymmetrical objects calls for methods to handle the center of the lit zone where the arc length is not well defined.
Bibliography


The program POLGEN which generates the objects for ray tracing

APPENDIX A

The program POLGEN which generates the objects for ray tracing

```fortran
program polgen

c This code generates a polygon file and a dxf file

c
real*8 xyz(2,3),hj(5,3),th(2),fi(2),r(3),r_func,rad,the,phi
integer num,j,i,laynum,n,ith,ifi,i2,j2,nfi,laynum_1
character q(0:15)
character*3 w3(10000),w
logical TH_CRIT,FI_CRIT
Pi= 3.141592653589793D0

do j=1,3
  xyz(1,j)= 1.d10
  xyz(2,j)=-1.d10
enddo

call gen_hex(w3) ! generate Entity handles

open(2,file='ut.pol')
rewind 2
write(2,*) 'Polygonfil' ! header pol
call system("cp start_file_R12 ut.dxf") ! start_file
open(3,file='ut.dxf',access='APPEND')
call start_Entities_R12 ! header Entities

n= 30
nfi= 2*n
write(6,*)'Number of polygons:',n*nfi
laynum= 0
num= 0 ! facet number
do ith= 0,n-1
  th(1)= Pi*ith/dfloat(n)
  th(2)= Pi*(ith+1)/dfloat(n)
  the= (th(1)+th(2))/2.d0
  do ifi= 0,nfi-1
    num= num+1
    fi(i)= 2*Pi*ifi/dfloat(nfi)
  enddo
```

[34]
fi(2) = 2*Pi*(ifi+1)/dfloat(nfi)
phi = (fi(1)+fi(2))/2.d0

if(ith.EQ.0 .OR. ith.EQ.n-1) then
  ant = 3
else
  ant = 4
endif

do i=1,ant
  do j=1,3
    hj(i,j)= r(j) ! set corner
  enddo
endo
call r_th_fi(th(i2),fi(j2),r) !compute corners

do j=1,3
  if(j .EQ. 1) then
    i2=1; j2= 1
  elseif(j .EQ. 2) then
    i2=2; j2= 1
  elseif(j .EQ. 3) then
    i2=2; j2= 2
  endif
endif
endo

TH_CRIT= the.LT.20.d0 .OR. the.GT.2.3d0
FI_CRIT= phi.LT.20.d0 .OR. phi.GT.3.0d0

if(TH_CRIT .AND. FI_CRIT) then !select part
  laynum_1= laynum
else
  laynum_1= 1
endif

write(2,*) 3*ant+2 !write polygon file
write(2,*) laynum_1 !layer number
write(2,*) num !polygon number

do 40 i= 1,ant
  do 41 j=1,3
    if(hj(i,j).ge.xyz(2,j)) xyz(2,j)=hj(i,j) ! max x,y,z
    if(hj(i,j).lt.xyz(1,j)) xyz(1,j)=hj(i,j) ! min x,y,z
  enddo
  if(laynum_1 .EQ. 0) then
    w= w3(50+num) !avoid low handles
    call wri3D_h_R12(hj,ant,num,w,laynum_1) !3DFACE objects
  endif
  write(6,*)'largest Ent. handle= ',w,' < $HANDSEED=A01'
  endif
endo
endo

write(2,*) 0
write(2,*) 'COMBI' !footer pol
call stop_Entities_R12 !footer dxf

write(6,*)'Limits:
  do j=1,3
    write(6,*) (sngl(xyz(i,j)),i=1,2)
```fortran
enddo
write(6,*) 'Default Aimingpoint:',
&(sngl((xyz(2,j)+xyz(1,j))/2.d0),j=1,3)
write(6,*) 'Widths Dx, Dy, Dz:',
&(sngl(abs(xyz(2,j)-xyz(1,j))),j=1,3)
close(2)
20 format(40A)
stop
end

c-----------------------------------------------------------------------------
subroutine start_Entities_R12
write(3,20)'SECTION' !start Entities
write(3,20)' 2'
write(3,20) 'ENTITIES'
write(3,20) ' 0'
20 format(40A)
return
end

c-----------------------------------------------------------------------------
subroutine stop_Entities_R12
write(3,20)'ENDSEC' !end of Entities
write(3,20)' 0' !footer Entities
write(3,20)'EOF' !end of file
20 format(40A)
return
end

c-----------------------------------------------------------------------------
subroutine genHex(w3)
c create 3 digit hex string
integer num,i,j,n
character*3 w3(10000)
character q(0:15)
q(0)='0'; q(1)='1'; q(2)='2'; q(3)='3'; q(4)='4'
q(5)='5'; q(6)='6'; q(7)='7'; q(8)='8'; q(9)='9'
q(10)='A'; q(11)='B'; q(12)='C'; q(13)='D'; q(14)='E'; q(15)='F'
um= 0
do i=0,15
do j=0,15
do n=0,15
num= num+1
w3(num)= q(i)//q(j)//q(n)
enddo
enddo
enddo
return
end

c-----------------------------------------------------------------------------
subroutine wri3D_h_R12(hj,ant,num,w,laynum)
c write 3DFACE object; AutoCAD version R12
integer ant,i,j,grk(5,3),num,laynum
real*8 hj(5,3)
character*3 w
grk(1,1)=10; grk(1,2)=20; grk(1,3)=30 !corner 1
grk(2,1)=11; grk(2,2)=21; grk(2,3)=31 !corner 2
grk(3,1)=12; grk(3,2)=22; grk(3,3)=32 !corner 3
grk(4,1)=13; grk(4,2)=23; grk(4,3)=33 !corner 4
```
write(3,20)'3DFACE'          !header 3DFACE
write(3,20) 5              !layer number; Group code 8
write(3,20)v               !Entity handle hex > hexmin
write(3,20)' 8'            !layer name, Group code 8
write(3,24)laynum          !layer number; 0
do i=1,ant
  do j=1,3
    write(3,23)grk(i,j)     !corner code
    if(hj(i,j) .GT. 0.d0) then
      write(3,21)hj(i,j)    !corner
    else
      write(3,22)hj(i,j)    !corner
    endif
  enddo
enddo
write(3,20)' 0'            !footer 3DFACE

20 format(40A)
21 format(F12.10)
22 format(F12.9)
23 format(I3)
24 format(I1)
return
end

c-----------------------------------------------------------------------------
subroutine r_th_fi(th,fi,r)
  c Specify geometry of object
  real*8 th,fi,x(3),th0,th1,rc,Pi,ellipse,eps,th2,a,ths,d,d_,thc
  logical SPHERE,SPHEROID,CONESPHERE_CUSP,CONESPHERE
  logical CYLINDER
  external ellipse
  Pi= 3.141592653589793D0
  SPHERE=.False.
  SPHEROID=.False.
  CONESPHERE_CUSP=.False. !with cusp
  CONESPHERE=.True. !continuous
  CYLINDER=.false. !cylindrical geom.
  if(SPHERE)then
    th1= th; rc= 1.d0; d= 0.d0
  elseif(SPHEROID)then
    a= 0.5d0; eps=-0.5d0; th2= 0.0d0 !eps > 0 prol., eps < 0 obl.
    th1= th; rc= ellipse(a,th,eps,th2); d= 0.d0
  elseif(CONESPHERE_CUSP)then
    th0= 0.3d0; d= 0.d0 !th0= tip angle
    if(th<th0) then
      th1= th; rc= 1.d0
    else
      th1= th0; rc= (Pi-th)/(Pi-th0)
    endif
  elseif(CONESPHERE)then
    ths= 0.3; th0= ths+ Pi/2.d0 !ths= tip angle
    if(th<th0) then
      th1= th; rc= 1.d0; d=0.d0
    else
      d = 1.d0*sin(ths) !d= tip shift
      th1= th0; rc= d*cos(ths)*(Pi-th)/(Pi-th0)
    endif
  elseif(CYLINDER)then
    th0= 1.4; d_= 3.d0; thc= Pi-th0
    if(th<th0) then
      th1= th; rc= 1.d0; d=d_
elseif(th>thc) then
    th1 = th; rc = 1.0; d = d_
else
    th1 = th0; rc = 1.0
    d = -d_ + 2*(th-th0)/(thc-th0) + (d_ + cos(th0))
endif
endif
stop 'no object specified'

r(1) = sin(th1)*cos(fi)*rc
r(2) = sin(th1)*sin(fi)*rc
r(3) = cos(th1)*rc - d
return
end

function ellipse(a, th, eps, th0)
c elliptical boundary in polar form; th0 rotation
real*8 a, l1, l2, eps, th0, Pi, limit
real*8 th, ellipse, th_num, ta, Rx, Ry, x, y, xyp, R
parameter(Pi=3.141592653589793D0, limit= 1.d-20)
l1 = a*(1.0+ eps)
l2 = a*(1.0- eps)
th_num = th - th0
if( abs(th_num) .LT.limit .OR. abs(abs(th_num)-Pi/2.d0).LT.limit .OR. abs(abs(th_num)-Pi).LT.limit ) then
    th_num = th_num + 2*limit !avoid sing. in tan
endif
    ta = sin(th_num)/cos(th_num)
 Rx = sqrt((l2/l1)**2 + ta**2)
Ry = sqrt((l1/l2)**2 + 1/ta**2)
x = l2/Rx !polar form
y = l1/Ry
R = sqrt((x**2+y**2)**2/4)
elipse = &
return
end}
A script in Mathematica on interpolation.

```mathematica
(* Script for arclength calculation on a unit sphere. *)
(* The arclength is approximated by means of diagonals *)
(* that correspond to a pixel screen. *)

\[ \rho[x_,y_] := \sqrt{x^2 + y^2} \]
\[ \zeta[x_,y_] := \sqrt{1 - \rho[x,y]^2} \]
\[ \chi[x_,y_] := \text{ArcTan}[\zeta[x,y]/\rho[x,y]] \]

\(\phi = 0.2; r_0 = 1.0\)
\(x_0 = r_0 \cos(\phi); y_0 = r_0 \sin(\phi)\)

\[ x_2[y_] := \sqrt{1-y^2}; y_2[x_] := \sqrt{1-x^2} \]

\[ L_{x2}[x_0,y_0] = x_2[y_0]+l; L_{y2}[x_0,y_0] = y_2[x_0]-y \]
Print["Lx = ", L_{x2}[x_0,y_0], " Ly= ", L_{y2}[x_0,y_0]]

(* Arclength along x and y axis - analytical solution *)
\[ L_{ax}[x_,y_] = x_2[y](\pi/2 - \text{ArcSin}[x/x_2[y]]) \]
\[ L_{ay}[x_,y_] = y_2[x](\pi/2 - \text{ArcSin}[y/y_2[x]]) \]
Print["Lax= ", L_{ax}[x_0,y_0], " Lay= ", L_{ay}[x_0,y_0]]

(* Preparing for interpolation between arclengths *)
\(\varepsilon = 10^{-10}\)
\(\text{Crit1}[x_,y_] = \rho[x,y]<1\)
\(\text{Crit2}[x_,y_] = x>x_0 \land y>y_0\)
\(\text{Crit}[x_,y_] = \text{Crit1}[x,y] \land \neg \text{Crit2}[x,y]\)
\(\phi[x_,y_] := \text{ArcTan}[y/(x+\varepsilon)]\)

(* Upper limit for diagonal integration *)
\(\alpha_{x}[x_,y_] := 3\pi/4 + \phi[x,y]\)
\[ \alpha_{x2}[x_,y_] := \text{Arctan}[2/1] + \phi[x,y]\]
\[ \alpha_{x12}[x_,y_] := \text{Arctan}[1/2] + \phi[x,y]\]
\[ \alpha_{x} := \text{Cos}[\alpha_{x}][x,y] + \rho[x,y]\]
\[ \alpha_{x2} := \text{Cos}[\alpha_{x2}][x,y] + \rho[x,y]\]
\[ \alpha_{x12} := \text{Cos}[\alpha_{x12}][x,y] + \rho[x,y]\]

\[ \text{Rdia}_{x}[x_,y_] := \text{car}[x,y] \cdot \text{Sqrt}[\text{car}[x,y]^2 - \rho[x,y]^2 + 1 \mp \varepsilon]\]
\[ \text{Rdia}_{x2} := \text{car}[x,y] \cdot \text{Sqrt}[\text{car}[x,y]^2 - \rho[x,y]^2 + 1 \mp \varepsilon]\]
\[ \text{Rdia}_{x12} := \text{car}[x,y] \cdot \text{Sqrt}[\text{car}[x,y]^2 - \rho[x,y]^2 + 1 \mp \varepsilon]\]
```
(* Infinitesimal arclength *)

\[
\begin{align*}
\begin{array}{l}
r_2 &= 1 / \sqrt{2}; \quad r_{15} = 1 / \sqrt{5}; \quad r_{25} = 2 / \sqrt{5} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
l_{z1}[x_-, y_-, t_] &= Sqrt[1 - (x + t*r_2)^2 - (y + t*r_2)^2] \\
l_{z15}[x_-, y_-, t_] &= Sqrt[1 - (x + t*r_{15})^2 - (y + t*r_{25})^2] \\
l_{z25}[x_-, y_-, t_] &= Sqrt[1 - (x + t*r_{25})^2 - (y + t*r_{15})^2] \\
l_{z1525}[x_-, y_-, t_] &= Sqrt[l_{z15}[x_-, y_-, t_] + 1] \\
l_{z2515}[x_-, y_-, t_] &= Sqrt[l_{z25}[x_-, y_-, t_] + 1] \\
l_{z251525}[x_-, y_-, t_] &= Sqrt[l_{z2515}[x_-, y_-, t_] + 1]
\end{align*}
\]

(* Arclength for diagonals *)

\[
\begin{align*}
\begin{array}{l}
L_{dia}[x_-, y_] &= If[Crit1[x_,y_], NIntegrate[l_{z1}[x_,y_,t_], \{t,0,R_{dia}[x_,y_]\}], 0] \\
L_{dia21}[x_-, y_] &= If[Crit1[x_,y_], NIntegrate[l_{z1525}[x_-, y_-, t_], \{t,0,R_{dia21}[x_,y_]\}], 0] \\
L_{dia12}[x_-, y_] &= If[Crit1[x_,y_], NIntegrate[l_{z2515}[x_-, y_-, t_], \{t,0,R_{dia12}[x_,y_]\}], 0]
\end{array}
\end{align*}
\]

(* Interpolation in phi *)

\[
\begin{align*}
\begin{array}{l}
fi21 &= ArcTan[2/1]; \quad fi12 &= ArcTan[1/2] \\
fi_{21} &= ArcTan[2/1]; \quad fi_{12} &= ArcTan[1/2]
\end{array}
\end{align*}
\]

(* Print " Linear 5 points" )

\[
\begin{align*}
\begin{array}{l}
L[x_-, y_] &= If[phi[x_,y_] < \pi/4, \\
& \quad If[phi[x_,y_] < fi_{12}, \\
& & \quad Lax[x_,y_] + (L_{dia12}[x_,y_]-Lax[x_,y_]) * phi[x_,y_] / fi_{12} \\
& & \quad \& \quad L_{dia12}[x_,y_] + (L_{dia}[x_,y_]-L_{dia12}[x_,y_]) * phi[x_,y_] / (\pi/4-fi_{12}) \\
& & \quad \& \quad L_{dia12}[x_,y_] + (L_{dia}[x_,y_]-L_{dia12}[x_,y_]) * phi[x_,y_] / (\pi/4-fi_{12})] \\
& \quad \& \quad \end{array}
\end{align*}
\]

(* Print " Quadratic 5 points general" )

\[
\begin{align*}
\begin{array}{l}
A &= (\pi/4)^2; \quad B = \pi/4; \\
F &= (\pi/4)^2; \quad G = \pi/4; \\
DX &= -B*F + A*G; \quad DY = B*F - A*G \\
L[x_-, y_] &= If[phi[x_,y_] < \pi/4, \\
& \quad If[phi[x_,y_] < fi_{12}, \\
& & \quad (B-G)*Lax[x_,y_]+G*L_{dia12}[x_,y_]-B*L_{dia}[x_,y_])/DX*phi[x_,y_]^2 + \\
& & \quad \& \quad (A-F)*Lax[x_,y_] + F*L_{dia12}[x_,y_]-A*L_{dia}[x_,y_]) /DY*phi[x_,y_] + Lax[x_,y_] \\
& \quad \& \quad \end{array}
\end{align*}
\]

Print[" Quadratic 5 points general "]

L[x_-, y_] = If[phi[x_,y_] < \pi/4, \\
A = fi_{12}^2; \quad B = fi_{12}; \\
F = (\pi/4)^2; \quad G = \pi/4; \\
DX = -B*F + A*G; \quad DY = B*F - A*G; \\
\begin{align*}
\begin{array}{l}
& (B-G)*Lax[x_,y_]+G*L_{dia12}[x_,y_]-B*L_{dia}[x_,y_]) /DX*phi[x_,y_]^2 + \\
& \quad \& \quad (A-F)*Lax[x_,y_] + F*L_{dia12}[x_,y_]-A*L_{dia}[x_,y_]) /DY*phi[x_,y_] + Lax[x_,y_] \\
& \quad \& \quad \end{array}
\end{align*}
\]

Print[" Quadratic 5 points general "]
\[ A = (f_2 - f_1 - \pi/4)^2; \]
\[ B = f_2 - f_1 - \pi/4; \]
\[ F = (\pi/4)^2; \]
\[ G = \pi/4; \]
\[ D_x = -B*F + A*G; \]
\[ D_y = B*F - A*G; \]
\[ ((B-G)*L_{dia}[x,y] + G*L_{dia21}[x,y] -B*L_{ay}[x,y])/D_x*\phi(x,y)-\pi/4)^2 + \]
\[ ((A-F)*L_{dia}[x,y] + F*L_{dia21}[x,y] -A*L_{ay}[x,y])/D_y*\phi(x,y)-\pi/4) + \]
\[ L_{dia}[x,y] \]

\text{arcl}[x_,y_]:= \text{If}[\text{Crit}[x,y], \text{xhi}[x,y], 0] (*
= \text{Plot3D}[\text{arcl}[x,y],[x,0,1],[y,0,1],\text{PlotPoints}->60,\text{DisplayFunction}->\text{Identity}]\text{Show}[a, \text{DisplayFunction}->\text{Identity}]\text{Export}["\text{UNDERV/Ray_Trace/arcl.ep}\text{s},\text{Show}[a, \text{DisplayFunction}->\text{Identity}]\text{Export}["\text{UNDERV/Ray_Trace/arcl.xy.eps},\text{Show}[b, \text{DisplayFunction}->\text{Identity}]\text{Export}["\text{UNDERV/Ray_Trace/arcl.D.ep}s,\text{Show}[c, \text{DisplayFunction}->\text{Identity}]\text{Export}["\text{UNDERV/Ray_Trace/arc_dia.ep}s,\text{Show}[d, \text{DisplayFunction}->\text{Identity}]\text{ClearAll (*}
= \text{(A,B),(C,D))}
\text{m} = \{(f_2-f_1,f_3-f_1)\}
\text{LinearSolve[m,\{(f_2-f_1,f_3-f_1)\}}
*)