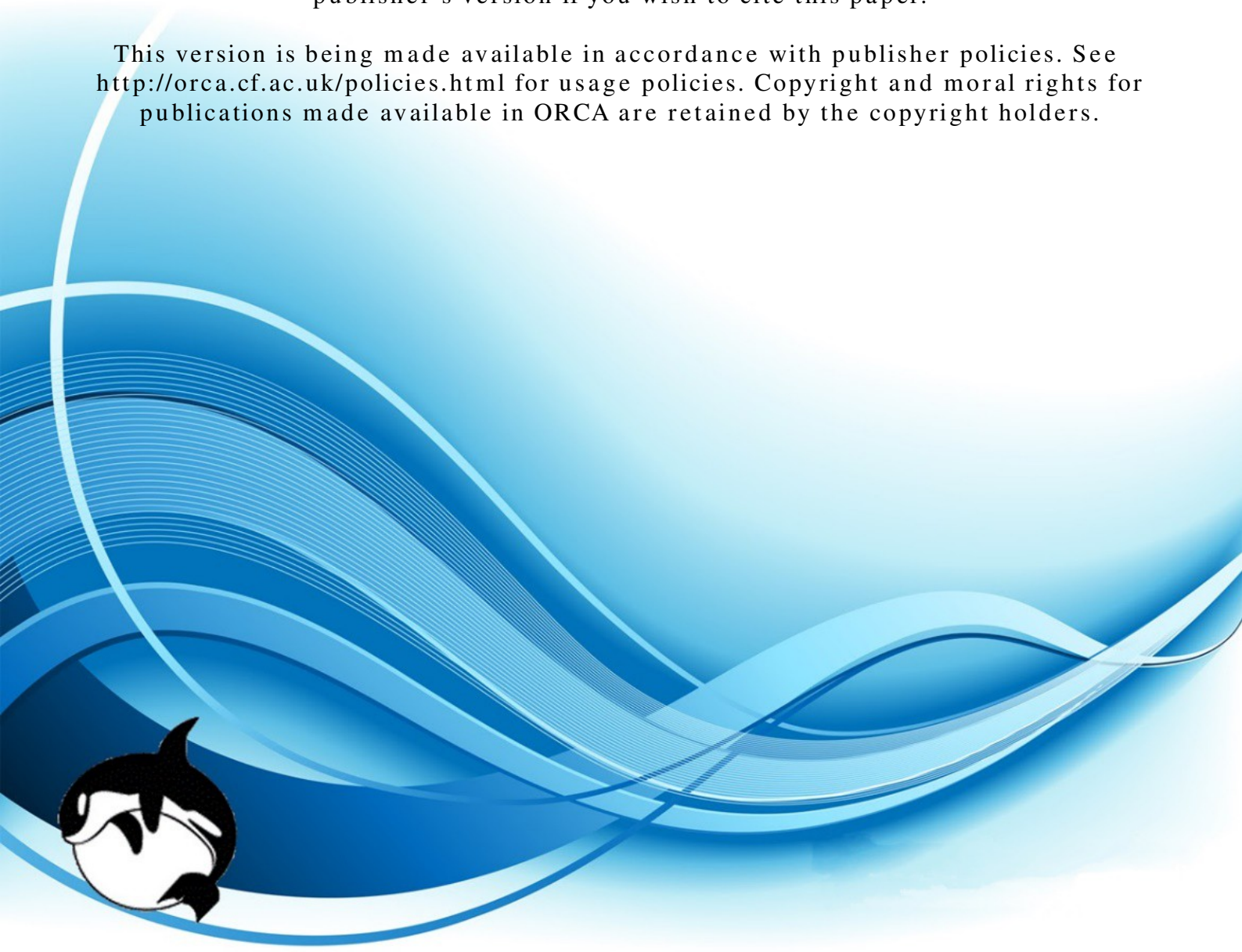

Using low-discrepancy sequences and the Crofton formula to compute surface areas of geometric models

Xueqing Li
Shandong University
Jinan, Shandong, China
E-mail:liyou@jn-public.sd.cninfo.net

Wenping Wang
University of Hong Kong
Pokfulam Road, Hong Kong
E-mail: wenping@cs.hku.hk

Ralph R. Martin
Cardiff University, UK
E-mail: ralph@cs.cf.ac.uk

Adrian Bowyer
University of Bath, UK
E-mail: A.Bowyer@bath.ac.uk

April 29, 2002

Abstract

The surface area of a geometric model, like its volume, is an important integral property that needs to be evaluated frequently and accurately in practice. In this paper we present a new quasi-Monte

Carlo method using low-discrepancy sequences for computing the surface area of a 3D object. We show that the new method is more efficient than a Monte Carlo method using pseudo-random numbers. This method is based on the Cauchy-Crofton formula from integral geometry, and it computes the surface area of a 3D body B by counting the number of intersection points between the bounding surface of B and a set of straight lines in E^3. Low discrepancy sequences are used to generate the set of lines in E^3 to reduce the estimation errors that would be caused by using statistically uniformly distributed lines. We study and compare two different methods for generating 3D random lines, and demonstrate their validity theoretically and experimentally. Experiments suggest that the new quasi-Monte Carlo method is also more efficient than the conventional approach based on surface tessellation.

1 Introduction

Length, area, volume, and moments are important integral properties of geometric objects that need to be computed frequently in solid modeling applications. There are various methods in the literature for computing the area, volume, and other measures, of geometric objects in different representations [3, 4, 7, 9, 10, 17, 19]. One approach to computing *volume* properties reduces them to computation of appropriate *surface* integrals by means of the divergence theorem [19]. Thus, computing integrals over the *surface* of the object is both important in its own right, and as a means of computing other mass properties of interest. We consider here the particular problem of computing the surface area of an object in CSG representation.

There are various approaches currently in use to performing this computation. A direct approach, used in GMSolid, is described by Sarraga [17]. The surface of each *primitive* is divided into small elements, whose sizes are chosen to meet a user-defined density. These elements are topologically rectangular and are bounded by parameter lines $u = u_1$, $u = u_2$ and $v = v_1$, $v = v_2$ (except e.g. at the poles of a sphere). Each element is then classified as lying *inside, outside* or *on* the model by means of point-membership classification, using a random test point belonging to the element. (A randomly chosen point is used to avoid aliasing effects.) The area of each element classified as *on* the surface of the solid is computed analytically using the usual

formula from differential geometry:

$$A = \int_{v_1}^{v_2} \int_{u_1}^{u_2} \sqrt{(\mathbf{r}_u.\mathbf{r}_u)(\mathbf{r}_v.\mathbf{r}_v) - (\mathbf{r}_u.\mathbf{r}_v)^2} du\, dv.$$

These areas are then summed to give an estimate for the surface area of the whole object.

Another class of approach is based on ray casting [11]. The area of a face can be approximated as the sum of the areas of rectangular strips that approximately cover the face. Each face is covered by a bounding rectangle, which is divided into strips; a face is assumed to be planar here for simplicity of description, but the algorithm can be generalised. Within each strip, a ray is fired across the rectangle, recording where the ray enters and leaves the face. Summing the lengths of the segments inside the face multiplied by the widths of each strip gives an approximation for the area of the face. A second approach also based on ray casting is to fire a regular grid of rays through the object, recording entry and exit points. Adjacent piercing points are connected to form triangles, and the sum of the areas of these triangles approximates the surface area of the object [14]. Clearly, this is not accurate where the rays are almost tangential to the object.

A third type of approach converts the CSG representation to a boundary representation. From this it is possible to produce a polygonal tessellation of the surface of the model, for example using the marching cubes algorithm [12]. An approximation of the surface area can then be computed by summing the areas of all polygons in the tessellation. More sophisticated variants of this approach have also been described [20, 22]; sampling rays are cast to accurately locate geometric intersections on the tessellated surface which is computed adaptively. Alternatively, the area can be computed directly from the boundary representation model using exact integral methods [9, 10] or numerical quadrature; it is also possible to convert surface integrals into line integrals using Green's theorem [19] as an alternative means of evaluation.

We present a new approach to computing the surface area of a CSG solid. Like the second class of previous approaches, our method is based on ray-casting, but it uses an entirely different principle for computing the area, and uses random rays. Furthermore, to obtain better performance than a naive application of the method, we use low-discrepancy sequences to obtain good statistical properties.

The method we present is most suited for use with CSG models, as it is based on ray casting and ray classification. However, the ideas have more

general applicability, and the method could also be applied to boundary representation or even surface models, too, if desired.

Our method is based on the Cauchy-Crofton formula (also known as Maurer-Cartan formula) from integral geometry [16], which relates the surface area of a 3D body B to the number of intersection points between the surface of B and a set of random straight lines in E^3. There are three main ideas that are used in this paper: 1) the Cauchy-Crofton formula; 2) two 4D space models which parameterize uniformly distributed lines in E^3; 3) low discrepancy sequences that produce a set of more evenly distributed lines than pseudo-random number generators do, thus improving the efficiency (or accuracy, for the same amount of time) of area computation.

Generating random lines in 3D space is far more tricky than generating random points in Euclidean space. In this paper we study in detail two models for generating 3D lines that have appeared in the literature. We demonstrate the theoretical validity of both models by verifying that the measure of the 3D line distribution in each case satisfies an established criterion in integral geometry, i.e., invariance under Euclidean transformations. We also present the effective application of both models in the quasi-Monte Carlo method through experiments, and demonstrate the superiority of the quasi-Monte Carlo method to a tessellation-based method.

The remainder of this paper is organized as follows. In Section 2 we introduce the Cauchy-Crofton formula from integral geometry and derive from it the formula for surface area computation. In Section 3 we discuss two models for uniformly distributed lines in E^3. In Section 4 we review the properties of low discrepancy sequences, and explain how to use them to generate lines in 3D with good statistical properties. In Section 5, we consider the problem of determining the number of intersections between a line and the bounding surface of a CSG solid. In Section 6 experimental results and comparions with two methods for computing surface area based on surface tessellation are presented, and we give conclusions in Section 7.

2 Formula for surface area computation

Studying the measure of a set of geometric figures is the central topic in integral geometry, a branch of geometry that is closely related to combinatorial geometry, convex geometry, and geometric probability. The result from integral geometry that is of interest to us here is the measure, or density,

of a set of lines in E^3. The reader may refer to [15, 16] for more detailed derivation of the following results.

Consider a surface element S in E^3. Let $\mathbf{n}$ be the unit normal vector of S. For a line L intersecting S, let (x, y) be the intersection point of L and S, and let ψ be the angle that L forms with normal vector $\mathbf{n}$. It can be shown [15] that the density of all lines intersecting S is given by

$$\mathrm{d}L = \cos\psi|\mathrm{d}s \wedge \mathrm{d}\sigma|, \tag{1}$$

where the exterior product $\mathrm{d}s = \mathrm{d}x \wedge \mathrm{d}y$ is the area element of S, and $\mathrm{d}\sigma$ is the area element on the unit sphere S^2, which is the domain of the unit vector $\mathbf{n}$. Here $\wedge$ stands for the exterior product of two differential forms. This density function is invariant under Euclidean transformations as are, for example, area and volume in Euclidean space.

Let Σ be a piecewise smooth surface (i.e., a collection of regular surface patches with G^0 continuity) of area s. Integrating the right-hand side of (1) over Σ and over directions of all the lines intersecting Σ at a point yields

$$\int |\cos\psi|\mathrm{d}s \wedge \mathrm{d}\sigma = 2\pi s, \tag{2}$$

because the integration of $|\cos\psi|\mathrm{d}\sigma$ over all directions gives 2π, which is the area of the projection of S^2 onto a planar section of S^2 through its centre. On the other hand, the integration on the left-hand side of (2) runs over all the lines. Since each line is counted twice at each of its intersection points with the surface Σ (due to the two hemispheres on the two sides of the surface), overall the line is counted $2m$ times, where m is the total number of intersection points of the line with Σ. Hence,

$$\int m\mathrm{d}L = \pi s. \tag{3}$$

This is the Cauchy-Crofton formula that relates the area of a surface to the number of intersections that the surface has with all lines in E^3.

Next we show how to use the above formula to compute the surface area of a given 3D body. Suppose that Σ is the bounding surface of a body B whose area s needs to be computed. Suppose further that Σ_1 is the bounding surface of a *reference* object B_1 which contains B. See Figure 1. We assume that the area s_1 of B_1 is known. Consider a set $\mathcal{L}$ of N lines that are randomly sampled from the set of lines that intersect B_1. Let n be the total number

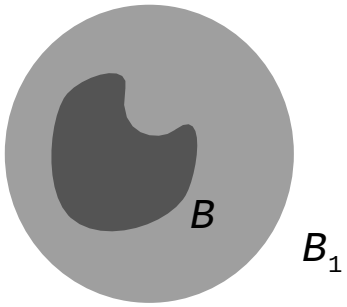

Figure 1: Body B is contained within the reference body B_1.

of intersection points of Σ with the lines in $\mathcal{L}$. Let n_1 be the total number of intersection points of Σ_1 with the lines in $\mathcal{L}$. According to equation (3), by integration approximation, we get

$$\frac{n}{N} \approx c\pi s \quad \text{and} \quad \frac{n_1}{N} \approx c\pi s_1,$$

where c is a constant of proportionality. It follows that

$$s \approx \frac{n}{n_1} s_1. \tag{4}$$

To summarize, the surface area s of Σ can be computed by formula (4) with the following algorithm.

1. Generate a set $\mathcal{L}$ of N random lines that sample the set $\bar{\mathcal{L}}$ of all the lines intersecting the reference object B_1.

2. Compute the number of intersections of the lines in $\mathcal{L}$ with the reference surface Σ_1 and the number of intersections of the lines in $\mathcal{L}$ with the surface Σ. Let n_1 and n denote these two numbers of intersections, respectively.

3. Approximate the area s of Σ by $\tilde{s} = \frac{n}{n_1} s_1$, i.e., equation (4).

In order for the formula (4) to be valid, it is essential to assume that the N lines of $\mathcal{L}$ form a well chosen sample of (for example, are uniformly distributed in) the set $\bar{\mathcal{L}}$ of all the lines that intersect the reference object B_1. The approximation error of $\tilde{s}$ to the exact area s of Σ can be attributed to the discrete sampling of the set $\bar{\mathcal{L}}$ by $\mathcal{L}$, and this error is also dependent on the evenness of the distribution of the lines of $\mathcal{L}$.

3 Generating uniformly distributed lines

In this section we consider how to generate a set of uniformly distributed lines in E^3. The set of lines in E^3 forms a 4D space and there are several different representations or models for this space. The best known representation for a line in E^3 probably comprises the Plücker coordinates [6], which are homogeneous coordinates $(L_1, L_2, L_3, L_4, L_5, L_6)$ satisfying $L_1L_4 + L_2L_5 + L_3L_6 = 0$. However, this representation does not provide a density measure that allows a simple way to generate uniformly distributed random lines.

An alternative might be to consider the density function

$$\mathrm{d}L = \cos\psi|\mathrm{d}s \wedge \mathrm{d}\sigma|$$

discussed in the last section. Since this density function is defined locally relative to a fixed planar surface element, it is natural to apply this density function over a finite planar patch, called a *base*; i.e., choose a point of the planar patch as a point through which the line passes, and then determine the direction of the line according to the density function. However, this method cannot generate a set of lines covering a 3D region like a sphere or a cube, since some of the lines required do not intersect the base, as illustrated in Figure 2.

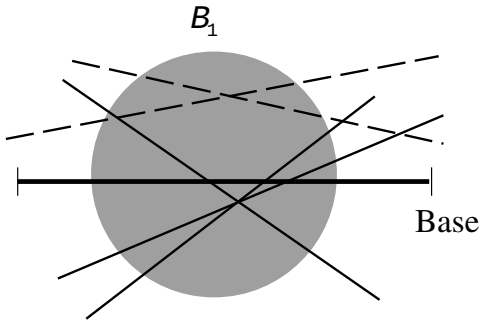

Figure 2: The dashed lines miss the base.

In the rest of this section we will study the application of two models for generating lines in 3D that are reported in [18] and [2], called the *chord model* and *tangent model*, respectively. The global nature of these models makes them particularly suitable for generating 3D uniformly distributed random lines. However, these models were originally proposed using only intuitive arguments [3,12]. Thus, we first demonstrate that these models

can be rigorously established by showing that the distribution of the lines generated by each of the two models has a density that is invariant under Euclidean transformations, as required in integral geometry for the correct application of Formula (4). Then procedures are presented for sampling the spaces of these models using uniform random variables in the interval [0,1].

3.1 The chord model

The *chord model* can be described as follows. A random line is defined to be a line passing through two independent uniformly distributed points on a sphere S_R of radius R in E^3. All such lines are all the lines in E^3 that intersect the sphere S_R. Since only uniformly distributed lines are acceptable in the application of Formula (4), we need to ensure the property that the random lines produced by the chord model have a uniform distribution. This property was derived in [18] by linking the chord model to another intuitive model using chord length distribution. In the following we provide a direct proof of this property by showing that random lines produced by the chord model have the density defined by Equation (2).

Consider the sphere S_R of radius R centred at the origin O. Let L be a line determined by two random points P_0 and P_1 on S_R. Since P_0 can be any point on S_R with equal probability, we define our coordinate system for this calculation so that $P_0 = (0, 0, -R)$. Let S denote a surface element of sphere S_R at the point P_0, i.e., S is tangential to S_R. Let $\mathrm{d}s$ denote the area element of S. Let $\bar{S}^2$ denote the unit sphere centred at P_0. Let S_R be parameterized by

$$Q_1(\beta, \alpha) = (R\sin\beta\cos\alpha, R\sin\beta\sin\alpha, R\cos\beta), \tag{5}$$

and let $\bar{S}^2$ be parameterized by

$$Q_2(\phi, \alpha) = (\sin\phi\cos\alpha, \sin\phi\sin\alpha, \cos\phi - R).$$

These two parameterizations are illustrated in Figure 3.

The area element at $Q_2(\phi, \alpha)$ on $\bar{S}^2$, which is $\sin\phi\,\mathrm{d}\phi\,\mathrm{d}\alpha$, is projected under the projection centred at P_0 to the surface area element of area

$$R^2\sin(\beta)\,\mathrm{d}\beta\,\mathrm{d}\alpha = 2R^2\sin(2\phi)\,\mathrm{d}\phi\,\mathrm{d}\alpha$$

at $Q_1(2\phi, \alpha)$ on S_R, since $Q_2(\phi, \alpha)$ is mapped to $Q_1(2\phi, \alpha)$, $\mathrm{d}\phi$ to $\mathrm{d}\beta = 2\mathrm{d}\phi$, and $\mathrm{d}\alpha$ to $R\mathrm{d}\alpha$. See Figure 4. Clearly, the unit direction vector of the line L,

which is $(P_1 - P_0)/|P_1 - P_0|$ and has one end attached at P_0, falls in a surface element of area $\sin\phi\,\mathrm{d}\phi\,\mathrm{d}\alpha$ at $Q_2(\phi,\alpha)$ on $\bar{S}^2$ if and only if point P_1 falls in a surface element of area $2R^2\sin(2\phi)\,\mathrm{d}\beta\,\mathrm{d}\alpha$ at $Q_1(2\phi,\alpha)$ on S_R. Hence, since P_1 is uniformly distributed on S_R, the density of the line that passes through P_0 and has direction vector $Q_2(\phi,\alpha) - P_0$ is

$$2R^2\sin(2\phi)\,\mathrm{d}\phi\,\mathrm{d}\alpha = 4R^2\cos\phi\sin\phi\,\mathrm{d}\phi\,\mathrm{d}\alpha = 4R^2\cos\phi\,\mathrm{d}\sigma,$$

where $\mathrm{d}\sigma = \sin\phi\,\mathrm{d}\phi\,\mathrm{d}\alpha$ is the area element on $\bar{S}^2$ for the direction of L. Dropping the proportionality constant $4R^2$, we obtain that the density of random lines produced by the chord model which intersect the planar surface element S at P_0 is $\cos\phi\ \mathrm{d}s\wedge\mathrm{d}\sigma$, where $\mathrm{d}s$ is the differential area of the element S. Since this density function is identical to (2), the random lines produced by the chord model have a uniform distribution.

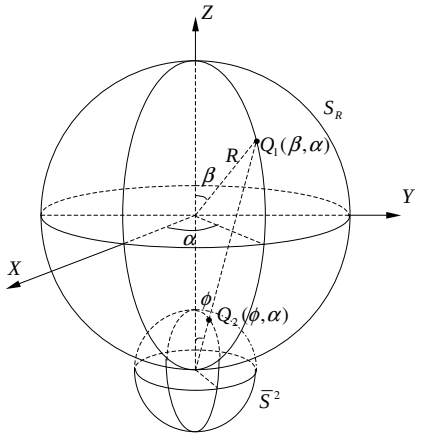

Figure 3: The parameterizations for the two spheres.

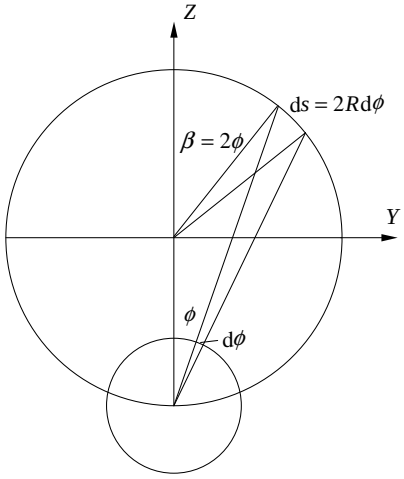

Figure 4: The 2D sectional illustration of the two spheres.

3.2 The tangent model

Beckers and Smeulder [2] derive a 4D model for lines in E^3 and the associated density of lines via intuitive invariance principles; we call this model the *tangent model*.

Let S_r denote a sphere of radius r centred at the origin. A point $\mathbf{x}$ on the sphere S_r is defined by three parameters r, θ, ϕ, where θ and ϕ are, respectively, the latitude and longitude of point $\mathbf{x}$ in the spherical coordinate system. Now consider the pencil of lines on the tangent plane of S_r at $\mathbf{x}$ with the centre of the pencil at $\mathbf{x}$. Let ψ be the angle for specifying a line L in this pencil with respect to a reference direction aligned with the appropriate great circle as shown in Figure 5. The domains of the parameters are $r \in [0, \infty)$, $\theta \in [0, \pi]$, $\phi \in [0, 2\pi)$, and $\psi \in [0, \pi)$. It is clear that a line in 3D is uniquely determined by the four parameters r, θ, ϕ, and ψ. As a matter of fact, there is a one-to-one correspondence between a line in 3D space and a parameter point in the 4D space of (r, θ, ϕ, ψ) unless $\theta = 0$ or π. Thus (r, θ, ϕ, ψ) can be regarded as a representation of lines in 3D space. We refer to this model as the *tangent model* because a line is represented as a tangent to some sphere of radius r in this model.

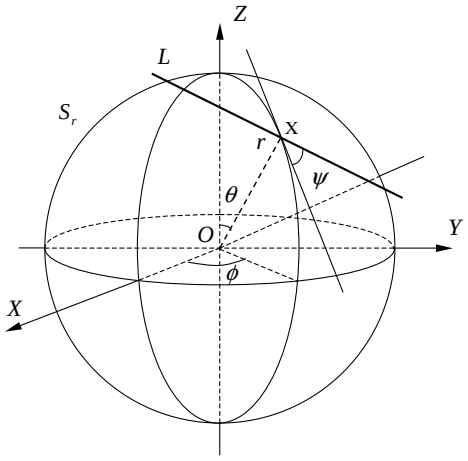

Figure 5: The four parameters (r, θ, ϕ, ψ) for a line in the tangent model.

It is shown in [2] that uniformly distributed lines which are defined in terms of some intuitive invariance principles in E^3 have density $cr \sin\theta$ in the tangent model, where c is a normalization constant. In the following we validate the tangent model by showing that, with the density function $cr \sin\theta$, it produces random lines with the same distribution as that of the random lines produced by the chord model.

For the sake of simplicity, and without loss of generality, we assume S_R to be the unit sphere S^2 centred at the origin when discussing intersections

of the tangent lines of S_r with S_R; thus $0 \le r \le 1$. The parameter domain of (r, θ, ϕ, ψ) is $[0,1] \times [0,\pi] \times [0, 2\pi) \times [0, \pi)$ for generating all the lines intersecting S^2. Clearly, the length ℓ of the chord that is on a tangent to S_r (a sphere of radius r centred at the origin) and intersected within S^2 satisfies $r = \sqrt{1 - (\ell/2)^2}$, where $0 \le r \le 1$. Let $g(\ell)$ be the density of ℓ, and $f(r)$ the density of r. Then

$$g(\ell) = f(r) \left| \frac{\mathrm{d}r}{\mathrm{d}\ell} \right|,$$

by a change of random variable. On the other hand,

$$f(r)|\mathrm{d}r| = \int cr \sin\theta |\mathrm{d}\theta \, \mathrm{d}\phi \, \mathrm{d}\psi||\mathrm{d}r| = 4c\pi^2 r|\mathrm{d}r|,$$

with the integration running over the domains of θ, ϕ, and ψ. Since

$$\left| \frac{\mathrm{d}r}{\mathrm{d}\ell} \right| = \frac{\ell}{2\sqrt{1 - (\ell/2)^2}} = \frac{\ell}{2r},$$

we get

$$g(\ell) = f(r) \left| \frac{\mathrm{d}r}{\mathrm{d}\ell} \right| = 4c\pi^2 r \cdot \frac{\ell}{2r} = 2c\pi^2 \ell.$$

Since the length ℓ of a chord within a unit sphere satisfies $0 \le \ell \le 2$, we have $\int_0^2 g(\ell) = 1$. It follows that $c = 1/(4\pi^2)$, *i.e.*, $g(\ell) = \ell/2$.

On the other hand, it is shown in [18] that the chord length of random lines produced by the chord model also has the density distribution $h(\ell) = \ell/2$. Because of the directional homogeneity of the two models, we conclude that the tangent model, like the chord model, generates uniformly distributed lines in 3D space.

The above 4D space of (r, θ, ϕ, ψ) serves as the starting point of our method for generating uniformly distributed lines in 3D space intersecting a sphere S_R of radius R. Since the density of lines in E^3 is proportional to $cr \sin\theta$, if we can generate a random point in the 4D parameter space (r, θ, ϕ, ψ) with probability density $cr \sin\theta$, then this random point will give a uniformly distributed random line in E^3. Hence we just need to generate a random point in (r, θ, ϕ, ψ) with density proportional to $cr \sin\theta$.

First, we may easily generate $\phi \in [0, 2\pi)$ and $\psi \in [0, \pi]$ with uniform distribution. Next, we need to generate $r \in [0, R]$ with density $k_0 r$ and $\theta \in [0, \pi]$ with density $k_1 \sin\theta$ for some constants k_0 and k_1; to make $k_0 r$ and

$k_1 \sin\theta$ legitimate probability density functions, we must set $k_0 = 2/R^2$ and $k_1 = 1/2$. Then it is easy to see that the cumulative probability distribution functions for r and θ are $G(r) = r^2/R^2$ and $H(\theta) = \frac{1}{2}(1 - \cos\theta)$, respectively.

Now we consider generating a random variable with a pre-specified cumulative distribution $F(y)$. Suppose that β is a random variable with distribution function $F(y)$, which is assumed to be a continuous and non-decreasing function. Define $F^{-1}(y) = \inf\{x|F(x) = y\}$. Let α be a uniformly distributed random variable in $[0, 1]$. Then we claim that $\beta = F^{-1}(\alpha)$ is a random variable with distribution $F(y)$, for $P(\beta \leq y) = P(F^{-1}(\alpha) \leq y) = P(\alpha \leq F(y)) = F(y)$, where $P()$ stands for the probability of an event. Hence, the desired random variable β with distribution $F(y)$ can be generated by function $\beta = F^{-1}(\alpha)$ of a uniform random variable α in $[0, 1]$. By a straightforward application of this argument, r and θ can be generated by $r = R\sqrt{\alpha}$ and $\theta = \arccos(1 - 2\alpha)$, respectively, where α is a uniform random variable in [0,1].

The above provides us with the means to generate a set of uniformly distributed lines that samples all the lines intersecting a sphere centred at the origin and of radius R. Hence, the reference body B_1 defined in Section 2 should be set to a sphere of radius R centred at the origin. To recap, there are two main steps: 1) Generate the four random parameter values (r, θ, ϕ, ψ) with their respective densities as given above; 2) Construct the unique line from the values of these four parameters. The detailed procedure for computing a parametric equation of the line is then straightforward.

4 Low discrepancy sequences

The approach to computing the surface area based on Formula (4), as we have pursued so far, is essentially a Monte Carlo method for numerical integration, with the domain of integration being the 4D parameter space (r, θ, ϕ, ψ) of 3D lines in the case of the tangent model. It is known [13] that uniformly distributed random points are not distributed as evenly as so-called *low discrepancy* sequences of points for the purpose of accurate numerical integration. Hence, we will use low-discrepancy sequences, instead of pseudo-random number generators, for generating the set $\mathcal{L}$ of evenly distributed lines. for this reason our method is called a quasi-Monte Carlo method. Such methods have already been used for volume computations in CSG modelling [3].

Below we briefly introduce the concept of low discrepancy sequences, following [13]. Given a set of numbers x_i, $i = 1, 2, \ldots, N$, and a set E contained in an interval I, define

$$A(E; N) = \sum f_E(x_i),$$

where $f_E(x)$ is the characteristic function of the set E, i.e., $f_E(x) = 1$ if $x \in E$ and $f_E(x) = 0$ otherwise. The discrepancy of the sequence x_i over the interval I is defined to be

$$D_N = \sup_J \left| \frac{A(J; N)}{N} - |J| \right|,$$

where J runs through all subintervals of I and $|J|$ is the length of J. Together with the regularity of the integrand, the discrepancy of a sequence of points provides an error bound for numerical integration using the sequence. For a function $f(x)$ with bounded variation $V(f)$ over $I = [0, 1]$, it can be shown [13] that

$$\left| \frac{1}{N} \sum_{i=1} f(x_i) - \int_0^1 f(x) \mathrm{d}x \right| \leq V(f) D_N^*,$$

where D_N^* has the same definition as D_N but with the subinterval J having the form $[0, t]$, $t \leq 1$. Clearly, $D_N^* \leq D_N$. Hence, the lower the discrepancy, the better is the distribution of the sequence, and the more accurate is the numerical integration.

Discrepancy can also be defined for a box $[0, 1]^s$ in a s-dimensional space. It is known that the expected value of the discrepancy of a statistically uniformly distributed random variable is $O(N^{-1/2})$. In contrast, the discrepancy of Niederreiter's sequence of points in the box $[0, 1]^s$ is $O(N^{-1} \log^s N)$. This means low discrepancy sequences, such as Niederreiter's, yield asymptotically smaller error bounds for numerical integration in our present setting.

When using low discrepancy sequences for multidimensional integration, we can only guarantee an improvement for a box-like region of integration in the variables concerned. In practice, we do get improvements for other shaped regions too. Hence, in our quasi-Monte Carlo method we use Niederreiter's 4D low discrepancy sequences of points in either the chord model or the tangent model. This is done in the 4D space $(\beta, \alpha, \beta', \alpha')$ for the chord model, where (β, α) and (β', α') are two independent pairs of parameters for

generating two independent points on the sphere S_R through $Q_1(\beta, \alpha)$ as defined in Section 3.1, and in the (r, θ, ϕ, ψ) space for the tangent model as defined in Section 3.2.

Figure 6 and Figure 7 illustrate the difference in distribution evenness of two sets of 1000 points on a sphere; the points in Figure 6 are generated using a pseudo-random number generator and the points in Figure 7 are generated using Niederreiter's 2D low discrepancy sequence, through the parameterization of a sphere given by equation (5). The sampling of the surface is clearly more even in the latter case. Note that we do not want a *perfectly* even spacing when performing numerical integration, since that can lead to unwanted aliasing artefacts.

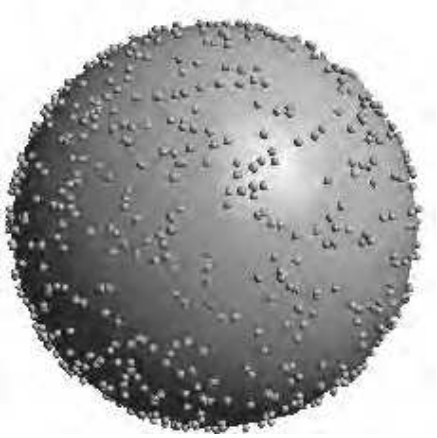

Figure 6: Pseudo-random points.

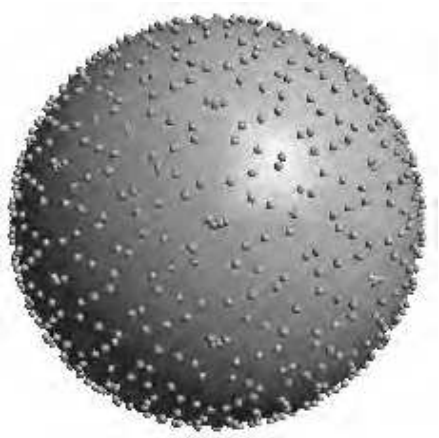

Figure 7: Niederreiter's low discrepancy sequence of points.

5 Intersecting a line with a CSG solid

Let Σ be the bounding surface of a CSG body B consisting of a number of primitive surfaces. Suppose that a set of uniformly distributed lines, denoted by $\mathcal{L}$, has been generated. The next step of our method for computing the surface area of Σ is to determine the number of intersection points between each of the lines in $\mathcal{L}$ and the surface Σ. This can be done by computing the

intersection points of a line ℓ with all the primitive surfaces of B and then merging the results according to the Boolean operations associated with B to obtain the number of intersection points of the line ℓ with the bounding surface of B. As the result of this merging step, intersection points not on the bounding surface of B are removed and only those on the surface of B are retained and counted. A detailed account of intersecting a line with a CSG object, but for ray-tracing purposes, can be found in [5].

Although only the *number* of intersections of the line ℓ with the body B matters eventually for computing the surface area of B, it is still necessary to determine the *locations* of the intersection points of ℓ with all the primitive surfaces in order to classify these intersections points with respect to the various surfaces. The intersection of a primitive surface and a line ℓ can be computed accurately if the primitive surface is an algebraic surface of degree 4 or less. Numerical root finding methods must be used if more general surfaces are used in the CSG definition. Since CSG primitives encountered in CAD/CAM applications are mainly planes, natural quadrics, and tori or cyclide surfaces, which are of degree 4 or less, our method is applicable to these most common cases without resorting to numerical solvers.

When one needs to compute the surface area of a part of just a single algebraic surface $f(x, y, z) = 0$ (not necessarily of low degree), the number of intersections (*without* their corresponding locations) between a line and the part of the surface can be determined efficiently using well known real root isolation techniques, such as Sturm sequences, without having to solve numerically for the roots of a possibly high degree polynomial equation.

6 Implementation and experiments

In this section we first give experimental results demonstrating the convergence of the estimated surface areas of some simple objects, i.e., a cube of side-length 10, a sphere of radius 10, and a cylinder of radius 10 and height 20, followed by some examples involving more complicated objects. These examples are computed using two different methods: one is the standard Monte Carlo method using 3D lines generated with pseudo-random numbers and the other is the quasi-Monte Carlo method using Niederreiter's low discrepancy sequences. Further, we compare the efficiency of the quasi-Monte Carlo method with a conventional surface tessellation method based on the marching cubes algorithm [12]. Finally, we compare our method with a re-

cursive tessellation method for surface area computation.

Pseudo-random numbers used in these tests were generated by the pseudo-random number generator provided by the Visual C++ library. For generating Niederreiter's sequences of points in the 4D space of (r, θ, ϕ, ψ), we used a C++ implementation adapted from the FORTRAN code available from [1]. The test program was implemented in C++ and ran on a PC using a Pentium II 233MHz CPU. For the simple objects (cube, sphere, and cylinder) used in this section, the enclosing reference body B_1 defined in Section 2 was a sphere of radius $R = 18.849$, except that an enclosing sphere of radius 179.07 was used for the CSG object in Figure 12. These spheres were chosen to be slightly larger than, and therefore contain, the tightest enclosing spheres of the rectangular bounding boxes of given objects; the bounding spheres of the simple objects have the same radius ($R = 18.849$) because these objects happen to have the same bounding boxes. Other types of bounding surfaces (i.e., reference surfaces), such a rectangular box or a cylinder, could also be used with the method; in general, for better accuracy, one should use a bounding surface which bounds a given object tightly and whose exact surface area is easy to obtain. All errors measured and presented below are relative errors.

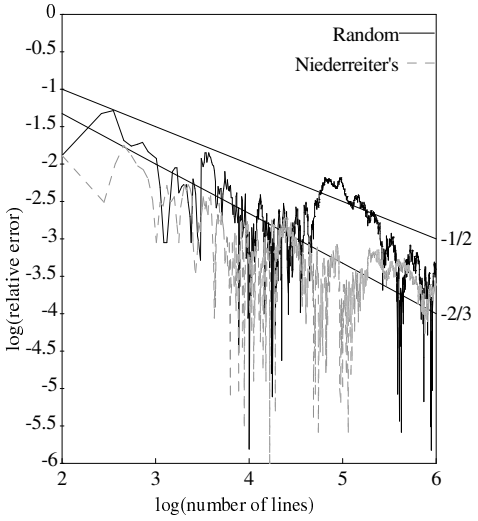

Figure 8: Errors for the cube, with the tangent model.

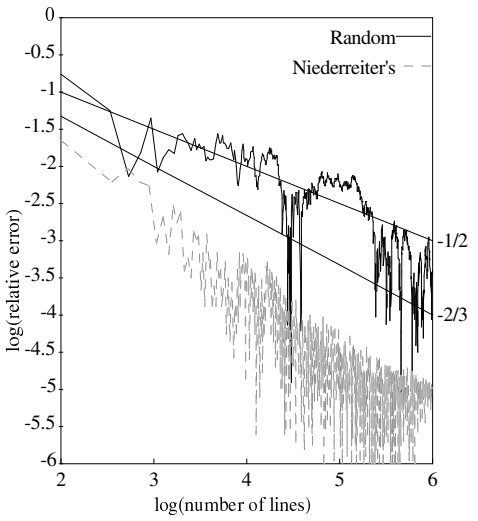

Figure 9: Errors for the sphere, with the tangent model.

Figures 8–10 show the curves of relative approximation errors generated

by the standard Monte Carlo method using pseudo-random numbers and the quasi-Monte Carlo method using Niederreiter's sequences for the cube, the sphere, and the cylinder, respectively, using the tangent model for lines (see Section 3.2). The comparisons in Figures 8–10 suggest that using the low discrepancy sequences leads to smaller approximation errors than using pseudo-random numbers. The reference lines marked with $-\frac{1}{2}$ and $-\frac{2}{3}$ in the figures are the graphs of the functions $n^{-1/2}$ and $n^{-2/3}$, respectively, for revealing the trend of the error curves; we expect the standard Monte Carlo method to have error $O(n^{-1/2})$, where n is the number of lines used.

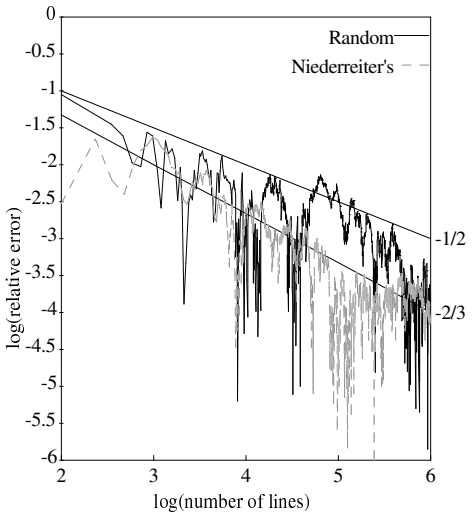

Figure 10: Errors for the cylinder, with the tangent model.

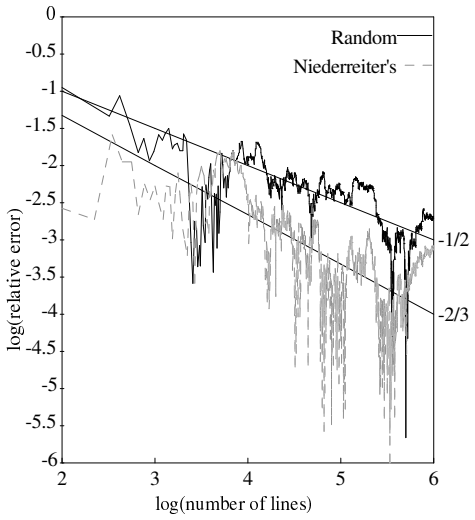

Figure 11: Errors for the object L±cylinder in Figure 12, with the tangent model.

Figure 11 shows the error curves generated by the standard Monte Carlo method and by the quasi-Monte Carlo method, respectively, for the CSG solid shown in Figure 12. The error curves in Figure 11 also show that the error arising from using low discrepancy sequences is in general smaller than the error arising from using pseudo-random numbers.

The surface area of an object can also be computed approximately by using the marching cubes method to generate a triangulation of the object's surface and summing the areas of the triangles in the triangulation. Figure 13 shows the error curves generated by the quasi-Monte Carlo method and

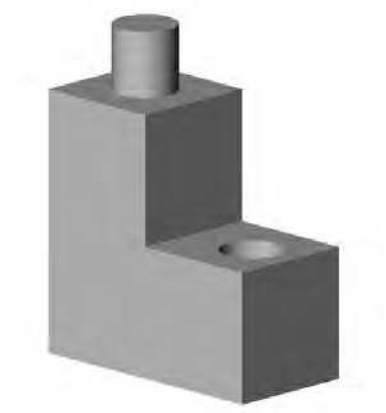

Figure 12: A CSG object: L±cylinder

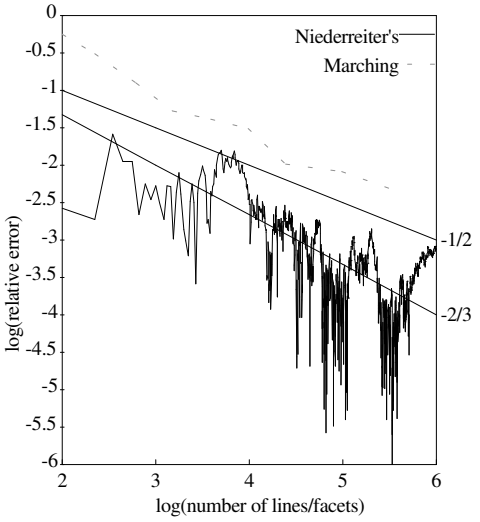

Figure 13: Comparison with surface tessellation.

by the marching cubes method, respectively, for the CSG object in Figure 12, which will be referred to as *L±cylinder*. We are careful to make comparable the number of lines used in the quasi-Monte Carlo method and the number of triangles generated by the marching cubes method; thus, for varying k, the errors resulting from using 10^{2k} lines in the former are compared in Figure 13 with the errors resulting from using 10^{3k} cubes in the latter, which generates $O(10^{2k})$ triangles on the surface of the L±cylinder. According to Figure 13, the error given by the marching cubes method, though decreasing more steadily, is always larger than the error given by the quasi-Monte Carlo method.

Figures 14–17 show the error curves for the same set of four objects, i.e., cube, sphere, cylinder, and L±cylinder, using the quasi-Monte Carlo method with the *chord model* for generating lines (see Section 3.1). These figures again demonstrate that, with the chord model, the quasi-Monte Carlo method using low discrepancy sequences produces better results than the standard Monte Carlo method using pseudo-random numbers.

Figure 18 compares the errors for the object L±cylinder in Figure 12 given by the chord model and the tangent model, respectively, used in the quasi-Monte Carlo method. The radius of the reference sphere used was

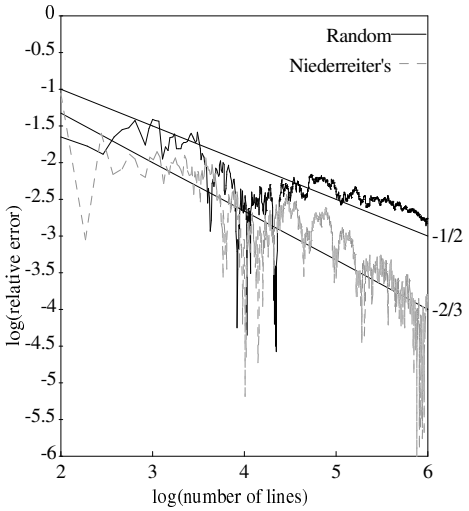

Figure 14: Errors for the cube, with the chord model.

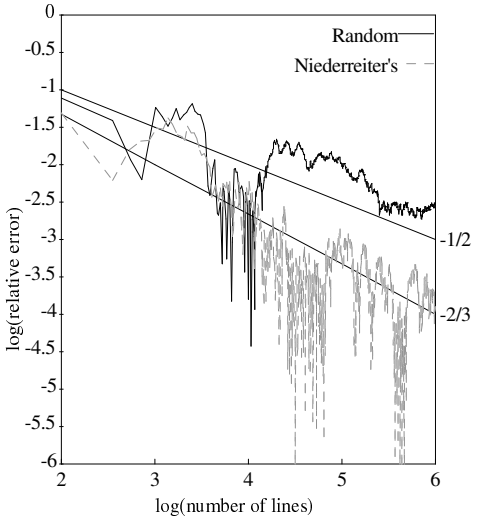

Figure 15: Errors for the sphere, with the chord model.

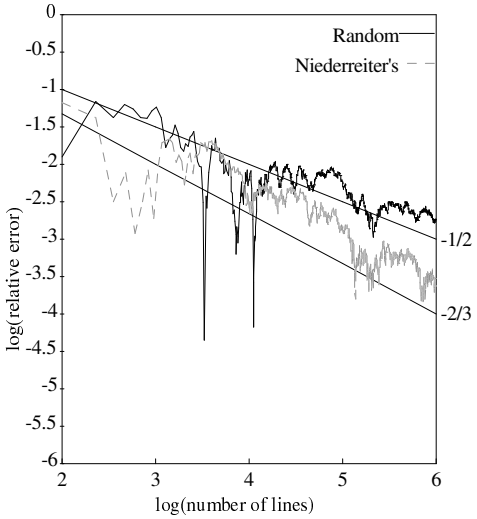

Figure 16: Errors for the cylinder, with the chord model.

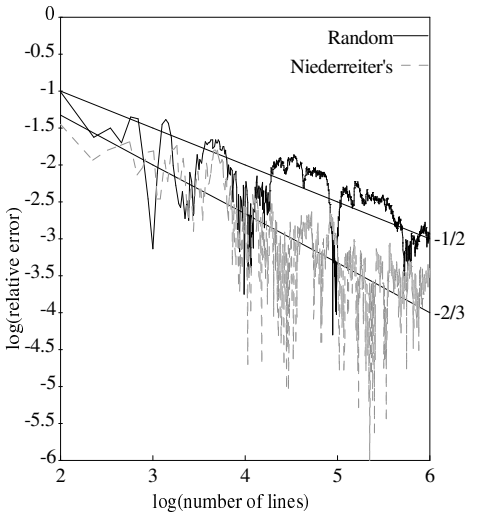

Figure 17: Errors for the L±cylinder in Figure 12, with the chord model.

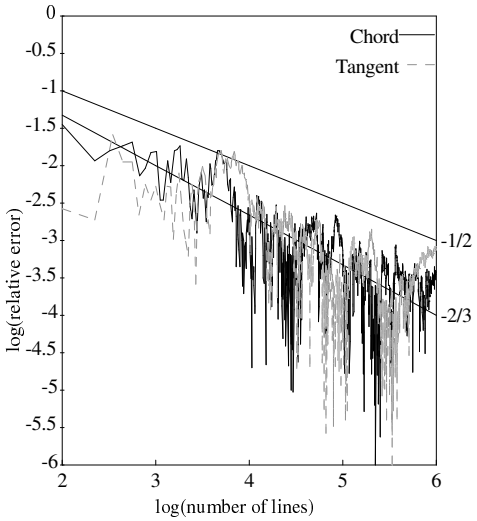

Figure 18: Comparison between the chord model and the tangent model for the L±cylinder in Figure 12.

179.07. We may see that neither model appears to have a clear advantage over the other in terms of accuracy; however, in our experience, the chord model is easier to implement and has faster running time than the tangent model.

Now we present two more examples of CSG objects of considerable complexity to show the efficiency of the quasi-Monte Carlo method. The first example is the CSG difference between a box of side-length 20 and a sphere of radius 13, as shown in Figure 19; we call this object the 'skeleton'. The second object is the 'gate' shown in Figure 20.

Figures 21 and 22 show the curves of relative approximation errors generated by the Monte Carlo method using pseudo-random numbers and the quasi-Monte Carlo method using Niederreiter's sequences for the 'skeleton' and the 'gate', respectively; the tangent model was used.

Figures 23 and 24 show the same using the chord model.

Next we compare our method with two other methods that use surface tessellation to compute surface areas: the marching cube method and the adaptive subdivision method. Table 1 lists the timings of computing the area of the L±cylinder in Figure 12 with three different methods: the standard Monte Carlo method, the quasi-Monte Carlo method, and the marching cube

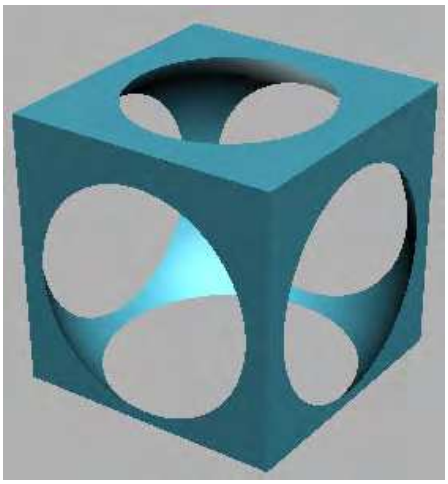

Figure 19: The 'skeleton'.

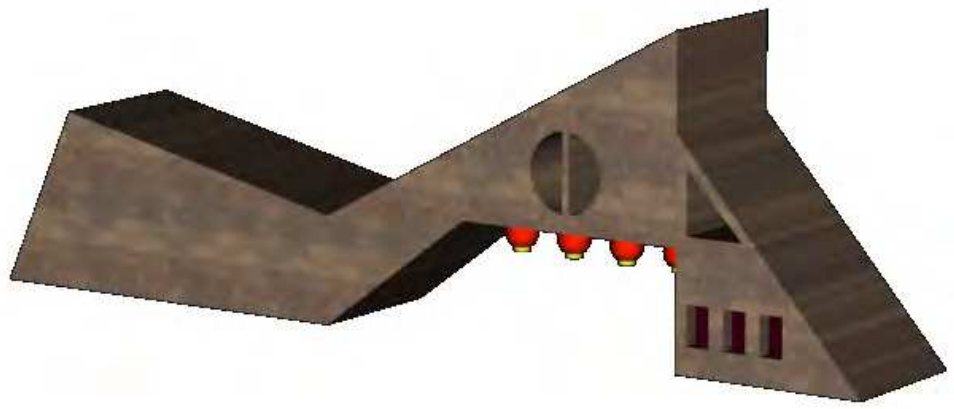

Figure 20: The 'gate'.

method; 10^4 lines are used in the first two methods, and 10^6 cubes used in the marching cube method.

Table 1 shows the superior timing efficiency of the quasi-Monte Carlo method, and even that of the standard Monte Carlo method, over the surface tessellation approach based on the marching cube method. As explained before, we use 10^6 cubes in the marching cube algorithm in order that the number of triangles generated is of the same order as the number of lines used in the other two methods; the number of resulting triangles is 29212,

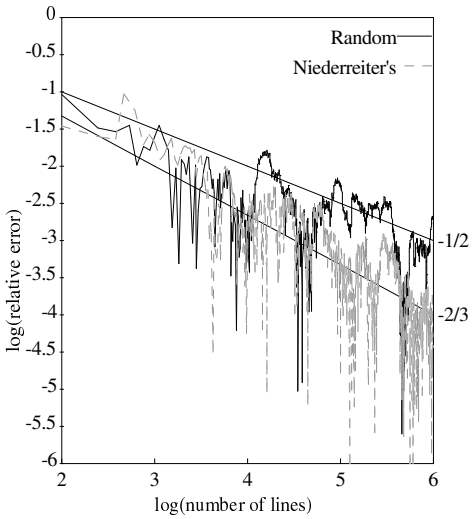

Figure 21: Errors for the 'skeleton', with the tangent model.

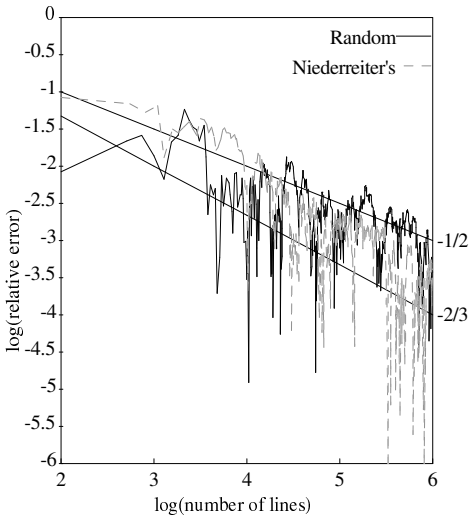

Figure 22: Errors for the 'gate', with the tangent model.

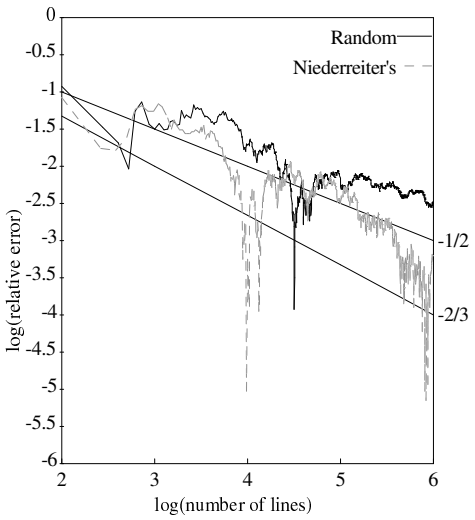

Figure 23: Errors for the 'skeleton', with the chord model.

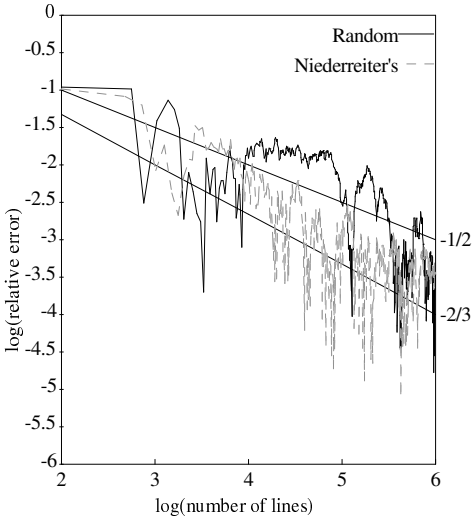

Figure 24: Errors for the 'gate', with the chord model.

but still produces much larger error than the quasi-Monte Carlo method does with 10000 lines.

Method	Timing	No. of elements	Relative error
Random	1.439 sec.	10000 lines	0.016615
Niederreiter	1.450 sec.	10000 lines	0.007674
Marching Cube	98.21 sec.	29212 triangles	0.025766

Table 1: Timing comparison with the marching cube method.

Since the marching cube method is not very efficient for surface tessellation, a more efficient surface tessellation method based on recursive subdivision implemented in svLis was compared with the method proposed in this paper. SvLis is a geometric modeller authored by Adrian Bowyer at the University of Bath. SvLis uses recursive spatial division combined with faceting to compute areas. A CSG geometric model consisting of unions, intersections, and complements of primitives represented by implicit functions of the three space variables is initially surrounded by an axis-aligned box large enough to contain the whole object being defined. This box is recursively divided into a binary tree of smaller boxes by further axis-aligned planes. As each new smaller child box is generated the set-theoretic expression defining the contents of its larger parent box is pruned to the child box – that is it is simplified using the normal rules of logic such as the Absorption Law and De Morgan's Law, together with information about which parent primitives lie wholly outside or wholly within the child box. This information is obtained by treating the boxes as three affine intervals in the three space variables, and substituting these intervals into the implicit functions defining the primitives. For more details of this process, see [21] or the svLis website [http://www.bath.ac.uk/~ensab/G_mod/Svlis/].

The terminating conditions for the recursive box division are two-fold: recursion stops when there are three or fewer primitives in a box and when each of those primitives has a range of grad vectors in the box smaller than a pre-determined level. The first condition divides down to surfaces, edges and corners, and the second divides curved surfaces until reasonably locally flat parts of them are found. This second division criterion is also used to decide which of the three possible division directions to use at each stage – the direction is chosen that splits the surfaces to give the best distribution

of grad vectors. Thus cylinders, for example, get divided into long strips parallel to their axes (though there is an aspect ratio constraint on this to stop the division boxes getting too long and thin).

Once the model has been divided, svLis facets it by decomposing each leaf box in the tree that contains model surface into a pattern of packed tetrahedra. The points where the primitives cut the edges of these tetrahedra are found by binary division, and triangles or quadrilaterals with vertices at those points are used to approximate the primitives.

The quality of the faceting, and thus the accuracy of the area calculation, therefore depends on the fineness of the grad vector criterion, and the effect that this has on whether a surface is considered flat enough to facet. The experiments for this paper were done by gradually refining this figure until two areas for a shape were found, one less accurate that the required error, and one more. The timing for the actual error being aimed at was found by linearly interpolating between the times that gave rise to those two area values.

Tables 2 and 3 show the comparison between the recursive subdivision method and the quasi-Monte Carlo method. Table 2 gives the timings in seconds required by the recursive subdivision method to achieve various relative errors for three solid models: a sphere of radius 20, the 'skeleton' shown in Figure 19 and the object L±cylinder shown in Figure 12. Table 3 gives the respective timing data for the new method. The true areas for the three objects are 5026.55, 1752.83, and 1557.08, respectively. The data in Table 2 were generated on a PC with 1.2GHz Pentium CPU running Linux RedHat 7.2. The data in Table 3 were generated on a PC with 1.2GHz Pentium CPU running MS Windows 2000.

It is noted that the new method is more efficient than the recursive subdivision method for the sphere and the 'skeleton' but is slower than the latter for the object L±cylinder. We speculate that the recursive subdivision method does not need to subdivide the surface of the object L±cylinder to a very fine level in order to obtain a good approximation of the surface by polygonal facets, since the major part of the surface of the L±cylinder is planar, while the new method based on line-surface intersection does not exploit the planarity of the object surface for a possible speedup; hence, the recursive subdivision method is faster in this case. This comparison also suggests that the new method tends to perform better for objects with mainly curved boundary surfaces.

Note also that Tables 2 and 3 lead us to believe that the quasi-Monte

Carlo method will scale better if even results of higher accuracy are required.

Relative Error	Sphere	Box-sphere	L±cylinder
8%	0.0686281	0.0897207	0.0852837
4%	0.0985547	0.408693	0.0942379
2%	0.517069	0.859189	0.0969864
1%	0.539595	1.11492	0.0989368
0.5%	1.91135	2.21722	0.099912
0.25%	2.05938	3.55824	0.180743

Table 2: Timings (seconds) for the recursive subdivision method.

7 Conclusion

We have presented a quasi-Monte Carlo method for computing the surface area of a CSG object. This method is based on a classical result in integral geometry, the Cauchy-Crofton formula. To devise a practical and efficient method, we have investigated the problem of generating a set of evenly distributed lines in 3D space using Niederreiter's low discrepancy sequences. Our experiments show that the quasi-Monte Carlo method delivers, in general, more accurate results and better timing performance for geometric models with largely curved boundary surfaces than the conventional surface tessellation approach or the standard Monte Carlo method using random lines with pseudo-random uniform distribution.

8 Acknowledgment

The authors would like to thank Y.T. Lee for useful discussions that helped improve an earlier version of this paper. Thanks also go to the two anonymous referees who gave valuable comments on this work.

Relative Error	Sphere	Box-sphere	L±cylinder
8%	0.010600	0.029900	0.029000
4%	0.042400	0.080800	0.094800
2%	0.074650	0.130150	0.161900
1%	0.106600	0.181550	0.228300
0.5%	0.138700	0.234000	0.293900
0.25%	0.169700	0.334700	0.359600

Table 3: Timings (seconds) for the quasi-Monte Carlo method.

References

[1] P. Bratley, B.L. Fox, H. Niederreiter, Algorithm 738: Programs to generate Niederreiter's low-discrepancy sequences, *ACM Transactions on Mathematical Software*, vol. 20, no. 4, pp. 494–495, 1994.

[2] A.L.D. Beckers, A.W.M. Smeulders, The probability of a random straight line in two and three dimensions, *Pattern Recognition Letters* vol. 11, pp. 233–240, April 1990.

[3] T. J. G. Davies, R. R. Martin, A. Bowyer, Computing volume properties using low-discrepancy sequences, *Computing [Suppl]*, vol. 14, pp. 55–72, 2001.

[4] I.D. Faux, M.J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood Publishers, 1985.

[5] J. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

[6] H. Pottmann, J. Wallner, *Computational Line Geometry*, Springer, 2001.

[7] C. Gonzales-Ochoa, S. McCamnon, J. Peters, Computing moments of objects enclosed by piecewise polynomial surfaces, *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 143–157, 1998.

[8] C.V. Howard, M.G. Reed, *Unbiased Stereology: Three-Dimensional Measurement in Microscopy*, Bios Scientific Publishers Limited, 1998.

[9] Y.T. Lee, A.A.G. Requicha, Algorithms for computing the volume and other integral properties of solids, Part I, *Communications of The ACM*, vol. 25, no. 9, pp. 635–641, 1982.

[10] Y.T. Lee, A.A.G. Requicha, Algorithms for computing the volume and other integral properties of solids, Part II, *Communications of The ACM*, vol. 25, no. 9, pp. 642–650, 1982.

[11] Y.T. Lee, Personal Communication, 2001.

[12] W. Lorensen, H.E. Cline, Marching cubes: a high resolution 3D surface construction algorithm, *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[13] H. Niederreiter, Quasi-Monte Carlo methods and pseudo-random numbers, *Bulletin of the American Mathematical Society*, vol. 84, no. 6, pp. 957–1041, 1978.

[14] Prisant, M.G. Applications of the ray representation to problems of protein structure and function, *Proceedings of CSG 96*, Information Geometers, Winchester, pp. 33–47, 1996.

[15] L.A. Santaló, *Introduction to Integral Geometry*, Hermann, Paris, 1953.

[16] L.A. Santaló, Integral geometry, *Studies in Global Geometry and Analysis*, The Press of The Mathematical Association of America, pp. 147–167, 1967.

[17] R. Sarraga, Computation of surface areas in GMSolid, *IEEE Computer Graphics and its Applications*, vol. 2, no. 7, pp. 65–70, 1982.

[18] H. Solomon, *Geometric Probability*, SIAM, Philadelphia, 1978.

[19] H. G. Timmer, J. M. Stern, Computation of global properties of solid objects, *Computer-Aided Design*, vol. 12, no. 6, pp. 301–304, 1980.

[20] R. F. Tobler, T. M. Galla, W. Purgathofer, ACSGM – an adaptive CSG meshing algorithm, *Proceedings of CSG 96*, Information Geometers, Winchester, pp. 17–31, 1996.

[21] I. Voiculescu, J. Berchtold, A. Bowyer, R. R. Martin, Q. Zhang, Interval and affine arithmetic for surface location of power- and Bernstein-form polynomials, in *The Mathematics of Surfaces IX*, R. Cipolla and R. Martin, eds., pp. 410–423, Springer, 2000.

[22] A. Wilkie, R. F. Tobler, W. Purgathofer, Photon radiosity lightmaps for CSG solids, *Proceedings of CSG 98*, Information Geometers, Winchester, pp. 155–167, 1998.

[23] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C*, Cambridge University Press, pp. 274–316, 1992.