Pattern Recognition 43 (2010) 3900–3909

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Surface area estimation of digitized 3D objects using quasi-Monte Carlo methods

Yu-Shen Liu [a,b,c,*], Jing Yi [a], Hu Zhang [a], Guo-Qin Zheng [a], Jean-Claude Paul [a,d]

[a] School of Software, Tsinghua University, Beijing 100084, China
[b] Key Laboratory for Information System Security, Ministry of Education of China, Beijing, China
[c] Tsinghua National Laboratory for Information Science and Technology, Beijing, China
[d] INRIA, France

ARTICLE INFO

Article history:
Received 2 January 2010
Received in revised form
29 May 2010
Accepted 3 June 2010

Keywords:
Surface area estimation
Digital geometry
Cauchy–Crofton formula
Quasi-Monte Carlo methods
Low-discrepancy sequences

ABSTRACT

A novel and efficient quasi-Monte Carlo method for estimating the surface area of digitized 3D objects in the volumetric representation is presented. It operates directly on the original digitized objects without any surface reconstruction procedure. Based on the Cauchy–Crofton formula from integral geometry, the method estimates the surface area of a volumetric object by counting the number of intersection points between the object's boundary surface and a set of uniformly distributed lines generated with low-discrepancy sequences. Using a clustering technique, we also propose an effective algorithm for computing the intersection of a line with the boundary surface of volumetric objects. A number of digitized objects are used to evaluate the performance of the new method for surface area measurement.

1. Introduction

Length, area and volume are important mass properties of geometric objects that need to be computed frequently in many medical, biological and industrial applications. In this paper, we study the particular problem of estimating the surface area of a digitized three-dimensional (3D) object in a volumetric representation. This problem may arise in various applications of digital geometry [3,9,17,24,36,38,41]. For instance, the cortical surface area is likely to be related to functional capacities in the analysis of the cortex in MR images [38,40]. In hemodialysis, the peritoneal surface area is considered an important factor of dialysis effectiveness [3]. Another class of applications is in 3D shape recognition and matching which focus on the area of geometric measurements [17]. Furthermore, surface area estimation of the landscape from digital elevation models (DEMs) plays an important role, especially in landscape analysis and studies of wildlife habitat [12].

Digital geometry analysis aims at measuring properties of the continuous world on the basis of digitized objects [17]. A digitized object consists of a finite number of lattice points or voxels. The area measurement can only be estimated in the digitized object, but rather in the original, pre-digitized object [24]. A number of previous studies have concerned this estimation. A direct way is to compute the total area of exposed voxel faces. However, since the voxel representation of smooth continuous surfaces is generally jagged, the computed area is usually much greater than that of the original continuous surface, resulting in over-estimation [24,38].

Another possible solution is to reconstruct a polygonal mesh of an iso-surface from the input volumetric object and then sum up the area of the resulting polygons. The classical approaches, such as the Convex Hull algorithm [2], consider a polygonalization of the boundary point set and estimate the surface area by summing the area of the facets. The marching cubes algorithm and its variants [27,30,39] are often used for extracting the iso-surfaces. However, there are some limitations on estimating surface area of volumetric objects based on reconstruction strategy [24,38]. First, when the marching cubes algorithm is used to create the triangulated representation, topology ambiguities may occur and holes may be generated. Second, the surface area estimated based on the local polyhedrization techniques does not converge to the true surface area as the resolution increases [16]. Furthermore, computational complexity of many reconstruction-based methods is rather high [24,38], as they require an explicit approximation of the boundary of the object. In contrast, the surface area estimation in our work operates directly on the volumetric objects without any explicit or implicit reconstruction procedure.

A third type of approaches is the voxel-based area estimation [24,29,37,38,41] by extending some existing planar perimeter

* Corresponding author at: School of Software, Tsinghua University, Beijing 100084, China. Tel.: +86 10 6279 0533; Mobile: +86 159 1083 1178.
E-mail addresses: liuyushen@tsinghua.edu.cn, liuyushen00@gmail.com (Y.-S. Liu).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.06.002

estimation methods (e.g. [7,20]). The main advantage is that the surface area is measured directly from the binary volume and no polygonal mesh is carried out. One early work was proposed by Mullikin and Verbeek [29], where their estimation is designed to be unbiased by minimizing the mean-square error for planes. The core of voxel-based approaches is to assign surface area weights to the voxels of a digitized object, then the total surface area can be estimated by summing up the weighted area contributed by all surface voxels of the object. Windreich et al. [38] delimited a region of interest on the surface of the digitized object and estimated its area using Mullikin and Verbeek's method. In contrast to [29], which used the six face-neighbors of a voxel, Lindblad improved the surface estimation by considering the $2\times2\times2$ neighborhood in Ref. [24] (the preliminary version appeared as [23]). Other extensions to surface area estimation based on weighted counts of voxels can be found in Refs. [35,41]. Since the choice of the weights is not unique, there is a number of competing choices motivated by geometric arguments [41]. Although the weights are assigned to the local configuration without need to explicitly reconstruct a polygonal mesh, these weight-based approaches are similar to the marching cubes algorithm and they are essentially the implicit reconstruction methods. In addition, Coeurjolly and Flin et al. [5,9] present the algorithm for surface area estimation based on discrete normal vector field integration. Several publications [5,18,19] have also studied the multigrid convergent surface area estimation; the reader may consult [17] for detailed expositions.

Alternatively, the surface area can be estimated in a statistical sense [11,21]. One strategy is the line intersection count method [11] that is a stereology estimation for 2D perimeter and 3D surface area. This estimation involves four steps: (1) generating a line probe, (2) finding object border in a digital object, (3) counting object-probe intersections, and (4) applying stereological formulas to obtain area estimation. Nevertheless, the stereology estimation strongly depends on the pre-selection of start position and direction of line probes for area estimation [11]. Recently, Legland et al. [21] presented a local estimation for the perimeter and surface area of a discretized 2D or 3D set by discretization of Crofton formula using filtering and look-up table transformations. In this paper, we present an alternatively stochastic algorithm for surface area estimation of digitized 3D objects. Like the previous stereology approach, our method is based on ray casting, but it uses an entirely different principle for computing the area by random rays. Furthermore, we use low-discrepancy sequences to obtain good statistical properties.

The work most related to ours is Ref. [22], in which a quasi-Monte Carlo method is presented for computing the surface area of a constructive solid geometry (CSG) model in computer aided design. It is based on the Cauchy–Crofton formula, and performs the area estimation by counting the number of intersection points between the boundary surface of the CSG model and a set of uniformly distributed lines. In essence, the Cauchy–Crofton formula transforms a problem of estimating the surface area into a problem of counting the intersection points. In the same spirit, this paper mainly discusses how to apply the quasi-Monte Carlo method to a digitized 3D object in the volumetric representation for quickly and efficiently estimating the area of the object's underlying surface. The main challenge in our work is to determine all intersection points between a line and the volumetric object. For this purpose, we present a new algorithm of intersecting a line with a digitized object's boundary, which is based on a clustering technique. Our research in this paper accomplishes the task of estimating surface area by operating the input volumetric object without reconstructing any polygonal mesh. This simplifies the overall system design and avoids difficulties, ambiguities and distortion that may arise due to the application of a polyhedrization process. Our method in this paper can be considered as an alternative and complementary tool for surface area estimation of digitized 3D objects.

2. Preliminaries

2.1. Formula for surface area estimation

To measure the boundary surface area of a digitized 3D object, we make use of the Cauchy–Crofton formula from integral geometry, which relates the area of a surface to the number of intersections that the surface has with all lines in $\mathbb{R}^3$ [15,22,26]. Consider a surface in $\mathbb{R}^3$, whose area is denoted as s. For a line L intersecting the surface, let $\mathrm{d}L$ be the density of all lines intersecting the surface. The Cauchy–Crofton formula can be written as

$$\int w\,\mathrm{d}L = \pi s, \tag{1}$$

where w represents the number of intersection points of a given line with the surface, and the integration is taken over the space of all possible lines in $\mathbb{R}^3$. The reader may consult Ref. [22] for more detailed derivation of the formula.

Next we show how to use the above formula to estimate the surface area of a given 3D object O. Suppose that S is the boundary surface of O whose area s needs to be computed. Suppose further that S_1 is the boundary surface of a *reference* object O_1 which contains O. We assume that the area s_1 of O_1 is known. Consider a set $\mathcal{L}$ of N lines that are randomly sampled from the set of lines that intersect O_1, as illustrated in Fig. 1. Let n_S and n_{S_1} be the total numbers of intersection points of the lines in $\mathcal{L}$ with S and S_1, respectively. According to Eq. (1), by the integration approximation, we have [22]

$$\frac{n_S}{N} \approx c\pi s \quad \text{and} \quad \frac{n_{S_1}}{N} \approx c\pi s_1,$$

where c is a constant of proportionality. Then, by combining these equations, the surface area s of O can be estimated by

$$s \approx \frac{n_S}{n_{S_1}} s_1. \tag{2}$$

Eq. (2) is essentially a Monte Carlo method for numerical integration approximation, and this technique has been applied

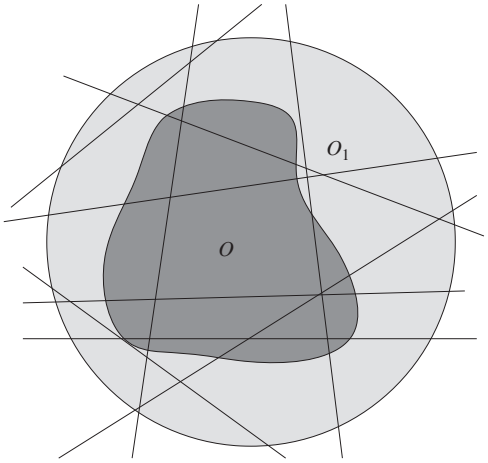

Fig. 1. Compute the boundary surface are of a body O contained within the reference body O_1 using the Cauchy–Crofton formula [26]. Random lines are chosen within O_1.

to area estimation for CSG models [22], point-sampled surfaces [26], and molecular surfaces [15]. The Cauchy–Crofton formula transforms a problem of surface area estimation into a problem of intersection counting.

2.2. Digitized 3D objects

We consider a digitized 3D object as a uniform 3D lattice consisting of object points O and background points $\overline{O}$. We represent the $3\times3\times3$ neighborhood of each lattice point $\mathbf{x}$ by $N_i(\mathbf{x})$ (i=6, 18, 26), which is a set of i points and each point (other than $\mathbf{x}$) that share a common grid edge, face, and cell with $\mathbf{x}$. The boundary of O is defined as [14]

$$\partial O = \{\mathbf{x} | \mathbf{x} \in O \quad \text{and} \quad N_i(\mathbf{x}) \cap \overline{O} \neq \emptyset\}, \tag{3}$$

where ∂O is a set of points sampled from some unknown underlying surface $S \in \mathbb{R}^3$. In this paper, we consider O as an 26-connected domain, i.e. $N_{26}(\mathbf{x})$. There have been numerous works on the volumetric representation, such as molecular shape comparison [8,25] and structure identification of cryo-EM models [14]. Our method presented in this paper is also suitable for other shape representations through converting them into the volumetric forms using some free softwares, such as *PolyMender* [13] and *binvox* [28], which produce a binary 3D voxel grid that approximates the original model. We generate a volumetric data by placing a 3D shape into a 3D cubic grid, compactly fitting the shape to the grid. Each lattice point is assigned either 1 or 0; 1 for object points O and 0 for background points $\overline{O}$.

3. Algorithm overview

We have just sketched the basis for computing the surface area of a given 3D body. Now we apply Eq. (2) to estimate the underlying surface area of a digitized object O. Suppose that the boundary ∂O of O describes some underlying surface S whose area s needs to be computed. Suppose that S_1 is the surface of a reference object O_1 whose area s_1 is known. To summarize, the new algorithm for estimating the surface area of O can be described as follows.

Algorithm 1 (*AreaEstimation*).

1. Generate the reference object O_1 containing O.
2. Generate a set $\mathcal{L}$ of N uniformly distributed lines that sample the set of all lines intersecting the reference object O_1.
3. Compute the number of intersection points of the lines in $\mathcal{L}$ with respect to the reference surface S_1 and the underlying surface S of the point set O. Let n_{S_1} and n_S denote those two numbers of intersection points, respectively.
4. Approximate the area s of S by Eq. (2).

3.1. The smallest enclosing ball

There are two requirements for generating a reference body O_1: (1) O_1 should been chosen as a simple object to simplify the intersection; (2) O_1 should enclose O as closely as possible such that the approximation error derived from the Monte Carlo method is small [33]. O_1 is generally chosen as a sphere to reduce the intersection of a line with O_1 [22], where the sphere is denoted by S_R. The simplest method for computing the enclosing sphere for the boundary ∂O of O is that the barycenter of ∂O is chosen as the center of S_R, and the maximum distance of ∂O with the barycenter is chosen as the radius of S_R; we call this method the *barycenter* method. However, the sphere generated by the barycenter method may be too loose for ∂O, leading to the larger approximation error of area estimation. Researching the smallest enclosing ball is a classical problem of computational geometry. In our implementation, we take advantage of Gärtner's *miniball* method [10] for computing the smallest enclosing ball of ∂O as S_R. Fig. 2 illustrates a 2D example for comparing Gärtner's miniball method with the barycenter method. Note that the inner circle generated by Gärtner's miniball method is smaller than the outer circle generated by the barycenter method. An alternative might be to consider a convex bounding box, such as an axis-parallel bounding box, as O_1, and the correlative sampling problem is discussed by Castro et al. [4].

3.2. Generating uniformly distributed lines using low-discrepancy sequences

The second step in our algorithm is generating a set $\mathcal{L}$ of N uniformly distributed lines that sample all lines intersecting the reference object O_1. Castro et al. [4] generated a uniform density of lines on a bounding sphere or a convex bounding box, where the convex bounding box might be to consider an axis-parallel bounding box of point sets. Li et al. [22] introduced one model for sampling on a sphere as follows: a random line is defined by a line passing through two independent uniformly distributed points on the surface of a sphere S_R with radius R. Uniformly distributed points (x, y, z) on a sphere can be generated from pairs (u, α) of uniformly distributed random numbers by using the following equation

$$(x, y, z) = (R\sqrt{1-u^2}\cos\alpha,\ R\sqrt{1-u^2}\sin\alpha, Ru), \tag{4}$$

where $u \in [-1,1]$ and $\alpha \in [0,2\pi)$. Further discussion of generating uniformly distributed points on spheres is given in Ref. [26].

It is known 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 [22,31,32]. In this paper, we use low-discrepancy sequences, instead of pseudo-random number generators, to generate the set $\mathcal{L}$ of evenly distributed lines. For this reason, our method is also called a *quasi-Monte Carlo method*. Let N be the number of sampling for approximating numerical integration. Using low-discrepancy sequences has theoretical error bound $O(N^{-1}\log^s N)$,

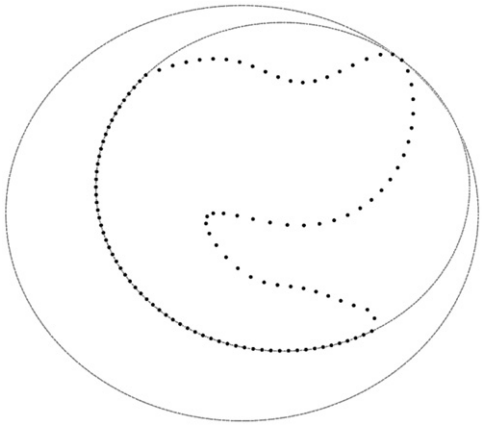

Fig. 2. The smallest enclosing ball of points in 2D. The inner circle is the smallest enclosing circle generated by Gärtner's method, whereas the outer circle is generated by the barycenter method.

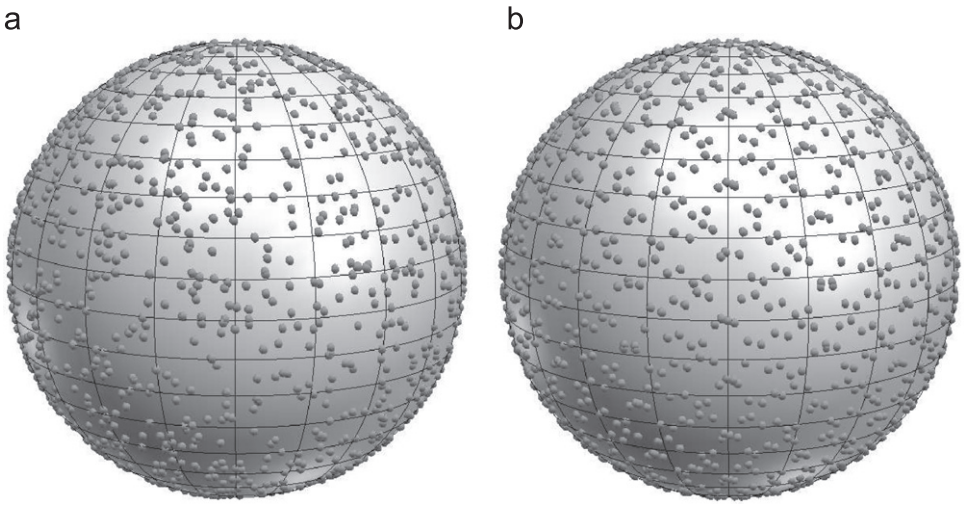

Fig. 3. The distribution of 1000 points on the surface of a sphere using Eq. (4): (a) shows pseudo-random points at slide view and (b) shows Niederreiter's low-discrepancy sequences of points at the same view. The sampling of the surface is clearly more even in the latter case.

which is much faster than the probabilistic error bound $O(N^{-1/2})$ of Monte Carlo methods using pseudo-random sequences [6,22,26,31,32], where discrepancy is defined in a s-dimensional space. There exists a lot of different low-discrepancy sequences: Halton, Sobol, Niederreiter, and others [32,33]. In our case, we use Niederreiter's 4D low-discrepancy sequences of points for Eq. (4). Fig. 3 shows the difference in distribution evenness of two sets of points on a sphere using a pseudo-random number generator and Niederreiter's low-discrepancy sequence in Eq. (4).

3.3. The intersection algorithm

A crucial part for surface area estimation is to count the number of intersection points between each line in $\mathcal{L}$ and the underlying surface S of a volumetric object O. S is available only in digital form ∂O. Since ∂O consists of a point array, we have to deal with the intersection problem between a line segment and a point set ∂O. To resolve this problem, we present a novel, fast, and robust intersection algorithm based on a clustering technique without requiring any surface reconstruction.

In some sense, our clustering idea can be considered as an extension of the ray tracing routine [34], which directly computed the intersection of a ray and a point cloud by intersecting a cylinder around the ray with the point cloud. Then, the intersection is computed as a weighted average of points that are inside the cylinder. However, [34] only finds one of the intersection points. We extend their method based on a clustering technique. In contrast to previous intersection approaches [1,34], our algorithm does not reconstruct or approximate a surface from a point cloud while finding all intersection points. The similar clustering idea has appeared in our previous work [26] to find the intersection points between a set of straight lines and a point cloud with associated the normal vectors. In Ref. [26], a major drawback is that the intersection strongly depends on the normal vector of points on the boundary surface. In contrast, our intersection algorithm presented in this section is normal-free. In Ref. [11], Huang and Klette represented a probe straight line as a sequence of voxels. A voxel visited along a digital line is an intersection iff it is a 26-border voxel of the object O. Although their method can test whether a digital line intersects with O, it can not count the number of all intersection points. Our intersection algorithm is described as follows.

Algorithm 2 (*Intersection*).

1. Detect whether an intersection has occurred between a cylinder around a line and a point set, and collect the points inside the cylinder.
2. Cluster the collected points by projecting them onto the line.
3. Classify the clusters.
4. Count the number of intersection points, which is equal to the number of the resultant clusters.

Fig. 4 gives an illustration of intersection between a line segment and a 2D digital object.

The first step of Algorithm 2 is similar to *intersection detection* described in Refs. [26,34]. We collect the candidate intersection points in the same way to [26] as follows. Let $\ell \in \mathcal{L}$ be a line with the parametric representation: $\ell(t) = \mathbf{o} + t\mathbf{n}$ for some $t \in \mathbb{R}$, where $\mathbf{o}$ and $\mathbf{n}$ is an origin and a unit direction of the line ℓ, respectively. We consider a cylinder around the line segment ℓ with the radius r (see Fig. 4(c)). To determine r, we need to obtain the density of the point set ∂O, where the density is the maximum size of a gap in ∂O. Suppose that d is the edge length of a voxel and it usually is set as a unit value (e.g. $d \equiv 1$) then the longest distance in the neighborhood around a voxel is $\sqrt{3}d$. Typically, we choose $r = \sqrt{3}d$ as the radius of the cylinder for obtaining sufficient intersection points and less time. We call these points of ∂O inside the cylinder the *inclusion points*. In our implementation we construct an axis-aligned octree of O to accelerate the speed of searching inclusion points, as described in Ref. [26]. In Fig. 5, the black points are inclusion points relative to a line segment ℓ surrounded by a cylinder of radius r.

After collecting the inclusion points, we then cluster them (see Fig. 4(d)) for counting the number of intersection points. Our cluster strategy maps 3D points into 1D parameter coordinates by projecting the inclusion points into the line segment ℓ. Suppose that $\{\mathbf{q}_i\} \subseteq \partial O$ is a set of inclusion points of ℓ and ∂O. Firstly, we project each point $\mathbf{q}_i$ onto ℓ, and get one corresponding parameter $t_i \in \mathbb{R}$. We also obtain a set $\{t_i\}$ of parameters. Secondly, the set $\{t_i\}$

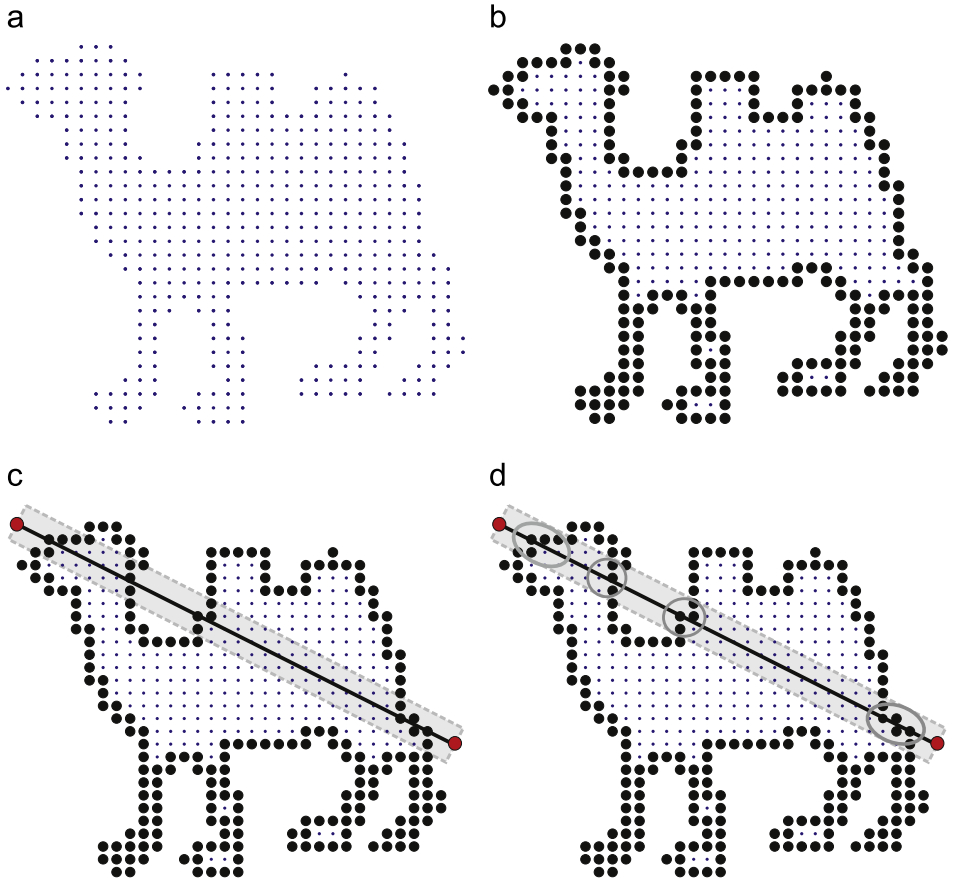

Fig. 4. The illustration of Algorithm 2 for a 2D digital object. (a) The input point array (blue points) is sampled from a camel image. (b) Extract the border (black points). (c) Collect the inclusion points that are inside a cylinder (gray) around a line segment connecting two endpoints (red). (d) Cluster the inclusion points, where clusters are highlighted in ellipses. Note that our cluster method maps 3D points into 1D parameter coordinates by projecting the inclusion points into the line segment. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

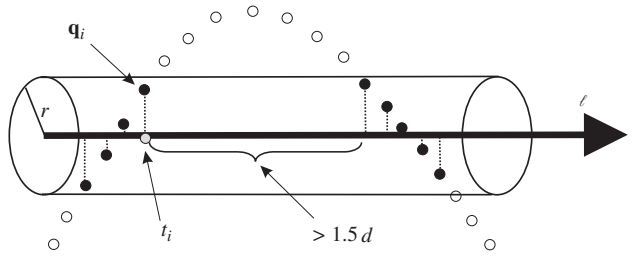

Fig. 5. Cluster the inclusion points by projection, where r is the radius of a cylinder around ℓ connecting two sample points and the black points denote the inclusion points inside the cylinder. We map each inclusion point $\mathbf{q}_i$ into 1D parameter coordinate $t_i \in \mathbb{R}$ by projecting $\mathbf{q}_i$ onto the line segment ℓ.

is sorted in increasing order. Here we suppose that $\{t_i\}$ has already been sorted. Finally, we build the clusters by $\{t_i\}$ as described below. Starting from the minimal parameter of $\{t_i\}$, a cluster Q_0, which is a set of some inclusion points in $\{\mathbf{q}_i\}$, is built by comparing the distance of adjacent parameters. This cluster is terminated when the distance of two adjacent parameters is larger than a maximum bound (we typically choose $1.5d$ as the bound). Then, starting from the terminated parameter, the next cluster Q_1 is built repetitively. Clustering is terminated until the maximal parameter is reached. After classifying clusters, we choose the number of resultant clusters as the number of intersection points. Fig. 5 shows the procedure of building clusters by projection. Alternatively, the representative intersection point of one cluster can be computed as the average of projecting the inclusion points onto the line segment [26]. Fig. 6 illustrates the procedure of surface area estimation using the quasi-Monte Carlo method (Fig. 7).

3.4. Computational complexity

In this section, we discuss the computation complexity of intersection in Algorithm 2. Assume that the input volumetric object O consists of n object points, where ∂O contains m boundary points ($m \ll n$). N uniformly distributed lines are sampled on the reference sphere S_R. In Algorithm 2, an upper bound of the running time to intersect a line with O is $O(m \log(m))$ because of sorting 1D parameter coordinates of inclusion points. As a result, the total complexity of intersection for N lines is $O(N\, m \log(m))$.

However, the real computational complexity is much lower than the theoretic one. Since the number of object points is volumetric level number, and the number of boundary points m is an area level number, while the average number m' of inclusion points in each cylinder can be considered as a line level number. Therefore, we can roughly approximate the relations among n, m, m' as $m = O(n^{2/3})$, $m' = O(n^{1/3})$. Hence our complexity is $O(Nm' \log m')$ for N lines, which is about $O(Nn^{1/3} \log n^{1/3})$. But in real case, the number of inclusion points m' is rather small and hence $m' \log m'$ can be roughly approximated as a smaller constant. As tested in our experiments, the running time for computing the intersection are very fast.

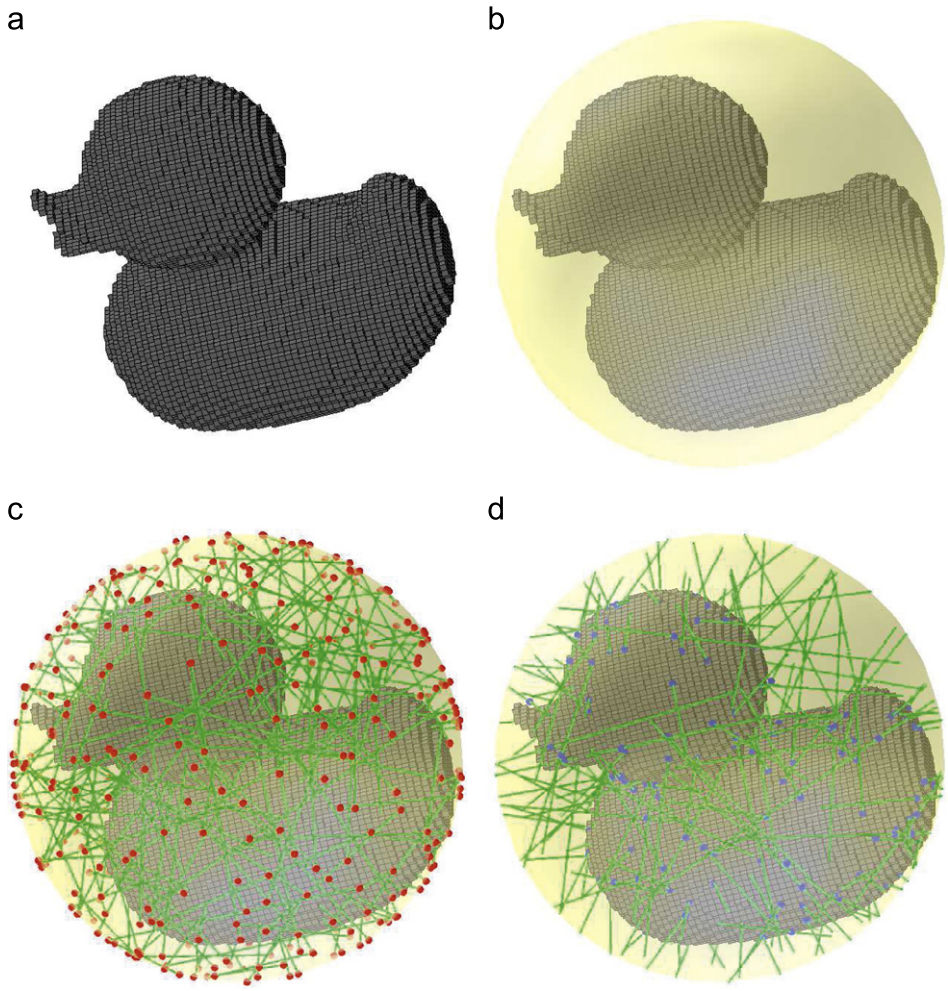

Fig. 6. The illustration of surface area estimation using the quasi-Monte Carlo method. (a) A digitized duck model with a given resolution ($64 \times 64 \times 64$). (b) The enclosing reference sphere. (c) A set of 200 uniformly distributed lines (green), and the corresponding endpoints (red) on the surface of the sphere. (d) The intersected lines and corresponding intersection points (blue) with the duck model. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

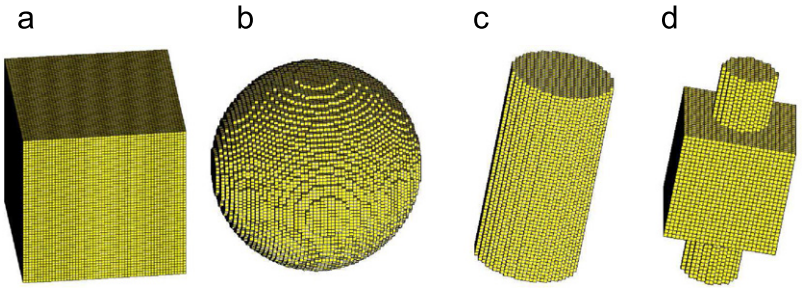

Fig. 7. Four synthetic objects sampled from some simple objects: (a) cube, (b) sphere, (c) cylinder, and (d) a CSG unit of cube and cylinder.

4. Results and discussions

We have implemented the quasi-Monte Carlo method for surface area estimation on different digitized 3D objects, which are digitized from some simple objects or obtained by some devices. The algorithms presented in this paper have been written in a C++ program. All the experiments were run on a PC with Pentium Dual-Core 2.60 GHz CPU and 2G RAM. In the preprocessing step, we construct an axis-aligned octree [26] with depth 5 and compute the smallest enclosing ball. This procedure can be performed in a short time. Table 1 gives the time in seconds for the preprocessing step of some volumetric models referred to in this paper.

4.1. Implementation and parameters

In our implementation, the enclosing reference body O_1 defined in Section 3.1 is a sphere with radius R, where R is chosen to be slightly larger than the radius of the smallest enclosing ball of the given object. Our method requires two parameters: the number N of uniformly distributed lines and the

Table 1
Time and error comparison for complex models with various resolutions.

Model	H	m	T_1[a] (s)	T_2[b] (s)	T_3[c] (s)	T_4[d] (s)	E_1[e] (%)	E_2[f] (%)
Duck	50	6497	0.016	0.016	1.797	1.531	1.137	10.218
	100	28,990	0.031	0.016	1.891	1.641	0.026	6.322
	150	67,152	0.047	0.031	2.063	1.781	0.335	4.371
	200	121,940	0.078	0.078	2.094	1.906	0.411	2.806
Bunny	50	6190	0.016	0.016	1.547	1.438	2.027	6.856
	100	26,695	0.031	0.016	1.672	1.563	1.095	4.323
	150	63,481	0.047	0.031	1.688	1.656	0.364	2.619
	200	112,338	0.047	0.063	1.703	1.781	0.579	1.576
Brain	50	6800	0.016	0.016	1.797	1.625	8.824	5.854
	100	30,350	0.047	0.016	2.000	1.766	3.859	2.155
	150	71,384	0.062	0.031	2.250	2.078	0.838	1.287
	200	128,322	0.094	0.063	2.500	2.188	0.255	1.073
Protein	50	6999	0.016	0.000	1.703	1.594	33.490	14.259
	100	31,432	0.031	0.016	1.797	1.766	8.783	3.160
	150	73,272	0.047	0.031	2.047	1.875	1.142	2.181
	200	132,642	0.062	0.094	2.250	2.016	0.786	1.633

Here, the errors are relative to the area of the original mesh using low-discrepancy sequences and pseudo-random numbers, respectively.

[a] "T_1" is the time of constructing an octree at depth 5.
[b] "T_2" is the time of computing the smallest enclosing ball.
[c] "T_3" is the time of estimating area using low-discrepancy sequences.
[d] "T_4" is the time of estimating area using pseudo-random numbers.
[e] "E_1" is the relative error using low-discrepancy sequences.
[f] "E_2" is the relative error using pseudo-random numbers.

radius r of the cylinder surrounding the intersection line. One may select a large N, which yields the better approximation result at the expense of computation time. We choose $r=\sqrt{3}d$ for obtaining sufficient intersection points and less time, where d is the edge length of a voxel unit. For the regular cubic grid, it usually is set as a unit value, i.e. $d=1$. A larger value for r yields the expense of computation time and increases the approximation error.

The relative error E_r [11] used for performance measure is defined by

$$E_r = \frac{|\tilde{s}-s|}{s}, \tag{5}$$

where $\tilde{s}$ is the estimated area value and s is the surface area of the original object (we typically call s the true surface area value). All errors measured and presented below are the relative errors. It may be of interest to other error measures, such as mean-square error [24,29].

4.2. Performance evaluation

4.2.1. Four simple models for the surface area measure

To evaluate the performance of surface area estimation, we test the presented method on four synthetic objects: cube, sphere, cylinder, and a CSG unit of cube and cylinder (referred to "Cube+Cylinder"), as shown in Fig. 7. The original object is first generated in the continuous space and then digitized using Gauss digitization in a 3D cubic grid. The cubic grid is considered as a 3D $H \times H \times H$ array, where the integer H is the grid *resolution* of the digitized object.

Fig. 8 shows the curves of relative errors generated by our method for the four synthetic objects at a constant resolution ($H = 100$), where the number of lines is specified from 100 to 100,000. The relative error curves in Fig. 8 show that the quasi-Monte Carlo method leads to small approximation errors with

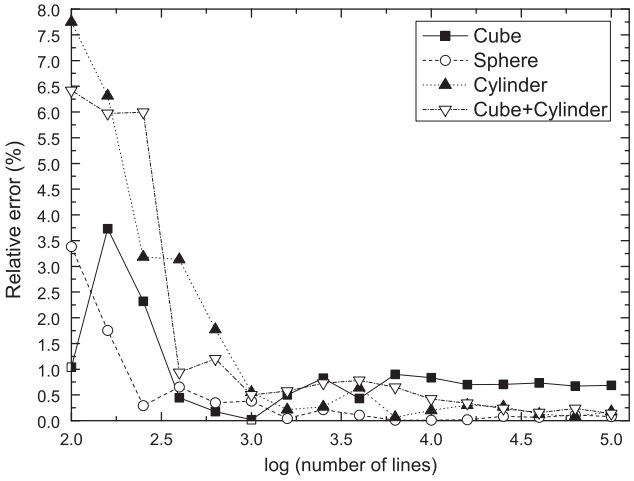

Fig. 8. The relative errors of surface area estimation for four synthetic objects at a given resolution ($H = 100$), where the horizontal axis denotes the number of lines used for the quasi-Monte Carlo method and the vertical axis denotes the relative errors.

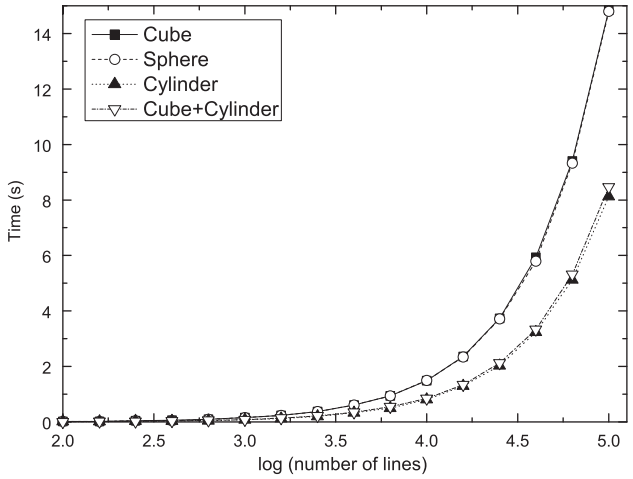

Fig. 9. Time comparison of estimating the surface area.

increasing the number of lines. When $N=5000$ lines are generated using Niederreiter's low-discrepancy sequences, it is expected to approximate the relative error $O(N^{-2/3})(\approx 0.34\%)$ for the quasi-Monte Carlo method [22]. In practice, we also find that 5000 lines for our method can lead to both small errors and little computation time. For the above four synthetic objects ($H = 100$), Fig. 9 gives the curves of computation time (in seconds), which are relative to the number of lines during estimating the area of four objects used. For the time curve of each model, the more lines are chosen, the larger computation time is consumed. The computation time is also related to the size of the input objects.

It is of interest to study behavior in the surface area estimation with respect to various resolutions. In Fig. 10, we typically test the four synthetic objects from $H=20$ to 128 with $N=5000$ lines using Niederreiter's low-discrepancy sequences. The results show that the estimated surface area converges to the true surface area as the resolution increasing.

In addition, we also test surface area estimation for the cylinder in different rotational orientations, as illustrated in Fig. 11. The rotations for the original objects slightly affect the estimated surface area with increasing the resolution.

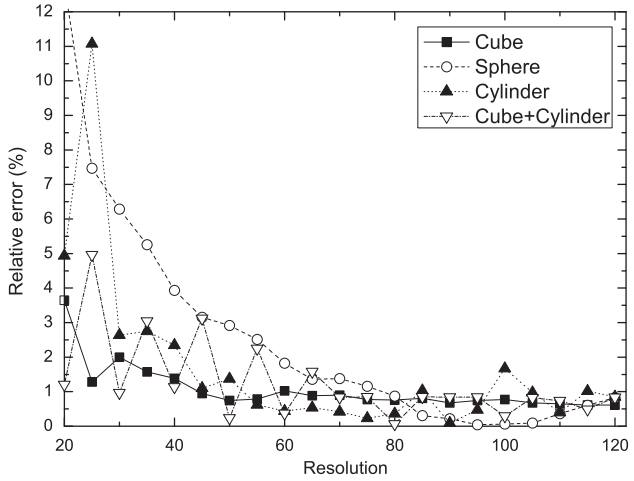

Fig. 10. The relative errors in estimating surface area of four objects with respect to the various resolutions.

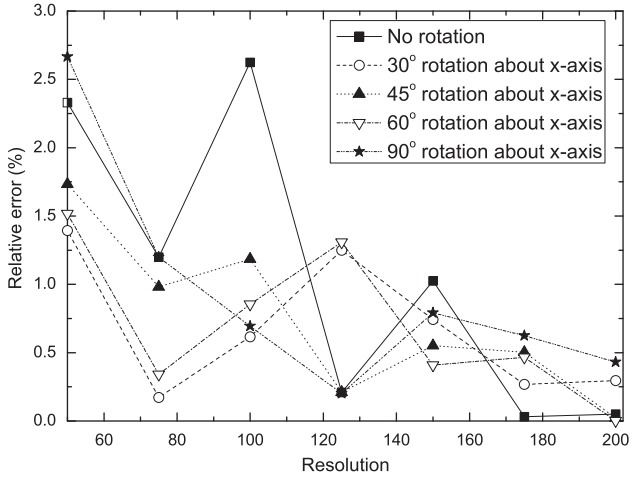

Fig. 11. The relative errors in estimating surface area of the cylinder in different rotational orientations as the resolution increasing.

4.2.2. Simulation results for complex models

Now, we demonstrate our method on several examples of digitized objets with great complexity, as shown in Figs. 12 and 6. Duck and bunny are acquired by 3D scanning devices, and the image volumes of protein and brain are from biology and medical research. Each model is originally represented in polygonal formats and digitized using Gauss digitization with various resolutions, where the surface area of the original mesh is referred to the true area. 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. Table 1 shows time and error comparisons for these complex models using the two methods with 5000 lines. In Table 1, "H" is the grid resolution, "m" is the number of boundary points, "T_1" and "T_2" and "T_3" and "T_4" give the respective timing data for the new method. The "E_1" and "E_2" items are defined by the error relative to the area of the original mesh using low-discrepancy sequences and pseudo-random number generator, respectively. The running time is less than 3.0 s for most cases. The results show that our method gives small approximation errors with increasing resolutions. The comparisons in Table 1 also suggest that using the low discrepancy sequences leads to smaller approximation errors than using pseudo-random numbers.

Note that the protein object in Fig. 12(c) occurs large errors ($\approx 33.490\%$) at resolution $H=50$. The reason is that the original molecular surface consists of many atoms which have complex structure. When the molecular surface is digitized as the set of voxels at low resolution, the digitized object is coarse and misses many local features. As the grid resolution increases, the digitized object approximates the molecular surface, resulting in the small approximation error ($\approx 0.786\%$ at resolution $H=200$).

4.3. Comparison with other works

Next, we compare our method with several other methods to estimate surface area for one same volumetric model. In Table 2, the tested sphere is digitized with 793,687 object points and 60,878 boundary points. The method "Voxel faces" estimates the surface area of a digitized 3D object as the sum of the total area of exposed voxel faces over the border of the object. This, however, results in rather big over-estimation ($\approx 19.784\%$). The classical approaches (the Convex Hull [2] and marching cubes algorithms) consider a polygonalization of the boundary point set and estimate the surface area by summing the area of the facets. Although the relative error using Convex Hull[1] is low, the convergence rates of its estimation are limit to convex objects. The marching cubes algorithm extracts an iso-surface and then sums up the area of the triangles in the iso-surface, but its relative error is worse than Convex Hull for the convex sphere. The surface area estimation is also tested in the "Local Estimation"[2] method [21], which approximates the surface area of a discretized set by assigning a weight value to each voxel and summing over the weighted discretized sets. Ref. [21] gives the low error ($\approx 1.078\%$) for the sphere model. The method presented by Lindblad [23][3] runs quite fast (0.015 s) and gives the low error ($\approx 1.648\%$). We also borrowed the results from Flin et al. [9], in which their method computes the normal vector field on the boundary voxels of digitized 3D objects and then estimates the surface area based on the computed normal vector field integration. Flin et al.'s method can give a low error ($\approx 0.726\%$). In contrast, the surface area estimation using our quasi-Monte Carlo (QMC) method exhibits lower error than the other methods.

In Table 3, we also extend the comparison with a more complex object, i.e. the digitized protein model with resolution $H=200$. The comparison shows that the "Voxel faces", Convex Hull and marching cubes methods produce rather big approximation errors for the complex object. "Local Estimation", Lindblad's method and our QMC method lead to small approximation errors. The comparisons demonstrate that our method tends to perform better results for objects with complex boundary surfaces than the other methods.

The idea in this work is similar to our previous work [26], in which we presented a quasi-Monte Carlo method for computing the area of a point-sampled surface with associated surface normal for each point. The method in [26] is based on the Cauchy–Crofton formula, and it transforms a problem of estimating the surface area into a problem of counting the intersection points with the point-sampled surface. Its core is the computation of the number of intersection points between a line and the point-sampled surface. In our current work, we borrow the similar idea, i.e. low-discrepancy sequences + Cauchy–Crofton formula, from [26], and adopt it to the surface area estimation of digitized 3D objects. Our current work can be considered as a simplified

[1] Convex Hull was obtained with "Qhull" at http://www.qhull.org

[2] The source code is available at http://w3.jouy.inra.fr/unites/miaj/public/perso/DavidLegland/Logiciels/localEstimation.html

[3] The source code is available at: http://www.cb.uu.se/~joakim/software/

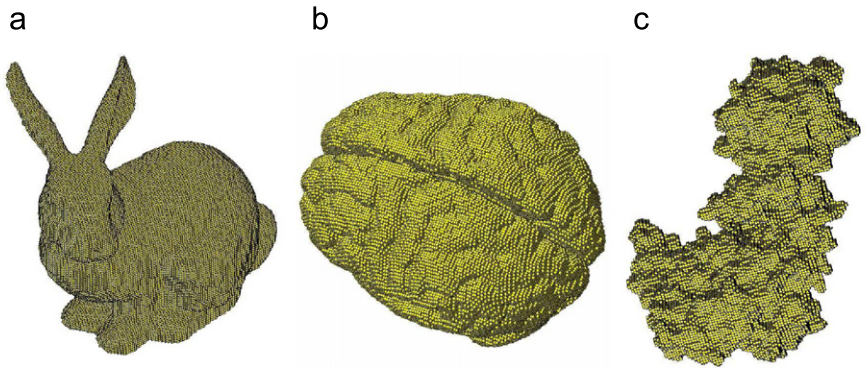

Fig. 12. Three complex volumetric models digitized by using Gauss digitization: (a) the bunny model; (b) the brain image volume; and (c) the protein image volume.

Table 2
Comparison our method with other works for the digitized sphere model with resolution $H=128$.

Methods	Time (s)	Error[a] (%)
Voxel faces	0.612	19.784
Convex Hull [2]	0.951	1.281
Marching cubes	1.963	8.801
Local estimation [21]	5.644	1.078
Lindblad [23]	0.015	1.648
Flin et al. [9]	–	0.726
QMC[b]	1.651	0.479

[a] "Error" is relative to the true area of the original sphere.
[b] Estimating the surface area using our quasi-Monte Carlo method.

Table 3
Comparison our method with other works for the digitized protein model with resolution $H=200$.

Methods	Time (s)	Error[a] (%)
Voxel faces	0.391	47.113
Convex Hull [2]	1.063	27.986
Marching cubes	10.172	8.799
Local estimation [21]	19.984	1.924
Lindblad [23]	0.078	3.201
QMC[b]	2.406	0.786

[a] "Error" is relative to the surface area of the original mesh.
[b] Estimating the surface area using our quasi-Monte Carlo method.

version of previous work [26] adapted to voxelized objects through taking advantage of the uniform binary voxel-structure. Thanks to the binary volumetric representation, which is considered as a uniform 3D lattice, we derive the normal-free intersection algorithm for the surface area estimation. To demonstrate the abilities of our method, we export the grid points that correspond to border voxels of the same digitized protein object in Table 3 as the point-sampled surface. Next we import the boundary grid points into our previous program in [26] and compute the normal vector of each point. Finally, we use default parameters for computing the area of point-sampled surface, which yields the larger approximation error ($\approx 2.929\%$) than the error using our current QMC method in this paper. The result shows that the current method gives a higher accuracy compared to the method for point-sampled models.

4.4. Discussions

Some discussions are presented below.

Parallel implementation: A variety of methods have been developed for surface area estimation of digitized 3D objects, but most of them are designed for traditional single-processor architectures. The current trend in computer hardware is towards increasingly parallel architectures. The technique presented in this paper estimates the surface area by counting the number of intersection points between the random lines and the target object. Because each sampling intersection line is completely independent from all other lines, our algorithm is a natural fit for a highly parallel architecture such as a GPU. A version of the parallel algorithm has been developed in Ref. [15] for estimating molecular surface area. In the future we plan to achieve a parallel implementation for estimating surface area of large volumetric data.

Non-regular grid: In this paper, our work is suited to the regular cubic grid, which is the most commonly used in digital geometry analysis. Our algorithm requires one user-specified radius r of the cylinder surrounding the intersection line for intersection detection. If the digitized object is the non-regular grid, which has a non-uniform 3D lattice, the current intersection technique may lead to large approximation errors. In order to eliminate the influence of non-regular, one possible strategy is to re-voxelize the non-regular grid for obtaining a uniform grid. However, the re-voxelization is non-trivial and may contain too many voxels such that it will spend more time on preprocessing and intersecting. Improving the effectiveness and robustness of line intersection is a core topic for future work.

Multi-surfaces: The Cauchy–Crofton formula can also be directly used for multiple objects by considering multi-surfaces as a collection of regular surface patches [22,26]. Some examples have been shown and discussed in [26]. In the same way, our work in this paper can be applied to multiple objects.

Fuzzy objects: The surface area estimation of digitized 3D objects with fuzzy borders seems to be attractive for applications, and it mainly deals with the evaluation of estimation at low resolutions. For improving precision, especially in the case of low resolution grids, the investigations of several measurements on fuzzy digital 2D and 3D objects were discussed [36]. As other future directions of research, we will explore estimating quantitative properties of fuzzy objects.

5. Conclusions

This research provided a fast and robust technique for estimating the surface area of a digital object. The input is a 3D binary digital image, i.e., a set of voxels. The volumetric representation is maintained and no triangulation is carried out. The suggested technique is a quasi-Monte Carlo method based on the Cauchy–Crofton formula from integral geometry. It is simple and does not need any complex data structure. The performance of surface area estimation is verified on a number of digitized

synthetic objects, and the experiments show that the quasi-Monte Carlo method is fast, robust and obtains the high accuracy. Our method can be considered an alternative approach for surface area estimation of digitized 3D objects as a complement of other existing techniques.

Acknowledgments

The authors appreciate the comments and suggestions of all anonymous reviewers, whose comments significantly improved this paper. The authors would like to thank Dr. Joakim Lindblad and Dr. Frédéric Flin for communicating and Dr. Tao Ju for the source code for processing the volumetric models. Chao Wang implemented the Gauss digitization of polygonal models in a 3D cubic grid. The research was supported by Chinese 973 Program (2010CB328001), the National Science Foundation of China (U0970155, 60625202, 90715043), and Chinese 863 Program (2007AA040401). Prof. Paul was supported by ANR-NSFC (60911130368).

References

[1] A. Adamson, M. Alexa, Ray tracing point set surfaces, in: Proceedings of Shape Modeling International, 2003, pp. 272–279.
[2] C.B. Barber, D.P. Dobkin, H.T. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software 22 (4) (1996) 469–483.
[3] E. Breton, P. Choquet, L. Bergua, M. Barthelmebs, B. Haraldsson, J.J. Helwig, A. Constantinesco, M. Fischbach, In vivo peritoneal surface area measurement in rats by micro-computed tomography (μCT), Peritoneal Dialysis International 28 (2008) 188–194.
[4] F. Castro, M. Sbert, Application of quasi-Monte Carlo sampling to the multi-path method for radiosity, in: Proceedings of the Third International Conference on Monte Carlo and Quasi Monte Carlo Methods in Scientific Computing, Lecture Notes in Computational Science and Engineering, Springer Verlag, Berlin, Germany, 1998.
[5] D. Coeurjolly, F. Flin, O. Teytaud, L. Tougne, Multigrid convergence and surface area estimation, 11th International Workshop on Theoretical Foundations of Computer Vision Geometry, Morphology, and Computational Imaging, Lecture Notes in Computer Science, vol. 2616, , 2003, pp. 101–119.
[6] T. Davies, R.R. Martin, Low-discrepancy sequences for volume properties in solid modelling, in: Proceedings of CSG'98, , 1998, pp. 139–154.
[7] L. Dorst, A. Smeulders, Length estimators for digitized contours, Computer Vision, Graphics, and Image Processing 40 (3) (1987) 311–333.
[8] Y. Fang, Y.-S. Liu, K. Ramani, Three dimensional shape comparison of flexible protein using the local-diameter descriptor, BMC Structural Biology 9 (29) (2009).
[9] F. Flin, J. Brzoska, D. Coeurjolly, et al., Adaptive estimation of normals and surface area for discrete 3-D objects: application to snow binary data from X-ray tomography, IEEE Transactions on Image Processing 14 (5) (2005) 585–596.
[10] B. Gärtner, Fast and robust smallest enclosing balls, Proceedings of the Seventh Annual European Symposium on Algorithms (ESA), Lecture Notes in Computer Science, vol. 1643, , 1999, pp. 325–338.
[11] Y. Huang, R. Klette, A comparison of property estimators in stereology and digital geometry, 10th International Workshop of Combinatorial Image Analysis (IWCIA 2004), Lecture Notes in Computer Science, vol. 3322, 2004, pp. 421–431.
[12] J. Jenness, Calculating landscape surface area from digital elevation models, Wildlife Society Bulletin 32 (3) (2004) 829–839.
[13] T. Ju, Robust repair of polygonal models, ACM Transactions on Graphics 23 (3) (2004) 888–895.
[14] T. Ju, M. Baker, W. Chiu, Computing a family of skeletons of volumetric models for shape description, Computer-Aided Design 39 (5) (2007) 352–360.
[15] D. Juba, A. Varshney, Parallel, stochastic measurement of molecular surface area, Journal of Molecular Graphics and Modelling 27 (2008) 82–87.
[16] Y. Kenmochi, R. Klette, Surface area calculation for digitized regular solids, Vision Geometry IX, SPIE-4117, 2000, pp. 100–111.
[17] R. Klette, A. Rosenfeld, Digital Geometry: Geometric Methods for Digital Picture Analysis, Morgan Kaufmann, San Francisco, 2004.
[18] R. Klette, H. Sun, A global surface area estimation algorithm for digital regular solids, Technical Report CITR-TR-69, Computer Science Department, University of Auckland, New Zealand, 2000.
[19] R. Klette, H. Sun, Digital planar segment based polyhedrization for surface area estimation, International Workshop on Visual Form 2001, Lecture Notes in Computer Science, vol. 2059, , 2001, pp. 356–366.
[20] J. Koplowitz, A. Bruckstein, Design of perimeter estimators for digitized planar shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (6) (1989) 611–622.
[21] D. Legland, K. Kien, M. Devaux, Computation of Minkowski measures on 2D and 3D binary images, Image Analysis and Stereology 26 (2) (2007) 83–92.
[22] X. Li, W. Wang, R.R. Martin, A. Bowyer, Using low-discrepancy sequences and the Crofton formula to compute surface areas of geometric models, Computer-Aided Design 35 (9) (2003) 771–782.
[23] J. Lindblad, Surface area estimation of digitized planes using weighted local configurations, Proceedings of the 11th International Conference on Discrete Geometry for Computer Imagery (DGCI), Lecture Notes in Computer Science, vol. 2886, , 2003, pp. 348–357.
[24] J. Lindblad, Surface area estimation of digitized 3D objects using weighted local configurations, Image and Vision Computing 23 (2005) 111–122.
[25] Y.-S. Liu, Y. Fang, K. Ramani, IDSS: deformation invariant signatures for molecular shape comparison, BMC Bioinformatics 10 (157) (2009).
[26] Y.-S. Liu, J.-H. Yong, H. Zhang, D.-M. Yan, J.-G. Sun, A quasi-Monte Carlo method for computing areas of point-sampled surfaces, Computer-Aided Design 38 (1) (2006) 55–68.
[27] W.E. Lorensen, H.E. Cline, Marching cubes: a high resolution 3D surface construction algorithm, Computer Graphics (SIGGRAPH'87) 21 (4) (1987) 163–169.
[28] P. Min, binvox: 3D mesh voxelizer 〈http://www.cs.princeton.edu/~min/binvox/〉.
[29] J.C. Mullikin, P.W. Verbeek, Surface area estimation of digitized planes, Bioimaging 1 (1) (1993) 6–16.
[30] B.K. Natarajan, On generating topologically consistent isosurfaces from uniform samples, The Visual Computer 11 (1) (1994) 52–62.
[31] H. Niederreiter, Quasi-Monte Carlo methods and pseudo-random numbers, Bulletin of the American Mathematical Society 84 (6) (1978) 957–1041.
[32] H. Niederreiter, Random Number Generation and Quasi-Monte Carlo Methods, SIAM Philadelphia, Philadelphia, 1992.
[33] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, second ed., Cambridge University Press, 1992.
[34] G. Schaufler, H.W. Jensen, Ray tracing point sampled geometry, in: Proceedings of the 11th Eurographics Workshop on Rendering, , 2000, pp. 319–328.
[35] K. Schladitz, J. Ohser, W. Nagel, Measuring intrinsic volumes in digital 3d images, Proceedings of the 13th International Conference on Discrete Geometry for Computer Imagery (DGCI), Lecture Notes in Computer Science, vol. 4245, , 2006, pp. 247–258.
[36] N. Sladoje, I. Nyström, P.K. Saha, Measurements of digitized objects with fuzzy borders in 2D and 3D, Image and Vision Computing 23 (2005) 123–132.
[37] P. Verbeek, L.J. van Vliet, An estimator of edge length and surface area in digitized 2D and 3D images, in: Proceedings of 11th IAPR International Conference on Pattern Recognition (ICPR), , 1992, pp. 749–753.
[38] G. Windreich, N. Kiryati, G. Lohmann, Voxel-based surface area estimation: from theory to practice, Pattern Recognition 36 (11) (2003) 2531–2541.
[39] G. Wyvill, C. McPheeters, B. Wyvill, Data structures for soft objects, The Visual Computer 2 (4) (1986) 227–234.
[40] X. Zeng, L.H. Staib, R.T. Schultz, J.S. Duncan, Segmentation and measurement of the cortex from 3D MR images using coupled-surfaces propagation, IEEE Transactions on Medical Imaging 18 (10) (1999) 927–937.
[41] J. Ziegel, M. Kiderlen, Estimation of surface area and surface area measure of three-dimensional sets from digitizations, Image and Vision Computing 28 (1) (2010) 64–77.

Yu-Shen Liu is an Associate Professor in School of Software at Tsinghua University, PR China. He received his BS in Mathematics from Jilin University, China, in 2000. He earned his PhD in the Department of Computer Science and Technology at Tsinghua University, China, in 2006. He spent three years as a Post Doctoral Researcher in Purdue University from 2006 to 2009. His current research interests include in 3D shape description and comparison, volumetric object processing, computer aided design & computer graphics. Dr. Liu is the author of nine international journal papers. He received the Best Student Paper Award of the international conference of CAD/Graphics 2005 hosted at Hong Kong University of Science & Technology. In 2006, he received the Excellent Doctoral Dissertation Award from Tsinghua University.