GPU-based Offset Surface Computation using Point Samples

Charlie C. L. Wang
Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong

Dinesh Manocha
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

We present an efficient algorithm to perform approximate offsetting operations on geometric models using GPUs. Our approach approximates the boundary of an object with point samples and computes the offset by merging the balls centered at these points. The underlying approach uses *Layered Depth Images* (LDI) to organize the samples into structured points and performs parallel computations using multiple cores. We use spatial hashing to accelerate intersection queries and balance the workload among various cores. Furthermore, the problem of offsetting with a large distance is decomposed into successive offsetting using smaller distances. We derive bounds on the accuracy of offset computation as a function of the sampling rate of LDI and offset distance. In practice, our GPU-based algorithm can accurately compute offsets of models represented using hundreds of thousands of points in a few seconds on GeForce GTX 580 GPU. We observe more than 100 times speedup over prior serial CPU-based approximate offset computation algorithms.

Key words: Offsetting, Structured points, Approximation, Highly parallel, Layered Depth Images

1. Introduction

Offsetting is a fundamental and important geometric operation in a variety of applications (ref. [1–4]), such as model smoothing and simplification, tolerance and clearance analysis for assembly, rapid prototyping and coordinate measuring machines (CMM), tool path generation for 3D numerically controlled (NC) machining, and robot path planning. An *offset surface* of a solid H is the set of points having the same offset distance r from the boundary ∂H of H. Offsetting a solid H can be performed on one side of its boundary surface ∂H. The *exterior offsetting* H_r^+ corresponds to a set of points outside H, while the result of *interior offsetting* H_r^- is a set of points inside H. Although the offsetting operation is mathematically well defined, offsetting a solid model exactly has proven to be difficult [5]. In recent years, approximation offsetting methods based on volumetric representations [6–8] and point-based algorithms [9, 10] have been proposed. These algorithms first generate volumetric grids (or sampling points) to approximate the offset model and then use a distance field (or collision checking) to calculate the implicit surface (or sample points). The distance field computation and collision detection can be computationally expensive. Moreover, it is hard to parallelize the existing mesh offsetting approaches to exploit the computational power of commodity many-core GPUs (*Graphics Processing Units*). In this paper, we develop an efficient algorithm to compute approximate offsetting of point-sampled geometric models using GPUs.

Problem Definition: Given a solid model H with its boundary surface ∂H approximated by the set P_H of sample points, we compute the boundary surface of exterior offset H_r^+ (or interior offset H_r^-) and represent that using a point set $P_{H_r^+}$ (or $P_{H_r^-}$ for interior offset).

In order to use the power of highly parallel computation in graphics hardware, we use Layered Depth Images (LDI) [11] to sample the boundary into structured points and compute their offsets [12–14]). The points of a solid represented by LDI are sampled and stored on the set of parallel rays passing through the centers of LDI pixels along the viewing direction. To compute the offset of a solid H, the offsetting balls (with radius r) centered at the sample points of P_H are merged together by a 'super' union algorithm, which is computed in parallel on the rays of LDI. The resultant point set $P_{H_r^+}$ (or $P_{H_r^-}$) in LDI representation can be converted back into triangular meshes by dual-contouring method running on GPUs [14].

The main contributions of our work include:

– A highly parallel algorithm based on structured point representation that is used to accelerate intersection computations between LDI rays and spheres centered at the sample points of LDI.
– An efficient load balancing algorithm that can distribute the merging operations on various GPU cores.

Email addresses: cwang@mae.cuhk.edu.hk (Charlie C. L. Wang), dm@cs.unc.edu (Dinesh Manocha).

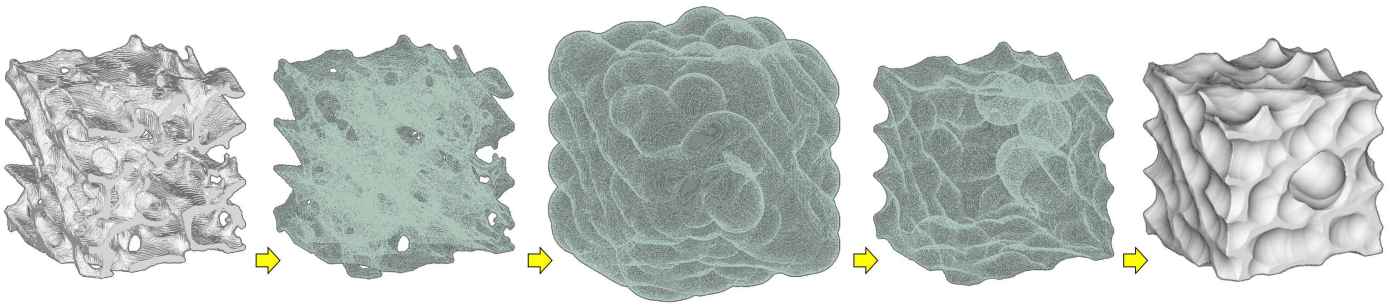


Fig. 1. A filleting operation (ref. [4]) on a scaffold model can be completed in a few seconds by our approach on GeForce GTX580: (from left to right) polygonal mesh of the scaffold model H, the sampled LDI model P_H with 480k points (taking 80ms), the exterior offsetting result $P_{H_r^+}$ (obtained in 3.2 sec. with r being 1/8 diagonal length of H's bounding box), the interior offsetting with the same distance r (completed in 2.4 sec.), and the resultant polygonal mesh of filleting (the mesh generation time is 103ms including the steps of normal evaluation and dual contouring).

– The shape error of approximate offsetting on structured points is analyzed, and we show that the bound on shape approximation error converges as a monotonic function of the sampling rate.

In contrast to previous approximate offsetting algorithms, our approach maps well to GPU architectures. As a result, we are able to compute approximate offsets of complex models in a few seconds (see Fig.1). With similar accuracy of approximation, more than 100 times speedup is observed over prior CPU-based approximate offset computation algorithms [7,15] on the same benchmark (as shown in Fig.8). Our GPU-based algorithm has lower memory overhead as compared to prior GPU-based techniques, which makes it possible to compute more accurate offset approximations using our algorithm.

The rest of the paper is organized as follows: we review related work in Section 2 and give an overview of our approach in Section 3. We present the many-core offset computation algorithm in Section 4 and describe its performance in Section 5.

2. Related Work

The problem of offset computations has been studied in CAD/CAM, robotics and related areas for more than three decades. Earlier approaches like [3, 16] first compute a superset of boundary features of offset surfaces by offsetting faces, edges and vertices into parallel faces, cylinders and spheres, respectively. Next, these methods trim this superset by subdividing its elements at their common intersections and clipping away pieces that do not belong to the boundary of the offset. These approaches are difficult to implement robustly because of the computational complexity of trimming operations, as well as numerical issues with intersection computations [5]. In order to avoid the computational issues with clipping and surface trimming, some approximate algorithms have been investigated. Qu and Stucker [17] offset a model by moving triangle vertices while maintaining the topology of original model. However, their method is limited to offset distances that are sufficiently small and assumes that the offset has the same topology as the original model.

Many previous approaches are based on distance field computation. Breen and Mouch [18] presented an offset method for *Constructive Solid Geometry* (CSG) models using distance volumes and the fast marching method. The resulting approach calculates the shortest distance to the CSG model from a set of points within a narrow band around the surfaces. It then propagates the information about the shortest distance and the closest points out to the remaining voxels in the volume. Frisken et al. [19] proposed adaptively sampled distance-fields (ADF) as a unifying representation of shape for a broad range of processing operations, including surface offsetting. The structure of this representation is simple and direct, and is effective for quality reconstruction of complex shapes. Varadhan and Manocha [6] presented a distance field-based algorithm to approximate the 3D Minkowski sum of a polyhedral model. The union of pairwise convex Minkowski sums is computed by generating a voxel grid, computing signed distances on the grid points, and performing isosurface extraction from the distance field. Adaptive sampling strategies [7, 8] have also been used to compute the offset of polygonal models. The major problem with these approaches is that they can have a high memory overhead in order to guarantee tight bounds on accuracy, especially for uniform sampling (e.g., [6, 18]). On the other hand, it is hard to parallelize adaptive methods (e.g., [19]) on current GPU architectures.

There have been a few offsetting algorithms based on the point (or ray) representation. Lien [9] proposed a point-based Minkowski sum computation algorithm. This algorithm uses a normal filter and collision detection algorithms to remove points, which do not belong on their boundaries, from the resultant models [20]. A ray-based representation (ray-rep) was employed by Hartquist et al. in [21] to perform offset, sweep, and Minkowski operations. A ray-rep of a solid model is a set of line segments that lie inside the solid and are generated by clipping a regular grid of lines against the model. The method of Chen and Wang [10] uses LDI representations to compute the uniform offsetting of polygonal mesh models. The self-intersections are removed

on the rays. However, it is difficult to directly map these approaches on GPUs.

There is considerable work on exploiting the computational power of GPUs for similar operations. The work of Yin et al. [22] computes adaptive distance-fields on GPUs at interactive rates for resolutions up to 512^3. However, their approach only computes accurate distance values near the surface of input models. Their refinement algorithm performs voxel/triangle overlap tests; it is hard to perform this test accurately for primitives that are away from the input surface. Li and McMains [23] have presented an efficient GPU-based Minkowski sum computation algorithm which can be used for offset computation; however, they use flooding algorithms to determine the exterior boundary of the resultant models, which misses inner voids. Our ball-union based offsetting algorithm does not have this limitation. Cao et al. [24] can compute the exact Euclidean distance transform for a binary image in 2D and higher dimensions at interactive rates on current GPUs (for resolution up to 512), but the memory overhead of these distance-field computation approaches increases as a cubic function of the resolution.

3. Offsetting by Super-Union of Balls

In this section, we present our offsetting algorithm. Both the input and the output of offsetting are represented by LDIs, which can be efficiently obtained from a B-rep and converted back to a B-rep on the GPUs (see [12, 14] for details of the conversion method). The idea of our offsetting algorithm is based on computing the *union* operation of many spheres in a highly parallel manner on the GPU with the help of LDI representation – called *super-union* below.

3.1. *Structured points in LDI*

The input/output solid model of our approach is represented by the sample points of LDI coupled with surface normal vectors (ref. [12]). Given a sampling parameter m, the sample points of LDI are obtained by intersecting the boundary surface of a solid model with $m \times m$ rays along the x-, y- and z-directions such that these rays intersect at $m \times m \times m$ nodes of uniform grids in $\Re^3$. Since the sample points of LDI are exactly located on rays, they are considered as well organized, and are called *structured points*. To store the LDI-based solids in the limited memory of graphics processors, the structured points of LDI sampled in one direction are stored in two arrays: an index array I and a float data array D. The index array I corresponds to a $m \times m$ matrix, where the (i, j)-th entry indicates the index of the first sample on the (i, j)-th ray in the data array D. The data array D saves depth values and normal vectors of sample points in an ascending order keyed by the index of rays and the depth values. Specifically, the first sample on a ray, (i, j), is $D[I[i, j]]$, and the number of samples on (i, j) can be obtained by the difference between $I[i, j]$ and the index of the first sample on its next ray, $I[i, j + 1]$. Details of this compact representation of a LDI solid can be found in [14].

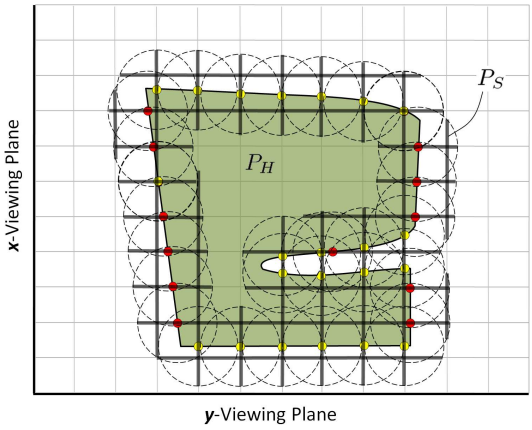


Fig. 2. The shell solid (computed approximately) according to the offsetting can be obtained by the super-union of spheres centered at the sample points of LDI representation, where the samples in red are obtained by the rays in x-direction and the yellow ones are on the rays in y-direction (i.e., the set of sample points P_H). The black line segments illustrate the 1D solids of P_S.

Boolean operations on two LDI solids, H_A and H_B, which have coincident rays can be easily computed. Briefly, the Boolean operations on 3D models are converted into computations on rays (i.e., 1D solids). A ray is subdivided into small segments by samples from both H_A and H_B. Whether each segment should be retained on the resultant 1D solid generated by a Boolean operation is based on the logic operations of 'inside' status of this segment on H_A and H_B: 'union' is defined by the logic 'OR', 'intersection' is defined by the logic 'AND', and 'subtraction' can be converted into 'intersecting a complement. Details can be found in [12].

3.2. *Union of balls on LDI*

The basic idea of our approach is to first compute a shell solid O_H (defined between the boundary surfaces of exterior offsetting solid H_r^+ and interior offsetting solid H_r^-) by merging the spheres with radius r centered at all the surface points of H. To simplify the union of an infinite number of balls, we only consider the set of H's surface sampling points, P_H. A sphere $S_{\mathbf{p}}$ centered at a sample $\mathbf{p} \in P_H$ located on the (i, j)-th ray in the τ-direction is converted into several 1D solids on the rays in τ-, $(\tau + 1)$- and $(\tau + 2)$-directions, respectively (see Fig.2 for an illustration). Here, the indices of directions are cycled along x, y and z according to the definition of LDI solids (ref. [12]). The intersections between $S_{\mathbf{p}}$ and the rays in τ-direction can be easily checked by searching the rays in the range of $(i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$, where w is the sampling distance of LDI (i.e, the distance between the (i, j)-th ray and the $(i, j+1)$-th ray). The value of w can be determined by the width of a model's bounding box divided by the sampling parameter m. For a ray (k, l) in this range, the intersection between $S_{\mathbf{p}}$

and the ray is converted into a line segment (i.e., 1D solid) on the ray if $(k-i)^2+(l-j)^2 < (r/w)^2$. A more critical issue is how to efficiently find the rays in the other two directions intersecting $S_{\mathbf{p}}$. The searching method presented in [25] checks all samples lying on the rays in the other two directions whose distance to the (i,j)-th ray in the τ-direction is not greater than r. As a result, $2m \times \lceil \frac{2r}{w} \rceil$ rays must be checked in total. This method contains too much redundancy, which is solved by our method proposed in Section 4.2.

Intersecting the spheres by the rays of LDI actually converts the spheres into 1D solids. The 1D solids on a ray (i,j) obtained by ray-sphere intersections are merged into the final 1D solid by the *union* operation. This turns out to be an approximation of O_H in LDI representation, P_S (as illustrated in Fig.2). The final exterior and interior offsetting results can be obtained by taking '$P_H \cup P_S$' or '$P_H \setminus P_S$', and these set operations corresponding to union or difference can be efficiently computed on GPUs. The approximation error of the offsetting result in the LDI representation is analyzed below.

3.3. *Approximation error in offset computation*

We analyze the approximation error based on the formulation of ϵ-covering of point-sampled geometry (ref. [9]).

Definition 1 (*d-regular*) A solid $H \subset \Re^3$ is called d-regular if, for each point $\mathbf{p} \in \partial H$, there exist two osculating open balls of radius d at $\mathbf{p}$ such that one lies entirely inside H and the other lies entirely out of H (ref. [26]).

From [26], it is known that the topology-bounded B-rep of a d-regular solid sampled by the rays of LDI representation with $w < d/\sqrt{3}$ can be reconstructed by the topology-preserved method. In CAD and manufacturing applications, the value of d relates to the smallest feature size of models that can be fabricated by machines. Without loss of generality, all the analysis below is based on the assumption that the input model H is d-regular. Generally, all polygonal models are not exactly d-regular; however, they are widely used in CAD systems to approximate d-regular solids which are smooth. For example, Guthe et al. [27] presented an efficient tessellation algorithm with error bounds for smooth surfaces.

Definition 2 (ϵ-*covering*) A set of points P is an ϵ-covering of a surface M if, for any point $\mathbf{q} \in M$, there exists a point in P whose distance to $\mathbf{q}$ is less than ϵ.

Remark 1 (ϵ-*covering* of LDI) For a d-regular solid H sampled from a B-rep into LDI representation, P_H, with w as the sampling distance ($w < d/\sqrt{3}$), P_H is a ϵ-covering point set of the boundary surface, ∂H, of H where $\epsilon = \sqrt{3}w$ (ref. [13]).

Moreover, as the sample points of the LDI solid are all located on the edges of cubes formed by the rays that intersect the boundary surface ∂H, we can compute the density of LDI in the following form.

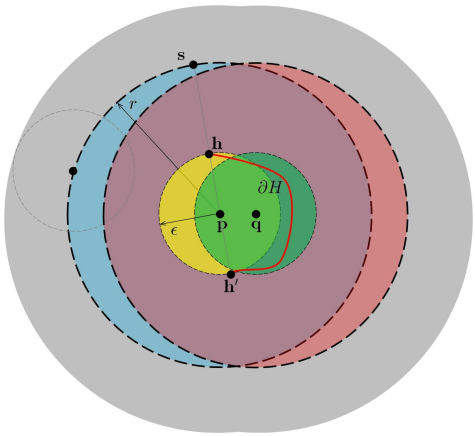


Fig. 3. Error in offset computation: The red curve corresponds to the exact boundary ∂H of the primitive H, the red and blue circles correspond to spheres with radius r centered at $\mathbf{q}$ and $\mathbf{p}$, respectively. Moreover, the grey region corresponds to the error or deviation between the exact and approximate offset computation. Similarly, the green and yellow circles correspond to spheres with radius ϵ centered at $\mathbf{q}$ and $\mathbf{p}$, respectively.

Remark 2 (*density* of LDI) For a d-regular solid H sampled from B-rep into LDI representation, P_H, with w as the sampling distance, $\forall \mathbf{p} \in P_H$, there exists another point $\mathbf{q} \in P_H$ that $\|\mathbf{p}-\mathbf{q}\| \leq \sqrt{3}w = \epsilon$.

This is because when any ray of a cube intersects with the boundary surface of a solid model, it must have another intersection point on the edges of this cube. By the conservative sampling method presented in [28], when the model tangentially contacts with a ray, the contact point will be counted as two samples on the ray. The maximal distance between two points on the edges of a cube with width w is $\sqrt{3}w$.

The boundary surface of the offset solid O_H can also be defined as a point set, $\partial O_H = \{\mathbf{p} | dist(\mathbf{p}, \partial H) = r\}$, by the distance function

$$dist(\mathbf{p}, S) = \inf_{\forall \mathbf{q} \in S} \|\mathbf{p}-\mathbf{q}\|. \quad (1)$$

This distance function is also used to derive the distance-bound of ball-union.

Proposition 1 (*distance-bound* of ball-union) If $r > \epsilon$, the union of spheres with radius r centered at the points in P_H results in a solid B_H such that its boundary surface ∂B_H gives the error bound: $\forall \mathbf{s} \in \partial B_H$, $|dist(\mathbf{s}, \partial H) - r| \leq \epsilon$.

Proof. Assuming that we can compute the union of spheres, all remaining points on the sphere $S_{\mathbf{p}}$ centered at the point $\mathbf{p}$ must be outside or on other spheres $S_{\mathbf{q}}$ ($\forall \mathbf{q} \in P_H,\ \mathbf{q} \neq \mathbf{p}$). As the point set P_H is an ϵ-covering of the surface ∂H, the continuous surface ∂H must lie in the region formed by merging all spheres with radius ϵ centered

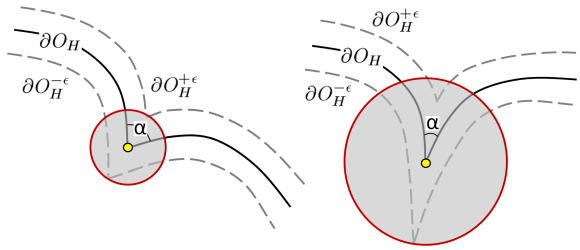


Fig. 4. The error bound (illustrated by the gray circles) between the surfaces of O_H and O_H^ϵ varies when the sharpness at the discontinuous regions on O_H changes. The sharpness can be measured by the angle α as illustrated – a smaller angle implies a higher measure of sharpness. When we decrease the value of α from the case on the left to the case on the right, the Hausdorff distance between O_H and O_H^ϵ increases, as indicated by the radii of gray circles.

at the points in P_H (e.g., the region formed by yellow and green circles in Fig.3) – called ϵ-spheres below.

For a point $\mathbf{s}$ on the sphere $S_\mathbf{p}$, when $\mathbf{p} \in \partial H$,

$$r - \epsilon \leq dist(\mathbf{s}, \partial H) \leq r.$$

Since $\|\mathbf{s} - \mathbf{p}\| = r$, the maximal value of $dist(\mathbf{s}, \partial H)$ is bounded by r. If the intersection point $\mathbf{h}$ between the line $\mathbf{sp}$ and the ϵ-sphere centered at $\mathbf{p}$ lies on ∂H, $\mathbf{h}$ is the closest point to $\mathbf{s}$ which gives $dist(\mathbf{s}, \partial H) = r - \epsilon$ (see Fig.3).

When $\mathbf{p}$ is not on ∂H but the $dist(\mathbf{p}, \partial H) \leq \epsilon$ (i.e., from an ϵ-covering point set of ∂H), the distance from $\mathbf{s}$ to ∂H satisfies this bound:

$$r - \epsilon \leq dist(\mathbf{s}, \partial H) \leq r + \epsilon.$$

Again, the minimal value of $dist(\mathbf{s}, \partial H)$ is given at the point $\mathbf{h}$. The maximal value of $dist(\mathbf{s}, \partial H)$ is given at the intersection point between the line $\mathbf{sp}$ (see the illustration in Fig.3) and the ϵ-sphere on the other side (i.e., the point $\mathbf{h}'$). □

By Eq.(1) and Proposition 1, it is known that points on ∂B_H fall in the region bounded by $\partial O_H^{+\epsilon} = \{\mathbf{p} | dist(\mathbf{p}, \partial H) = r + \epsilon\}$ and $\partial O_H^{-\epsilon} = \{\mathbf{p} | dist(\mathbf{p}, \partial H) = r - \epsilon\}$. The region is denoted by O_H^ϵ. When the surfaces of O_H are not smooth, some sharp curves will be formed at the places having discontinuous normals. The two-side Hausdorff distance between the surfaces of O_H and O_H^ϵ depends on the sharpness of these discontinuous regions. As shown in Fig.4, when the crease becomes sharper (i.e., the angle α becomes smaller), the Hausdorff distance between O_H and O_H^ϵ increases. Generally, there is no bound on the sharpness of a resultant offset surface (i.e., the value of α varies from case to case). The point set P_S is sampled from ∂B_H; thus, that Hausdorff distance between P_S and ∂O_H is also not bounded. However, according to definition of the offsetting operator and its relationship to the Minkowski sum of balls (ref. [3]), we know the relationship explained in Remark 3.

Remark 3 (*Convergency to Exact Offset*) When $w \to 0$, the result of the ball-union converges to exact offset.

Moreover, when the exact offset surfaces do not have very sharp regions, the Hausdorff distance between ∂B_H and ∂O_H could be bounded by a value δ. In these cases, the error bound of ball-union in LDI representation can be derived (see Appendix).

4. GPU-based Parallel Offset Computation

This section presents the GPU-based algorithms for computing the approximate offset of solids with the help of LDI representation.

4.1. *Primary scheme*

LDI samples resulting from ball-union can be computed on rays in parallel. The whole algorithm consists of three major steps.

Step 1) For each ray (i, j) in the τ-direction, we search its neighboring rays in the same direction – i.e., the rays with indices $(i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$ also in the τ-direction. If a sphere centered at a sample on the ray (k, l) intersects the ray (i, j), two intersection points with depth values d_l and d_u will be obtained. These two samples are merged into the existing samples on the ray (i, j) in an efficient way. Firstly, as the existing samples on (i, j) are placed in an ascending order based on their depth values, the locations for the insertion of d_l and d_u on the ray can be computed by using a binary search. Next, the computation corresponding to removing the existing samples and adding new samples to generate the merging result can be performed based on different configurations. We perform the search and compute the ball-union on different rays in parallel using multiple cores on a GPU. During the search and ball-union computation, merging results on rays are stored in a local temporary array. After that, the merging results are added into a global data buffer array Π. In order to avoid *read-modify-write* (RMW) hazards in graphics global memory, the *atomicInc* operation [29] is used to determine the starting index of resultant samples on different rays. Meanwhile, the number of samples on a ray (i, j) and the starting index of the ray (i, j)'s samples in Π are stored in other two 2D arrays at the (i, j)-th element.

Step 2) This step searches the rays in other directions to check if the samples on those rays will intersect the ray (i, j). If so, the intersections will be merged into existing samples on (i, j) by the same method described above. For rays in the $(\tau + 1)$-direction, only rays in the range $(1 \cdots m, i \pm \lceil r/w \rceil)$ need to be searched. Similarly, for rays in the $(\tau+2)$-direction, we check sphere-ray intersections for rays in the range $(j \pm \lceil r/w \rceil, 1 \cdots m)$.

Step 3) In this step, the *scan* primitive [30] is employed to count the required size of elements before allocating the resultant data array D; information for the resultant index array I can be obtained during the scan. At last, a kernel is used to copy the samples from Π into D.

After obtaining the ball-union result in LDI representation (i.e., P_S), a Boolean operation [12, 14] is performed on

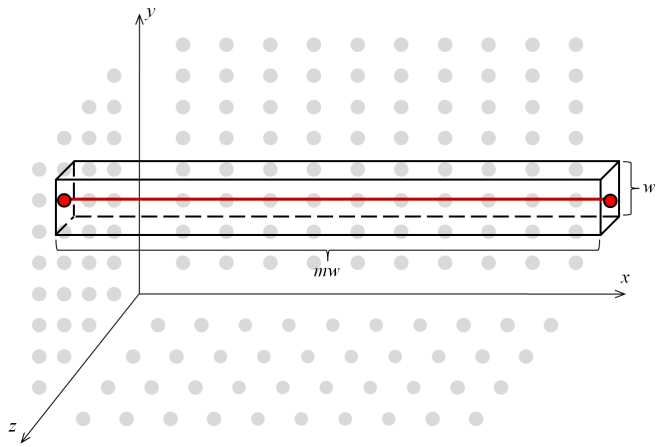


Fig. 5. A rectangular bar (with dimension: $mw \times w \times w$) is constructed around a ray (displayed in red) along the x-axis, which defines the bin of spatial hashing.

P_S and P_H to generate the final offsetting result. Normal vectors of samples can be obtained from the spherical surfaces that these samples belong to. However, normal vectors obtained in this way must be stored during the entire ball-union computation procedure, which has a high cost in terms of running time and memory overhead. A more efficient way is to reconstruct normal vectors in parallel by using an orientation-aware *Principle Component Analysis* (PCA) based on the neighborhoods of a sample (ref. [8]), which can be efficiently computed by searching the rays around a sample point. Note that the orientation of a sample can be obtained by its location on a ray (i.e., whether it is a point *entering* a 1D solid or *leaving* the solid).

4.2. *Spatial hashing by LDI*

By the primary scheme described in Section 4.1, ball-union computation running on the GPU results in 15-30 times speedup as compared to a CPU based single-core implementation [25]. The second of the three steps of ball-union corresponds to a bottleneck – searching intersections between a ray and the spheres centered at the points on rays in other directions. For the example shown in Fig.1, the second step takes more than 80% of the total time in the grown offsetting and more than 85% of the total time in the shrunk offsetting. During this step, for each ray (i, j) in the τ-direction, all samples on around $2m \times \lceil \frac{2r}{w} \rceil$ rays in $(\tau + 1)$- and $(\tau + 2)$-directions are searched. However, spheres centered at most of these samples do not intersect the ray (i, j). A spatial hashing method is used to remove the redundant searches on these rays.

Rectangular bars (in total of $m \times m$) are constructed by locating their centers at the rays along the a-direction. Each bar has the width $mw \times w \times w$ in the τ-, $(\tau+1)$- and $(\tau+2)$-directions, respectively (see Fig.5 for an illustration). The bar along the (i, j) ray is labelled with the index $(im + j)$. The samples on rays in $(\tau + 1)$- and $(\tau + 2)$-directions are stored in a sorted array Θ keyed by the indices of bars containing these samples. The sorted array can be computed by the highly parallel *sort* primitive (ref. [30]). The index table Υ of spatial hashing, which stores the location of the first sample of a rectangular bar in Θ, can be generated in parallel by checking all the elements in Θ to compute the element whose index of the containing bar is different from its previous element. By this index table Υ and the sorted array Θ, the samples close to the ray (i, j) are stored in Θ from $\Theta[\Upsilon[i, j]]$ to $\Theta[\Upsilon[i, j + 1] - 1]$.

Therefore, in the second step of ball-union, we only check the samples contained by the bars in the range of $(i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$ to generate spheres that intersect the ray (i, j). Specifically, the bars with index (k, l) are excluded if $(k - i)^2 + (l - j)^2 > (r/w + \sqrt{2})^2$ because the spheres centered at points in these bars will not intersect the ray (i, j). This spatial hashing technique can reduce the search time in the second step by up to 80%.

4.3. *Balancing workload on rays*

According to our empirical results, we observe that the number of merging operators taken in the second step of ball-union could be quite different on neighboring rays. As a result, if we run the ray-based search and merging in parallel on many-cores of the GPU, the workload in different threads could be considerably different – and the performance will be governed by the slowest thread.

To solve this problem, we add a workload estimation step before the search and merging steps. The number of merging operations to be performed on each ray is first estimated by checking the number of spheres that intersect that specific ray. Next, the rays are sorted according to the number of intersections, which indicates the workload on each ray during the ball-union computation. The search and merging on the rays with zero intersecting spheres is discarded. Starting from the first ray with non-zero intersecting spheres, sorted rays are packaged into bins, where each has around n_r rays.

$$n_r = c \times \#block\_per\_grid \times \#thread\_per\_block \quad (2)$$

We use $c = 10$. The rays classified into the same bin will have similar workloads, and the search and merging on them are conducted in parallel in the multiple threads of the GPU. This ray-packaging technique is similar to the workload balancing strategy used in [31] and can reduce computing time by up to 60% in our parallel ball-union approach.

The pseudo-code of our parallel ball-union approach is listed in **Algorithm** *ParallelBallUnionOnLDI*, which can be implemented easily with the help of the *scan*, *sort*, and *compact* primitives [30].

4.4. *Successive offsetting*

Both the search range of sphere-ray intersections and the number of spheres intersecting a ray depend on the off-

Algorithm 1: ParallelBallUnionOnLDI

Input: the LDI solid P_H and the offsetting distance r
Output: the ball-union result P_S in LDI-rep

Construct an empty LDI solid P_S ;
foreach *axis* τ *in* $\{x, y, z\}$ **do**
  foreach *ray* (i, j) *of* P_H *in* τ*-direction* ***in parallel*** **do**
    foreach *ray* (k, l) *in the same direction with* $(k, l) \in (i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$ **do**
      Compute the intersections between the ray (i, j) and the sphere centered at each sample on the ray (k, l);
      Merge the points of intersected 1D solids into the existing samples on the ray (i, j);
    end
  end
  Initialize the index table Υ and the data array Θ for spatial hashing;
  Insert the samples of rays of P_H in the $(\tau + 1)$- and $(\tau + 2)$-directions into the data array Θ, each sample is keyed with the index of the rectangular bar containing the sample;
  Sort the samples of Θ in an ascending order;
  Use the *scan* primitive to build the index table Υ;
  foreach *ray* (i, j) *of* P_H *in* τ*-direction* ***in parallel*** **do**
    Search the samples contained by the bars in the range of $(i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$, and count the number of intersection points;
  end
  Sort rays in τ-direction by the number of intersection points, which indicate the workload for ball-union on a ray;
  Package the sorted rays into bins;
  foreach *bin of rays in* τ*-direction* **do**
    foreach *ray* (i, j) *of* P_H *in the same bin* ***in parallel*** **do**
      Compute the intersections between the ray (i, j) and the spheres centered at samples contained by the bars in the range of $(i \pm \lceil r/w \rceil, j \pm \lceil r/w \rceil)$;
      Merge the points of intersected 1D solids into the existing samples on the ray (i, j);
    end
  end
  Fill the ball-union result on rays in the τ-direction into the LDI solid P_S;
end
return

setting distance r. Obviously, having an extremely large r (e.g., the diagonal length of the model's bounding box) will make the computation of point-based offsetting very slow. The order of computational complexity is around $O(\lceil r/w \rceil^2)$. Based on the general bounds for solid offset computation [3], it is known that offsetting of a solid with distance r can be decomposed into n successive offsetting operations with distance $r_1, r_2, \ldots, r_n$ if the offsetting distances satisfy: 1) $r = \sum_{i=1}^{n} r_i$ and 2) all r_is and r have the same sign. Without loss of generality, if $r_i = r/n$, the complexity of computation becomes $O(n\lceil r/nw \rceil^2) \approx O(\frac{1}{n}\lceil r/w \rceil^2)$.

In this paper, our point-based method computes the approximate offsetting. As analyzed in Section 3.3, the computed shell solid O_H in LDI representation slightly enlarges the shape approximation error of the input LDI solid from ϵ to $\sqrt{\delta^2 + (1 + \frac{\delta}{r})\frac{\epsilon^2}{4}}$. Therefore, performing the offset too many times in succession will generate a result with a large approximation error. According to our experiences, the following decomposition

$$n = \lceil r/(5w) \rceil, \quad r_i = r/n, \tag{3}$$

gives a good trade-off between speed and error in approximate offset computation.

Table 1
Statistics of Computing Time (ms) on a GeForce GTX 580 GPU

LDI resolution: 256 × 256					
	Input	Relative Offset Distance†: r/L_d			
Models	Point #	−0.05	0.025	0.05	0.1
Octa-flower	136,454	452	343	749	1,747
Vase-lion	180,730	811	483	874	2,215
Filigree	107,308	266	281	618	1,482
Buddha	106,950	453	297	681	1,607
LDI resolution: 512 × 512					
	Input	Relative Offset Distance†: r/L_d			
Models	Point #	−0.025	0.0125	0.025	0.05
Octa-flower	553,648	1,389	952	1,840	3,947
Vase-lion	731,052	2,246	1,404	2,543	5,101
Filigree	438,590	1,373	1,466	2,344	4,540
Buddha	432,078	1,357	967	1,794	3,635

† The offset distances are specified as relative values w.r.t. the diagonal length of the enclosed bounding box of the model, L_d.

5. Results and Discussion

In this section, we highlight the performance of our algorithm on different benchmarks. All the timings reported here were recorded on a machine using an Intel Core i5 2.67GHz CPU and 4GB memory. We implemented our ball-union algorithm using CUDA on a GeForce GTX 580 GPU with 1GB of video memory.

Figures 6, 7 and 8 show the four benchmarks used in our experimental tests and the results of both grown and shrunk offsetting. The statistics of computing time have been listed in Table 1. It is easy to conclude that offsetting on all these benchmarks can be computed at almost interactive rates.

Table 2
Offset Computation Time[§] (sec.) on GPU vs. CPU

Methods	Relative Offsetting Distance[†]: $r = -0.025L_d$		
	Vase-lion	Filigree	Buddha
CPU (1-core)	639	448	462
CPU (4-cores)	207 (×3.1)	146 (×3.1)	129 (×3.6)
GPU Primary	20.8 (×31)	15.9 (×28)	16.3 (×28)
GPU SH	9.03 (×71)	7.24 (×62)	5.40 (×86)
GPU SH+P	5.71 (×112)	2.39 (×187)	3.40 (×136)
GPU SH+P+Succ	2.25 (×284)	1.37 (×327)	1.36 (×340)
Methods	Relative Offsetting Distance[†]: $r = 0.0125L_d$		
	Vase-lion	Filigree	Buddha
CPU (1-core)	216	188	187
CPU (4-cores)	69.6 (×3.1)	58.5 (×3.2)	58.0 (×3.2)
GPU Primary	12.0 (×18)	9.38 (×20)	9.22 (×20)
GPU SH	2.82 (×77)	3.60 (×52)	2.01 (×93)
GPU SH+P	1.62 (×133)	1.39 (×135)	1.19 (×157)
GPU SH+P+Succ	1.40 (×154)	1.47 (×128)	.967 (×193)
Methods	Relative Offsetting Distance[†]: $r = 0.025L_d$		
	Vase-lion	Filigree	Buddha
CPU (1-core)	446	368	374
CPU (4-cores)	151 (×3.0)	122 (×3.0)	123 (×3.0)
GPU Primary	25.8 (×17)	18.3 (×20)	18.5 (×20)
GPU SH	5.37 (×83)	3.06 (×120)	2.76 (×136)
GPU SH+P	5.15 (×87)	2.93 (×126)	3.15 (×119)
GPU SH+P+Succ	2.54 (×176)	2.34 (×157)	1.79 (×209)

[§] The numbers in parenthesis are the speedups compared to a single-core CPU implementation.
[†] The offset distances are specified as relative values w.r.t. the diagonal length of the enclosed bounding box of the model, L_d.
[‡] The input LDI models have resolution 512×512.

In order to study the performance of the spatial hashing, ray-packaging and successive offsetting techniques used in our approach, the timing results are shown in Table 2. The implementations on single and multi-core CPUs, by using a GPU-based primary scheme (Section 4.1), by using spatial hashing (SH), by using SH and ray-packing (SH+P), and by successive offsetting (SH+P+Succ) are compared on three benchmarks. We can observe up to about 6.8 times speedup on spatial hashing as compared to the primary scheme. The improvement by packaging rays according to the workload estimation is significant when the offset distance is relatively small (e.g., about $70 - 200\%$ speedup when r is 0.0125 in Table 2). When the value of r is large, the workloads on different rays are similar to each other. In these cases, the workload estimation and the sorting of rays take extra time and the overall algorithm spends more time on SH+P than on SH only (see the cases with $r = 0.025$ in Table 2). After decomposing the offsetting with a large distance into successive operations with smaller distances, generally about $16 - 150\%$ improvement can be observed on various benchmarks. Note that the approximation error can be large when taking many successive offsets with small distances. There is one special case in which SH+P+Succ is slower than SH+P (see the filigree model with $r = 0.0125$ in Table 2), which happens when the value of r is not very big – only about two successive operations are applied.

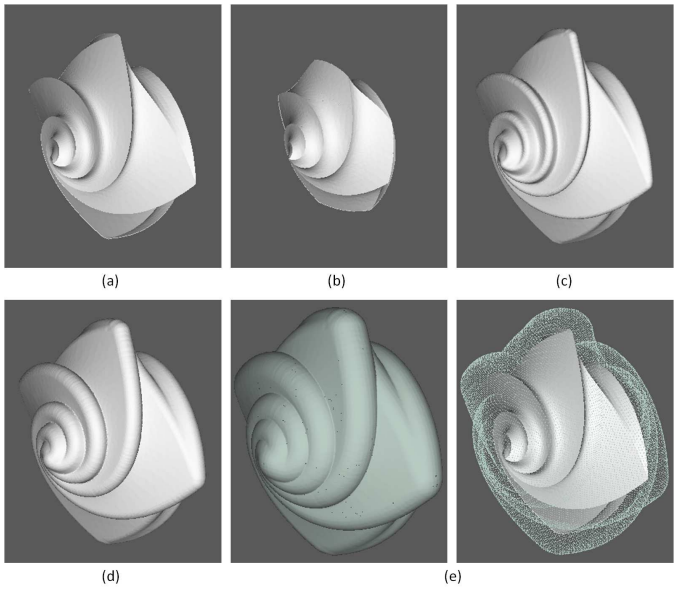


Fig. 6. Offsetting the Octa-flower model with different distances: (a) input model, (b) $r = -0.025L_d$, (c) $r = 0.0125L_d$, (d) $r = 0.025L_d$, and (e) $r = 0.05L_d$. L_d is the diagonal length of bounding box.

The Buddha model shown in Fig.8 is also used in [7, 15] to compute the offset with distance at 2% of the bounding box's diagonal length. The offsetting computation takes more than 3,000 seconds by [7] and around 180 seconds by [15], respectively. Both are computed on the distance-fields at the 512^3 resolution. By our approach, the offset can be obtained in 1.69 seconds by 512×512 LDI representation. More than 100 times speedup is observed as compared to [15].

Figure 9 shows the performance of our algorithm on different benchmarks. These results show that our algorithm works well on three different GPUs. Compared to the last generation of GPUs (i.e., GeForce GTX 295), the algorithm achieves 4 - 11x speedups on a GeForce GTX 580 for different models at different resolutions.

We also study the accuracy of offsetting results in our experimental tests. The distances between all the points in P_S and the boundary surface ∂H of the given solid H are measured by the publicly available PQP separation distance computation library [32]. Two error metrics, the average error and the maximal error, are employed in our analysis.

$$E_{avg} = \frac{1}{n(P_S)|} \sum_{\forall \mathbf{p} \in P_S} |dist(\mathbf{p}, \partial H) - r|, \tag{4}$$

$$E_{max} = \max_{\forall \mathbf{p} \in P_S} |dist(\mathbf{p}, \partial H) - r|, \tag{5}$$

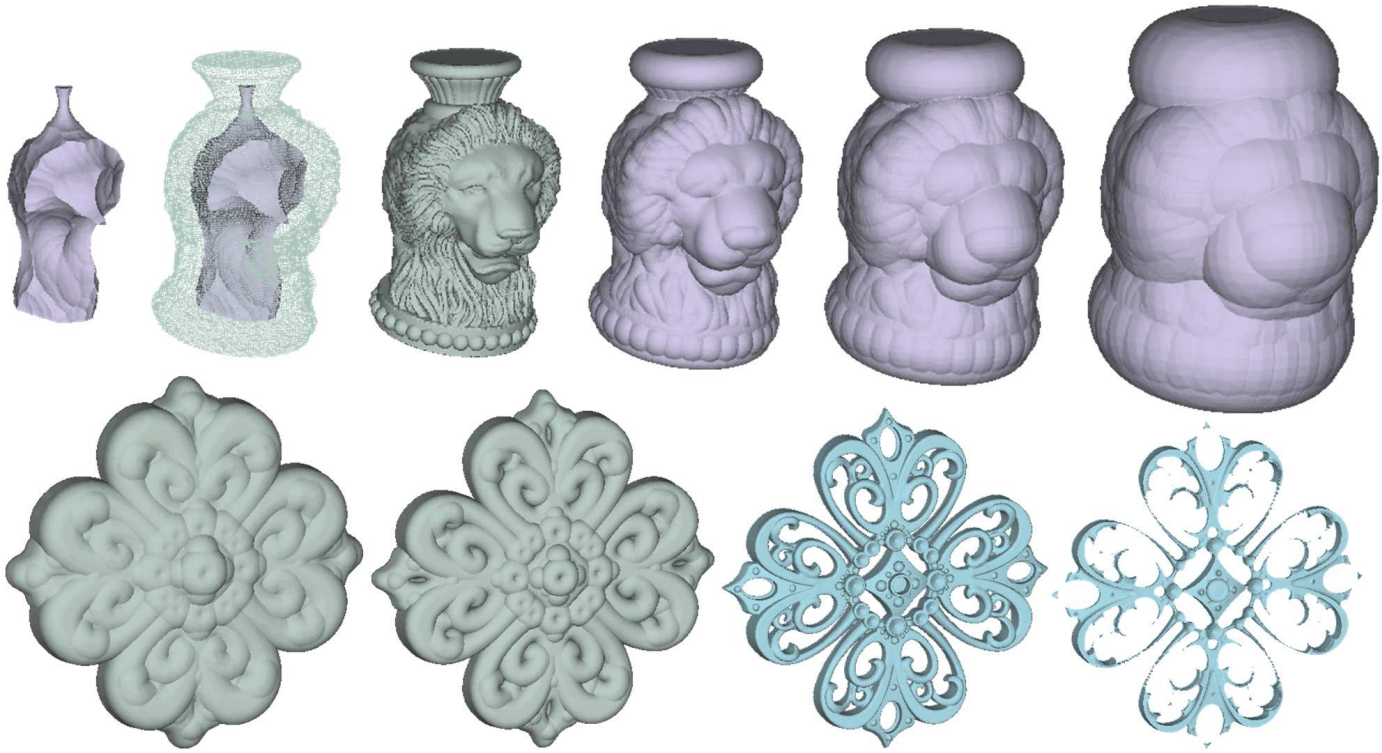


Fig. 7. The Vase-lion and Filigree models used in our tests of parallel offsetting: (top row) the offset of Vase-lion with distances as (-0.05), 0.025, 0.05 and 0.1 of the diagonal length L_d of the bounding box, and (bottom row) offsetting the filigree model with distances of 0.05, 0.025 and (-0.0125) of L_d. Note that the points are displayed sparsely, and the actual representation is more dense.

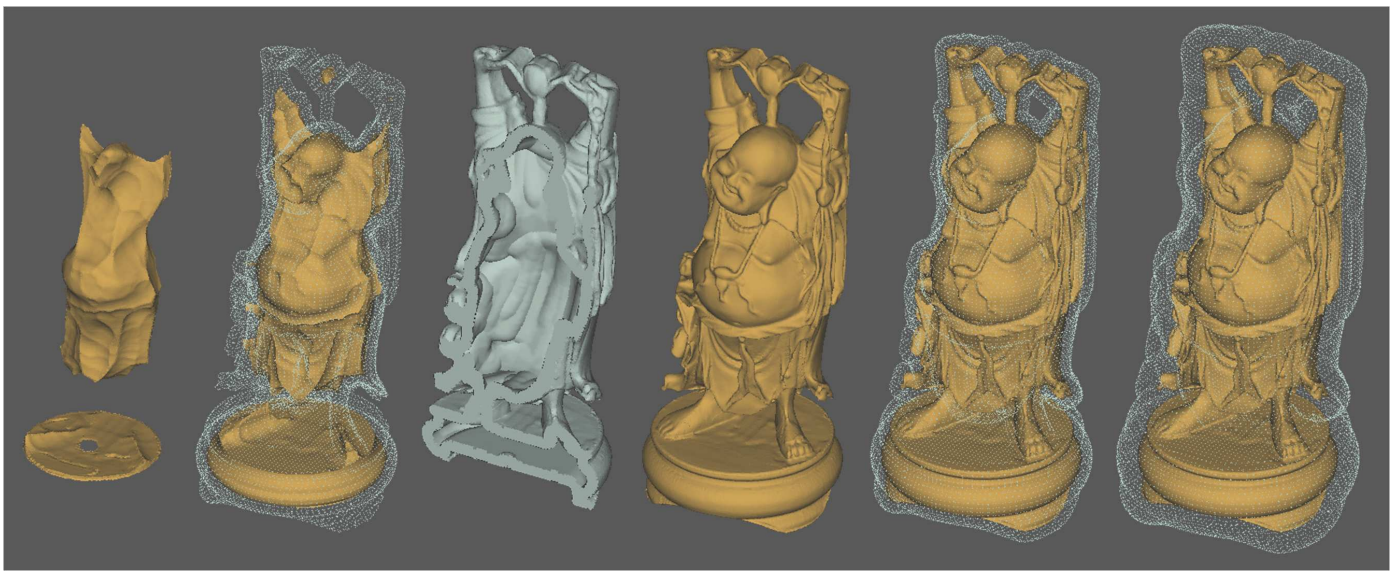
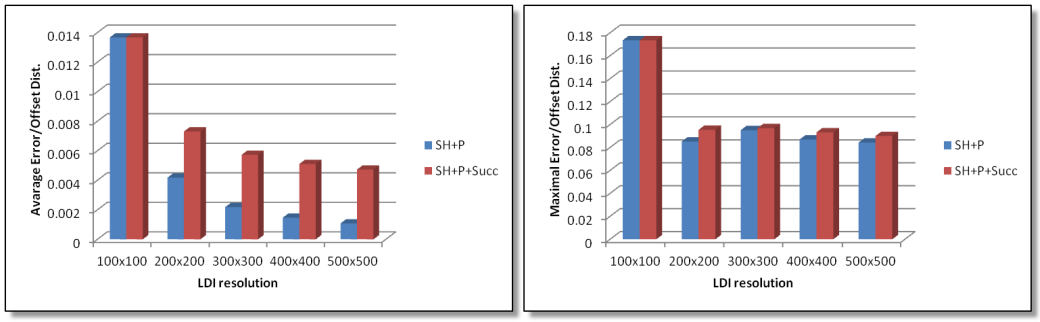
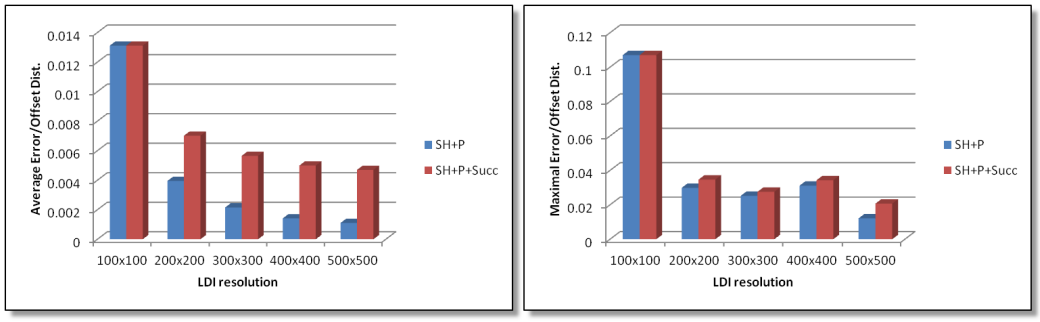


Fig. 8. Offsetting Buddha model with different distances. From left to right: $r = -0.0125L_d$; $r = -0.025L_d$; the hollowing model obtained by offsetting and Boolean; the given model; $r = 0.025L_d$; and $r = 0.05L_d$. L_d denotes the diagonal length of bounding box. For the visualization purpose, the points are displayed more sparsely than the real density.

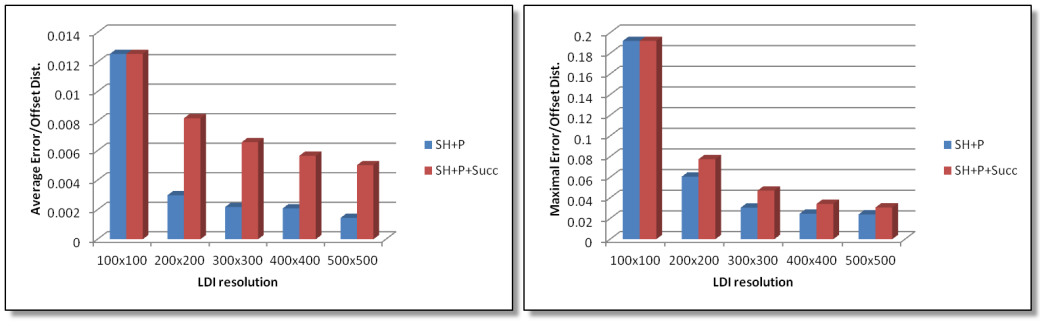
where $n(P_S)$ denotes the number of points in P_S. Figure 10 shows the charts for the accuracy analysis on three cases. Both the average error and the maximal error are reported with reference to the offset distance (i.e., E_{avg}/r and E_{max}/r). Moreover, the offsetting results without vs. with successive decomposition ('SH+P' vs. 'SH+P+Succ') are also studied. It is not surprising to find that the results with 'Succ' present larger errors in both E_{avg} and E_{max}. The results with the same offset distance but different LDI resolutions (i.e., different ϵ-coverings) are compared; the value of offset distance r is chosen as $25w$ at the LDI resolution 500×500 in all the cases of Fig.10. The shape approximation error generated in our approach converges while increasing the sampling rate in LDI representation.



(a) Grown offsetting of Buddha: $r = 0.0790$



(b) Shrunk offsetting of Buddha: $r = -0.0790$



(c) Grown offsetting of Filigree: $r = 0.0554$

Fig. 10. Error analysis of offsetting results shows that the approximation errors are reduced when increasing the resolutions of LDI, where the errors are reported w.r.t. the offset distance r.

The GPU-based *Euclidean Distance Transform* (EDT) proposed in [24] can be used to compute the distance-field to a set of points on uniformly sampled grid nodes. We test their approach on the same PC equipped with a GeForce GTX 580 graphics card, and compare its performance with the approximate offset proposed in this paper. Different resolutions are tested on both EDT and our ball-union approach on the Buddha model shown in Fig.8. As shown in Table 3, when increasing the resolution of the distance field to be greater than 512, the memory is used up very soon by the EDT approach. We also check the performance of our approach on different offsetting distances. It can be observed that our approach is faster than EDT when computing on high resolution LDI with small offset distances. Although our method becomes slower while increasing the offset distance, the LDI based approach can compute the offset surface for high sampling rates (1024^2 or higher), which is important for applications that require more accuracy.

Table 3
Comparison with EDT-based method [24]

Res.	Input Point #	Time (ms) of EDT	Time (ms) of Our Approach $r = 3w$	$r = 5w$	$r = 8w$	$r = 11w$
256	106,950	78	110	171	234	312
512	432,078	686	327	468	873	1,311
1024	1,740,424	n/a	936	1,747	3,806	6,318
2048	6,988,304	n/a	5,180	9,189	n/a	n/a

§ 'n/a' stands for the case that memory is used out.
† Buddha model shown in Fig.8 is used in this test.
‡ The 'SH+P' scheme of our approach is tested to conduct the comparison.

Limitation on topological error

In Section 3.3, only the geometric error bounds are given for the ball-union algorithm presented in this paper. From the analysis in [26], we know that for the LDI representation P_H of a solid model H, if H is d-regular with $d > \sqrt{3}w$

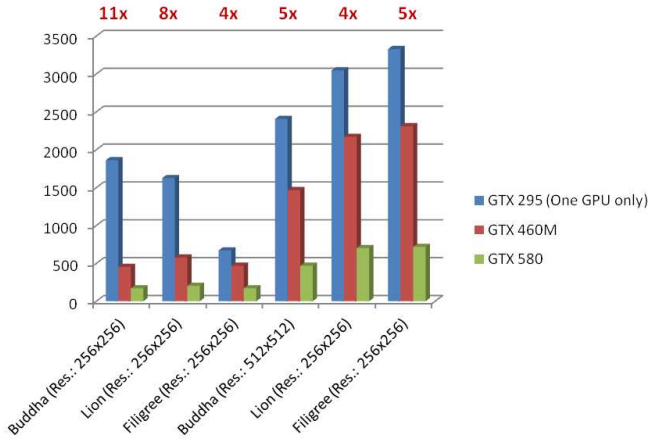


Fig. 9. Performance comparison of our algorithm on three different GPUs. Models are sampled into LDI with 256×256 and 512×512 resolutions, and the results are computed with offsetting distance $r = 5w$. Note that, GTX 295 is a dual-GPUs graphics card; here, only one GPU is used in our tests.

and w being the sampling distance between rays of LDI, the boundary surface M of P_H generated by a *topology-preserving* method on cubic grids formed by the rays of LDI is d-homeomorphic to the exact surface boundary, ∂H. However, the resultant exterior/interior offsetting solids with different offsetting distances could have different bound, d, for the d-regular. Specifically, for any point $\mathbf{p}$ on the boundary of H_r^+ (or H_r^-), the maximal osculating ball that lies entirely outside H_r^+ (or inside H_r^-) may have a radius less than d. As a result, the approximate offsets computed with a fixed sampling distance could have different topology as compared to the results of exact offsetting.

6. Conclusion

In this paper, we present a highly parallel method to compute the approximate offsetting on models represented by structured points in *Layered Depth Images* (LDI). The main approach is to compute the *super-union* of all the balls centered at the input points, which can be conducted in a highly parallel manner on the GPU with the help of LDI representation. Our approximate offsetting algorithm computes the boundary whose accuracy is governed by the sampling rate of LDI. As compared to prior CPU-based approximation offsetting algorithms, our approach results in more than 100 times speedups. Overall, this is the first approach that can compute the offset approximation of complex solid models at almost interactive rates on current GPUs.

In terms of future work, we would like to generalize this approach and use it to compute non-uniform offsetting, which has many applications including biomedical engineering [33]. We would like to further improve its performance to perform the computations at interactive rates and also give tight bounds on the topological error.

Acknowledgment

This work was partially supported by the Hong Kong RGC/GRF Grant (CUHK/417508 and CUHK/417109) and the Direct Research Grant (CUHK/2050518), and it was also supported by NSF grants 1000579 and 1117127.

References

[1] T. Maekawa, An overview of offset curves and surfaces, *Computer-Aided Design*, vol.31, no.3, pp.165-173. (1999)
[2] B. Pham, Offset curves and surfaces: a brief survey, *Computer-Aided Design*, vol.24, no.4, pp.223-229. (1992)
[3] J.R. Rossignac and A. Requicha, Offsetting operations in solid modeling, *Computer Aided Geometric Design*, vol.3, no.2, pp.129-148. (1986)
[4] J. Williams and J. Rossignac, Mason: Morphological simplification, *Graphical Models*, vol.67, no.4, pp.285-303. (2005)
[5] C.M. Hoffmann, *Geometric and Solid Modeling: An Introduction*, Morgan Kaufmann Pub. (1989)
[6] G. Varadhan and D. Manocha, Accurate Minkowski sum approximation of polyhedral models, *Graphical Models*, vol.68, no.4, pp.343-355. (2006)
[7] D. Pavić and L. Kobbelt, High-resolution volumetric computation of offset surfaces with feature preservation, *Computer Graphics Forum*, vol.27, no.2, pp.165-174. (2008)
[8] S. Liu and C.C.L. Wang, Fast intersection-free offset surface generation from freeform models with triangular meshes, *IEEE Transactions on Automation Science and Engineering*, vol.8, no.2, pp.347-360. (2011)
[9] J.M. Lien, Covering Minkowski sum boundary using points with applications, *Computer Aided Geometric Design*, vol.25, no. 8, pp.652-666. (2008)
[10] Y. Chen and C.C.L. Wang, Uniform offsetting of polygonal model based on Layered Depth-Normal Images, *Computer-Aided Design*, vol.43, no.1, pp.31-46. (2011)
[11] J. Shade, S. Gortler, L.-W. He, and R. Szeliski, Layered depth images, *Proc. of SIGGRAPH '98*, pp.231-242. (1998)
[12] C.C.L. Wang, Y.-S. Leung, and Y. Chen, Solid modeling of polyhedral objects by Layered Depth-Normal Images on the GPU, *Computer-Aided Design*, vol.42, no.6, pp.535-544. (2010)
[13] C.C.L. Wang, Approximate Boolean operations on large polyhedral solids with partial mesh reconstruction, *IEEE Transactions on Visualization and Computer Graphics*, vol.17, no.6, pp.836-849. (2011)
[14] H. Zhao, C.C.L. Wang, Y. Chen, and X. Jin, Parallel and efficient Boolean on polygonal solids, *The Visual Computer*, vol.27, no.6-8, pp.507-517. (2011)
[15] S. Liu and C.C.L. Wang, Fast intersection-free offset surface generation from freeform models with triangular meshes, *IEEE Transactions on Automation Science and Engineering*, vol.8, no.2, pp.347-360. (2011)
[16] M. Forsyth, Shelling and offsetting bodies, *Proc. ACM Symposium on Solid Modeling and Applications*, pp.373-381. ACM, Salt Lake City, Utah. (1995)
[17] X. Qu and B. Stucker, A 3D surface offset method for STL-format models, *Rapid Prototyping Journal*, vol.9, no.3, pp.133-141. (2003)
[18] D.E. Breen and S. Mauch, Generating shaded offset surfaces with distance, closest-point and color volumes, *Proc. International Workshop on Volume Graphics (Swansea, UK)*, pp.307-320. (1999)
[19] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, *Proc. of ACM SIGGRAPH 2000*, pp.249-254. ACM, New Orleans, LA. (2000)

[20] S. Gottschalk, M.C. Lin, and D. Manocha, OBBTree: A hierarchicial structure for rapid interference detection, *Proc. of ACM SIGGRAPH 96*, pp.171-180. ACM, New York, NY. (1996)
[21] E.E. Hartquist, J.P. Menon, K. Sursh, H.B. Voelcker, and J. Zagajac, A computing strategy for applications involving Offsets, Sweeps, and Minkowski operations, *Computer-Aided Design*, vol.31, no.3, pp.175-183. (1999)
[22] K. Yin, Y. Liu, and E. Wu, Fast computing adaptively sampled distance field on GPU, *Proc. of Pacific Graphics 2011*, Short Paper. 2011
[23] W. Li and S. McMains, Voxelized Minkowski sum computation on the GPU with robust culling, *Computer-Aided Design*, vol.43, pp.1270-1283. (2011)
[24] T.-T. Cao, K. Tang, A. Mohamed, and T.-S. Tan, Parallel banding algorithm to compute exact distance transform with the GPU, *Proc. of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp.83-90. (2010)
[25] C.C.L. Wang, Computing on rays: a parallel approach for surface mesh modeling from multi-material volumetric data, *Computers in Industry*, vol.62, no.7, pp.660-671. (2011)
[26] P. Stelldinger, L.J. Latecki, and M. Siqueira, Topological equivalence between a 3D object and the reconstruction of its digital image, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.29, no.1, pp.126-140. (2007)
[27] M. Guthe, A. Balazs, and R. Klein, GPU-based trimming and tessellation of NURBS and T-Spline surfaces, *ACM Transactions on Graphics*, vol.24, no.3, pp.1016-1023. (2005)
[28] Y.-S. Leung, and C.C.L. Wang, Conservative sampling of solids in image space, *IEEE Computer Graphics and Applications*, accepted. (2012)
[29] CUDA programming guide for cuda toolkit 3.0, http://developer.nvidia.com/object/gpucomputing.html, NVIDIA Corporation, 2010.
[30] M. Harris, S. Sengupta, and J.D. Owens, Parallel prefix sum (scan) with CUDA. In H. Nguyen, editor, *GPU Gems 3*. Addison Wesley. (2007)
[31] J. Pan and D. Manocha, GPU-based parallel collision detection for fast motion planning, *International Journal of Robotics Research*, vol.31, no.2, pp.187-200. (2012)
[32] E. Larsen, S. Gottschalk, M.C. Lin, and D. Manocha, Fast proximity queries with swept sphere volumes, *Proceedings of International Conference on Robotics and Automation*, pp.3719–3726. (2000)
[33] C. Bajaj and S. Goswami, Multi-component heart reconstruction from volumetric imaging, *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*, pp.193-202. (2008)

Appendix

The error-bound of LDI-based offsetting can be derived on the cases that the Hausdorff distance between ∂B_H and ∂O_H is bounded.

Proposition 2 (*error-bound* of ball-union on LDI) When the Hausdorff distance between ∂B_H and ∂O_H is δ, the Hausdorff distance between ∂O_H and the point set P_S is bounded by $\sqrt{\delta^2 + (1+\frac{\delta}{r})\frac{\epsilon^2}{4}}$.

Proof. First, as $P_S \subset \partial B_H$, the bound δ of the Hausdorff distance between ∂B_H and ∂O_H gives this bound:

$$\forall \mathbf{a} \in \partial P_S,\ dist(\mathbf{a}, \partial O_H) \le \delta.$$

The bound can also be expressed as:

$$\forall \mathbf{a} \in \partial O_H,\ dist(\mathbf{a}, \partial B_H) \le \delta.$$

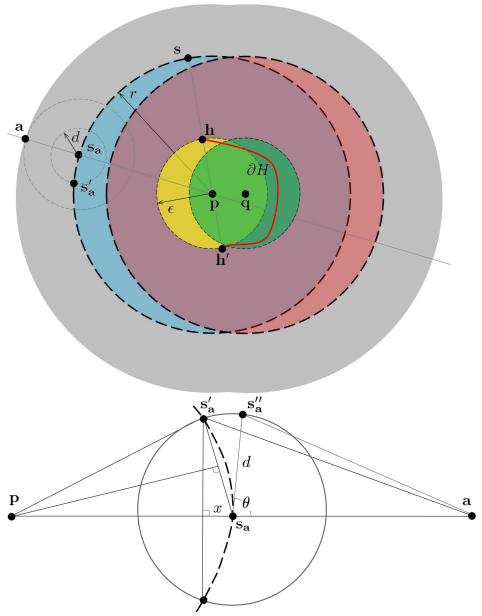


Fig. 11. An illustration for Proposition 2.

For example, the gray region shown in Fig.11 is described by this formula. However, the closest point $\mathbf{s_a}$ ($\mathbf{s_a} \in \partial B_H$) to the point $\mathbf{a}$ may not be a member of P_S when the ball-union is computed on the rays of LDI representation. As shown in Remark 1 and 2, we can know that there must be a point $\mathbf{s'_a} \in P_S$ in the region such that their distance $\|\mathbf{s_a s'_a}\| \le \frac{\epsilon}{2}$.

If there is a point $\mathbf{s'_a}$ that is located on the same sphere that contains $\mathbf{s_a}$ (i.e., $S_\mathbf{p}$) with $\|\mathbf{s_a s'_a}\| = d$, since $\|\mathbf{s_a p}\| = \|\mathbf{s'_a p}\| = r$, this implies that

$$\frac{x}{\|\mathbf{s_a s'_a}\|} = \frac{\|\mathbf{s_a s'_a}\|/2}{\|\mathbf{s_a p}\|}$$

which leads to $x = \frac{d^2}{2r}$ (see the bottom of Fig.11). Therefore, the distance between $\mathbf{a}$ and $\mathbf{s'_a}$ is

$$\|\mathbf{s'_a a}\| = \sqrt{(x+\delta)^2 + (d^2 - x^2)} = \sqrt{\delta^2 + (1+\frac{\delta}{r})d^2} \qquad (6)$$

As $d \le \epsilon/2$, it has $\|\mathbf{s'_a a}\| \le \sqrt{\delta^2 + (1+\frac{\delta}{r})\frac{\epsilon^2}{4}}$.

For the point $\mathbf{s''_a} \in P_S$ that $\|\mathbf{s_a s''_a}\| = d \le \frac{\epsilon}{2}$ and does not lie inside the sphere $S_\mathbf{p}$, it must satisfy the relationship $\|\mathbf{s''_a p}\| \ge r$. Otherwise, this point will not lie on the boundary surface of the union of 1D solid obtained from the sphere $S_\mathbf{p}$. As illustrated in the bottom of Fig.11, $\mathbf{s''_a}$ must be located in the right side of the dashed circular arc. Without loss of generality, having $\|\mathbf{s_a} - \mathbf{a}\| = \delta$, we can show that

$$\begin{aligned} \|\mathbf{s''_a a}\| &= \sqrt{(\delta - d\cos\theta)^2 + (d\sin\theta)^2} \\ &= \sqrt{\delta^2 + d^2 - 2\delta d\cos\theta}, \end{aligned} \qquad (7)$$

and this is an increasing function of θ when $\theta \in [0, \pi]$ with a fixed d. Thus, a maximum is obtained if $\mathbf{s''_a}$ and $\mathbf{s'_a}$ are coincident. In other words, $\|\mathbf{s''_a a}\| \le \|\mathbf{s'_a a}\|$. Therefore,

$$\forall \mathbf{a} \in \partial O_H,\ dist(\mathbf{a}, \partial P_S) \le \sqrt{\delta^2 + (1+\frac{\delta}{r})\frac{\epsilon^2}{4}}.$$

The proposition has been proved. □