GPU-Based Super-union for Minkowski Sum

Yuen-Shan Leung[1], Charlie C. L. Wang[1] and Yong Chen[2]

[1]The Chinese University of Hong Kong,
yleung@mae.cuhk.edu.hk
cwang@mae.cuhk.edu.hk

[2]University of Southern California,
yongchen@usc.edu

ABSTRACT

We present an efficient and robust algorithm to approximate the 3D Minkowski sum of two arbitrary polyhedra on Graphics Processing Unit (GPU). Our algorithm makes use of the idea of super-union, in which we decompose the two polyhedra into convex pieces as usual, but the way we perform pairwise convex Minkowski sum and merge the pairwise sums one by one is changed to group by group on GPU. The core technique involved is to directly compute the convex hull of pairwise sum in image representation and utilize voxelization to perform massive union operations. Despite the lack of accuracy, we guarantee that the voxelization of Minkowski sum is conservative, and the inner voids are well preserved. Our algorithm is also scalable and fits well into GPU's streaming architecture.

Keywords: Minkowski sum, voxelization, inner void, GPU, MAX/MIN blending.
DOI: 10.3722/cadaps.2013.475-487

1 INTRODUCTION

Minkowski sum is fundamental to many applications, such as solid modeling, computer-aided design, robot motion planning, penetration depth estimation and dynamic simulation. Given two point sets P and Q in $\Re^n$, the Minkowski sum of P and Q is defined as

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\} \tag{1.1}$$

In geometry, it is formed by translating the set Q by all vectors $p \in P$ or vice versa for polyhedra in $\Re^3$ (polygon in $\Re^2$). Although the definition looks simple, it is complicated in implementation and has a high computation cost. Let the computation of P and Q consist of m and n primitives respectively, the

Minkowski sum may have a complexity of up to $O(m^3n^3)$. A lot of research has been conducted to compute it accurately and efficiently [5, 9, 12, 24].

Generally, two main approaches have been adopted for the Minkowski sum computation of complex polyhedra. The first approach decomposes both polyhedra into convex pieces, computes their pairwise Minkowski sums and the union of pairwise Minkowski sums [25]. It is relatively easy to implement this but the union step becomes the bottleneck of performance when the number of pairs is large. The second approach is convolution- based. It generates a superset of the Minkowski sum and filters out primitives that do not belong to the final boundary surface [10]. The hardest part of this approach is to compute the arrangement of convolution and the wind number robustly in $\Re^3$. In view of the high complexity of computing the exact Minkowski sum, we have developed an approximation algorithm.

Main results: In this paper, we present an efficient and robust approach for computing voxelized 3D Minkowski sums of polyhedra on GPU. Unlike most existing approximation algorithms [16, 17, 20], which adopt a convolution- based approach and thus lead to the missing of inner voids, our method is decomposition- based and aims at computing the Minkowski sum of arbitrary polyhedra and preserving inner voids in terms of voxel. For some explicit applications like motion planning, these voids cannot be neglected as they usually represent critical pathways. We know that even the decomposition- based approaches are guaranteed to generate Minkowski sum with enclosed voids [23] and allow voids within the input models, their performance are degraded when a large number of components are united.

In order to solve the bottleneck of unions, this paper introduces a GPU- based streaming algorithm to perform a massive union of pairwise Minkowski sum on GPU. We create solid voxelization of the Minkowski sum and transmit it back to CPU once the whole union step is finished. Surface voxelization can also be easily obtained. The accuracy of our results is governed by the volumetric resolution and the quality of decomposition. As we use a resolution of 1024^3 in this paper, holes can be preserved as long as their length in one of the direction is larger than 10^{-4} in magnitude.

2 RELATEDWORK

There has been much work on finding practical algorithms for robust computation of Minkowski sums of complex polyhedral models in three dimensions. This goal is relatively easy to achieve if the models are convex and the process of Minkowski sum of polyhedra P and Q with m and n features respectively takes $O(mn)$ time. Ghosh presented a unified algorithm to compute Minkowski operations of polygons and polyhedra using a new representation scheme, called *slope diagram representation* [10]. He merged the diagrams of two polyhedra and extracted the boundary of Minkowski sum from the result. Some researchers followed this idea [3, 5, 9] but their implementations again work only for convex objects. Lately, Barki et al. [4] improved his work and his new approach can be applied to nonconvex- convex pair of polyhedra and deal with non- manifold situation.

Nevertheless, it is much more difficult to handle the Minkowski sum of two non- convex polyhedra as it could have complexity $O(m^3n^3)$ for the worst case. One of the common approaches is based on decomposing the non- convex objects into simpler and convex polyhedra [7, 8], computing all pairwise Minkowski sums of the convex pieces, and merging the pairwise sums [2]. Another similar strategy is decomposing the polyhedra into affine cells instead of convex pieces [21]. Although the idea is simple, decomposition and union steps can be difficult to implement robustly and efficiently [1, 13]. Hachenberger [12] introduced a robust and exact 3D Minkowski sum of two non- convex polyhedra with a decomposition- based algorithm. His work was built upon an existing library CGAL [22] and is able to optimize the union step through the use of Nef polyhedra. However, his results were still

constrained by the number of convex pieces. At present, no existing union method can efficiently compute the union of thousands of pairwise Minkowski sums. To reduce computational complexity, an approximation of the union step was introduced by Varadhan and Manocha [23]. They generated samples of Minkowski sum boundary in distance field and extracted the meshes using marching cube technique.

Convolution is an alternative approach to compute Minkowski sum of non- convex objects. Guibas et al. introduced the concept of convolution and proposed an output sensitive algorithm to compute 2D convolution curves [11]. Kaul et al. [15] constructed supersets of boundaries of Minkowski sums and provided a set of criteria to remove facets which do not belong to the actual boundaries. This concept can also be used in our algorithm to remove unnecessary facets of each pairwise Minkowski sum. Wein [24] first described a robust and exact software implementation for non- convex polygons based on a convolution method. His work can handle inputs with many degeneracies and yield topologically correct results. Later on Lien [18] proposed a nearly exact algorithm for polyhedra. He first generated a brute- force convolution of two input polyhedra and computed the 2D arrangement of

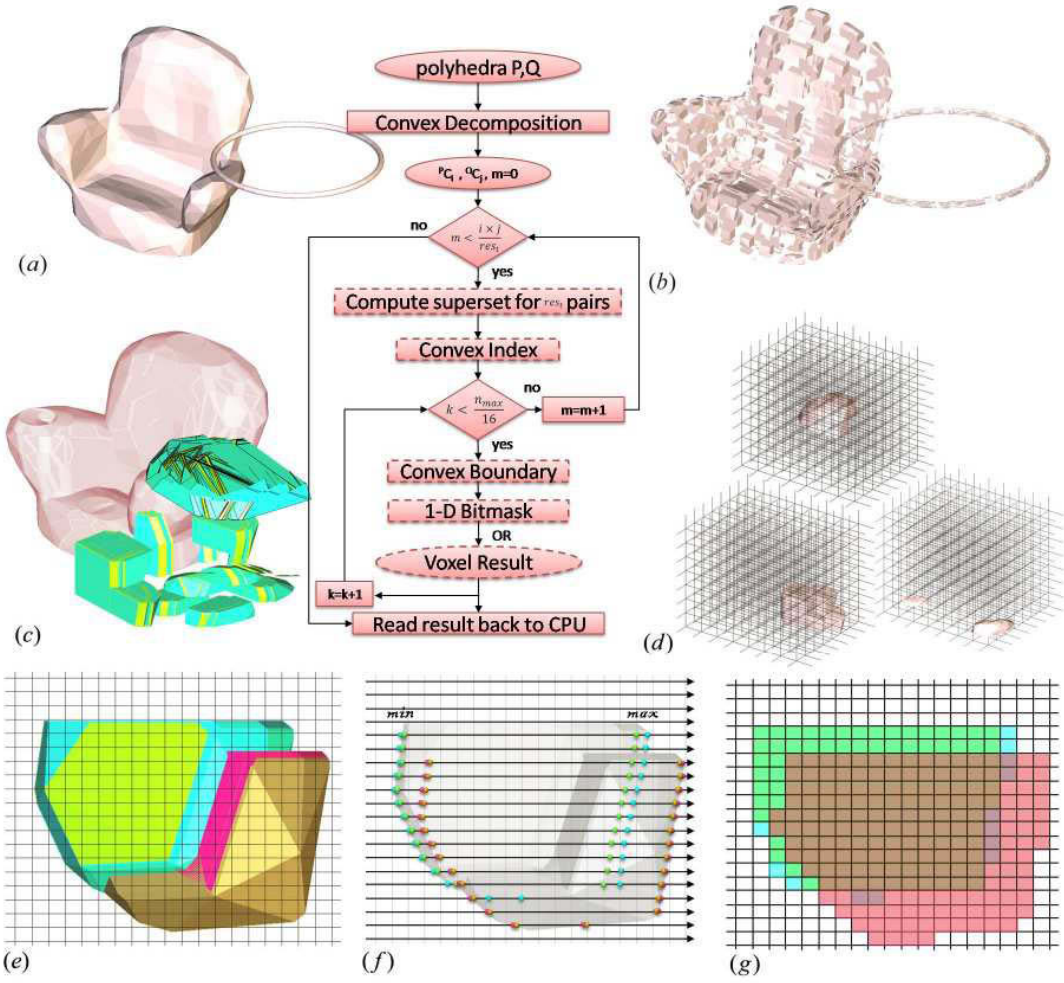

Fig. 1: Algorithm flow chart for computing Minkowski sum in voxels. Dotted boxes indicate sections that can run on GPU. (a) Giving an example of Armchair⊕Torus in triangular meshes; (b) Decompose two meshes, either exactly or approximately, into two set of convex pieces; (c) Compute the superset of

Minkowski sum of convex components using convolution-based approach; (d) Rasterize the resultant sum components group by group; (e)-(g) Union algorithm on a group of components. (e) Detect which components intersect a ray; (f) Obtain the maximum and minimum depth values on ray for each component (i.e. Convex hull of the Minkowski sum of convex components); (g) Voxels are set to '1' in the range [min,max], and then united by binary operator OR.

facet-facet intersections within the convolution. Then he grouped the facets into small regions and filtered the inside using a collision detector. Recently, Campen et al. [6] presented a technique for an efficient extraction of boundary from sweep operations including Minkowski sum. Approaches with approximated representation were also explored [16, 17]. However, convolution-based methods include difficulties in removing facets inside the Minkowski sum boundary and maintaining inner voids.

Our algorithm in this paper appears to be similar to Varadhan-Manocha's approach. However, there are crucial differences between our approach and theirs. We employ uniform voxel grids instead of distance field to represent union results. Our union algorithm is designed for fully utilizing the computational power and parallelism of modern graphics hardware. Moreover, we propose to decompose a complex non-convex model into approximated convex pieces, producing fewer components compared with the exact approach.

3 MINKOWSKI SUM COMPUTATION

In this section we introduce a robust algorithm for generating voxelized Minkowski sum with enclosed void regions preserved. Preserving the region is important in some applications like CAD, collision detection and medical imaging.

3.1 Algorithm Overview

Fig.1 shows the flow of our work. The algorithm starts with decomposing two input polygonal meshes into convex pieces either exactly or approximately. If the approximate approach is sufficiently accurate, lesser number of components can be produced and processed; thus it is able to reduce the overall computational complexity. After making all convex pieces ready the pairwise Minkowski sums are performed. It is well known that a common method for computing Minkowski sum of two convex polyhedral is to calculate the convex hull of all vector sums of their vertex pairs. In this stage, we adopt a similar approach but work completely on GPU. The approach first computes the supersets of pairwise Minkowski sums using a parallel convolution based method and then extracts their boundaries in image space representation. To handle large number of supersets, we develop a GPU-based streaming algorithm to perform super-union for Minkowski sum. The idea is like this: loading the supersets group by group; in which all supersets in one group are assigned an index and being rasterized to obtain the corresponding maximal and minimal samples at each pixel location, then fill up the voxels within the max-min distance; finally using binary operator OR to accumulate the voxel results from group, one by one. Notice that the maximal and minimal samples are just a variation of the boundary of a superset; as a result it is able to simplify the computation of pairwise Minkowski sum and fit well on GPU architecture.

3.2 Convex Decomposition

The problem of decomposing a polyhedron into a minimum number of pieces is known to be NP-hard. Chazelle [7] proposed a basic decomposition method which generates $O(r^2)$ convex pieces in $O(nr^3)$ time for an input model with complexity n and reflex edges r. However, constructing an exact

decomposition can be expensive in processing time and result in an extremely large number of components. Computing Minkowski sum with these components is not practical at all. Therefore, we prefer using an approximate approach to partition complex models into few components. In addition, we found that detail features are usually eliminated in Minkowski sum — that means an approximated decomposition is sufficient to some models (see Fig.2 and Fig.5).

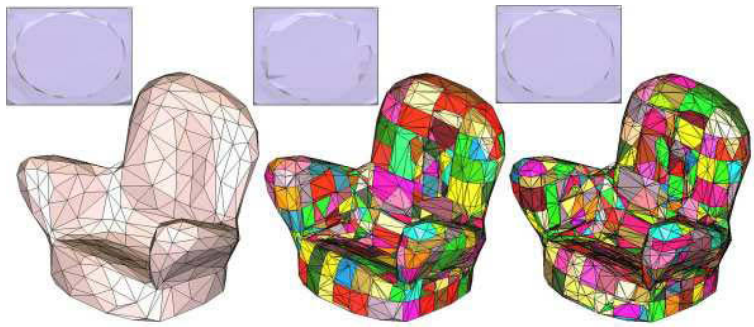

Fig. 2: A convex decomposition result in different degrees of simplification. Left: original model, Middle & Right: simplified model consists of 656 and 1055 components (i.e. Vol. diff: + 0.5% and 0%) respectively. The zoom- in view shows that there is no significant difference between the original and simplified model.

In this paper we employ the approach of Huang and Wang [14], which is a general solid simplification algorithm that can produce conservative bounding of simplified models with watertight and intersection- free guaranteed. Combined with the voxelization process in a conservative manner, the final result must be conservative as well. Note that any other exact or approximate convex decomposition approaches can be used (conservativeness may not guaranteed) as long as they generate convex pieces (e.g.[19]).

3.3 Pairwise Minkowski Sum

Given two polyhedra P and Q which consist of m and n convex pieces respectively, the pairwise Minkowski sum generates $m \times n$ pairs, and we denote them as M_{ij}. It is well known that the Minkowski sum of two convex polyhedra C_P^i and C_Q^j is a convex polyhedron, and M_{ij} can be obtained by computing the convex hull of the vector sum of vertices in C_P^i and C_Q^j. Although this standard convex hull algorithm is simple to be implemented, it is time consuming and cannot make full use of GPU parallelism. A more efficient algorithm is to use convolution to compute M_{ij}[11]. However, the constant factors in the time complexity can be high and it is non- trivial to implement them robustly. Until recently, Li and McMains [16] provided a convenient way to perform convolution on GPU. Here we combine their work with our culling technique to robustly generate a complete Minkowski sum boundary of M_{ij} on GPU, which is applied to the following steps:

- For each M_{ij}, compute a superset S_{ij} of the Minkowski sum boundary of C_P^i and C_Q^j, where M_{ij} is a subset of S_{ij}. The S_{ij} is generated by translating triangles in C_P^i by vectors in C_Q^j (and vice versa) and sweeping edges in C_P^i along edges in C_Q^j. The facets of S_{ij} have to fulfill several propositions described in [15].
- Rasterize the facets and obtain the intersecting points of Minkowski sum boundary M_{ij} per ray.

The second step is introduced in next section since the rest of implementation is based on rendering S_{ij}. The $\cup S_{ij}$ is written to the Vertex Buffer Object (VBO) as the intermediate result and is

directly rendered using OpenGL. Note that any other approaches (e.g. [3, 11]) that can compute the Minkowski sum boundary of two convex polyhedra are also acceptable in our union method.

3.4 Super-union in Image Space

In this section, we present our GPU-based algorithm for merging massive convex objects, which is usually the bottleneck of the decomposition-based approach. Most existing exact union approaches cannot handle a large number of objects robustly and efficiently. This limitation prevents their usage in practical applications. Our streaming algorithm here has the benefit of accepting void-enclosed polyhedra and producing voxelized void-enclosed polyhedra, where each unit can unify at most 1024 components simultaneously. Despite the lack of accuracy, we achieve a good trade-off between complexity and quality of results. Our approach consists of the following steps as a main GPU-based kernel for one unit (see Fig. 1(e)-(g)):

1. Convex Indexing: assign an index $i \times j$ for each superset S_{ij}.
2. Compute Convex Boundary: render all S_{ij} and generate the fragments of boundary of M_{ij} with MAX/MIN blending operator.
3. Construct 1-D bitmask: assign a column of voxel for each fragments of M_{ij} and perform the binary operation OR.

3.4.1 Convex Indexing

After the superset of pairwise Minkowski sum is done, $\cup S_{ij}$ is written to VBO, which can be directly rendered using OpenGL. As the first step of the union, we index each S_{ij} with $i \times j$ so that after rendering it is easy to find which S_{ij} the ray is intersecting. For implementation, we simply put $\cup S_{ij}$ into one single VBO and then render them separately with an array of indices. The array is constructed with size $i \times j$ and stores the total number of triangles of each S_{ij}. Through the rasterization process of VBO, the index of S_{ij} is voxelized into $res \times res \times res_t$ grids. This indicates the existence of S_{ij} per ray. Note that res_t is not necessary equal to res as res_t refers to the maximum unit size of S_{ij}. If we set $res_t =$ 1024, it means that we allow at most 1024 components can be found on a ray for each rendering. A 3D texture, denoted as T_1, is used to store the voxelization results.

3.4.2 Computation of Convex Boundary

The goal of this step is to compute the boundary of S_{ij} along one direction. Once the VBO is rendered, every fragment of S_{ij} can be located according to its order in T_1 and the corresponding index $i \times j$. What we already know is that every S_{ij} is convex; correspondingly there are two outermost fragments for each S_{ij} on their intersecting ray. In our implementation, we reserve two channels (odd and even) in texture for S_{ij} on their intersecting ray to store the maximum and minimum values. To compute the minimum/maximum depth values, we assign each incoming depth value d_f to both channels as follows:

$$\begin{cases} p_i = \max(1 - d_f) \\ p_{i+1} = \max(d_f) \end{cases} \tag{3.1}$$

With the aid of the Max blending function in OpenGL, we can get the fragments on the boundary of S_{ij} with values stored in another texture which is denoted as T_2.

3.4.3 Construction of 1-D bitmask

Current graphics hardware supports at most 32-channels for blending in one single pass. In other words, only 16 convex boundaries (M_{ij}) on rays can be collected simultaneously. By rendering a

rectangle primitive with size $res \times res$, the union of these 16 components (32 values) in voxel on each ray can be obtained easily. We simply fill the voxels within the range of two channels, p_i and p_{i+1}, and then integrate them with others using the OR operator.

To generate a united voxelization of res_t components, our algorithm may require iteratively running $res_t/16$ times of the steps of Computation of Convex Boundary and Construction of 1-D bitmask. But this rarely happens since not all res_t components intersecting the same ray. We now focus on merging a large number of S_{ij}. If the convex decomposition produces an incredibly large number of components, one possible solution is to subdivide the problem into subproblems until the subproblems can be dealt with. In our work, we partition the components into a set of groups. Each group (unit) consists of at most 1024 components that are possible to be applied in the step of Convex Indexing and be rendered multiple times in the other two sections. A larger size of the group can reduce the transition time between the CPU and the GPU, but at the cost of the memory space. Moreover, since each pairwise Minkowski sum S_{ij} can be completely voxelized and merged independently, we do not have to store all primitives into GPU. Instead, only one 3D texture with size $res \times res \times res$ storing the voxelization result is necessary to be kept throughout the whole process. The union kernel is easy and simple to be implemented since there is no complicated testing involved. The pseudocode for our framework is given in Algorithm 1.

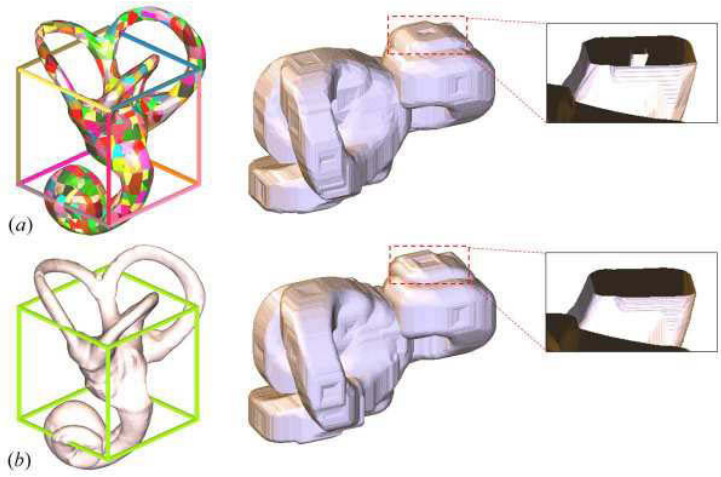

Fig. 3: Minkowski sum that (a) having voids and (b) without voids. From left to right: the input models, outer view of Minkowski sum, (zoom in) inner view of Minkowski sum. This figure shows that voids are usually ignored but our approach can generate a correct boundary of the Minkowski sum.

4 EXPERIMENTAL RESULTS AND DISCUSSION

In this section we show the experimental results of our Minkowski sum algorithm. All tests were carried out on a PC with Intel Core i5 CPU 750 2.67GHz + 4GB RAM and GeForce GTX580 graphics card. Tab. 2 gives the solid voxelization timing of Fig.3 and 4 under two different resolutions. Our implementation runs on both CPU and GPU — the CPU is used to decompose the polyhedra and compute superset of pairwise Minkowski sum while GPU is used to perform union continuously. The superset is also possible to be computed on GPU as [16] provided a detailed description of how to efficiently generate a superset on CUDA.

Fig. 3 shows that the Minkowski sum results can be look alike from the outside but completely different on the inside. Enclosed voids are usually ignored by some previous work for ease of

computation. For example, Li- McMain 's approach [16], to the best of our knowledge, is the fastest way to compute the Minkowski sum of two arbitrary polyhedra. However, their approach can't handle enclosed voids. See the cases of Void⊕Sphere and Grate2⊕Frame as shown in Fig. 4, which either input models with enclosed void or yield voxelized Minkowski sum with enclosed voids. These two cases are not applicable to the outer- boundary- only approaches. Moreover, our method is able to run out- of- core, which means polyhedra with large number of convex pieces can also be handled in limited memory space. This is comparatively more practical since the workload is dividable and the main core part is able to be parallel.

Concerning the ways to achieve the best performance, determining the size of group is important. Two factors need to be considered for choosing an appropriate value for res_t: the maximum number of components per ray and the computing power of GPU. From Tab.1 we can see that if res_t is large, the number of components per ray (n_{max}) will be large too, meaning that more looping is required in the parallel union kernel. If res_t is small, the number of components per ray will be small; this requires more looping in creating VBO. Based on the experimental result, we found that if the value of res_t can control the maximum number of components on rays (n_{max}) to be within 14- 30, the algorithm would yield the best result. Small number is preferred because the parallel power is constrained by the limited color channels for blending. In addition, since a high resolution will reduce the parallel power while a large number of components in sequential streaming will degrade the performance, a medium size of res_t is a better choice.

In addition, the level of the approximation involved in our algorithm is determined by two parameters. A typical one is the resolution of voxelization (res). Since voxelization is a sampling process, the result is always discretized and loses continuous information. Another parameter is the volume difference (Vol. diff) between the approximated result and the original model in convex decomposition. The approximation provides a result of ignoring certain features of the original model. In the case of fig. 2, although the edge of the decomposed chair at the bottom is rougher than the input one, it still provides a similar result in Minkowski sum with better runtime performance achieved (fig.5).

4.1 SPEEDUP

To achieve faster performance, we may consider multi- GPU acceleration, which allows us to speed up because our algorithm involves a large number of independent computations. We have previously shown computations run in a reasonable amount of time using a single GPU. Hence, our method with the help of GPUs, is able to solve computationally expensive problems quickly. In addition, our implementation does not necessarily require a particular advanced graphics card or a large amount of memory as our algorithm is streaming- based; therefore, the cost of speedup is relatively small.

5 CONCLUSION

In this paper, we present a practical algorithm to compute the voxelized 3D Minkowski sum of two arbitrary polyhedra on Graphics Processing Unit (GPU) efficiently and robustly. Our algorithm employs a BSP tree to decompose polyhedra into convex pieces conservatively. Then we perform pairwise Minkowski sum by generating the superset and extracting the boundary in image representation. The image representation are then converted to solid voxels and the final result is merged by logic operation OR. Our algorithm is out- of- core and parallel. Nevertheless we guarantee that the voxelization of Minkowski sum is conservative and the inner voids are well preserved. The experimental result shows that our algorithm can handle a massive number of unions efficiently and takes advantage of the parallel computation power of GPU.

res_t	*Avg. of* n_{max}	*Avg. of VBO*	*GPU time*	
			512	*1024*
12	11.4	31.0K	3.52	8.74
24	21.6	62.1K	3.14	8.77
48	39.8	124.2K	3.45	9.11
96	66.4	248.4K	4.04	11.7
192	96.2	496.7K	4.77	N/A

Tab. 1: Performance of different res_t on Bunny⊕Sphere in second.

A⊕B	*Convex Pieces (i×j)*	*CPU Time*	*GPU Time*			
			512	res_t	*1024*	res_t
Void⊕Sphere	965	3.24	2.65	36	7.3	36
Ear⊕Frame	14,424	9.62	12.5	96	38.1	96
Grate2⊕Frame	696	0.46	3.20	48	10.5	24
Grate2⊕Grate	1,624	1.01	7.28	56	20.7	28
Bunny⊕Sphere	960	6.38	3.14	24	8.77	24
Armchair⊕Torus	746,940	152.7	378.2	180	1295.3	200
Armchair_Simp⊕Torus	464,448	118.7	301.9	160	1060.2	180

Tab. 2: Timing for computing voxelized Minkowski sum from convex decomposed models. CPU time (in second) includes all the computation time for generate supersets of pairwise Minkowski sums and the transmission time for uploading VBO to GPU. GPU time (in second) includes all the computation time of union.

	Trgl.	*Num. Piece.*	*Volume. Diff*
Void	13.3K	965	0%
Ear	24.2K	1202	+ 5%
Grate2	0.67K	58	0%
Torus	11.3K	708	0%
Bunny	39.1K	960	+ 2%
Armchair	14.0K	1055	0%
Armchair_Simp	10.4K	656	+ 0.5%
Sphere	0.82K	1	0%
Frame	0.14K	12	0%
Grate	0.34K	28	0%

Tab. 3: Convex decomposition of input models. For each model, from left to right: the face number, the number of pieces and the volume difference of the approximated result and the original model. 0% indicates that an exact decomposition is performed.

* N/A indicates that the computation is out of memory.

Algorithm 1 *Tile-based Union in Image Space*

Input: Pairwise Minkowski sum $\rightarrow S_{m\times n}$
Output:T_3

1: for $i = 1 \longrightarrow (m \times n)/res_t$ do
2: $vbo \longleftarrow$ gl.UnloadToVBO ($S_{(i-1)\times res_t}$ *to* $S_{i\times res_t}$);
3: glEnable(or);
4: *//Rendering vbo with index by shader*
5: for all fragments f_k do
6: $z \Leftarrow$ GetVoxelPosition (IndexOf(f_k));
7: $T_1 \Leftarrow$ Update voxel in position z ;
8: end for
9: glDisable(or);
10: *//Rendering a quad by shader*
11: $n_{max} \longleftarrow$ GetMaxIntersectMKS(T_1);
12: for $j = 0 \longrightarrow ceil(n_{max})/16$ do
13: glEnable(BLEND);
14: glBlendEquation(MAX);
15: *//Rendering vbo with index by shader*
16: for all fragments f_k do
17: $p \Leftarrow$ GetIntersectMKSpos(IndexOf(f_k));
18: $T_2(x, y, p) \Leftarrow 1 - depth(f_k)$;
19: $T_2(x, y, p + 1) \Leftarrow depth(f_k)$;
20: end for
21: glDisable(BLEND);
22: glEnable(or);
23: *//Rendering a quad by shader*
24: $T_3 \Longleftarrow$ VoxelRange($T_2(p), T_2(p + 1)$);
25: glDisable(or);
26: end for
27: $vbo \longrightarrow$ gl.ClearVBO();
28: end for

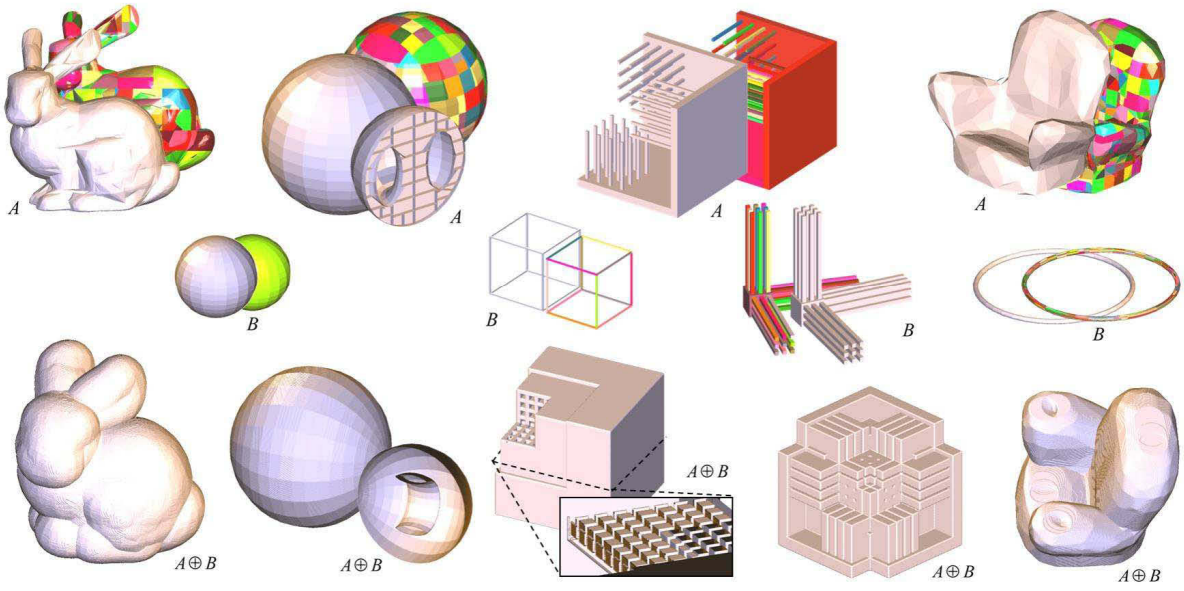

Fig. 4: Voxelized Minkowski sum results of five pairs of models in resolution 512 × 512 × 512. From left to right: Bunny⊕Sphere, Void⊕Sphere, Grate2⊕Frame, Grate2⊕Grate, Armchair⊕Torus.

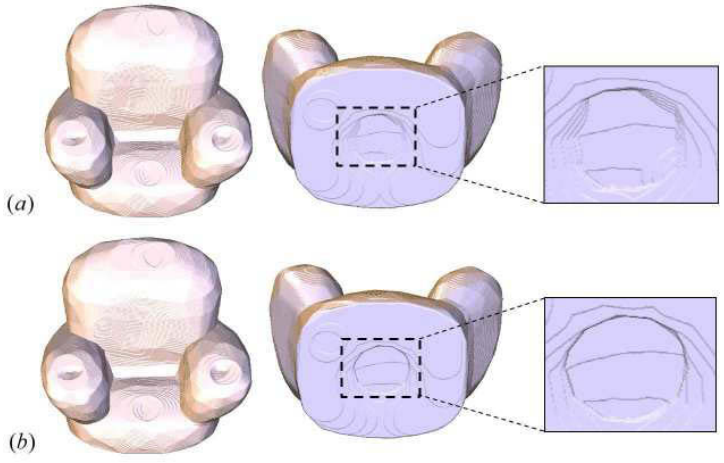

Fig. 5: A comparison of voxelized Minkowski sum results of Armchair⊕Torus using (a) approximating decomposition and (b) exact decomposition approaches in resolution of 512 × 512 × 512. The result shows no significant difference in appearance but great improvement in the runtime performance.

ACKNOWLEDGEMENT

The third author acknowledges the support of National Science Foundation grant CMMI- 0927397.

REFERENCES

[1] Agarwal, P.; Flato, E.; Halperin, D.: Polygon decomposition for efficient construction of Minkowski sums, In M. Paterson (Ed.), Algorithms –ESA 2000, volume 1879 of Lecture Notes in Computer Science, 20- 31, Springer Berlin/Heidelberg, 2000.

[2] Aronov, B.; Sharir, M.; Tagansky, B.: The union of convex polyhedra in three dimensions, SIAM J. Computing, 26, 1670- 1688, 1997.

[3] Barki, H.; Denis, F.; Dupont, F.: Contributing vertices- based Minkowski sum computation of convex polyhedra, Computer- Aided Design, 41, 525- 538, 2009.

[4] Barki, H.; Denis, F.; Dupont, F.: Contributing vertices- based Minkowski sum of a nonconvex-convex pair of polyhedra, ACM Transactions on Graphics, 30, 3:1- 3:16, 2011.

[5] Bekker, H.; Roerdink, J.B.T.M.: An efficient algorithm to calculate the Minkowski sum of convex 3d polyhedral, 619- 628, 2001.

[6] Campen, M.; Kobbelt, L.: Polygonal boundary evaluation of Minkowski sums and swept volumes, Computer Graphics Forum, 29(5), 1613- 1622, 2010.

[7] Chazelle, B.M.: Convex decompositions of polyhedra, In Proceedings of the thirteenth annual ACMsymposium on Theory of computing (STOC '81).ACM, New York, NY, USA, 70- 79, 1981.

[8] Ehmann, S.A.; Lin, M.C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition, Computer Graphics Forum, 500- 510, 2001.

[9] Fogel, E.; Halperin, D.: Exact and efficient construction of Minkowski sums of convex polyhedra with applications, Computer- Aided Design, 39, 929- 940, 2007.

[10] Ghosh, P.K.: A unified computational framework for Minkowski operations, Computers & Graphics, 17(4), 357- 378, 1993.

[11] Guibas, L.; Seidel, R.: Computing convolutions by reciprocal search, In Proceedings of second annual symposium on Computational geometry, SCG'86 New York, NY, USA: ACM, 90- 99, 1986.

[12] Hachenberger, P.: Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces, Algorithmica, 55, 329- 345, 2009.

[13] Halperin, D.: Robust geometric computing in motion, I.J. Robotic Res., 21(3), 219- 232, 2002.

[14] Huang, P.; Wang, C.C.L.: Volume and complexity bounded simplification of solid model represented by binary space partition. In Proceedings of 14th ACM Symposium on Solid and Physical Modeling, SPM'10 New York, NY, USA: ACM, 177- 182, 2010.

[15] Kaul, A.; Rossignac, J.R.: Solid- interpolating deformations: Construction and animation of pips, Computers & Graphics, 16(1), 107- 115, 1992.

[16] Li, W.; McMains, S.: A gpu- based voxelization approach to 3d Minkowski sum computation, In Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, SPM'10 New York, NY, USA: ACM, 31- 40, 2010.

[17] Lien, J.M.: Point- based Minkowski sum boundary, In Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG '07), IEEE Computer Society, Washington, DC, USA, 261- 270, 2007.

[18] Lien, J.M.: A simple method for computing Minkowski sum boundary in 3d using collision detection, In Proceedings of WAFR' 08, 401- 415, 2008.

[19] Lien, J.M.; Amato, N.M.: Approximate convex decomposition of polyhedra and its applications, Computer- Aided Geometric Design, 25, 503- 522, 2008.

[20] Peternell, M.; Steiner, T.: Minkowski sum boundary surfaces of 3d- objects, Graphical Models, 69, 180- 190, 2007.

[21] Evans, R.C.; O'Connor, M.A.; Rossignac, J.R.: Construction of Minkowski sums and derivatives morphological combinations of arbitrary polyhedra in CAD/CAM systems, US Patent 5159512, 1992.

[22] The CGAL Project: CGAL User and Reference Manual, CGAL Editorial Board, 3.9 Edition, http://www.cgal.org/, 2011.

[23] Varadhan, G.; Manocha, D.: Accurate Minkowski sum approximation of polyhedral models, Graphical Models, 68, 343-355, 2006.

[24] Wein, R.: Exact and efficient construction of planar Minkowski sums using the convolution method, In Proceedings of the 14th conference on Annual European Symposium - Volume 14(ESA'06), Yossi Azar and Thomas Erlebach (Eds.), Vol. 14, Springer-Verlag, London, UK, 829-840, 2006.

[25] de Berg, M.; Cheong, O.; van Kreveld, M.; Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag, Berlin, 1997.