A Real Time Visual Boolean Operation on Triangular Mesh Models

Shouxin Chen[1], Ming Chen[2] and Shenglian Lu[3]

[1]School of Computer and engineering, Guangxi Normal University, chen.csx@outlook.com
[2]School of Computer and engineering, Guangxi Normal University, hustcm@hotmail.com
[3]School of Computer and engineering, Guangxi Normal University, lsl@gxnu.edu.cn

Corresponding author: Ming Chen, hustcm@hotmail.com

Abstract. In the product design stage, it is necessary to repeatedly perform Boolean operations to inspect whether the shapes of design products meet the requirements or not. When the involved models are complex triangular mesh models, the existing Boolean operations be performed in real time, which is important for trial and error at the product design phase. This paper uses the ray tracing technique to very quickly obtain sampled ray segments and perform Boolean operations between the sampled intervals, converting 3D model Boolean operations on mesh models into one-dimensional Boolean operations on 1D intervals. The 1D Boolean operation result is finally rendered as an image to obtain the visual resultant effect of Boolean operation on triangular mesh models. The test results have shown that the proposed method is much faster than the commercial CAD package, i.e., Rhino and the state-of-the-art method based on the LDI (Layered depth image) or LDNI (Layered depth-normal image), and can obtain the visual results of Boolean operation in real-time and have a potential application in simulation and CAD/CAM.

Keywords: Boolean operation, Ray tracing, Real time Boolean operation, constructive solid modeling, geometric modeling
DOI: https://doi.org/10.14733/cadaps.2022.470-480

1 INTRODUCTION

Boolean operation plays one important role in constructive solid modeling (CSG), which constructs complex models through a series of Boolean operations on simple primitives, which are usually represented as triangular mesh models. With the rapid development of 3D printing technology in recent years, manufacturing complex shaped models has become a reality. To model such complex shapes during the product design phase, Boolean operations will be repeatedly done on primitives and the results are preferred to be inspected in real time such that one can judge whether the involved primitives need be amended or not.

In some applications, due to the large number of models and the huge scale of the models, current techniques cannot support real-time Boolean operations on dense mesh models of millions of triangles, thereby limiting product design efficiency. To solve this problem, one ray tracing-based method is proposed in the paper to sample 3D models as depth-orderly interval models efficiently, and transform 3D Boolean operation as 1D one. The results of 1D Boolean operation are rendered by image rendering technique. The above steps are defined as "Visual Boolean Operation", short for VBO. In this field, the most common method is using Surfel rendering [1] and layered depth image, i.e., LDI [17] technique, both of which sample the involved models as points with normal information. As graphic cards developed rapidly recently and ray tracing can be very efficiently performed [14] [21], the paper makes full use of the advantage to sample 3D triangular mesh models into ordered intervals and perform 1D Boolean operation parallelly. The major difference between the method proposed in the paper and the above two methods lie in the sampling step. The above two methods usually need much more time for dense mesh models, but in the proposed method, the sampling step can be significantly speeded up, achieving a real-time VBO result.

2 RELATED WORK

Boolean operation is heavily studied and the efficiency is a key bottleneck. The review in the paper focuses on how to improve the efficiency of Boolean operations. Boolean operations can be divided into approximate Boolean operations and exact Boolean operations. The boundary presentation (B-rep) is the most commonly used for exact Boolean operations, in which the intersection calculation of the primitives and the extraction of the intersection loops takes much time, which is unacceptable at the design phase. In order to improve the efficiency, two methods, i.e., bounding volumes and spatial partitioning, are applied. The bounding volumes methods include axis-aligned bounding box (AABB) [2], oriented bounding box (OBB) [4], K-Dop [13], and bounding sphere [11]. For example, Qin *et al*. [16] used axis-aligned bounding box binary tree to accelerate the triangle-triangle intersection test. The collision detection among the models can be quickly completed by the bounding volumes methods. But a more accurate intersection test should be done to extract intersection loops. With the increase of the complexity of models, the detection efficiency of the bounding volumes method will decrease significantly. The spatial partitioning methods include octree[10, 15], BSP tree[3], kd-tree[8], and uniform grid[9]. The most popular method among them is the octree. The octree recursively divides the space into 8 subspace cubes, which can quickly locate the intersecting triangles among the mesh models. Douze *et al*.'s QuickCSG [8] uses the *k-d* tree index to accelerate and implements very high-speed Boolean operations between mesh models through parallel computing. Zhou [24] used winding numbers and combined BSP tree space division to propose a new Boolean operation method. This method performs Boolean operations on multiple models at the same time and eliminate self-intersections. Campen and Kobbelt [5] proposed a hybrid method that divides the BSP trees in the octant of the octree, which balanced the efficiency and memory consumption. In comparison to the bounding volumes method, space partitioning avoids unnecessary intersection tests and improves the overall efficiency of Boolean operations.

Approximate Boolean operation is based on volumetric representation. More explicitly, the mesh model is converted to a voxel model and then the Boolean operation is performed. After the Boolean operation is completed, the result is converted back to a mesh model. Via this method, the robustness has been improved. However, the accuracy of the method depends heavily on the resolution of the voxelization. The geometric details of the mesh models are inevitably lost in the converting procedure, especially in the regions where the intersections occur. For more accuracy, with the increasement of voxel resolution, the memory consumption will increase exponentially. Vardhan *et al*. [18] uses adaptive segmentation techniques to reduce the memory cost. Pavić *et al*. [15] performs Boolean operations by adopting polygonal and voxel hybrid presentation. This method uses an adaptive octree to generate voxels by surface-volume conversion only in the intersection areas of models. The amount of data will be reduced a lot compared with the method of volumetric representations. For non-intersected areas, B-rep is used so that the resultant mesh retains the original input geometric features of the model.

In addition to the above methods, some scholars also use Ray-rep to accelerate mesh Boolean operations. This type of methods often uses the image space technique of the LDI or layered depth-normal image (LDNI) [6] to accelerate Boolean operations. Chen *et al*. [7] used LDNI and topology information to accelerate the inside/outside classification of the Boolean operation, which greatly improve the efficiency and performance. Wang *et al*. [6] used LDNI to discretize the model, determined the inside and outside of the sampling point cloud to obtain the Boolean operation result, and reconstruct it into a polygonal mesh through the dual contour method [12]. Wang's mesh/LDI-hybrid representation [20] is an approximate Boolean operation that can keep the model features of the non-intersected areas. Another method proposed by Wang employs LDI to sample the model, and performs a membership classification based on the ray sampling results. The intersecting area is reconstructed in a voxel-based manner with the help of an octree to generate a new mesh model. Similar to this, Zhao *et al*. [25] adopted the compact LDI (CLDI) for ray sampling, and its memory consumption is much reduced compared with LDI and LDNI.

3 METHOD OUTLINE

The proposed algorithm can be outlined as three steps in Figure 1: 1) first, emit multiple rays from the pixel center of the current view plane in an orthogonal projection manner and sample the involved mesh models (see Figure 1(a)) as the ray interval models (see Figure 1(b)); 2) perform a one-dimensional Boolean operation on the ray interval models to obtain the resultant interval model after 1D Boolean operations (see Figure 1.(c)); 3) render the kept points of Figure 1(c) and use the specified lighting environment and materials to render the result of Figure 1(c) as an image, representing the visual result of Boolean operation on mesh models of Figure 1(a). once the viewpoint changes or either of the involved model is modified, steps 1 to 3 will be re-executed.

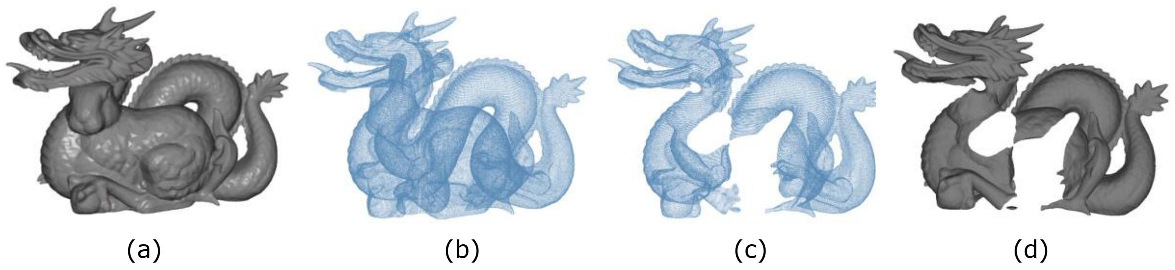

Figure 1: The steps of the proposed method: (a) The input triangular mesh models for a "difference" Boolean operation (Dragon-Bunny), (b) Sample the input models as ray interval models by ray tracing, (c) Perform one-dimensional Boolean operation on the ray interval models, (d) Render the result of (c) as an image, which outputs the visual effect of the Boolean operation result of (a).

Note that the sampling procedure is performed using ray tracing technique, which can be very quickly performed by ray-tracing engine. In the sampling step, we do not use the octree method to recursively spatially split models as ray-tracing engines have implemented this acceleration data structure. The sampling resolution is set to be the current viewing window pixel resolution.

4 VISUAL BOOLEAN OPERATION

As mentioned before, the whole procedure of the proposed method consists of three steps, i.e., ray tracing sampling, 1D Boolean operation on ray-segment models and image rendering. Next come the detailed introductions of the three steps.

4.1 Ray Tracing Sampling

Different from the method of LDI [7, 17, 21, 23] or LDNI [6, 20, 24] , in which rasterized rendering for ray sampling is used, this paper uses a ray tracing method to sample models as ray intervals, which is much efficient. The involved models' bounding boxes will be first calculated and the union region of their occupied area will be regarded as region of interests (ROI). For the ROI, a series of parallel rays will be emitted from the center of pixels of the screen and the hit points of the rays and the models are regarded as the sampling points, which are ordered by their depth values *t*. Figure 2 shows one YZ-plane-sampling slice on two models denoted as M_a and M_b when rays shoot along the x axis.

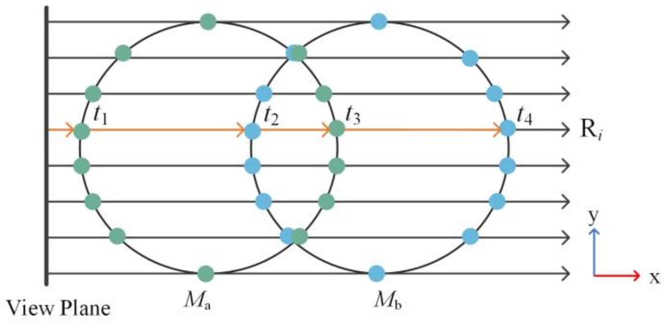

Figure 2: Ray sampling on the YZ plane along the X axis.

For a NVIDIA graphics card with RTX, a programmable ray tracing pipeline can perform ray tracing via three programs, i.e., *ray generation*, *closest-hit*, and *miss-hit*. *Ray generation* initializes one ray by *Ray* (*orig*, *dir*), where *orig* and *dir* represent the ray tracing start point and the direction of the tracing a ray, respectively. One sampling ray corresponds to one program fragment, which can be executed parallelly and efficiently. When one ray hits an intersection point, the *closest-hit* program fragment will be automatically called and the intersection point coordinate can be calculated in *closest-hit* program. The RTX's built-in ray triangle intersection program will quickly evaluate the barycentric coordinate (u, v) of the intersection point in the triangle hit by the ray, and its corresponding cartesian coordinate P can be calculated by Eq. (1) as below:

$$P = uP_1 + vP_2 + (1 - u - v)P_3 \tag{1}$$

where P_1, P_2 and P_3 are the coordinates of the vertices of the triangle hit by the ray. when a ray has no intersection points within the specified ray tracing range t, *miss-hit* is called to terminate the sampling procedure.

The sampling process is shown in Figure 3. After ray generation is executed, all rays will be traced. The acceleration structure will be traversed to determine whether a triangle intersects any ray. Within the specified ray tracing range [$t_{\min}$, $t_{\max}$], once the intersection occurs, *closest-hit* will be carried out at the intersection point closest to *orig*. Next, update the ray start point *orig* to be the intersection point and continue the ray tracing along the direction *dir,* the emitting direction of the rays. Actually, a preset small positive offset ε should be added to the hit point for the next ray tracing to avoid a dead loop. As shown in Figure 4, R_i should hit the triangle T_j at point P, but the obtained point may be P_1 out of floating-point computing errors. Once P_1 is behind P and updated as the new ray tracing start point, R_i will hit T_i again, resulting in a dead loop. Thus, a small number ε ($\varepsilon = 10^{-5}$ in the paper) is added to P obtaining a new point P_2 , and P_2 will be used as a new starting point for ray tracing. The value of the ray tracing range is important: an unreasonable range will result in incomplete sampling or ray tracing failure. In Figure 2, the ray R_i intersects the two mesh models M_a and M_b at t_1, t_3, and t_2, t_4, respectively; if the range [$t_{\min}$, $t_{\max}$] is set to be [t_2, t_4], R_i can only obtain the sampling points at t_2, t_3, and t_4 and the sampling point at t_1 will be lost as t_1 is out of the range of [t_2, t_4], resulting in incomplete sampling. In order to obtain a complete sampling, $t_{\max}$ can be easily set as an infinite positive number and one strategy should be devised to determine $t_{\min}$. In the paper, each sampling model's bounding box is first calculated and ensure the value of $t_{\min}$ and $t_{\max}$ are set to be outside of the bounding box, avoiding the incomplete sampling problem. when R_i hit any primitives at t', a positive offset ε will be added to t' and continue ray tracing with a new ray tracing range, i.e., $[t'+\varepsilon, t_{max}]$, preventing the ray from repeatedly sampling at the same position and falling into a dead loop. If no intersection occurs in the range, do *miss-hit* and stop ray tracing. The above sampling can be very quickly performed.

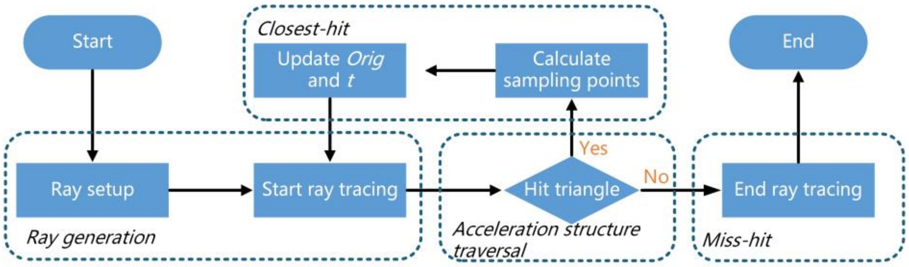

Figure 3: The flowchart of ray sampling by ray tracing.

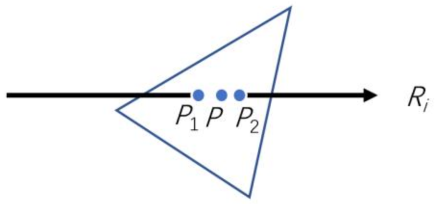

Figure 4: Ray sampling with ε offset.

4.2 1D Boolean Operation on Rays

Ideally, the involved primitives are watertight. In this case, for one ray, entry points and exit points appear alternately and the number of sampling points is even; when isolated edges or holes appear, odd number of sampling points may occur, leading to wrong inside/outside classifications and incorrect results of Boolean operations. The methods in the literatures [6][24] are both applicable to the situation where the number of sampling points on one ray is even. As the built-in Ray-Triangle intersection program of the ray tracing engine returns no intersections, when the emitted rays are coplanar to the sampling primitives and an odd number of sampling points will be generated. If a perspective projection is used, since the rays are not parallel to each other, the probability of the rays being tangent to the model will be much larger than that of orthogonal projection, which is used in the paper.

To conquer the failure of the inside/outside classification caused by the odd number of sampling points, this paper proposes a new scheme to determine the inside/outside classification of sampling points. As shown in Figure 5, it is assumed that the green and blue sampling points represent the sampling points of M_a and M_b, respectively. By judging the angle between the normal of the sampling point and the ray direction, the sampling point's entry/exit status can be defined. When the angle is acute, the point is an exit point and if the angle is obtuse, the point is an entry one. After determining the entry/exit statuses of hit points, the inside/outside classifications of one model's sampling points can be figured out with the help of the entry/exit statuses of their nearest neighboring sampling points of the other model. Take Figure 5 for an example, it is assumed that P_2, P_3 and P_4 belong to M_a and the other points M_b. Point P_2 is one exit point of M_a, thus P_1 is located inside of M_a. Point P_1 is one entry point of M_b, thus P_2, P_3 and P_4 are located inside of M_b. Similarly, P_5 is outside of M_a as

P_4 is the exit point of M_a. According to the rule, all sampling points' inside/outside classification can be easily decided.

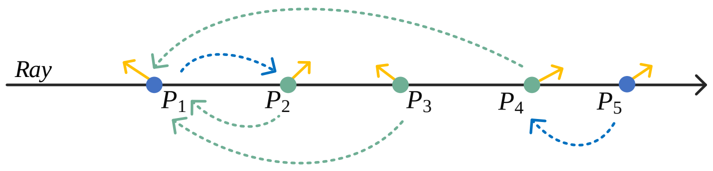

Figure 5: The schematic diagram of inside/outside classification: the yellow arrow stands for the directions of normal vectors of the sampling points and the green and blue represent that they are sampled from different models.

Once the inside/outside step is completed, the sampling points of M_a and M_b can be kept or removed using the rule of Tab.1, and all kept sampling points can be image or Surfel rendered, representing the final visual result of Boolean operation.

Operation	*M_a Kept*	*M_b Kept*
$M_a \cap M_b$	Inside M_b	Inside M_a
$M_a \cup M_b$	Outside M_b	Outside M_a
$M_a - M_b$	Outside M_b	Inside M_a
$M_b - M_a$	Inside M_b	Outside M_a

Table 1: The kept rules of sampling points.

4.3 Image Based Rendering

After 1D Boolean operation, the kept points can be rendered through the Surfel method or the image-based rendering method. The Surfel method converts points to surfel and then renders them through rasterization [18]. Due to the limitation of sampling resolution, low-resolution sampling often needs to interpolate before rendering to obtain the correct intersection area visual effect [23]. In the paper, our focus is how to get the visual result in the least time. Therefore, we directly use ray tracing for image-based rendering and then employed OpenGL to show the rendered image with 2D texture. The image-based rendering method is adopted to render the final results out of its more excellent efficiency.

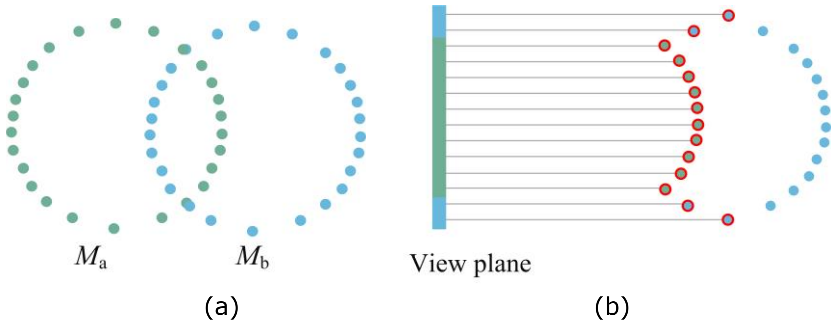

Figure 6: 1D Boolean result of M_b-M_a at one slicing layer: (a)The ray sampling models of M_a and M_b at one slicing layer; (b) 1D Boolean result of M_b-M_a at one slicing layer and only the circled points (the foremost point along each ray) will be rendered as one part of the final rendering image.

5 RESULTS

The method in [6] and the commercial package Rhino are adopted as benchmarks. The test models are all triangular mesh models listed in Tab.2. The tested PC is configured with Intel(R) Xeon(R) Silver 4110 CPU @2.10ghz CPU × 2, 64GB DDR4, NVIDIA Quadro P4000 graphics card and 8GB video memory.

Figure 7 shows the final visual effect (images) by the proposed algorithm. The difference of Dragon and Bunny is performed in Test 1 and the sampling intervals, the result of 1D Boolean operation and the rendered result are shown in the second row of Fig. 6(a); In Test 2 (see Figure 7(a)), a cuboid subtracts 144 Flexes models; One product design in jewelry is presented in Test 3, in which one big ring subtracts 100 small rings and the results are shown in Fig. 6(c). The time consumption of Rhino, the proposed method and the method in [6] are all counted and the statistical data is listed in Table 2 - Table 4. Table 2 lists the triangle numbers of each involved mesh models. From Table 3, we can easily find that it is impossible to obtain a real time result by current commercial CAD package such as Rhino if one mesh model Boolean operation is directly done: in Test 3, it will take 32 seconds for Rhino, which cannot be accepted in product design phase as the Boolean operation will be repeatedly done to inspect the final product appearance and the real-time effect is a must. Thus, quick visual Boolean operation is one complementary step before doing real mesh Boolean operation. In Figure 7(c), the result of VBO can be directly rendered as one image with different materials, i.e., silver, gold, plastic and emerald.

For all the tests, the proposed method can obtain the visual result in the least time: at 512 ×512 resolution, the three tests can be done in 66 ms, 184.5 ms and 154 ms, respectively, which will not be perceptible (regarded as real time). As for the resolution 1024×1024, a high-resolution rendering image can be obtained instantly, i.e., 148.5 ms, 488 ms, 435 ms, for the three tests, respectively. If LDNI [6] is done for visual Boolean operation, the proposed algorithm is much faster, as the sampling time in the paper is much shorter than that of the benchmark in [6]. The sampling time and 1D Boolean operation time by the proposed method and the method in [6] are illustrated in Table 4. Note that once the view window is update after rotation, zoom in/out or pan, the VBO will be executed again, updating the resultant rendered image. As the VBO can be done quickly, the rendered image can be updated in a real time.

Test Mesh Models	*Test 1*		*Test 2*		*Test 3*	
	Dragon	*Bunny*	*Cuboid*	*144Flexes*	*Ring*	*100Rings*
Vertices	50K	3.5K	0.7K	216K	1.2K	123K
Triangles	100k	6.9K	1.2K	265K	2K	209K

Table 2: The tested models' triangle number

Resolution	*Rhino(sec)*	*LDNI (ms)*	*Ray tracing(ms)*
	Test 1/2/3	*Test 1/2/3*	*Test 1/2/3*
128×128	1.3/15/32	399.5/471/408	15.2/24.3/28
256×256		425.0/477/454	28.8/60.8/60
512×512		495.7/525/490	66/184.5/154
1024×1024		664.2/716/706	148.5/488/435

Table 3: The time cost by the benchmarks and the proposed algorithm.

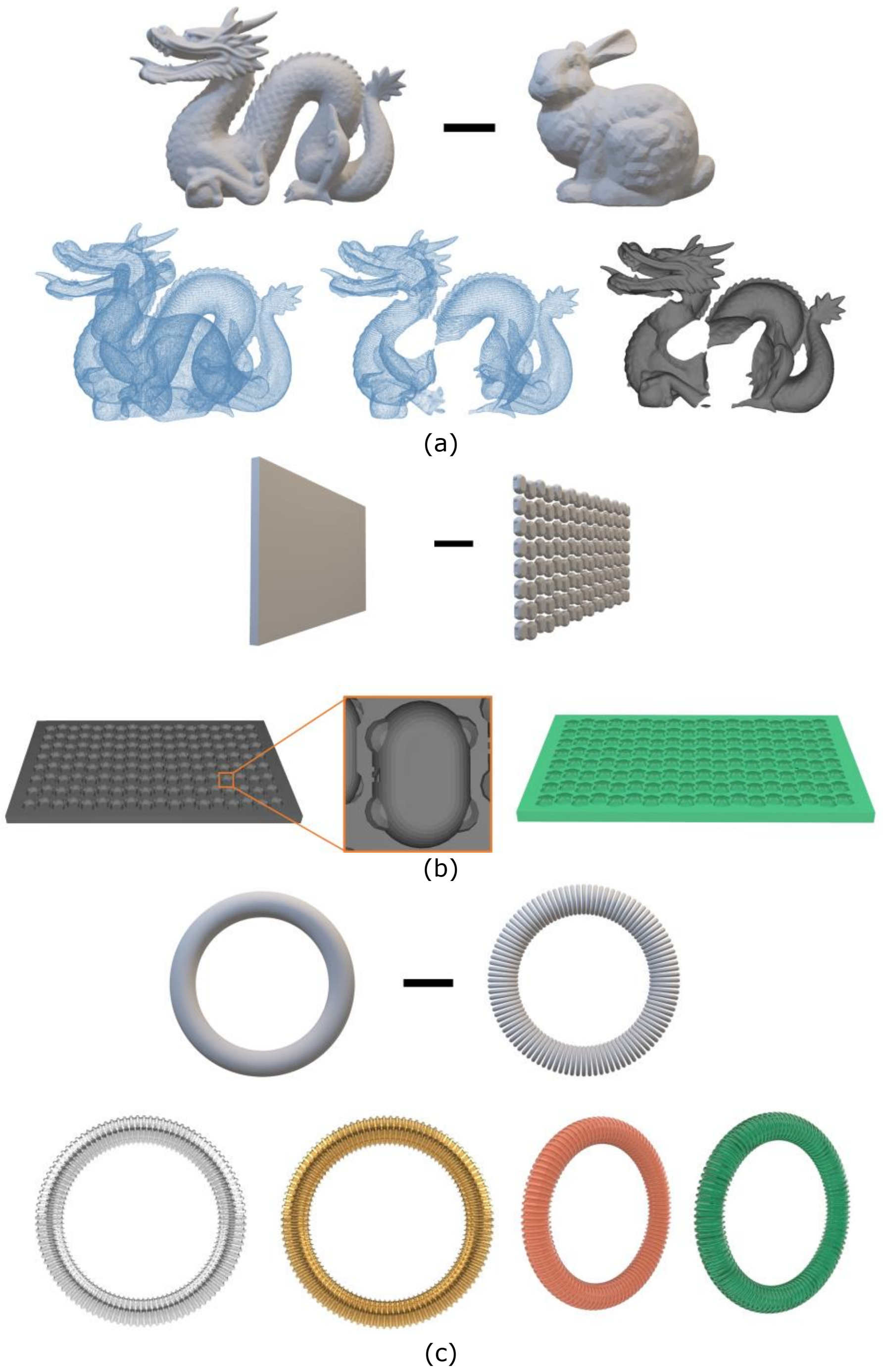

Figure 7: The visual results of VBO by the proposed method: (a) The sampling model and the result of VBO for Test 1; (b) The VBO result of Test 2; (c) The VBO result of Test 3 and it is rendered with different materials including silver, gold, plastic and emerald.

Resolution	*LDNI (ms)*		*Ray tracing(ms)*	
	Sampling	*1D Boolean*	*Sampling*	*1D Boolean*
128×128	398	1.5	15.25	Less than 1
256×256	419.75	5.25	27.75	1
512×512	487.5	8.25	63	3
1024×1024	645	19.25	138	9.5

Table 4: The sampling and 1D Boolean operation time by [6] and the proposed method.

Since rays are emitted from the center of each pixel of the screen, the accuracy of the proposed visual Boolean operations is at pixel level. As shown in Figure 8(a), Dragon and Armadillo perform an intersection operation, and the result can be shown as a full view in Figure 8(b) or as different zooming-in views as in Figure 9(c)-(d). better visual effects can be obtained by bringing the viewpoint closer to the model, all of which are at pixel accuracy level. Once the view is updated after zoom/pan/rotate, the visual Boolean operation will be repeatedly executed to obtain a new rendered image, a visual resultant effect.

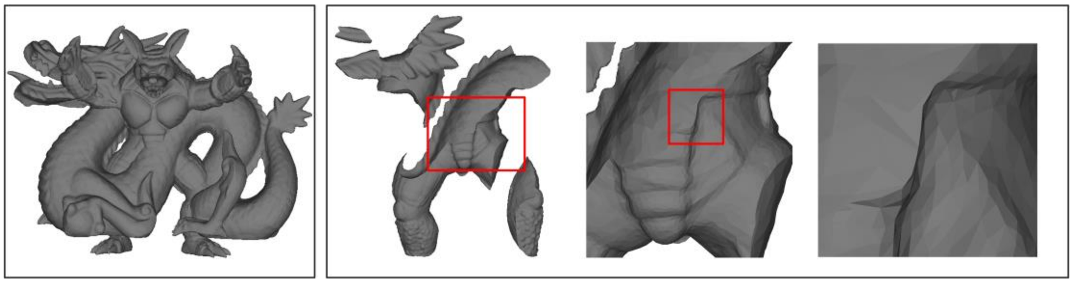

Figure 8: The visual results of VBO (Dragon∩Armadillo): (a) The visual result in a full view, (b) The result of the full view, (c) The visual result when zooming in to the particular parts bounded by the red rectangle in (b), (d) The visual result when zooming in to the particular parts bounded by the red rectangle in (c).

6 CONCLUSIONS

Aiming at the problem that Boolean operations on large-scale mesh models cannot be quickly performed in the design phase. The paper proposes one ray tracing-based method to achieve the real-time visual Boolean operation result, which samples involved models as ray interval models and 1D Boolean operation, defined as "Visual Boolean operation" is done to show the visual result of Boolean operation. In this paper, the ray tracing technique is used to acquire a ray segment model, which can be very quickly done by a third-party ray tracing engine. The results show that the proposed method can achieve visual Boolean operation in real time, which is of a significant sense in designing complex shapes by lots of Boolean operations.

7 ACKNOWLEDGEMENTS

The authors of this paper are supported by the funding of Natural Science Foundation of China (No: 61662006, 62062015) and the Innovation Project of School of Computer Science and Information Engineering, Guangxi Normal University under the contract number JXXYYJSCXXM-002 and Guangxi

100 oversea talents plan. The author of the paper is also partially supported by the Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing.

Shouxin Chen, https://orcid.org/0000-0002-4756-7763
Ming Chen, https://orcid.org/0000-0003-0506-5308
Shenglian Lu, https://orcid.org/0000-0002-4957-9418

REFERENCES

[1] Adams, B.; Dutré, P.: Interactive Boolean operations on surfel-bounded solids, ACM SIGGRAPH, 2003, 651-656. https://doi.org/10.1145/882262.882320

[2] Bergen, G. V. D.: Efficient collision detection of complex deformable models using AABB trees, Journal of graphics tools, 2(4), 1997, 1-13. https://doi.org/10.1080/10867651.1997.10487480

[3] Bernstein, G.; Fussell, D.: Fast, exact, linear Booleans, Computer Graphics Forum, 28(5), 2009, 1269-1278. https://doi.org/10.1111/j.1467-8659.2009.01504.x

[4] Chang, J.-W; Wang, W.; Kim, M.: Efficient collision detection using a dual OBB-sphere bounding volume hierarchy, Computer-Aided Design, 42(1), 2010, 50-57. https://doi.org/10.1016/j.cad.2009.04.010

[5] Campen, M.; Kobbelt, L.: Exact and robust (self-) intersections for polygonal meshes, Comput Graph Forum, 29(2), 2010, 397-406. https://doi.org/10.1111/j.1467-8659.2009.01609.x

[6] Chen, Y.; Wang, C. C. L.: Layer Depth-Normal Images for Complex Geometries: Part One-Accurate Modeling and Adaptive Sampling, International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 43277, 2008, 717-728. https://doi.org/10.1115/DETC2008-49432

[7] Chen, M.; Chen, X.-Y.; Tang, K.; Yuen, M. M. F.: Efficient Boolean operation on manifold mesh surfaces, Computer-Aided Design & Applications, 7(3), 2010, 405-415. https://doi.org/10.3722/cadaps.2010.405-415

[8] Douze, M.; Franco, J. S.; Raffin, B.: QuickCSG: Fast Arbitrary Boolean Combinations of N Solids. arXiv preprint arXiv:1706.01558, 2017.

[9] de Magalhães, S. V. G.; Franklin, W. R.; Andrade, M. V. A.: An Efficient and Exact Parallel Algorithm for Intersecting Large 3-D Triangular Meshes Using Arithmetic Filters, Computer-Aided Design, 120, 2020, 102801. https://doi.org/10.1016/j.cad.2019.102801

[10] Feito, F. R.; Ogáyar, C. J.; Segura, R. J.; Rivero, M. L.: Fast and accurate evaluation of regularized Boolean operations on triangulated solids, Computer-Aided Design, 45(3), 2013, 705-716. https://doi.org/10.1016/j.cad.2012.11.004

[11] Hubbard, P. M.: Collision detection for interactive graphics applications, IEEE Transactions on Visualization and Computer Graphics, 1(3), 1995, 218-230. https://doi.org/10.1109/2945.466717

[12] Ju, T.; Losasso, F.; Schaefer, S.; Warren, J.: Dual contouring of hermite data, Proceedings of the 29th annual conference on Computer graphics and interactive techniques, 2002, 339-346. https://doi.org/10.1145/566570.566586

[13] Klosowski, J. T.; Held, M.; Mitchell, J. S.; Sowizral, H.; Zikan, K.: Efficient collision detection using bounding volume hierarchies of k-DOPs, IEEE transactions on Visualization and Computer Graphics, 4(1), 1998, 21-36. https://doi.org/10.1109/2945.675649

[14] Parker, S. G.; Bigler, J.; Dietrich, A.; Friedrich, H.; Hoberock, J.; Luebke, D.; McAllister, D.; McGuire, M.; Morley, K.; Robison, A.; Stich, M.: OptiX: a general purpose ray tracing engine, ACM transactions on graphics (tog), 29(4), 2010, 1-13. https://doi.org/10.1145/1778765.1778803

[15] Pavić, D.; Campen, M.; Kobbelt, L.: Hybrid Booleans, Computer Graphics Forum, 29(1), 2010, 75-87. https://doi.org/10.1111/j.1467-8659.2009.01545.x

[16] Qin, Y.; Luo, Z.; Wen, L.; Feng, C.; Zhang, X.; Lan, M.; Liu, B.: Research and application of Boolean operation for triangular mesh model of underground space engineering-Boolean

operation for triangular mesh model, Energy Science & Engineering, 7(4), 2019, 1154-1165. https://doi.org/10.1002/ese3.335

[17] Shade, J.; Gortler, S.; He, L.; Szeliski, R.: Layered depth images, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998, 231-242. https://doi.org/10.1145/280814.280882

[18] Sainz, M.; Pajarola, R.: Point-based rendering techniques, Computers & Graphics, 28(6), 2004, 869-879. https://doi.org/10.1016/j.cag.2004.08.014

[19] Varadhan, G.; Krishnan, S.; Sriram, T. V. N.; Manocha, D.: Topology preserving surface extraction using adaptive subdivision, Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, 235-244. https://doi.org/10.1145/1057432.1057464

[20] Wang, C. C. L.; Leung, Y. S.; Chen, Y.: Solid modeling of polyhedral objects by layered depth-normal images on the GPU, Computer-Aided Design, 42(6), 2010, 535-544. https://doi.org/10.1016/j.cad.2010.02.001

[21] Wang, C. C. L.: Approximate boolean operations on large polyhedral solids with partial mesh reconstruction, IEEE transactions on visualization and computer graphics, 17(6), 2010, 836-849.https://doi.org/10.1109/TVCG.2010.106

[22] Wald, I.; Woop, S.; Benthin, C.; Johnson, G. S.; Ernst, M.: Embree: a kernel framework for efficient CPU ray tracing, ACM Transactions on Graphics (TOG), 33(4), 2014, 1-8. https://doi.org/10.1145/2601097.2601199

[23] Yang, Z.-L.; Chen, M.: Quick visual Boolean operation on heavy mesh models, Comput Appl, 37(7), 2017, 2050-2056.

[24] Zeng, L.; Lai, L. M. L.; Qi, D.; Lai, Y.-H.; Yuen, M. M. F.: Efficient slicing procedure based on adaptive layer depth normal image, Computer-Aided Design, 43(12), 2011, 1577-1586. https://doi.org/10.1016/j.cad.2011.06.007

[25] Zhou, Q.; Grinspun, E.; Zorin, D.; Jacobson, A.: Mesh arrangements for solid geometry, ACM Transactions on Graphics (TOG), 35(4), 2016, 1-15.https://doi.org/10.1145/2897824.2925901

[26] Zhao, H.; Wang, C. C. L.; Chen, Y.; Jin, X.: Parallel and efficient Boolean on polygonal solids, The Visual Computer, 27(6), 2011, 507-517.https://doi.org/10.1007/s00371-011-0571-1