Title:
A Real Time Visual Boolean Operation on 3D Models in Product Design Phase

Authors:
Shouxin. Chen, chen.csx@outlook.com, School of Computer and Information Engineering, Guangxi Normal University
Ming Chen*, hustcm@hotmail.com, School of Computer and Information Engineering, Guangxi Normal University
Shenglian Lu, lsl@gxnu.edu.cn, School of Computer and Information Engineering, Guangxi Normal University

Keywords:
Boolean operation, Ray tracing, Real time Boolean operation

DOI: 10.14733/cadconfP.2021.16-20

Introduction:

Boolean operation plays one important role in CAD modeling technology. Many 3D products with complex shapes usually require lots of Boolean operations on multiple primitives to construct the final shapes. With the rapid development of 3D printing technology in recent years, manufacturing complex shaped models has become a reality, which asks for an urgent requirement on the efficiency of Boolean operations. To model such complex shapes in the product concept design phase, Boolean operations are repeatedly done many times on two or more 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 immediate 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 model very 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 LDNI technique [2], 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.[4] [7], the paper makes full use of the advantage to sample 3D shapes into points and perform 1D Boolean operation in parallel. 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 lots of time for dense mesh models, but in the proposed method, the sampling step can be significantly speeded up, achieving a real-time VBO result.

Main Idea:

Ray tracing sampling

Different from using LDI [3][5][8] or LDNI [2][6][9] to use rasterized rendering for ray sampling, this paper uses ray tracing method to sample models as ray segments more efficiently. As shown in Fig.1, when sampling the models M_a and M_b, a set of parallel rays are emitted from the center of pixels of the screen, and the intersection points between the rays and the models are regarded as the sampling points, which are ordered by their depth values.

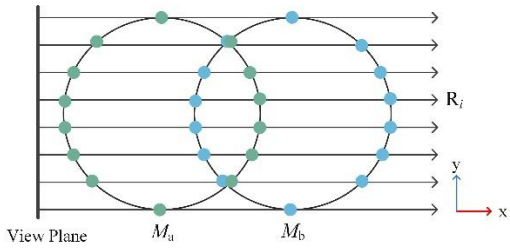

Fig. 1: Ray sampling on *YZ* plane along the *X* axis.

In graphic card, 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 as *R* (*orig*, *dir*), where *orig* and *dir* represent the start point and the direction of the tracing ray, respectively. One sampling ray corresponds to one program fragment, which can be executed parallelly and efficiently. When one ray hits an intersection point, *closest-hit* will be automatically called and the intersection point can be calculated in *closest-hit*; when the ray has no intersection points within the specified range *t*, *miss-hit* is called to terminate the sampling procedure. The sampling process is shown in Fig.2. After *ray generation* is executed, all rays will be traced. The acceleration structure will be traversed to determine whether a triangle intersects any ray. If the intersection occurs, *closest-hit* will be carried out at the intersection point closest to *orig*. Update the ray start point *orig* to be the intersection point, adjust the ray tracing range *t* and keep the ray direction unchanged to continue the ray tracing. If no intersection occurs, do *miss-hit* and stop ray tracing.

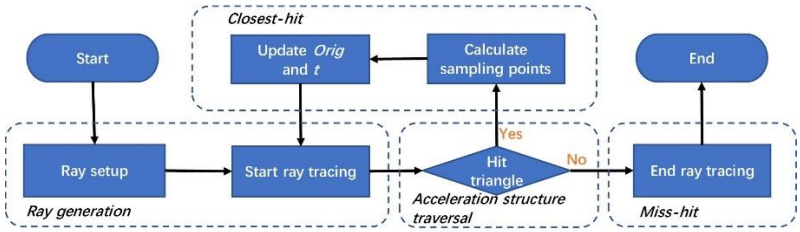

Fig. 2: The flowchart of ray sampling by ray tracing.

The ray tracing range *t* is very important. An unreasonable value of *t* will result in incomplete sampling or failure. The above sampling can be very quickly performed.

1D Boolean operation on rays

Ideally, 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 or when the ray is tangent to the sampling primitive, odd number of sampling points may occur, leading to wrong inside/outside classifications and incorrect results of Boolean operations. The methods in the literatures [2][8] are both applicable to the situation where the number of sampling points on one ray is even. This paper proposes a new scheme to determine the inside/outside classification of sampling points. As shown in Fig.3, 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. One sampling point's inside/outside classification can be figured out with the help of the entry/exit status of its nearest neighboring sampling points of another primitive. Take Fig.3 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_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. Once the inside/outside step is completed, the sampling points of M_a and M_b should 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.

Fig. 3: 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.

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

Tab. 1: Rules of sampling points kept.

Image based rendering

After 1D Boolean operation, the kept points can be rendered through the Surfel method or the image-based rendering method. In the paper, our focus is how to get the visual result in the least time. The image-based rendering method is adopted to render the final results.

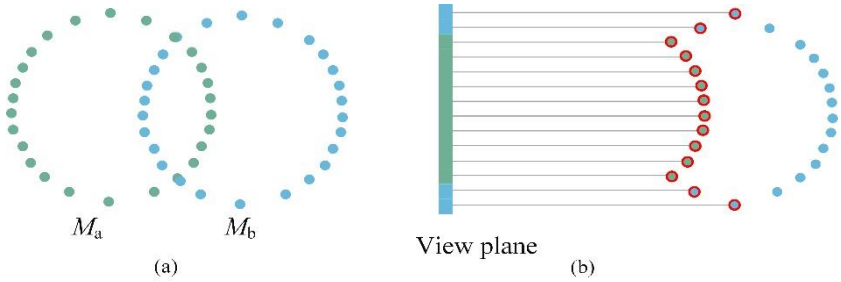

Fig. 4: 1D Boolean result of M_b-M_a at one slicing layer: (a) Ray sampling result of M_a and M_b at one slicing layer; (b) 1D Boolean result of M_b-M_a at one slicing layer and only the circled points (the foremost point along each ray) will be rendered as one part of the final rendering image.

Fig. 4 shows the 1D Boolean operation result of M_b-M_a at one slicing layer. For the kept points after 1D Boolean operation, only the foremost point (circled points in Fig.4(b)) along each ray will be rendered as one pixel. After calculating each pixel's rendering information such as normal vector, light and material texture, all these kept foremost points will be rendered as one image, presenting the final visual result. Once one rotates, pans, zooms in/out or modifies any primitive, three steps, namely, sampling, 1D Boolean operation, image rendering, should be re-executed. The above three steps can be done in real time.

Results:

The method in [2] 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.Fig.5 shows the final visual effect (images) by the proposed algorithm. For Rhino, the time of mesh model

Boolean operation is counted. For the proposed method and the method in [2], the time of VBO is counted. All the above data is listed in Tab.2. From Tab.3, we can find that it is impossible to obtain a real time result by Rhino if one mesh model Boolean operation is directly done: for Test 3, if 100 difference Mesh Boolean operations are done one time, it will take 32 seconds for Rhino package. Thus, quick visual Boolean operation is one complementary step before doing real mesh Boolean operation. For all three tests, the proposed method can obtain the final 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 [2] is done for visual Boolean operation, the proposed algorithm is much faster, as the sampling time is much shorter than that of the benchmark in [2]. The sampling time and 1D Boolean operation time by the proposed method and the method in [2] are illustrated in Tab.4.

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

Tab. 2: Tested models

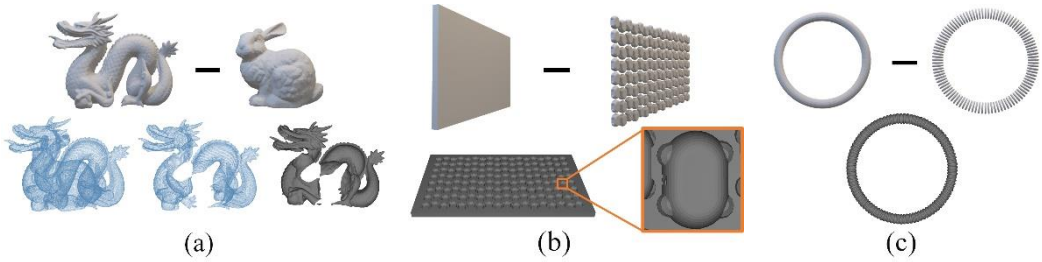

Fig. 5: The visual results of VBO by the proposed method: (a) the Test1's sampling result and VBO; (b) the VBO result of Test 2; (c) the VBO result of Test 3.

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

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

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

Tab. 4: The sampling time and 1D Boolean operation time by [2] and the proposed method.

Conclusion:

Aiming at the problem that Boolean operations on large-scale 3D models cannot be quickly performed in the design phase. The paper proposes one ray tracing-based method to achieve real-time visual Boolean operation result, which samples involved models as point models and 1D Boolean operation, defined as "Visual Boolean operation" is done to show the visual result of Boolean operation. The results show that the proposed method can achieve visual Boolean operation in real time, which is of significant sense in designing complex shapes by lots of Boolean operations.

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.

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] 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, 2008, 43277, 717-728. https://doi.org/10.1115/DETC2008-49432

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

[4] 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), 2010, 29(4), 1-13. https://doi.org/10.1145/1778765.1778803

[5] 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

[6] 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

[7] 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), 2014, 33(4), 1-8. https://doi.org/10.1145/2601097.2601199

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

[9] 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, 2011, 43(12), 1577-1586. https://doi.org/10.1016/j.cad.2011.06.007