Supplementary Document of *DiffCSG*: Differentiable CSG via Rasterization

HAOCHENG YUAN, The University of Edinburgh, United Kingdom
ADRIEN BOUSSEAU, Inria, Université Côte d'Azur, France
HAO PAN, Microsoft Research Asia, China
CHENGQUAN ZHANG, Nanjing University, China
NILOY J. MITRA, University College London, Adobe Research, United Kingdom
CHANGJIAN LI, The University of Edinburgh, United Kingdom

ACM Reference Format:
Haocheng Yuan, Adrien Bousseau, Hao Pan, Chengquan Zhang, Niloy J. Mitra, and Changjian Li. 2024. Supplementary Document of *DiffCSG*: Differentiable CSG via Rasterization. In *SIGGRAPH Asia 2024 Conference Papers (SA Conference Papers '24), December 3–6, 2024, Tokyo, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3680528.3687608

In this supplemental material, we provide details on intersection edge calculation, our benchmark evaluation, and comparisons against SDF-based and derivative-free CSG optimization.

1 INTERSECTION EDGE CALCULATION

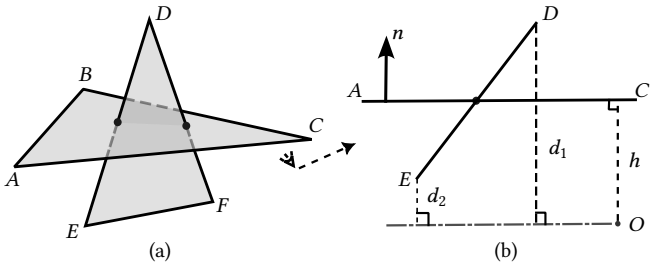

Fig. 1. **Intersection edge calculation**.

As shown in Fig. 1 (a), given two triangles with vertices A, B, C and D, E, F, we find the intersection point of each edge to the other triangle. Taking triangle ABC and edge DE as an example, a side view orthogonal to the triangle normal is shown in Fig. 1 (b). Denoting O a spatial point out of the shape volume, we compute the unsigned distance h of O to the triangle, along with the unsigned distances d_1 and d_2 of D and E to the plane parallel to the triangle and running through O. We then construct similar triangles to find the relative position of the intersection point on the edge ED as:

$$\alpha = \frac{d_1 - h}{d_1 - d_2}. \tag{1}$$

If α is between $[0, 1]$, we calculate the intersection point as $(1 - \alpha)D + \alpha E$, otherwise, we treat it as a non-intersection. Note that we discard cases where d_1 and d_2 are nearly equal, which corresponds to cases where the edge is nearly parallel to the triangle. Having the interpolated point, we simply check if it is within the triangle ABC to accept it as an intersection point. We calculate the intersection points for each edge in parallel. In case of duplicates, we select one of the intersection points randomly. In the end, if there are two intersection points, we record them for the following anti-aliasing step.

Complexity and scalibilty. We have implemented the intersection edge calculation in parallel using PyTorch. For a scene with K primitives, the total number of triangles in the scene is $n = \sum_{i=1}^{K} N[i]$, where $N[i]$ indicates the number of triangles of the i-th primitive. Our method computes intersections between all triangle pairs where the two triangles are from different primitives. The total amount of triangle pairs to be processed for that scene is $P = (\sum_{i=1}^{K} N[i])^2 - \sum_{i=1}^{K} N[i]^2$. In our implementation, we construct a $2 \times P \times 3$ matrix for all the triangle pairs and compute the intersection in parallel. That means the memory complexity can be approximately considered as $O(n^2)$, while the ideal running time complexity is $O(1)$ if the GPU behavior is more tractable.

In Fig. 2, we further display the intersection edge calculation time versus the number of *triangle pairs* by changing the subdivision level of two intersecting spheres. Up to *16 million* triangle pairs, the edge calculation takes around 0.5s.

Although the edge calculation is not the bottleneck since it is faster than the Goldfeather algorithm, this step could be accelerated by fast triangle-triangle intersection tests or a polygonal ID buffer. Besides, because our method optimizes primitive parameters rather than primitive vertices, we can use low-poly geometry for optimization and only resort to high-poly geometry for downstream applications (visualization, fabrication), or adopt a coarse-to-fine strategy that would progressively increase mesh resolution throughout optimization.

Authors' addresses: Haocheng Yuan, The University of Edinburgh, 10 Crichton Street, Edinburgh, United Kingdom, H.C.Yuan@ed.ac.uk; Adrien Bousseau, Inria, Université Côte d'Azur, 2004 route des lucioles, Valbonne, France, adrien.bousseau@inria.fr; Hao Pan, Microsoft Research Asia, No.5 Danling Rd, Beijing, China, haopan@microsoft.com; Chengquan Zhang, Nanjing University, 163 Xianlin Rd, Nanjing, China, zhangandresnju@gmail.com; Niloy J. Mitra, University College London, Adobe Research, 169 Euston Square, London, United Kingdom, n.mitra@cs.ucl.ac.uk; Changjian Li, The University of Edinburgh, 10 Crichton Street, Edinburgh, United Kingdom, Changjian.li@ed.ac.uk.

SA Conference Papers '24, December 3–6, 2024, Tokyo, Japan

ACM ISBN 979-8-4007-1131-2/24/12
https://doi.org/10.1145/3680528.3687608

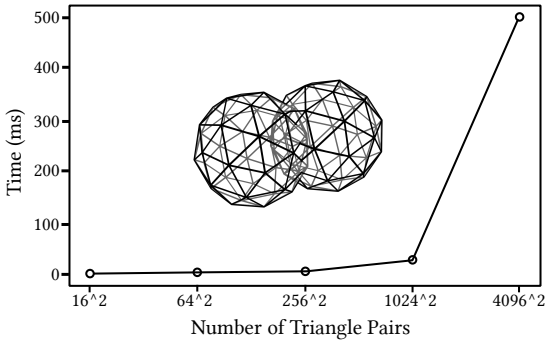

Fig. 2. **Running time vs. triangle pairs.** We render a scene of two spheres with different amount of triangles by controlling the sub-division levels of the sphere mesh.

2 BENCHMARK EVALUATION

By combining the Goldfeather algorithm with differentiable rasterization, our solution enables fast image-based optimization of CSG parameters. We evaluate the success and accuracy of this optimization on a benchmark composed of CSG models that we perturbed to form ground-truth source and target shapes.

DiffCSG-100 dataset. We have collected 50 CAD shapes from both *CADTalk-Real* [Yuan et al. 2023] and *Fusion360* [Willis et al. 2021] to form our benchmark dataset. While CADTalk-Real is a dataset of CSG programs, Fusion360 is a dataset of sketch-extrude construction sequences that we converted to CSG programs made of sketch-extrude primitives and boolean operations.

These 50 shapes vary in complexity and number of optimizable parameters. On the one hand, some of the CSG models in CADTalk-Real have been designed to only expose a few (5-20) editable hyper-parameters, while other parameters (50-200) are hard-coded. On the other hand, Fusion360 programs have no hyper-parameters. We manually created augmented versions of these programs, either to create hyper-parameters for the Fusion360 programs, or to expose all parameters of CADTalk-Real programs. This augmentation results in 100 shapes in total, distributed in two tracks that we denote H-Params (with hyper-parameters) and D-Params (with dense parameters). See Tab. 1 for statistics on the average number of primitives, optimizable parameters, and different types of boolean operations involved.

Table 1. **Benchmark statistics.** Average number of primitives, optimizable parameters, and boolean operators for the different tracks of our benchmark.

	CADTalk-20		Fusion360-30	
	H-Params	D-Params	H-Params	D-Params
#Primitive	13.1	13.1	2.5	2.5
#Parameters	6.4	202.0	9.33	108.03
#Union	9.9	9.9	0.5	0.5
#Difference	2.1	2.1	1.0	1.0
#Intersection	0.15	0.15	0.1	0.1

We treat these 100 shapes as targets for optimization, and we create source shapes to be optimized by perturbing the parameters of each shape. Specifically, we have created three source shapes for each target by multiplying each parameter by a factor randomly sampled from a Gaussian distribution $\mathcal{N}(1.0, \sigma^2)$ with σ set to be 0.1, 0.2, or 0.4 to obtain low, medium, and high perturbation levels, respectively. See Fig. 3 for two examples.

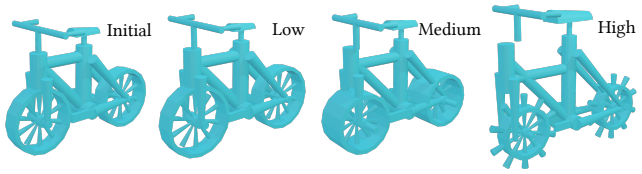

Fig. 3. **CSG shape perturbation.** Three levels of perturbations are applied to a given CSG shape, leading to the corresponding alternations.

Evaluation task. We perform the gradient-based optimization task using multi-view images as the target and the pixel-wise L_2 loss in the optimization. In order to capture the shape change as much as possible, *eight* 3/4 views are selected. For the target image, we experimented with both vertex normal maps and per-primitive pure colors.

Metrics. To evaluate the effectiveness, we use the Chamfer distance (S_{CD}) and Hausdorff distance (S_{HD}) as error metrics. Additionally, we categorize a test case as successfully converged if the average image-based objective error over the past 20 iterations falls below the same loss threshold as the one used to terminate the optimization. We define the success rate (*Succ*) as the percentage of successfully converged test cases out of the total number of test cases.

Result analysis. Our primary evaluation statistics are presented in Table 2. For the error metrics, we provide the Chamfer and Hausdorff distances between the initial and target shapes before optimization as a reference. As can be seen, our method can reduce both metrics over 50% when the normal signal is used, and 30% when the color signal is used. This indicates the overall effectiveness of the proposed differentiable CSG optimization. The last two rows reveal more information about the perturbation level and optimizable parameters. Firstly, increasing the perturbation level significantly decreases the success rate, especially in the D-Params track, where more parameters are optimizable. Secondly, the success rate across all perturbation levels is higher for H-Params than for D-Params, indicating that fewer hyper-parameters are easier to optimize to fit a target. Thirdly, even though the hyper-parameters are more semantically meaningful in CADTalk-20 than in Fusion-360, the success rate does not differ significantly. This suggests that the optimization does not depend heavily on the hyper-parameter design, and further semantic-aware regularizers could be proposed to better leverage the parameter design insights. Regarding the optimization signal, it is worth noting that the *Succ* for the color signal is higher than the normal signal. This is because color encodes less geometric information, making it easier to optimize, but it does not result in

Table 2. **Statistical Evaluation.** Three metrics are presented for four sub-tracks from CADTalk-20 and Fusion360-30 on three different perturbation levels. The superscript *before* means the error metrics measured between the initial and target shapes, while the *color* and *normal* indicate the optimization with per-primitive pure color or vertex normal as the target image.

	CADTalk-20						Fusion360-30					
	H-Params			D-Params			H-Params			D-Params		
Perturbation	Low	Medium	High	Low	Medium	High	Low	Medium	High	Low	Medium	High
${S_{CD}}^{before}(ref.)$	0.019	0.023	0.048	0.019	0.032	0.050	0.048	0.062	0.13	0.032	0.057	0.103
${S_{CD}}^{color} \downarrow$	0.015	0.017	0.026	0.018	0.021	0.038	0.029	0.027	0.073	0.025	0.036	0.037
${S_{CD}}^{normal} \downarrow$	0.014	0.014	0.016	0.012	0.020	0.031	0.029	0.022	0.078	0.019	0.033	0.040
${S_{HD}}^{before}$	0.043	0.048	0.096	0.041	0.089	0.167	0.153	0.210	0.404	0.108	0.174	0.311
${S_{HD}}^{color} \downarrow$	0.031	0.036	0.060	0.033	0.072	0.125	0.061	0.074	0.253	0.077	0.101	0.12
${S_{HD}}^{normal} \downarrow$	0.021	0.022	0.036	0.019	0.079	0.116	0.082	0.093	0.218	0.065	0.090	0.134
$Succ^{color}(\%) \uparrow$	100	100	94	98	90	85	98	95.3	90	94.7	88.7	78
$Succ^{normal}(\%) \uparrow$	100	98	92	96	85	73	97.3	93.3	81.3	96.7	82	71.3

more accurate geometry. This can be observed from the higher S_{CD} and S_{HD} when using the color signal.

3 COMPARISON

SDF-based CSG optimization. We further evaluate our method by comparing it against the closest competitor, UCSGNet [Kania et al. 2020], which represents CSG models using SDFs. UCSGNet takes a point cloud as input and outputs a CSG representation composed of primitives and their boolean combinations. Since we only optimize continuous parameters, we adapted UCSGNet by providing it with a fixed CSG tree and the corresponding primitives. Consequently, UCSGNet functions as an optimization algorithm that takes an initial CSG model as input and adjusts its parameters to match a target point cloud.

Since UCSGNet relies on analytic SDF functions to represent the primitives, it can only handle a limited set of primitive types, *i.e.*, cylinder, sphere, or box. We have selected from the benchmark 5 test cases which are entirely composed of these three types of primitives. For each shape, we used the augmented counterpart (*i.e.*, the hyper- and dense-parameter versions) as well as the three corresponding perturbations (*i.e.*, low, medium, and high) to obtain 45 (source,target) pairs to conduct the comparison. Statistically, the average *Succ* of USCGNet drops from 100% to 75% and further to 50% for the low, medium, and high levels, respectively, while our method succeeds in all cases, achieving a 100% *Succ* for all levels. For the successful cases, their S_{CD} are slightly lower than ours because UCSGNet uses the 3D point cloud as the target, which contains direct 3D information for shape optimization. Regarding the running time, on average, USCGNet takes 1342.5s, 4148.8s, and 934.8s for the three perturbation levels, while our method only takes 3.26s, 9.72s, and 4.32s, respectively, which is three orders of magnitude faster in reaching the solution.

We can attribute the differences to two factors. First, the SDF-based representation by UCSGNet needs to densely sample the whole 3D space to evaluate the SDF values, which takes cubic complexity in contrast to the quadratic complexity of our rendering-based approach. Meanwhile, our approach makes full use of the parallel processing pipeline of modern graphics processing units. Second, the boolean operations over SDF functions are approximated by α clipping, which transforms the SDF to binary occupancy. The clipping parameter must be carefully tuned during optimization, to balance between providing smooth gradients for optimization and sharp occupancy for accurate boolean operations. The scheduling of the clipping parameter is heuristic and causes slow convergence and even failure to converge in many cases. In contrast, our differentiable rendering of CSG models consistently produces sharp images and provides effective gradients along intersection edges, without requiring any heuristic tuning.

Dreivative-free CSG optimization. We have compared with CMA-ES [Hansen et al. 2003] that alternates between perturbing parameter values and selecting the best values for further optimization. A visual comparison is shown in Fig. 4. For simple shapes with few parameters, CMA-ES can converge to the target state successfully. However, it is sensitive to the scale of each parameter, which makes it hard to apply to CSG shapes where parameters typically have different and unknown scales leading to more optimization time and lower accuracy. Another advantage of making CSG-rendering differentiable is to enable the joint optimization of different scene parameters, such as CSG shapes and texture maps. In contrast, a gradient-free method like CMA-ES would need to be combined with differentiable rasterization in an ad-hoc manner since the two algorithms would optimize shape and texture parameters separately.

REFERENCES

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11, 1 (2003), 1–18.

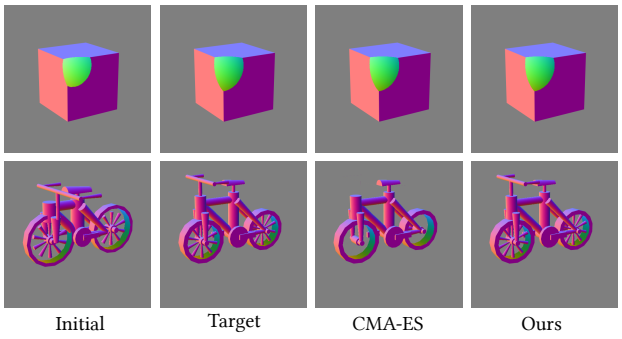

Fig. 4. **Comparison with derivative-free optimization of CSG models from CMA-ES [Hansen et al. 2003].** For the shape with few parameters (first row), both CMA-ES and our method successfully find the result in a few seconds (3s vs. 5s, respectively). While for complex shapes with parameters of different scale levels (second row), CMA-ES fails to find the desired result after 10 minutes; our method fits the target shape correctly within the same time.

Kacper Kania, Maciej Zieba, and Tomasz Kajdanowicz. 2020. UCSG-NET-unsupervised discovering of constructive solid geometry tree. *Advances in neural information processing systems* 33 (2020), 8776–8786.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. 2021. Fusion 360 gallery: a dataset and environment for programmatic CAD construction from human design sequences. *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2021).

Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy Mitra, and Changjian Li. 2023. CADTalk: An Algorithm and Benchmark for Semantic Commenting of CAD Programs. *arXiv preprint arXiv:2311.16703* (2023).