Pacific Graphics 2025
M. Christie, N. Pietroni, and Y.-S. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 7

PaMO Supplementary Materials

Seonghun Oh[1†], Xiaodi Yuan[2†], Xinyue Wei[2†], Ruoxi Shi[2], Fanbo Xiang[3], Minghua Liu[3], Hao Su[2,3‡]

[1]Yonsei University [2]University of California, San Diego [3]Hillbot Inc.

1. Intersection-free Dual Marching Cubes

1.1. Look-Up Table of DualMC

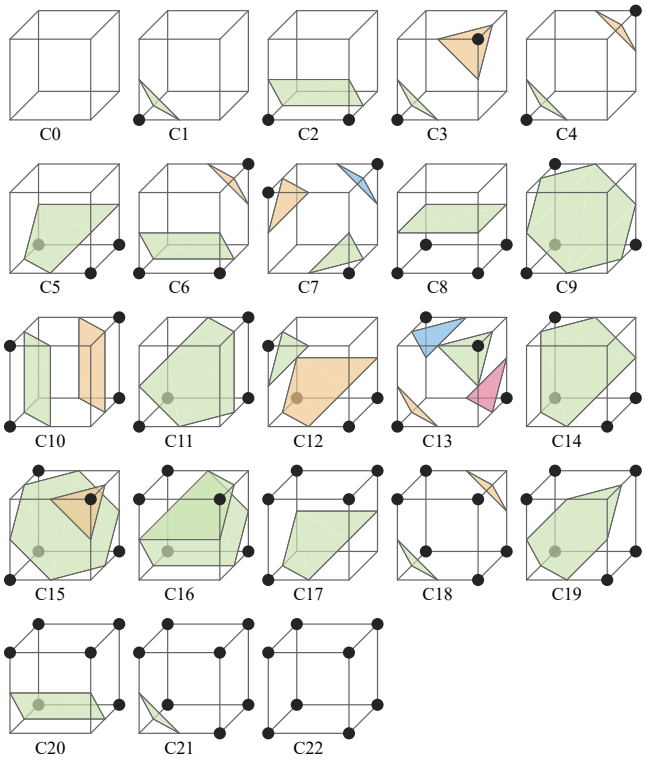

Figure 1: *Look-Up table of DualMC cases. Figure adapted from [Nie04].*

Here we show the look-up table for building DualMC patches. The original DualMC [Nie04] may introduce non-manifold structures in very rare cases, we follow the solution in [Wen13] to solve the problematic C16 and C19 cases in which ambiguous faces are shared.

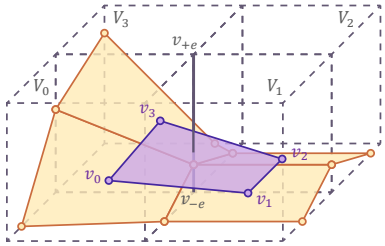

Figure 2: *Patches (in yellow) and a quad (in purple) in DualMC. (v_{+e}, v_{-e}) is a valid edge.*

2. Detailed Algorithms

2.1. DualMC Self-intersection Correction

2.1.0.1. Handling numerical instability In DualMC, the patch vertex v is determined through linear interpolation between the corresponding edge vertices (v_e^0, v_e^1) based on their SDF values:

$$v = v_e^0 + t(v_e^1 - v_e^0), \quad t = \frac{-f(v_e^0)}{f(v_e^1) - f(v_e^0)}, \tag{1}$$

where $f(\cdot)$ represents the SDF function and t is within $[0,1]$.

We found extreme t values can introduce numerical instability and cause inter-quad intersections. To address this, we apply a sigmoid smoothing function to t and use t' in the linear interpolation to mitigate extreme values:

$$t' = \frac{1}{1+\exp(-\beta(t-0.5))} \tag{2}$$

In this work, $\beta = 5$ is chosen based on experimental observations.

2.1.0.2. Quad Division We adopt the concept of the *envelope* from [JU06, Wan09] to eliminate intersections caused by quad division. As shown in Figure 2, a quad $Q = (v_0, v_1, v_2, v_3)$ is defined around a valid edge $e = (v_{+e}, v_{-e})$. The envelope E_e is defined using $v_{+e}, v_{-e}, v_0, v_1, v_2, v_3$. The concavity of the vertices in Q is determined using the scalar triple product. A vertex v_i is defined as

a *concave vertex* if either of the following condition holds:

$$(v_i - v_{+e}) \cdot \left[(v_i^l - v_{+e}) \times (v_i^r - v_{+e}) \right] < 0, \tag{3}$$

$$(v_i - v_{-e}) \cdot \left[(v_i^l - v_{-e}) \times (v_i^r - v_{-e}) \right] > 0. \tag{4}$$

where v_i^l and v_i^r are the two adjacent vertices of v_i in the quad.

The quad is divided based on the distribution of concave vertices:

- If only one diagonal of Q contains concave vertices, the quad is divided along that diagonal.
- If both diagonals contain concave vertices, Q is divided into four triangles, with an additional vertex v_e placed on edge e.
- If no concave vertices exist, we opt for a division that maximizes the minimal angle of each triangle.

When computing the additional v_e in the second case, we determine its position using linear interpolation based on the edge vertices SDF values, combined with the sigmoid smoothing function to more accurately represent the iso-surface.

We implemented the quad division strategy using a 2-kernel Gather approach to ensure parallelism. First, we determine whether each vertex is concave and record the number of triangles to be generated within each quad along with the count of newly created vertices, and then perform the appropriate quad division accordingly.

2.2. Computing the Energy Functions

In the safe projection stage (stage 3), as we optimize the augmented energy function $B(X)$ using Newton's method, we need to compute the gradient $\nabla B(X)$ and the Hessian matrix $\nabla^2 B(X)$ at each iteration. In this section, we provide details about the computation of each term in $B(X)$ and their gradients and Hessian matrices.

2.2.0.1. Distance energy E_{S2M} At the t-th iteration, we update the nearest neighboring point $\tilde{y}_i \in \mathcal{M}_{\text{in}}$ for every $i \in V(\mathcal{S})$:

$$\tilde{y}_i(X^t) = \underset{y \in \mathcal{M}_{\text{in}}}{\operatorname{argmin}} \|x_i^t - y\|^2. \tag{5}$$

With the nearest neighbors fixed,

$$\tilde{E}_{\text{S2M}}(X; \{\tilde{y}_i\}_{i=1}^n) = \sum_{i \in V(\mathcal{S})} s_i^0 \|x_i - \tilde{y}_i\|^2 \tag{6}$$

in the following iterations as an estimation of E_{S2M} before we update the nearest neighbors again. The Hessian matrix is block-diagonal, where every block is 3×3.

2.2.0.2. Distance energy E_{M2S} At the t-th iteration, we update the nearest neighboring triangle $(\tilde{i}_s, \tilde{j}_s, \tilde{k}_s) \in F(\mathcal{S})$ for every sample $y_s (1 \le s \le m)$:

$$(\tilde{i}_s, \tilde{j}_s, \tilde{k}_s)(X^t) = \underset{(i,j,k) \in F(\mathcal{S})}{\operatorname{argmin}} \; d(\triangle_{ijk}(X^t), y_s) \tag{7}$$

where $d(\triangle_{ijk}(X^t), y_s)$ is the distance between point y_s and the triangle whose vertices are (x_i^t, x_j^t, x_k^t). These point-triangle distances can further be classified into point-point distances, point-line distances, and point-plane distances according to the position of the nearest neighboring point of y_s on $\triangle_{ijk}(X^t)$. Each distance category corresponds to a differentiable closed-form formula. We refer the reader to [LFS*20] for the detailed formulas. Similar to E_{S2M}, we estimate E_{M2S} as

$$\tilde{E}_{\text{M2S}}(X; \{(\tilde{i}_s, \tilde{j}_s, \tilde{k}_s)\}_{s=1}^m) = \frac{A(\mathcal{M}_{\text{in}})}{m} \sum_{s=1}^m d(\triangle_{\tilde{i}_s \tilde{j}_s \tilde{k}_s}(X), y_s). \tag{8}$$

The Hessian matrix $\nabla_X^2 \tilde{E}_{\text{M2S}}(X)$ is a block-sparse matrix, where every block is at most 9×9 and corresponds to a triangle $(\tilde{i}_s, \tilde{j}_s, \tilde{k}_s)$.

2.2.0.3. Elastic energy E_{elas} We refer the reader to [SB12] for the computation of the gradient and Hessian of the St. Venant-Kirchhoff energy. The Hessian matrix $\nabla_X^2 E_{\text{elas}}(X)$ is a block-sparse matrix, where every block is at most 9×9 and corresponds to a triangle $(\tilde{i}_s, \tilde{j}_s, \tilde{k}_s)$.

2.2.0.4. Bending energy E_{bend} We refer the reader to [TG13] for the computation of the gradient and Hessian of the bending energy. The Hessian matrix $\nabla_X^2 E_{\text{bend}}(X)$ is a block-sparse matrix, where every block is at most 12×12 and involves 4 vertices of $\mathcal{S}$ belonging to two adjacent triangles.

2.2.0.5. Barrier function $b(d_c(X))$ We refer the reader to [LFS*20] for the computation of the gradient and Hessian of the barrier function. The Hessian matrix $\nabla_X^2 \sum_{c \in C} b(d_c(X))$ is a block-sparse matrix, where every block is at most 12×12 and involves 4 vertices of $\mathcal{S}$. The 4 vertices correspond to an activated ($d_c < \hat{d}$) contact pair, which is either a point-triangle pair or an edge-edge pair.

3. Intersection tests

3.1. Coplanar triangle intersection analysis

Coplanar triangle intersections in libigl and Blender often struggle to differentiate between non-intersecting adjacent pairs that share vertices and actual self-intersections. To address this issue, we have implemented a 2D intersection test that categorizes triangle pairs based on the number of shared vertices. The complete algorithm for our self-intersection detection is shown in Algorithm 1. Firstly, we construct a BVH tree to accelerate our triangle-triangle query process (Line 10). For each triangle in the mesh, we compute its intersection candidates by querying the BVH tree and checking for AABB overlaps (Line 12). Then, for non-coplanar cases, we utilize *Devillers' algorithm* to detect intersections; for coplanar cases, we divide the algorithm into three scenarios (Line 2 - Line 7) based on the number of shared vertices.

Here we explain the details of how we handle the detection of intersections among coplanar triangles.

3.1.0.1. Coplanar with no shared vertex We also use *Devillers' algorithm* to detect intersections; for those with one shared vertex

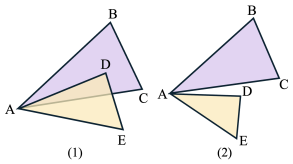

Figure 3: *Single-vertex-sharing cases.*

3.1.0.2. Single vertex sharing case When there is one shared vertex, we propose *angle-overlap test*. We begin by designating a shared vertex as A from triangles $\triangle ABC$ and $\triangle ADE$, as shown in Figure 3. We aim to determine

ALGORITHM 1: Parallel Self-Intersection Detection
$F(\mathcal{M})$ denotes the face set.

Procedure `CoplanarIntersect` (T,C,s) : // App. Sec. 3.1
  if $s = 0$ **then** // Fig. 7(1)
    return CoplanarNonShareIntersect(T,C)
  else if $s = 1$ **then** // Fig. 7(2)(3)
    return AngleOverlapIntersect(T,C)
  else if $s = 2$ **then** // Fig. 7(4)(5)
    return SameSideIntersect(T,C)
  return True
Procedure `IntersectionCheck` $(\mathcal{M})$:
  Input: Mesh $\mathcal{M}$
  Output: Self-intersecting triangle pairs $\mathcal{T}$
  BVH $\leftarrow$ ConstructLBVHTree$(\mathcal{M})$ // parallel
  foreach $T \in F(\mathcal{M})$ **do** // parallel
    $\mathcal{C}_T \leftarrow$ BVHQuery(BVH, T) // intersection candidates
  foreach (T,C) *s.t.* $T \in F(\mathcal{M}), C \in \mathcal{C}_T$ **do** // parallel
    $s \leftarrow$ CountSharedVertices(T,C)
    if IsCoplanar(T,C) **then**
      if `CoplanarIntersect` (T,C,s) **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(T,C)\}$
    else if $s \neq 2$ **then**
      $l \leftarrow$ GetIntersectLine(T,C)
      if $\|l\| > 0 \wedge \neg(\|l\| \leq \epsilon \wedge s = 1)$ **then** // App. Sec. 3.2
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{(T,C)\}$
  return $\mathcal{T}$

whether an edge of one triangle, such as vector $\overrightarrow{AD}$ or $\overrightarrow{AE}$, lies between the edges $\overrightarrow{AB}$ and $\overrightarrow{AC}$ of the other triangle.

Therefore, we first calculate the cross products for vectors $\overrightarrow{AB}$, $\overrightarrow{AC}$, $\overrightarrow{AD}$, and $\overrightarrow{AE}$ originating from A. We store the results of these calculations in variables corresponding to each pair's cross product—specifically:

$$\vec{c}_1 = \overrightarrow{AB} \times \overrightarrow{AD}, \qquad \vec{c}_2 = \overrightarrow{AB} \times \overrightarrow{AE},$$
$$\vec{c}_3 = \overrightarrow{AC} \times \overrightarrow{AD}, \qquad \vec{c}_4 = \overrightarrow{AC} \times \overrightarrow{AE}.$$

By comparing the signs of $\vec{c}_1$, $\vec{c}_2$, $\vec{c}_3$, and $\vec{c}_4$, we can infer the following:

- If $\vec{c}_1 \cdot \vec{c}_3 \leq 0$, $\overrightarrow{AD}$ may lie between $\overrightarrow{AB}$ and $\overrightarrow{AC}$.
- If $\vec{c}_2 \cdot \vec{c}_4 \leq 0$, $\overrightarrow{AE}$ may lie between $\overrightarrow{AB}$ and $\overrightarrow{AC}$.
- If $\vec{c}_1 \cdot \vec{c}_2 \leq 0$, $\overrightarrow{AB}$ may lie between $\overrightarrow{AD}$ and $\overrightarrow{AE}$.
- If $\vec{c}_3 \cdot \vec{c}_4 \leq 0$, $\overrightarrow{AC}$ may lie between $\overrightarrow{AD}$ and $\overrightarrow{AE}$.

While direction comparison can filter out most cases, the nature of the cross-product means that there can be instances where the directions differ but no actual overlap exists. To address this, after initial filtering using the directions of the $\vec{c}_i$, we further verify the situation by calculating the sum of the angles between the vector in question and the other two vectors. Specifically, if the directions differ, we compute this sum. If the sum of these angles is less than or equal to 180°, we confirm the presence of an intersection.

Let $\theta_{AB,AD}$ be the angle between $\overrightarrow{AB}$ and $\overrightarrow{AD}$, $\theta_{AB,AE}$ be the angle between $\overrightarrow{AB}$ and $\overrightarrow{AE}$, $\theta_{AC,AD}$ be the angle between $\overrightarrow{AC}$ and $\overrightarrow{AD}$, and $\theta_{AC,AE}$ be the angle between $\overrightarrow{AC}$ and $\overrightarrow{AE}$, where

$$\theta_{AB,AD}, \theta_{AB,AE}, \theta_{AC,AD}, \theta_{AC,AE} \in [0, \pi].$$

The conditions for confirming overlap are given by:

$$\begin{cases} \theta_{AB,AD} + \theta_{AC,AD} < \pi, & \text{if } \vec{c}_1 \cdot \vec{c}_3 < 0, \\ \theta_{AB,AE} + \theta_{AC,AE} < \pi, & \text{if } \vec{c}_2 \cdot \vec{c}_4 < 0, \\ \theta_{AD,AB} + \theta_{AE,AB} < \pi, & \text{if } \vec{c}_1 \cdot \vec{c}_2 < 0, \\ \theta_{AD,AC} + \theta_{AE,AC} < \pi, & \text{if } \vec{c}_3 \cdot \vec{c}_4 < 0. \end{cases}$$

To distinguish edge cases where the cross product is zero, as illustrated in Figure 4 cases (1) and (2), we can utilize the sign of the dot product. Specifically, for cases where the cross product is zero, we can compare the sign of the dot product to determine the intersection.

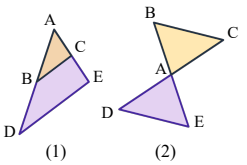

Figure 4: *Zero cross-product cases.*

For vectors $\vec{u}$ and $\vec{v}$, if $|\vec{u} \times \vec{v}| = 0$, we further compare the signs of the dot products to ensure accurate intersection checks.

The edge case handling can be described as follows: If $|\vec{u} \times \vec{v}| = 0$, then compare sign$(\vec{u} \cdot \vec{w})$ and sign$(\vec{v} \cdot \vec{w})$ for a vector $\vec{w}$.

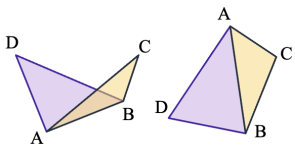

Figure 5: *Edge-sharing cases.*

3.1.0.3. Edge sharing case When two triangles share two vertices, A and B, we propose the *same-side test* to detect intersections. This test involves extending the shared edge AB into a line and determining the positions of the remaining vertices C and D. If C and D lie on opposite sides of the line AB, it indicates a simple edge sharing. Otherwise, if they are on the same side, it indicates a self-intersection.

3.2. 3D triangle intersection analysis

Drawing on the capabilities of [DG02], which allows for the determination of the intersection line when triangles intersect robustly, we propose a method to manage these intersections based on the length of the intersection line. If two triangles share a vertex and the intersection line is very short, the intersection likely occurs at the shared vertex, and thus, should not be considered a self-intersection. Conversely, if the intersection line is long despite the vertices being shared, it suggests that the triangles, although sharing vertices, do intersect themselves. Since in 3D scenarios, triangles sharing an edge cannot occur intersection, we can simply skip the case.

References

[DG02] DEVILLERS O., GUIGUE P.: *Faster triangle-triangle intersection tests*. PhD thesis, INRIA, 2002. 3

[JU06] JU T., UDESHI T.: Intersection-free contouring on an octree grid. In *Proceedings of Pacific graphics* (2006), vol. 2006, Citeseer. 1

[LFS*20] LI M., FERGUSON Z., SCHNEIDER T., LANGLOIS T., ZORIN D., PANOZZO D., JIANG C., KAUFMAN D. M.: Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Trans. Graph. (SIGGRAPH) 39*, 4 (2020). 2

[Nie04] NIELSON G. M.: Dual marching cubes. In *IEEE visualization 2004* (2004), IEEE, pp. 489–496. 1

[SB12] SIFAKIS E., BARBIC J.: Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses* (New York, NY, USA, 2012), SIGGRAPH '12, Association for Computing Machinery. URL: https://doi.org/10.1145/2343483.2343501, doi:10.1145/2343483.2343501. 2

[TG13] TAMSTORF R., GRINSPUN E.: Discrete bending forces and their Jacobians. *Graphical Models 75*, 6 (Nov. 2013), 362–370. URL: https://linkinghub.elsevier.com/retrieve/pii/S1524070313000209, doi:10.1016/j.gmod.2013.07.001. 2

[Wan09] WANG C. C.: *Intersection-free dual contouring on uniform grids: an approach based on convex/concave analysis*. Tech. rep., Technical Report, 2009, http://www2. mae. cuhk. edu. hk/˜ cwang/pubs . . . , 2009. 1

[Wen13] WENGER R.: *Isosurfaces: geometry, topology, and algorithms*. CRC Press, 2013. 1