gDist: Efficient Distance Computation between 3D Meshes on GPU Supplemetary Material

PENG FAN, Zhejiang University, China
WEI WANG, Zhejiang University, China
RUOFENG TONG, Zhejiang University, China
HAILONG LI, Shenzhen Poisson Software Co., Ltd., China
MIN TANG*, Zhejiang University, Zhejiang Sci-Tech University, China

ACM Reference Format:
Peng Fan, Wei Wang, Ruofeng Tong, Hailong Li, and Min Tang. 2024. gDist: Efficient Distance Computation between 3D Meshes on GPU Supplemetary Material. 1, 1 (August 2024), 4 pages. https://doi.org/10.1145/3680528.3687619

1 ALGORITHM FOR F12-BVH CONSTRUCTION

Algorithm 1 outlines how the GPU parallelly constructs our f12-BVH. The input parameter t denotes the thread ID, where the t-th thread processes the t-th leaf node of the BVH. n represents the number of triangles, $depth$ is the BVH depth, which can also be calculated from n, and $Triangles$ is the input array of triangles.

In lines 1-12 of the algorithm, we determine the indices of triangles that should be included in the t-th leaf node. The while loop starting from line 4 can be viewed as a search process from the root node to leaf node t, with each search incrementing the depth of the current node (line 11). The condition $OnTheRight$ checks if leaf node t is in the right subtree of the current node, a condition easily implemented with bitwise operations.

Lines $13-16$ of the algorithm build the bounding box for the t-th leaf node, with each leaf node containing only $1-2$ triangles.

Finally, the BVH is constructed in a bottom-up manner (line 17), following the approach outlined in [Chitalu et al. 2020].

2 ALGORITHMS FOR MAXIMUM DISTANCE COMPUTING

Algorithm 2 and 3 outline how we calculate the maximum distance between two BVHs. The overall process of the algorithm is consistent with the minimum distance computing, with only slight differences. During the traversal process, we maintain a lower bound on the maximum distance lower$_{\text{global}}$ (Algorithm 3, line 10) and eliminate all BVTT nodes whose upper bounds on the maximum distance are smaller than lower$_{\text{global}}$ (Algorithm 3, line 8).

In addition, since the maximum distance between meshes is equivalent to the distance between their vertices, the primitives stored in our BVH will change from triangle facets to vertices.

Algorithm 1 F12-BVH Construction

```
Input: $t, n, deep, Triangles$
Output: $BVH$
1:  $l \leftarrow 0$
2:  $r \leftarrow n$
3:  $d \leftarrow 0$
4:  while NotReachLeaf($d, deep$) do
5:      if OnTheRight($t, d, deep$) then
6:          $l \leftarrow l + \lceil r/2 \rceil$
7:          $r \leftarrow \lfloor r/2 \rfloor$
8:      else
9:          $r \leftarrow \lceil r/2 \rceil$
10:     end if
11:     $d \leftarrow d + 1$
12: end while
13: Initialize $AABB_{leaf}$
14: for $i = l$ to $r - 1$ do
15:     $AABB_{leaf} \leftarrow Triangles_i$
16: end for
17: ConstructBVHBottomUp($AABB_{leaf}, BVH$)
18: return $BVH$
```

Algorithm 2 Parallel Maximum Distance Computing

```
Input: Two BVHs: $root_A, root_B$
Output: The maximum disance: $D_{max}$
1:  Initialize $buffer_1, buffer_2$  // BVTT fronts
2:  $lower, upper \leftarrow$ calculateDistanceBounds($root_A, root_B$)
3:  $buffer_1 \leftarrow root_A, root_B, lower, upper$  // Update the front
4:  $lower_{global} \leftarrow lower$
5:  while Not Reach Leafs do
6:      $step \leftarrow$ calculateAdaptiveDepth($buffer_1^{length}$)
7:      $buffer_2 \leftarrow$ expandBvtt($buffer_1, lower_{global}, step$)
8:      swapBuffer($buffer_1, buffer_2$)
9:  end while
10: $D_{max} \leftarrow lower_{global}$
11: return $D_{max}$
```

*Corresponding author, https://min-tang.github.io/home/gDist/, This work was supported by the Leading Goose R&D Program of Zhejiang under Grant No. 2024C01103.

Authors' addresses: Peng Fan, Zhejiang University, China, yingtix2021@163.com; Wei Wang, Zhejiang University, China, wangweiydyk@zju.edu.cn; Ruofeng Tong, Zhejiang University, China, trf@zju.edu.cn; Hailong Li, Shenzhen Poisson Software Co., Ltd., China, lihailong@poissonsoft.com; Min Tang, Zhejiang University, Zhejiang Sci-Tech University, China, tang_m@zju.edu.cn.

ACM XXXX-XXXX/2024/8-ART
https://doi.org/10.1145/3680528.3687619

Algorithm 3 Expand BVTT Buffer(Maximum Distance)

```
Input: $buffer_{input}, lower_{global}, step$
Output: $buffer_{output}$
 1: Initialize $buffer_{output}$
 2: for each $BVTT \in buffer_{input}$ do
 3:     $children_A \leftarrow$ getChildren($BVTT_A, step$)
 4:     $children_B \leftarrow$ getChildren($BVTT_B, step$)
 5:     for each $child_A \in children_A$ do
 6:         for each $child_B \in children_B$ do
 7:             $lower, upper \leftarrow$ calcDistanceBounds($child_A, child_B$)
 8:             if $upper > lower_{global}$ then
 9:                 $bufferd_{output} \leftarrow child_A, child_B, upper$
10:                 $lower_{global} \leftarrow \max(lower_{global}, lower)$
11:             end if
12:         end for
13:     end for
14: end for
15: return $buffer_{output}$
```

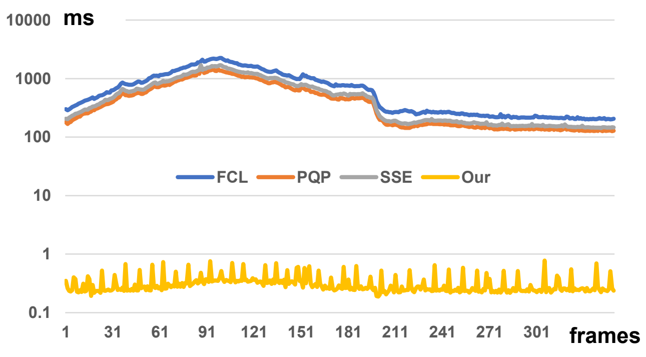

Fig. 1. **Performance Comparison for Benchmark Tools:** The figure compares the performance of distance calculations for each frame of rigid body motion. Compared to PQP, we observe above 1500X speedups on average.

3 PERFORMANCE COMPARISON

Fig. 1 is a performance comparison for Benchmark Tools between PQP [Larsen et al. 2014], FCL [Pan et al. 2012], SSE [Shellshear and Ytterlid 2014] and our algorithm, gDist. To facilitate a more intuitive comparison, the vertical axis in the figure is annotated in exponential notation.

Fig. 2 is a performance comparison for Benchmark Rings at each motion frame. To facilitate a more intuitive comparison, the vertical axis in the figure is annotated in exponential notation.

Fig. 3 is a performance comparison for Benchmark Comet-Tool at each motion frame. To facilitate a more intuitive comparison, the vertical axis in the figure is annotated in exponential notation.

Fig. 4 is a performance comparison for Benchmark Penetration at each motion frame. To facilitate a more intuitive comparison, the vertical axis in the figure is annotated in exponential notation.

Fig. 5 highlights the performance of Benchmark Deformable at each deforming frame. The average query time is about 0.51 milliseconds per frame on an NVIDIA GeForce 4090.

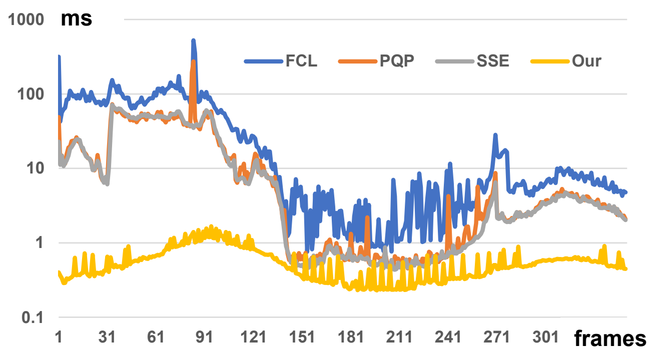

Fig. 2. **Performance Comparison for Benchmark Rings:** The figure compares the performance of distance calculations for each frame of rigid body motion. Compared to PQP, we observe above 26X speedups on average.

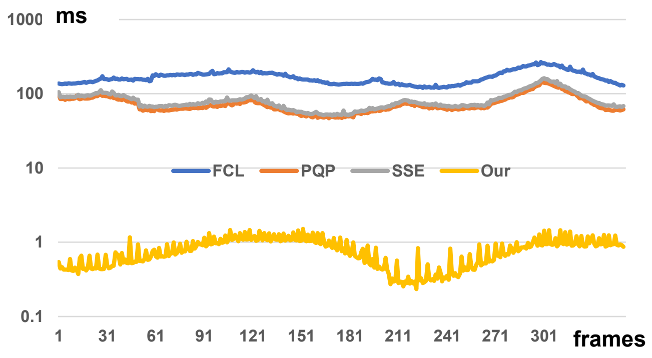

Fig. 3. **Performance Comparison for Benchmark Comet-Tool:** The figure compares the performance of distance calculations for each frame of rigid body motion. Compared to PQP, we observe about 92X speedups on average.

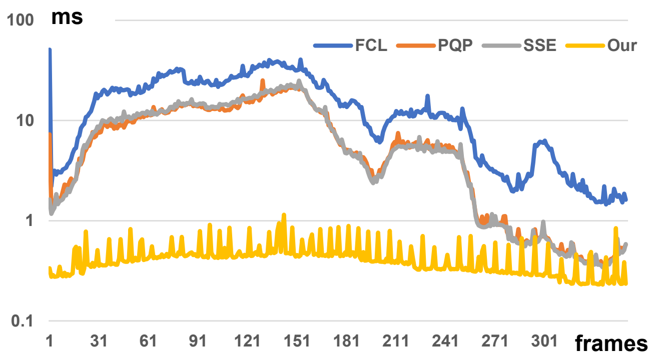

Fig. 4. **Performance Comparison for Benchmark Penetration:** The figure compares the performance of distance calculations for each frame of rigid body motion. Compared to PQP, we observe about 19.56X speedups on average.

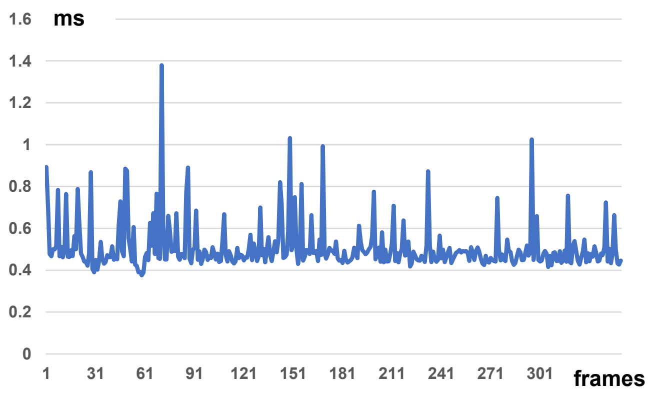

Fig. 5. **Performance of Benchmark Deformable:** The figure highlights the performance of Benchmark Deformable at each deforming frame.

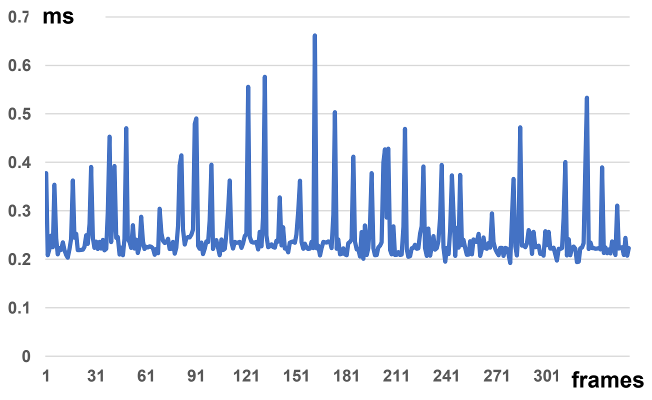

Fig. 6. **Performance of Benchmark MaxDist:** The figure highlights the performance of maximum distance computation for Benchmark MaxDist at each motion frame.

Fig. 6 highlights the performance of maximum distance computation for Benchmark MaxDist at each motion frame. The average query time is about 0.25 milliseconds per frame on an NVIDIA GeForce 4090.

4 TEMPORAL COHERENCY

The inherent temporal coherency of deformable bodies is strategically exploited using a lightweight approach. Specifically, we record model vertices (V_a, V_b) in proximity to the nearest point pair (P_a, P_b) from the previous frame. In the subsequent frame, these recorded vertices (V_a, V_b) aid in initializing estimates for maximum and minimum distances, using $Dist(V_a, V_b)$. This strategy incurs minimal computational cost and yields notable benefits. Notably, this temporal coherency utilization extends beyond deformable bodies, proving advantageous for rigid bodies undergoing continuous motion. Empirical results demonstrate a noteworthy $10\% - 12\%$ speedup compared to an implementation devoid of this temporal coherency strategy.

Fig. 8 illustrates a comparison between distance computations with and without temporal coherency on Benchmark Tools. The utilization of temporal coherency in this benchmark results in a notable performance improvement of 10.6%. In Fig. 7, two nested Voronoi balls undergo wavelike deformations along their surface normal vectors. Each ball model is composed of 130K triangles (Fig. 7). The average query time is about 0.51 milliseconds per frame on an NVIDIA GeForce 4090.

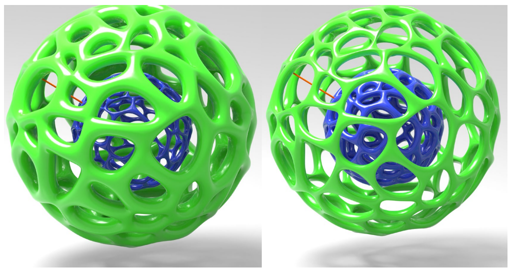

Fig. 7. **Benchmark Deformable:** Two nested Voronoi balls undergo wavelike deformations along their surface normal vectors. Each ball model is composed of 130K triangles. The average query time is about 0.51 milliseconds per frame on an NVIDIA GeForce 4090.

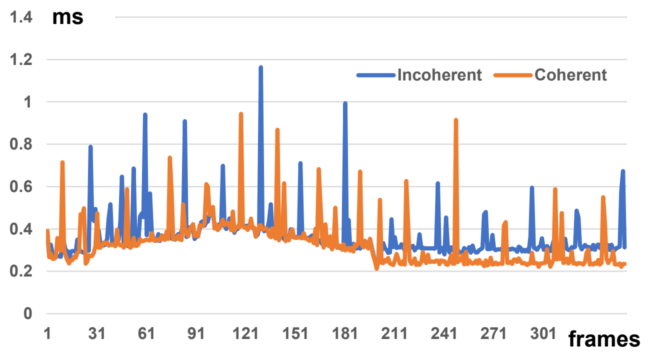

Fig. 8. **Speedups with temporal coherency on Benchmark Tools:** Figure 8 illustrates a comparison between distance computations with (in orange) and without (in blue) temporal coherency on Benchmark Tools. The utilization of temporal coherency in this benchmark results in a notable performance improvement of 10.6%.

Temporal coherence provides a good estimate of the upper bound of the minimum distance in our algorithm. However, as we can obtain a reasonably good estimate even without temporal coherence after a few expansions, its impact on the subsequent execution of the algorithm is limited.

In the event of a significant distance between the previous and subsequent frames, the algorithm's reliance on model vertices V_a and V_b from the previous frame may have a smaller acceleration effect. However, this optimization is designed to provide a reasonable initial estimate of the upper bound of the minimum distance. Even with a large deformation between frames, it won't lead to incorrect results.

Fig. 9 illustrates a comparison between distance computations with and without temporal coherency on Benchmark Rings. The utilization of temporal coherency in this benchmark results in a notable performance improvement of 12.6%.

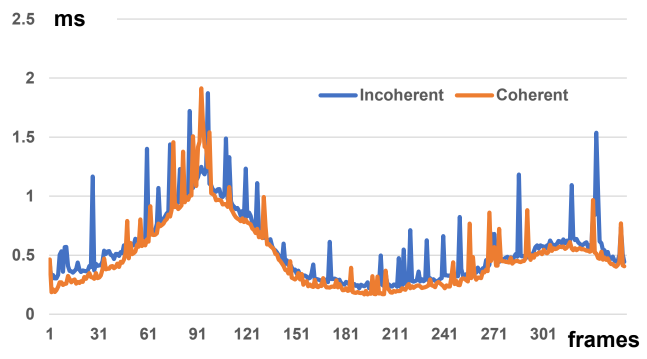

Fig. 9. **Speedups with temporal coherency on Benchmark Rings:** Figure 9 illustrates a comparison between distance computations with (in orange) and without(in blue) temporal coherency on Benchmark Rings. The utilization of temporal coherency in this benchmark results in a notable performance improvement of 12.6%.

5 COMPARISON WITH CPU/GPU IMPLEMENTATIONS

Our algorithm is generic, as demonstrated by testing on three different GPUs. We did not conduct any specific tuning for the targeted GPU; however, the tuning could further improve performance.

The insights gained from our approach may not be directly applicable to accelerate CPU-based algorithms. Our algorithm is tailored for GPU architecture. While the tighter distance bounds benefit CPU performance, the fine-grained BVTT expansion, optimized for GPU parallelism, might lead to suboptimal CPU performance.

6 BVH SHARING

Regarding BVH sharing, we deliberately chose not to share the BVHs of the two identical meshes. While BVH sharing is a common practice, we opted for algorithmic generality, treating each mesh independently. Considering body-space BVH for scenes with rigid bodies could offer advantages by avoiding the cost of mesh memory transfer and BVH construction, but it may compromise the algorithm's generality, especially for scenes with deformable objects.

7 PROOF OF ENHANCED BOUND

Given two meshes $Mesh_A$ and $Mesh_B$ with their bounding AABB V_a and V_b respectively. V_a and V_b are defined as:

$$V_a = \{P | A_i^{min} \leq P_i \leq A_i^{max}, i = 1, 2, 3\} \tag{1}$$

$$V_b = \{P | B_i^{min} \leq P_i \leq B_i^{max}, i = 1, 2, 3\} \tag{2}$$

We term an AABB as tight only when at least one point on each of its six bounding rectangles coincides with a vertex of its encompassing model.

Theorem 1: *The upper bound, d^*, for the minimum distance between AABBs, is,*

$$d^* = min\{max_{P \in S_i^{V_a}, Q \in S_j^{V_b}} ||PQ||\}, \tag{3}$$

where S represents the six bounding rectangles of each AABB, and $i = 1, 2, ..., 6$, $j = 1, 2, ...6$. Further, d^* also is an upper bound for the minimum distance between $Mesh_A$ and $Mesh_B$.

Proof: Given that two AABBs are convex polyhedra, the points of minimum distance between them necessarily fall on their boundary faces, ie., F_a and F_b. So the minimum distance between the two AABBs, d, must fulfill:

$$d \leq d^* = min\{max_{P \in S_i^{V_a}, Q \in S_j^{V_b}} ||PQ||\}. \tag{4}$$

Moreover, since the AABBs here are tight, there must exist vertices V_a in F_a and V_b in F_b such that the distance $d(V_a, V_b) < d^*$. Thus, d^* also serves as an upper bound for the minimum distance between $Mesh_A$ and $Mesh_B$.

Theorem 2: *Given two tight AABB V_a and V_b, there is always an inequality that holds:*

$$d_{min}^U = d^* \leq \hat{d}^* = \hat{d}_{min}^U, \tag{5}$$

where $\hat{d}_{min}^U$ is determined by the fastest points contained by the two AABBs.

Proof: Given that two AABBs are convex polyhedra, the points of maximum distance between them necessarily fall on their boundary faces, ie., F_a and F_b. So the maximum distance between the two AABBs, $\hat{d}^*$, must fulfill:

$$\hat{d}^* = max\{max_{P \in S_i^{V_a}, Q \in S_j^{V_b}} ||PQ||\}. \tag{6}$$

Since

$$d^* = min\{max_{P \in S_i^{V_a}, Q \in S_j^{V_b}} ||PQ||\}, \tag{7}$$

so we conclude: $d^* \leq \hat{d}^*$.

REFERENCES

Floyd M. Chitalu, Christophe Dubach, and Taku Komura. 2020. Binary Ostensibly-Implicit Trees for Fast Collision Detection. *Computer Graphics Forum* 39, 2 (2020), 509–521.

Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. 2014. *PQP*. http://http://gamma.cs.unc.edu/SSV/

Jia Pan, Sachin Chitta, and Dinesh Manocha. 2012. FCL: A General Purpose Library for Collision and Proximity Queries. In *2012 IEEE International Conference on Robotics and Automation*. 3859–3866. https://doi.org/10.1109/ICRA.2012.6225337

Evan Shellshear and Robin Ytterlid. 2014. Fast Distance Queries for Triangles, Lines, and Points using SSE Instructions. *Journal of Computer Graphics Techniques* 3, 4 (2014), 86–110.