Helper-Lane Optimized Triangulation of Polygons

Róbert Bene[1] and Gábor Valasek[1,2]

[1] Shapr3D Limited [2] Eötvös Loránd University, Hungary

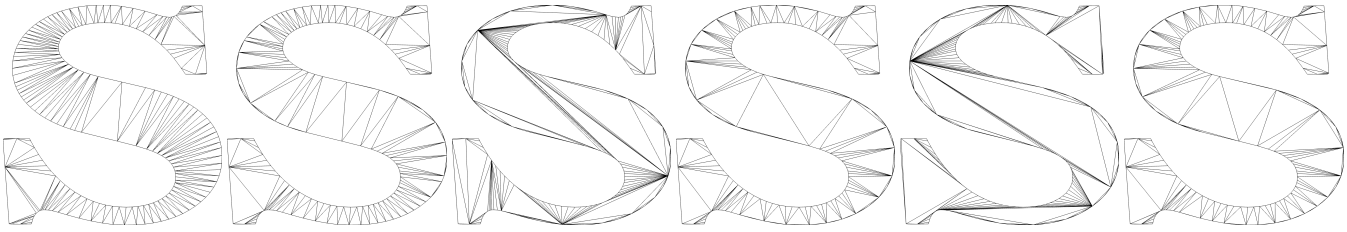


Figure 1: *Triangulations for the same polygon: constrained Delaunay triangulation with (CDT opt.) and without (CDT) our proposed edge flipping postprocessing. Ear clipping with (ear clipping opt.) and without our postprocessing. Greedy maximum-area (GMA) proposed by Persson [Per09]. Minimum-weight triangulation (MWT) that minimizes the total edge length.*

Abstract

We propose to use total edge length minimizing triangulation of polygons for high-performance rendering. We show that this reduces the number of helper-lane invocations and improves effective GPU utilization. We show that helper-lane count is an indicator of performance, however, it is not the only factor. We propose an edge flipping postprocessing algorithm to improve the rendering performance of arbitrary baseline triangulations. Our comparisons are carried out on vector graphics data.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Rasterization;***

1. Introduction and Related Work

Modern GPUs use triangles as rasterization primitives. Consequently, polygons, the ubiquitous representation of piecewise-linear planar faces, need to be triangulated for rendering. This is not a mere conversion between two equivalent forms, as different triangulation strategies measurably alter execution times on the GPU. This is particularly pronounced in the presence of complex fragment shaders, MSAA, high overdraw, or transparency.

A key mechanism behind this effect on performance is that GPUs dispatch fragment work in quads, i.e., in units of 2×2 fragments. In what follows, we refer to each fragment in a quad as a *lane*. If a quad contains a triangle edge, some of its lanes become helper invocations to provide derivative and other quad-scoped information, even though their corresponding pixels fall outside the triangle. Triangulations resulting in larger total edge lengths may increase the number of partially covered quads, raising the ratio of helper lanes and reducing SIMD utilization.

This was noted by Persson, observing that triangulating a circular polygon as a triangle fan results in suboptimal rasterization performance [Per09]. They report that replacing the triangle fan with a greedy maximum-area heuristic (GMA) substantially increased frame rates, in some cases by up to $3\times$. Their investigation was restricted to the triangle fan and GMA. We extend their research by considering triangulations adopted in other application domains and compare their processing and rendering performance.

One widely used such baseline is the constrained Delaunay triangulation (CDT) [Che89]. Generally, the result of GMA differs from that of the CDT. We demonstrate that this does not translate to optimal edge density or performance empirically.

Probably the most likely default triangulation algorithm for polygons relies on their anatomy rather than the statistics of the resulting triangulations. Ear clipping leverages that there is always a boundary triangle on a polygon whose interior does not contain any vertex. By iteratively clipping these, one can efficiently convert an arbitrary polygon into triangles [Mei75].

Finally, we consider the use of minimum weight triangulations (MWT) [Kli80] to explicitly minimize total edge lengths and potentially improve helper-lane efficiency. These triangulations mini-

DOI: 10.2312/egs.20261006

mize the total edge length of the final triangle set. Although proven to be NP-hard in general for polygons with holes, these triangulations still provide a viable offline preprocessing option for highly performance-sensitive use cases, especially for simple polygons for which $\Theta(n^3)$ solutions exist. Our paper makes the processing-time versus rendering performance trade-off quantifiable.

As a contribution, we adapt length-minimizing edge flips into an efficient postprocessing step for baseline ear clipping and CDT triangulations. Using a max-heap, our method iteratively flips edges within convex quadrilaterals to reduce total edge length.

In Section 2, we discuss the details of the aforementioned triangulations and variants thereof. Section 3 presents our performance measurements that support that minimization of total edge length is indicative of performance both on tiled and non-tiled GPUs, however, it is not solely driven by neither this nor helper-lane counts. Finally, Section 4 concludes our findings.

2. Triangulation Techniques

We used the following triangulation methods to quantify the trade-off between preprocessing time, GPU frame time, and boundary complexity. Our implementations handle convex and concave polygons without holes. Extending the methods to handle holes is feasible but beyond our current scope.

Ear Clipping. We used the `mapbox-earcut` implementation [Kog25], a robust ear-clipping variant that supports polygons with holes and common degeneracies. Classic ear clipping starts from the boundary vertex list and iteratively removes any ear (v_{i-1}, v_i, v_{i+1}) that is convex and contains no other polygon vertex in its interior. Each removed ear is emitted as a triangle, and the process repeats until a single triangle remains. In the worst case, ear clipping runs in $\mathcal{O}(n^2)$ time and uses $\mathcal{O}(n)$ memory.

Greedy Max-Area Triangulation. We use a recursive greedy heuristic that repeatedly extracts the largest-area valid triangle from the current sub-polygon, following Persson's line of thought [Per09]. For a contiguous boundary chain sub-polygon Q, we evaluate all vertex triples and select

$$(i,j,k) = \arg\max_{\substack{i<j<k \\ \phi(i,j,k)}} A(i,j,k), \tag{1}$$

where $A(i,j,k)$ is the signed triangle area and $\phi(i,j,k)$ is a validity predicate. $\phi(i,j,k)$ requires that $T = \triangle(v_i, v_j, v_k)$ is internal to Q: its edges are either polygon edges or feasible diagonals (lie inside or on the boundary of Q and do not intersect the boundary except at endpoints), and the interior of T contains no other vertex of Q. We accelerate testing by precomputing a diagonal-feasibility table. After emitting T, we recurse on the induced sub-polygons until only triangles remain. The triangulation runs in $\mathcal{O}(n^4)$ time.

Constrained Delaunay Triangulation (CDT). As a robust baseline, we compute a constrained Delaunay triangulation using Shewchuk's Triangle [She96]. We convert the input polygon to a planar straight-line graph and request a triangulation that preserves boundary constraints. CDT construction is typically $\mathcal{O}(n \log n)$ time for well-behaved inputs and uses $\mathcal{O}(n)$ memory.

Minimum-Weight Triangulation. We triangulate an n-vertex polygonal boundary via the standard interval dynamic programming for minimum-weight triangulation, augmented with a feasibility predicate to support concave inputs. Let $P = (v_0, \ldots, v_{n-1})$ be the ordered boundary and define $\mathrm{dp}[i,j]$ as the minimum cost to triangulate the sub-chain $[i,j]$:

$$\mathrm{dp}[i,j] = \min_{\substack{i<k<j \\ \phi(i,k,j)}} \Big(\mathrm{dp}[i,k] + \mathrm{dp}[k,j] + \|v_i - v_k\| + \|v_k - v_j\| \Big), \tag{2}$$

where $\phi(i,k,j)$ enforces geometric validity. For concave polygons, we precompute the diagonal feasibility table to accelerate geometric validity checks, and accept a candidate triangle (v_i, v_k, v_j) iff each of its three edges is either a boundary edge or a feasible diagonal. The method runs in $\Theta(n^3)$ time and $\Theta(n^2)$ memory.

Edge length reduction by edge-flipping. Both GMA and MWT require significant computational effort even for simple polygons. While length-minimizing edge flips are an established theoretical approximation for minimum-weight triangulations [DMM95], prior empirical evaluations of this heuristic relied primarily on uniformly distributed point clouds and low-complexity, organic polygonal contours. We adapt this heuristic to a practical, post-processing step focused on high-density, highly concave vector graphics and procedural shapes. By introducing a max-heap prioritized by the magnitude of local length reduction, we efficiently refine fast baseline triangulations. We demonstrate that this approach directly accelerates modern GPU rasterization by significantly reducing helper-lane invocations and tile amplification.

After building triangle adjacency, each interior edge shared by two triangles is a flip candidate. If the incident triangles form a convex quadrilateral (a,b,c,d), we replace the current diagonal (a,b) with (c,d) whenever this shortens the diagonal. Candidates are prioritized in a max-heap by the improvement $\Delta = \|a-b\|^2 - \|c-d\|^2$. On each pop, we revalidate convexity and improvement, perform the flip, and re-enqueue the local neighborhood. The procedure stops when no improving flips remain or when a user-defined limit on flips/heap pops are reached. Although this implementation does not guarantee finding a global optimum in terms of summed unique edge lengths, it provides a greedy local approximation while preserving geometric validity.

3. Test Results

We performed all measurements on two devices, an Apple M3 Max MacBook Pro with 14 CPU and 30 GPU cores equipped with 32 GB unified memory, and a PC desktop system with an Intel i9-13900K, 128 GB RAM, and an NVIDIA A4500 GPU. Our test application uses Metal 3.2 on Apple and DirectX12 via Falcor 8 on Windows. We chose these platforms to compare how triangulation affects performance across two fundamentally different GPU architectures: a tile-based deferred renderer (TBDR, Apple M-series) and an immediate-mode renderer (NVIDIA). The source code for the Apple and Windows implementations is publicly available at `https://github.com/benerobert42/HelperLaneViz` and `https://github.com/benerobert42/HelperLaneWindows`, respectively.

Table 1: *Performance: frame time t_f [ms] and helper-lane invocation count N_{hl} across test scenes, per MSAA mode and instance count.*

(a) *SVG scenes on NVIDIA A4500 GPU*

Method	MSAA = None												MSAA = 4×											
	1						100×100						1						100×100					
	S		7		2		S		7		2		S		7		2		S		7		2	
	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}
Ear clipping	0.032	83821	**0.028**	38414	0.031	76680	0.309	2047785	0.173	1323160	0.253	2081998	0.121	89873	0.108	40205	0.121	81065	0.782	4604424	0.377	2562780	0.685	4419187
Ear clipping (opt.)	**0.031**	58989	**0.028**	**27282**	**0.030**	45464	**0.272**	2137325	**0.169**	1382344	0.246	2004410	**0.116**	61865	**0.105**	**28729**	**0.113**	47277	0.616	4871694	**0.365**	2520659	0.525	3996707
CDT	0.032	75937	**0.028**	38654	0.031	71236	0.315	2759781	0.182	1532532	0.267	2422930	0.120	77232	0.109	39862	0.120	72229	0.703	7200225	0.393	3377746	0.614	6459293
CDT (opt.)	**0.031**	60109	**0.028**	30946	**0.030**	48224	0.296	2382453	0.176	1446292	0.226	2066550	0.117	61808	**0.105**	32354	0.114	49435	0.633	5572750	0.373	2795973	0.531	4413810
GMA	0.032	80845	**0.028**	33298	0.031	68556	0.302	**1745845**	0.181	**1056360**	0.263	**1639858**	0.120	89223	0.106	37992	0.118	75765	0.739	**4039864**	0.367	**1948647**	0.663	**3341594**
MWT	**0.031**	**57925**	**0.028**	27494	**0.030**	**45292**	0.288	2206689	0.180	1411660	**0.218**	1988542	**0.116**	**60553**	**0.105**	29007	**0.113**	**46795**	**0.614**	4992025	0.370	2542371	**0.522**	4033427

(b) *SVG scenes on Apple M3 Max (binned) GPU*

Method	MSAA = None												MSAA = 4×											
	1						10×10						1						10×10					
	S		7		2		S		7		2		S		7		2		S		7		2	
	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}
Ear clipping	0.186	37148	0.170	12023	0.181	25004	0.189	557900	0.173	197456	0.182	442872	0.246	37148	0.208	12023	0.233	25004	0.268	557900	0.225	197456	0.254	442872
Ear clipping (opt.)	0.175	23226	**0.168**	**8039**	**0.173**	13518	0.184	332780	**0.172**	**125286**	0.178	245692	0.222	23226	0.207	**8039**	**0.214**	13518	**0.253**	332780	0.223	**125286**	0.244	245692
CDT	**0.174**	17594	0.185	23241	0.188	33164	0.187	435036	0.186	490438	0.196	613534	0.224	17594	0.240	23241	0.244	33164	0.264	435036	0.253	490438	0.275	613534
CDT (opt.)	**0.174**	**17016**	**0.168**	8589	**0.173**	13674	0.185	331346	**0.172**	155014	0.178	248602	0.222	**17016**	0.207	8589	0.215	13674	0.256	331346	0.224	155014	**0.243**	248602
GMA	0.181	42302	0.169	17221	0.180	32712	0.189	597456	0.173	225190	0.184	436992	0.233	42302	0.209	17221	0.229	32712	0.267	597456	0.227	225190	0.255	436992
MWT	**0.174**	18044	**0.168**	9025	**0.173**	**12998**	**0.183**	**318858**	**0.172**	125328	**0.177**	**237122**	**0.220**	18044	**0.206**	9025	**0.214**	**12998**	0.254	**318858**	**0.222**	125328	**0.243**	**237122**

(c) *Circle (C) and Ellipse (E) on NVIDIA A4500 GPU*

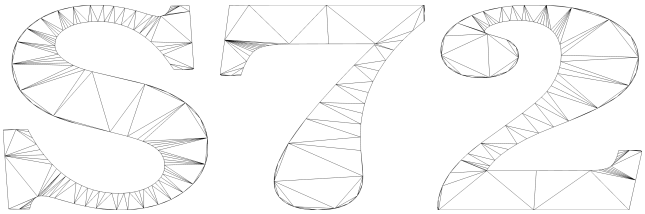
Method	MSAA = None								MSAA = 4×							
	1				100×100				1				100×100			
	C		E		C		E		C		E		C		E	
	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}
Ear clipping	**0.029**	26220	**0.024**	19976	0.401	1077008	0.401	718120	0.106	30652	0.095	24216	**0.454**	1686789	0.432	1268971
Ear clipping (opt.)	**0.029**	26220	**0.024**	17376	**0.400**	1077008	**0.400**	**628408**	0.106	30652	**0.094**	21628	0.508	1686789	0.485	1077650
CDT	**0.029**	27784	0.029	132092	**0.400**	1074660	**0.400**	1002264	0.106	32941	0.112	138578	0.457	1794261	0.460	4050816
CDT (opt.)	**0.029**	25944	0.025	23808	0.401	1038540	0.401	781656	**0.105**	30535	0.095	27564	0.509	1659918	0.485	1726840
GMA	**0.029**	25848	**0.024**	19080	0.401	1022818	0.401	658068	**0.105**	30684	0.095	23733	0.456	1628169	0.436	1174960
MWT	**0.029**	**25404**	**0.024**	**16556**	0.401	**999516**	0.401	629000	**0.105**	**29888**	**0.094**	**21473**	0.456	**1603460**	**0.431**	**1065602**

(d) *Circle (C) and Ellipse (E) on Apple M3 Max (binned) GPU*

Method	MSAA = None								MSAA = 4×							
	1				10×10				1				10×10			
	C		E		C		E		C		E		C		E	
	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}	t_f	N_{hl}
Ear clipping	**0.189**	35380	0.175	30711	**0.196**	287708	0.193	229402	0.264	35380	**0.221**	30711	0.273	287708	0.248	229402
Ear clipping (opt.)	0.190	35380	0.174	26967	0.197	287708	0.193	209300	0.264	35380	0.222	26967	0.273	287708	0.248	209300
CDT	0.191	39160	**0.172**	45109	**0.196**	321632	0.198	1813826	0.265	39160	0.225	45109	0.277	321632	0.310	1813826
CDT (opt.)	0.190	**33394**	0.175	23639	**0.196**	**252804**	**0.192**	251754	**0.263**	**33394**	0.222	23639	**0.272**	**252804**	0.248	251754
GMA	0.192	47384	0.176	35181	0.197	384404	0.193	297688	0.268	47384	0.224	35181	0.277	384404	0.250	297688
MWT	0.191	39822	0.176	**23253**	0.198	293620	**0.192**	**180480**	0.265	39822	**0.221**	**23253**	**0.272**	293620	**0.247**	**180480**

We evaluated five scenes: three SVG files of different complexity (Figure 2), and a procedurally generated circle and ellipse with 512 vertices, using axis ratio 0.5 for the latter. For each scene, all triangulations used the same vertex and triangle counts. Every scene was rendered with a single instanced indexed draw call. To normalize the triangulations from the various methods for GPU vertex consumption, we applied meshoptimizer's [Kap] vertex cache, overdraw, and vertex fetch optimizations to every triangulation.

Each scene is rendered on all platforms with both no MSAA and with MSAA at 4×. We also measured grid-based instancing using 1 instance (baseline) and a higher-load patterned setting: 10×10 on Apple and 100×100 on NVIDIA. We use different instance counts to ensure stable timing. On the NVIDIA GPU, low load yields disproportionately high frame-time variance, whereas on Apple, overly dense instancing can make primitives too small to span multiple tiles, masking tile-related effects. Tile size is set to 32×32 pixels on the Apple device. The viewport is 2800×1800 on the Apple and 1920×1080 on the NVIDIA system. All runs use optimized builds and a fixed fragment shader that binds a 2K texture and performs texture sampling. We report the median frame time over 200 consecutive frames, summarized in Table 1.



Scene 'S' *Scene '7'* *Scene '2'*

Figure 2: *The polygon conversion of the SVG files that were used in our performance measurements.*

We additionally report a summed helper-lane invocation count N_{hl}, computed per 2×2 fragment quad and accumulated over

Table 2: *Total unique edge length (L_e) and triangulation time [ms] (t_{tri}) measured on the NVIDIA device. EC: ear clipping (Earcut). Opt: edge flip postprocessing.*

Scene	Metric	Method					
		CDT	CDT opt	EC	EC opt	MWT	GMA
S	$L_e \downarrow$	41.79	34.57	49.76	34.32	**33.76**	48.88
	$t_{\text{tri}} \downarrow$	1.51	1.98	**1.42**	1.61	48.55	65.15
7	$L_e \downarrow$	21.39	17.92	23.37	16.42	**16.39**	23.65
	$t_{\text{tri}} \downarrow$	1.45	1.44	1.43	**1.38**	6.59	7.64
2	$L_e \downarrow$	39.11	28.37	43.84	27.45	**27.33**	42.13
	$t_{\text{tri}} \downarrow$	1.50	1.57	2.06	**1.47**	28.99	43.62
Circle	$L_e \downarrow$	22.29	20.98	**20.57**	**20.57**	**20.57**	20.78
	$t_{\text{tri}} \downarrow$	0.54	1.95	**0.44**	0.44	69.72	319.15
Ellipse	$L_e \downarrow$	132.31	22.83	16.09	15.34	**15.28**	16.01
	$t_{\text{tri}} \downarrow$	0.52	0.92	**0.41**	0.46	69.48	322.24

the frame using hardware-appropriate quad/wave operations. While the accumulation mechanism differs across platforms, the reported metric represents the same per-quad helper participation. Helper-lane counting was disabled in performance measurements.

Edge length statistics are computed from the raw 2D vertex positions prior to any transformation. Because all measurements are performed on planar geometry rendered without transforms, these lengths correspond to screen-space edge lengths up to a constant scale factor. We ensure that no edge is counted twice in total edge length computations.

To quantify tile-based amplification on Apple GPUs, we compute conservative triangle-tile overlap statistics in pixel space. For each triangle, we form its axis-aligned bounding box (AABB), convert the AABB extent to a tile range, and count all tiles whose AABBs intersect the triangle AABB. We accumulate triangles-per-tile (Tr/T) by inserting triangle IDs into per-tile sets and taking the median and P95 over non-empty tiles. Although the overlap test is conservative, using AABB-AABB rather than exact triangle-tile coverage, these metrics capture tile list pressure and per-tile edge workload. In Table 3, Tri/T_{med} exhibits strong correlation with median frame time, while the tail metric Tri/T_{95} captures hot-tile amplification that becomes more pronounced under MSAA 4×.

Table 3: *Apple TBDR tile-amplification metrics vs. median frame time [ms] per scene. Lower is better. Measurements were done with MSAA off and instance count set to 10x10.*

Method	S			7			2		
	Tri/T_{med}	Tri/T_{95}	t_f	Tri/T_{med}	Tri/T_{95}	t_f	Tri/T_{med}	Tri/T_{95}	t_f
CDT	15.00	34.00	0.187	25.00	47.00	0.186	24.00	63.00	0.196
CDT opt.	**13.00**	27.00	0.185	**8.00**	19.00	**0.172**	13.00	25.00	0.178
EC	20.00	36.00	0.189	10.00	20.00	0.173	17.00	37.00	0.182
EC opt.	**13.00**	28.00	0.184	**8.00**	18.00	**0.172**	13.00	**24.00**	0.178
GMA	18.00	40.00	0.189	10.00	21.00	0.173	17.00	33.00	0.184
MWT	**13.00**	**26.00**	**0.183**	**8.00**	**17.00**	**0.172**	**12.00**	**24.00**	**0.177**

4. Conclusions

Across architectures and at fixed triangle count, median GPU frame time is largely explained by edge-length. On NVIDIA, this component is reflected by helper-lane participation N_{hl}, whereas on Apple it manifests more directly through tile amplification and per-tile edge workload (captured by Tri/T_{med} and Tri/T_{95}), with N_{hl} remaining informative mainly through co-variation (Tables 1a, 1b and 3). Accordingly, triangulations that reduce triangle boundary-driven coverage work tend to reduce median frame time, and the coupling strengthens under MSAA or high instance counts. Translations within the quad and planar rotations do not meaningfully alter the relationship between t_f and N_{hl}.

On NVIDIA, triangulation choice measurably changes median frame time under load. In the SVG scenes at 100×100 patterns, smaller total length reduces t_f by up to ≈ 20–25% in our tests, especially under MSAA 4× (e.g., scene S: 0.782 → 0.614 ms from ear clipping to MWT; scene 2: 0.685 → 0.522 ms, Table 1a). With MSAA off, the effect remains significant on edge-heavy inputs (scene 2: CDT 0.267 → 0.218 ms for MWT, Table 1a). MWT and GMA typically achieve the lowest boundary-driven metrics but at substantially higher preprocessing cost, reaching two orders of magnitude on the 512-vertex analytic shapes (Table 2). This makes these more applicable to static geometry. For fast baselines, CDT/ear clipping with edge-flip refinement often narrows the GPU performance gap at modest CPU cost (e.g., scene S at 100×100 and MSAA 4×: 0.782 → 0.616 ms for ear clipping opt., Table 1a).

Importantly, total unique Euclidean edge length L_e and N_{hl} are only partial proxies for the relevant rasterization cost. Local diagonal shortening can reduce L_e yet increase helper activity (scene S, MSAA off, 100×100: ear clipping L_e 49.76 → 34.32 while N_{hl} 2.05 M→ 2.14 M, Tables 1a and 2), and under MSAA the median frame time can deviate from helper-lane-implied trends (ellipse, MSAA 4×, 100×100: CDT opt. reduces N_{hl} 4.05 M→ 1.73 M yet increases t_f 0.460 → 0.485 ms, Table 1c). These findings indicate that triangle shape, orientation, and per-sample/tile coverage terms can dominate beyond raw edge length and helper counts. Practically, we therefore recommend guiding triangulation and refinement by hardware-proximate metrics (N_{hl} on immediate-mode GPUs and tile amplification on TBDR GPUs) and validating under the intended MSAA/instancing regime, rather than relying on total edge length, L_e, alone.

References

[Che89] CHEW L. P.: Constrained delaunay triangulations. *Algorithmica 4* (1989), 97–108. doi:10.1007/BF01553881. 1

[DMM95] DICKERSON M. T., MCELFRESH S. A., MONTAGUE M. H.: New algorithms and empirical findings on minimum weight triangulation heuristics. In *Proc. 11th Annu. Symp. Comput. Geom. (SoCG)* (1995), ACM, pp. 238–247. 2

[Kap] KAPOULKINE A.: meshoptimizer. URL: https://github.com/zeux/meshoptimizer. 3

[Kli80] KLINCSEK G. T.: Minimal triangulations of polygonal domains. In *Annals of Discrete Mathematics*, vol. 9. North-Holland, 1980, pp. 121–123. doi:10.1016/S0167-5060(08)70044-X. 1

[Kog25] KOGLER S.: mapbox-earcut. Python Package Index (PyPI), 2025. Version 2.0.0 (released 2025-11-16). URL: https://pypi.org/project/mapbox-earcut/. 2

[Mei75] MEISTERS G. H.: Polygons have ears. *The American Mathematical Monthly 82*, 6 (1975), 648–651. doi:10.2307/2319703. 1

[Per09] PERSSON E.: Triangulation. Humus (personal website), comments page, Jan. 2009. Published January 14, 2009. Accessed 2025-12-20. URL: https://www.humus.name/index.php?page=Comments&ID=228&start=0. 1, 2

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry* (1996), vol. 1148 of *LNCS*, Springer, pp. 203–222. 2