VERTAISARVIOITU
KOLLEGIALT GRANSKAD
PEER-REVIEWED
www.tsv.fi/tunnus

# A simple technique for unstructured mesh generation via adaptive finite elements

Tom Gustafsson

**Summary**  This work describes a concise algorithm for the generation of triangular meshes with the help of standard adaptive finite element methods. We demonstrate that a generic adaptive finite element solver can be repurposed into a triangular mesh generator if a robust mesh smoothing algorithm is applied between the mesh refinement steps. We present an implementation of the mesh generator and demonstrate the resulting meshes via examples.

## Introduction

Many numerical methods for partial differential equations (PDE's), such as the finite element method (FEM) and the finite volume method (FVM), are based on splitting the domain of the solution into primitive shapes such as triangles or tetrahedra. The collection of the primitive shapes, i.e. the computational mesh, is used to define the discretisation, e.g., in the FEM, the shape functions are polynomial in each mesh element, and in the FVM, the discrete fluxes are defined over the cell edges or faces.

This article describes a simple approach for the triangulation of two-dimensional polygonal domains. The process can be summarised as follows:

1. Find a constrained Delaunay triangulation (CDT) of the polygonal domain using the corner points as input vertices and the edges as constraints.

2. Solve the Poisson equation with the given triangulation and the FEM.

3. Split the triangles with the largest error indicator using adaptive mesh refinement techniques.

4. Apply centroidal patch tesselation (CPT) smoothing to the resulting triangulation.

5. Go to step 2.

It is noteworthy that the steps 2, 3 and 5 correspond exactly to what is done in any implementation of the standard adaptive FEM; cf. Verfürth [21] who calls it the *adaptive process*.

The goal of this work is to demonstrate that if the mesh smoothing algorithm of step 4 is chosen properly, the adaptive process tends to produce reasonable meshes even if the initial mesh is of low quality. Thus, we demonstrate that the adaptive process—together with an implementation of the CDT and a robust mesh smoothing algorithm—can act as a simple triangular mesh generator.

## Prior work

Two popular techniques for generating unstructured meshes are those based on the *advancing front technique* [14] or the *Delaunay mesh refinement* [8, 17, 19]. In addition, there exist several less known techniques such as *quadtree meshing* [23], *bubble packing* [20], and hybrid techniques combining some of the above [15].

Some existing techniques bear similarity to ours. For example, Bossen–Heckbert [4] start with a CDT and improve it by relocating the nodes. Instead of randomly picking nodes for relocation, we use a finite element error indicator that guides the refinement. Instead of doing local modifications, we split simultaneously all triangles that have their error indicators above a predefined threshold.

Persson–Strang [16] describe another technique based on iterative relocation of the nodes. An initial mesh is given by a structured background mesh which is then relaxed by interpreting the edges as a precompressed truss structure. The structure is forced inside a given domain by expressing the boundary using signed distance functions and interpreting the signed distance as an external load acting on the truss. In contrast to the present approach, the geometry description is implicit, i.e. the boundary is defined as the zero set of a user-given distance function.

We do not expect our technique to surpass the existing techniques in the quality of the resulting meshes or in the computational efficiency. However, the algorithm can be easier to understand for those with a background in the finite element method and, hence, it may be a viable candidate for supplementing adaptive finite element solvers with basic mesh generation capabilities.

## Components of the mesh generator

The input to our mesh generator is a sequence of $N$ corner points

$$\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N), \quad \mathcal{C}_j \in \mathbb{R}^2, \quad j = 1, \ldots, N,$$

that form a polygon when connected by the edges

$$(\mathcal{C}_1, \mathcal{C}_2), (\mathcal{C}_2, \mathcal{C}_3), \ldots, (\mathcal{C}_{N-1}, \mathcal{C}_N), (\mathcal{C}_N, \mathcal{C}_1).$$

We do not allow self-intersecting polygons although the algorithm generalises to polygons with polygonal holes. The corresponding domain is denoted by $\Omega_\mathcal{C} \subset \mathbb{R}^2$.

### Constrained Delaunay triangulation

A *triangulation* of $\Omega_\mathcal{C}$ is a collection of nonoverlapping nondegenerate triangles whose union is exactly $\Omega_\mathcal{C}$. Our initial triangulation $\mathcal{T}_0$ is a *constrained Delaunay triangulation*

(CDT) of the input vertices $\mathcal{C}$ with the edges $(\mathcal{C}_1, \mathcal{C}_2)$, $(\mathcal{C}_2, \mathcal{C}_3)$, ..., $(\mathcal{C}_{N-1}, \mathcal{C}_N)$, $(\mathcal{C}_N, \mathcal{C}_1)$ constrained to be a part of the resulting triangulation and the triangles outside the polygon removed; cf. Chew [7] for the exact definition of a CDT and an algorithm for its construction.

An example initial triangulation of a polygon with a spiral-shaped boundary is given in Figure 1. It is obvious that the CDT is not always a high quality computational mesh due to the presence of arbitrarily small angles. Thus, we seek to improve the initial triangulation by iteratively adding new triangles, and smoothing the mesh. Note that the remaining steps do not assume the use of CDT as an initial triangulation—any triangulation with the prescribed edges will suffice.
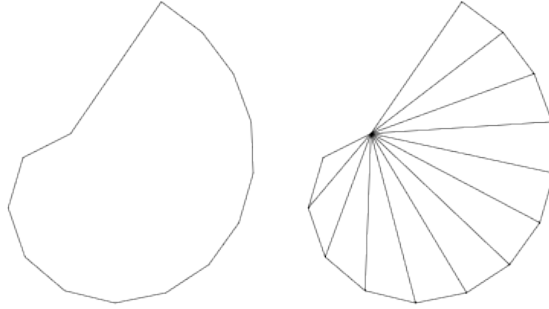


Figure 1: A spiral-shaped boundary approximated by linear segments and the corresponding CDT with the triangles outside of the polygon removed. An example from the documentation of the Triangle mesh generator [18].

*Solving the Poisson equation*

In order to decide on the placement of the new vertices and triangles, we solve the Poisson equation[1] using the FEM and evaluate the corresponding a posteriori error estimator. The triangles that have the highest values of the error estimator are refined, i.e. split into smaller triangles.

The problem reads: find $u : \Omega_{\mathcal{C}} \to \mathbb{R}$ satisfying

$$-\Delta u = 1 \quad \text{in } \Omega_{\mathcal{C}}, \tag{1}$$
$$u = 0 \quad \text{on } \partial\Omega_{\mathcal{C}}. \tag{2}$$

The finite element method is used to numerically solve the weak formulation: find $u \in V$ such that

$$\int_{\Omega_{\mathcal{C}}} \nabla u \cdot \nabla v \, \mathrm{d}x = \int_{\Omega_{\mathcal{C}}} v \, \mathrm{d}x \quad \forall v \in V, \tag{3}$$

where $w \in V$ if $w|_{\partial\Omega_{\mathcal{C}}} = 0$ and $\int_{\Omega_{\mathcal{C}}} (\nabla w)^2 \, \mathrm{d}x < \infty$.

We denote the $k$th triangulation of the domain $\Omega_{\mathcal{C}}$ by $\mathcal{T}_k$, $k = 0, 1, \ldots$, and use the piecewise linear polynomial space

$$V_h^k = \{ v \in V : v|_T \in P_1(T) \ \forall T \in \mathcal{T}_k \},$$

---

[1]The choice of the Poisson equation is motivated by the following heuristic observation: a *quality mesh* is often synonymous with a *good mesh for the finite element solution of the Poisson equation*.

where $P_1(T)$ denotes the set of linear polynomials over $T$. The finite element method corresponding to the $k$th iteration reads: find $u_h^k \in V_h^k$ such that

$$\int_{\Omega_C} \nabla u_h^k \cdot \nabla v_h \, dx = \int_{\Omega_C} v_h \, dx \quad \forall v_h \in V_h^k. \tag{4}$$

The local a posteriori error estimator reads

$$\eta_T(u_h^k) = \sqrt{h_T^2 A_T^2 + \frac{1}{2} h_T \int_{\partial T \setminus \partial \Omega_C} ([\![ \nabla u_h^k \cdot \boldsymbol{n} ]\!])^2 \, ds}, \quad T \in \mathcal{T}_k, \tag{5}$$

where $A_T$ is the area of the triangle $T$ and $h_T$ is the length of its longest edge, $[\![ w ]\!]|_{\partial T \setminus \partial \Omega_C}$ denotes the jump in the values of $w$ over $\partial T \setminus \partial \Omega_C$, and $\boldsymbol{n}$ is a unit normal vector to $\partial T$. The error estimator $\eta_T$ is evaluated for each triangle after solving (4). Finally, a triangle $T \in \mathcal{T}_k$ is marked for refinement if

$$\eta_T > \theta \max_{T' \in \mathcal{T}_k} \eta_{T'}, \tag{6}$$

where $0 < \theta < 1$ is a parameter controlling the amount of elements to split during each iteration. [21]

### Red-green-blue refinement

The triangles marked for refinement by the rule (6) are split into four. In order to keep the rest of the triangulation conformal, i.e. to not have nodes in the middle of an edge, the neighboring triangles are split into two or three by the so-called red-green-blue (RGB) refinement; cf. Carstensen [5]. Using RGB refinement to the example of Figure 1 is depicted in Figure 2.
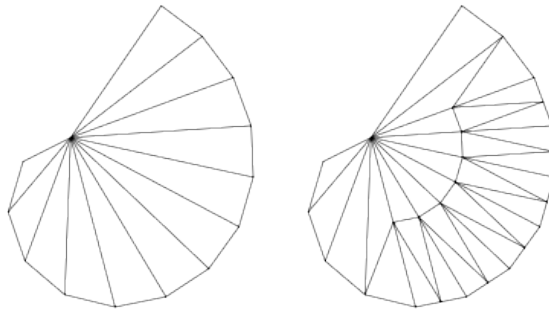


Figure 2: (Left.) The initial triangulation. (Right.) The resulting triangulation after a solve of (4) and an adaptive RGB refinement.

### Centroidal patch triangulation smoothing

We use a mesh smoothing approach introduced by Chen–Holst [6] who refer to the algorithm as *centroidal patch triangulation* (CPT) smoothing. The idea is to repeatedly move the interior vertices to the area-weighted averages of the barycentres of the surrounding triangles. The CPT smoothing is combined with an edge flipping algorithm, also described in Chen–Holst [6], to improve the quality of the resulting triangulation. The mesh smoother is applied to the spiral-shaped domain example in Figure 3.
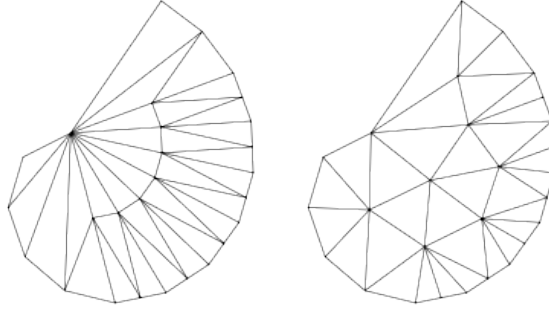
Figure 3: (Left.) The adaptively refined triangulation. (Right.) The resulting triangulation after smoothing and edge flipping.

## The mesh generation algorithm

In previous sections we presented an overview of all the components of the mesh generation algorithm. The resulting mesh generator is now summarised in Algorithm 1. The total number of refinements $M$ is a constant to guarantee the termination of the algorithm. Nevertheless, in practice and in our implementation the refinement loop is terminated when a quality criterion is satisfied, e.g., when the average minimum angle of the triangles is above a predefined threshold. The entire mesh generation process for the spiral-shaped domain example is given in Figure 4.

---

**Algorithm 1** Pseudocode for the triangular mesh generator

---

**Precondition:** $\mathcal{C}$ is a sequence of corner points for a polygonal domain
**Precondition:** $M$ is the total number of refinements

1: **function** ADAPTMESH($\mathcal{C}$)
2:     $\mathcal{T}_0' \leftarrow \text{CDT}(\mathcal{C})$
3:     $\mathcal{T}_0 \leftarrow \mathcal{T}_0'$ with triangles outside of $\mathcal{C}$ removed
4:     **for** $k \leftarrow 1$ to $M$ **do**
5:         $\mathcal{T}_k' \leftarrow \text{RGB}(\mathcal{T}_{k-1}, \{\eta_T : T \in \mathcal{T}_{k-1}\})$
6:         $\mathcal{T}_k \leftarrow \text{CPT}(\mathcal{T}_k')$
7:     **end for**
8:     **return** $\mathcal{T}_N$
9: **end function**

---

## Implementation and example meshes

We created a prototype of the mesh generator in Python for computational experiments [9]. The implementation relies heavily on the scientific Python ecosystem [22]. It includes source code from pre-existing Python packages `tri` [3] (CDT implementation, ported from Python 2) and the older MIT-licensed versions of `optimesh` [2] (CPT smoothing) and `meshplex` [1] (edge flipping). Moreover, it performs adaptive mesh refinement using `scikit-fem` [11] and visualisation using `matplotlib` [13].

Some example meshes are given in Figure 5. By default, our implementation uses the *average* triangle quality[2] above 0.9 as a stopping criterion which can lead to individual

---

[2]Triangle quality is defined as two times the ratio of the incircle and circumcircle radii.
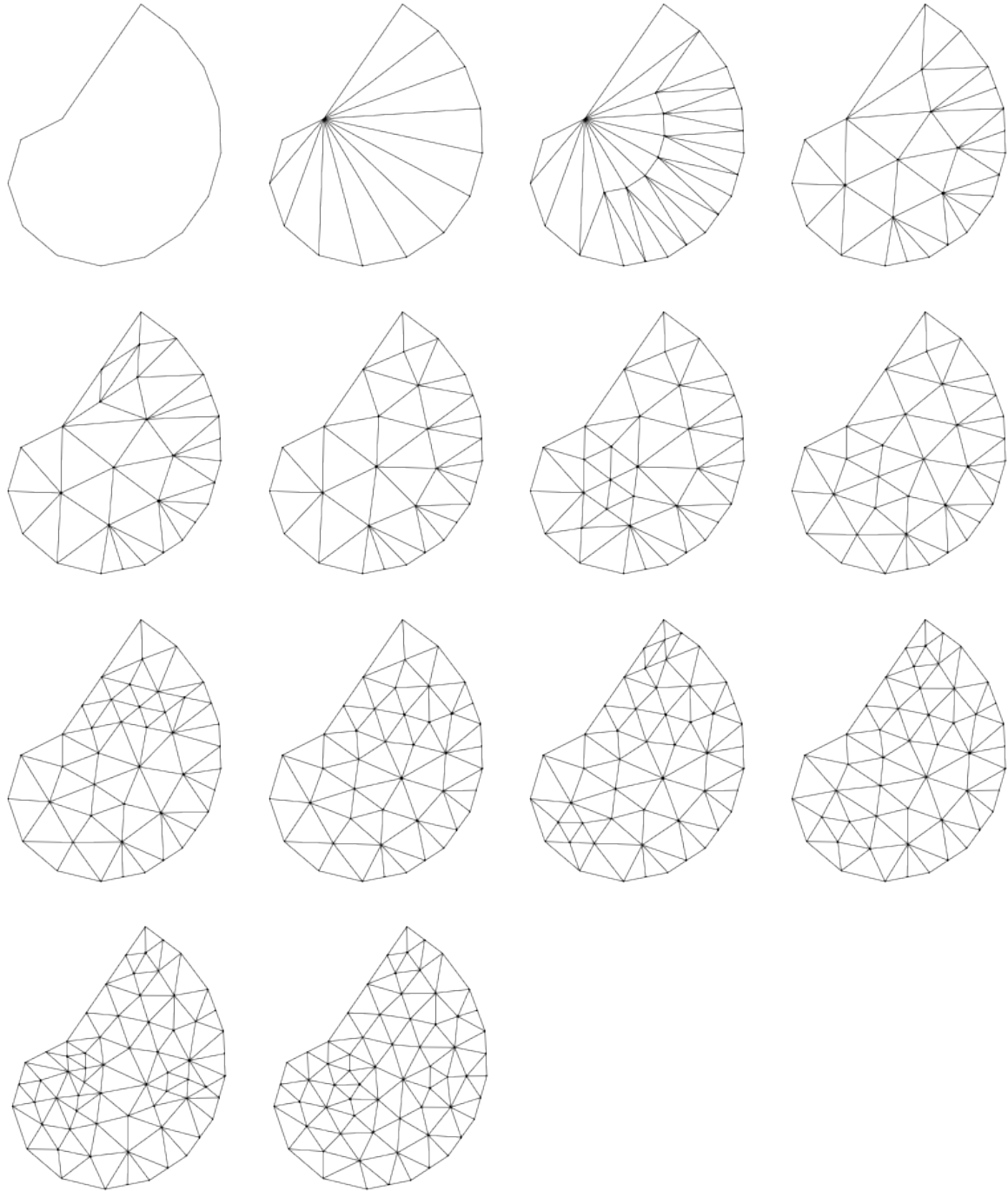
Figure 4: The entire mesh generation process for the spiral-shaped domain example from left-to-right, top-to-bottom.

slit triangles. This is visible especially in the last two examples that have small interior angles on the boundary.
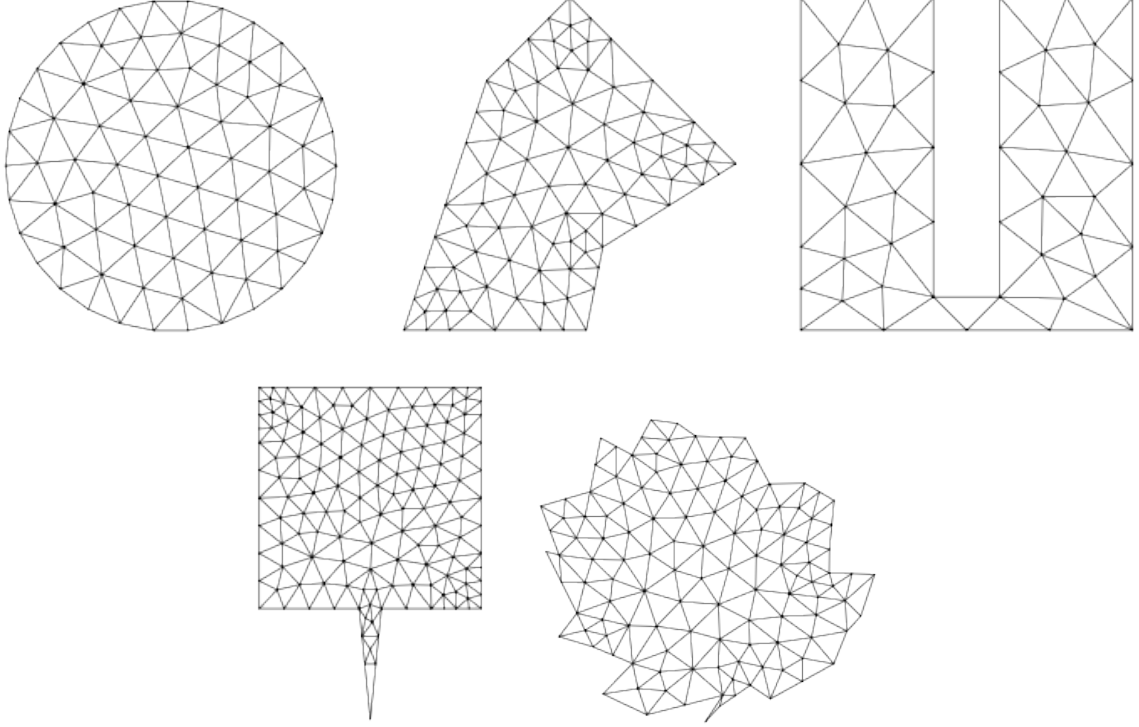


Figure 5: Some example meshes generated using our implementation of the proposed algorithm. The source code for the examples is available in [10].

### An example application

This example utilises a variant of Algorithm 1 with steps 2 and 3 modified to allow for the inclusion of polygonal holes [9]. The holes are treated similarly as the sequence of corner points $\mathcal{C}$ in step 2 and the triangles inside the holes are removed in step 3. We consider the domain

$$\Omega = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \geq a^2, -4a \leq x \leq 4a, -2a \leq y \leq 2a\}$$

which is approximated by the triangular mesh given in Figure 6. We split the boundary of the domain into two as $\partial\Omega = \Gamma \cup (\partial\Omega \setminus \Gamma)$ where $\Gamma = \{(x, y) \in \mathbb{R}^2 : |x| = 4a\}$, and consider the following linear elastic boundary value problem: find $\boldsymbol{u} : \Omega \to \mathbb{R}^2$ satisfying

$$\operatorname{div} \boldsymbol{\sigma}(\boldsymbol{u}) = \mathbf{0} \qquad \text{in } \Omega, \tag{7}$$

$$\boldsymbol{\sigma}(\boldsymbol{u})\boldsymbol{n} \cdot \boldsymbol{n} = g \qquad \text{on } \Gamma, \tag{8}$$

$$\boldsymbol{\sigma}(\boldsymbol{u})\boldsymbol{n} \cdot \boldsymbol{t} = 0 \qquad \text{on } \Gamma, \tag{9}$$

$$\boldsymbol{\sigma}(\boldsymbol{u})\boldsymbol{n} = \mathbf{0} \qquad \text{on } \partial\Omega \setminus \Gamma, \tag{10}$$

where

$$\boldsymbol{\sigma}(\boldsymbol{u}) = 2\mu\,\boldsymbol{\epsilon}(\boldsymbol{u}) + \lambda\operatorname{tr}\boldsymbol{\epsilon}(\boldsymbol{u})\boldsymbol{I} \in \mathbb{R}^{2\times2}, \quad \boldsymbol{\epsilon}(\boldsymbol{u}) = \frac{1}{2}\left(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T\right) \in \mathbb{R}^{2\times2},$$
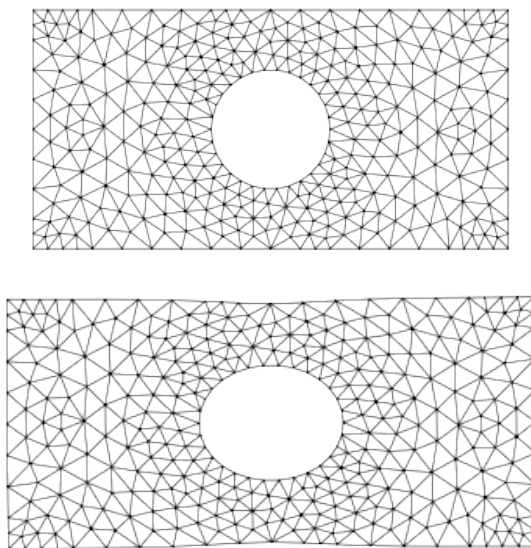
Figure 6: The original and the deformed meshes: the vertices of the original mesh are deformed using the finite element approximation of $\boldsymbol{u}$.
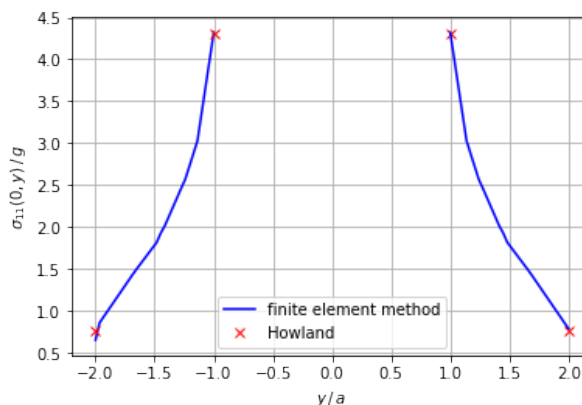


Figure 7: A comparison of the finite element approximation of the stress $\sigma_{11}(0, y)$ for $y \in [-2a, 2a] \setminus (-a, a)$ and the pointwise reference values from Howland [12].

and $\boldsymbol{n} \in \mathbb{R}^2$ denotes the outward unit normal, $\boldsymbol{t} \in \mathbb{R}^2$ is the corresponding unit tangent, $\boldsymbol{I} \in \mathbb{R}^{2 \times 2}$ is the identity matrix. Additional parameters describing the problem are the Lamé parameters which are chosen to be equal $\mu = \lambda$, the radius of the hole $a$, and a uniform tensile force $g$.

We solve the above problem using the finite element method and piecewise-quadratic elements [21]. The rigid body motion is eliminated by constraining the bottommost node on $x = 0$ to zero. The resulting deformed mesh, with the vertices of the original mesh displaced by the finite element approximation of $\boldsymbol{u}$, is given in Figure 6. A reference value of the stress $(\boldsymbol{\sigma}(\boldsymbol{u}(x, y)))_{11} = \sigma_{11}(x, y)$ at internal and external boundaries along $x = 0$ is given by Howland [12] via successive approximation. The reference values are $\sigma_{11}(0, 2a) = \sigma_{11}(0, -2a) \approx 0.75g$ and $\sigma_{11}(0, a) = \sigma_{11}(0, -a) \approx 4.3g$. A comparison of the finite element approximation and the reference values is given in Figure 7.

## Conclusions

We introduced an algorithm for the generation of triangular meshes for explicit polygonal domains based on the standard adaptive finite element method and centroidal patch triangulation smoothing. We presented a prototype implementation which demonstrates that many of the resulting meshes are reasonable and have an average triangle quality equal to or above 0.9.

A majority of the required components are likely to be available in an existing implementation of the adaptive finite element method. Therefore, the algorithm can be a compelling candidate for supplementing an existing adaptive finite element solver with basic mesh generation capabilities. Technically the approach extends to three dimensions although in practice the increase in computational effort can be significant and the quality of the resulting tetrahedralisations has not been investigated.

## Aknowledgement

## References

[1] *meshplex 0.12.1*. https://web.archive.org/web/20201023074645/https://pypi.org/project/meshplex/0.12.1/. Accessed: 2020-10-23.

[2] *optimesh 0.6.1*. https://web.archive.org/web/20201023080856/https://pypi.org/project/optimesh/0.6.1/. Accessed: 2020-10-23.

[3] *tri 0.3*. https://web.archive.org/web/20201023101325/https://pypi.org/project/tri/0.3/. Accessed: 2020-10-23.

[4] F. J. BOSSEN AND P. S. HECKBERT, *A pliant method for anisotropic mesh generation*, in 5th International Meshing Roundtable, 1996, pp. 63–74.

[5] C. CARSTENSEN, *An adaptive mesh-refining algorithm allowing for an $H^1$ stable $L^2$ projection onto Courant finite element spaces*, Constructive Approximation, 20 (2004), pp. 549–564, https://doi.org/10.1007/s00365-003-0550-5.

[6] L. CHEN AND M. HOLST, *Efficient mesh optimization schemes based on Optimal Delaunay Triangulations*, Computer Methods in Applied Mechanics and Engineering, 200 (2011), pp. 967–984, https://doi.org/10.1016/j.cma.2010.11.007.

[7] L. P. CHEW, *Constrained Delaunay triangulations*, Proceedings of the third annual symposium on computational geometry - SCG '87, (1987), https://doi.org/10.1145/41958.41981.

[8] L. P. CHEW, *Guaranteed-quality triangular meshes*, (1989), https://doi.org/10.21236/ada210101.

[9] T. GUSTAFSSON, *kinnala/adaptmesh v0.2.0*, (2021), https://doi.org/10.5281/zenodo.4453588.

[10] T. GUSTAFSSON, *kinnala/paper-meshgen v3*, (2021), https://doi.org/10.5281/zenodo.4574540.

[11] T. GUSTAFSSON AND G. D. MCBAIN, *scikit-fem: A Python package for finite element assembly*, Journal of Open Source Software, 5 (2020), p. 2369, https://doi.org/10.21105/joss.02369.

[12] R. HOWLAND, *On the stresses in the neighbourhood of a circular hole in a strip under tension*, Philosophical Transactions of the Royal Society of London, Series A, 229 (1930), pp. 49–86, https://doi.org/10.1098/rsta.1930.0002.

[13] J. D. HUNTER, *Matplotlib: A 2d graphics environment*, Computing in science & engineering, 9 (2007), pp. 90–95, https://doi.org/10.1109/MCSE.2007.55.

[14] R. LÖHNER AND P. PARIKH, *Generation of three-dimensional unstructured grids by the advancing-front method*, International Journal for Numerical Methods in Fluids, 8 (1988), pp. 1135–1149, https://doi.org/10.1002/fld.1650081003.

[15] D. J. MAVRIPLIS, *An advancing front Delaunay triangulation algorithm designed for robustness*, Journal of Computational Physics, 117 (1995), pp. 90–101, https://doi.org/10.1006/jcph.1995.1047.

[16] P.-O. PERSSON AND G. STRANG, *A simple mesh generator in MATLAB*, SIAM review, 46 (2004), pp. 329–345, https://doi.org/10.1137/S0036144503429121.

[17] J. RUPPERT, *A Delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms, 18 (1995), pp. 548–585, https://doi.org/10.1006/jagm.1995.1021.

[18] J. R. SHEWCHUK, *Triangle: Engineering a 2d quality mesh generator and Delaunay triangulator*, in Workshop on Applied Computational Geometry, Springer, 1996, pp. 203–222, https://doi.org/10.1007/BFb0014497.

[19] J. R. SHEWCHUK, *Delaunay refinement algorithms for triangular mesh generation*, Computational Geometry, 22 (2002), pp. 21–74, https://doi.org/10.1016/s0925-7721(01)00047-5.

[20] K. SHIMADA AND D. C. GOSSARD, *Bubble mesh*, Proceedings of the third ACM symposium on solid modeling and applications, (1995), https://doi.org/10.1145/218013.218095.

[21] R. VERFÜRTH, *A Posteriori Error Estimation Techniques for Finite Element Methods*, Oxford University Press, 2013, https://doi.org/10.1093/acprof:oso/9780199679423.001.0001.

[22] P. VIRTANEN, R. GOMMERS, T. E. OLIPHANT, M. HABERLAND, T. REDDY, D. COURNAPEAU, E. BUROVSKI, P. PETERSON, W. WECKESSER, J. BRIGHT, ET AL., *SciPy 1.0: fundamental algorithms for scientific computing in Python*, Nature methods, 17 (2020), pp. 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[23] M. YERRY AND M. SHEPHARD, *A modified quadtree approach to finite element mesh generation*, IEEE Computer Graphics and Applications, 3 (1983), pp. 39–46, https://doi.org/10.1109/mcg.1983.262997.

Tom Gustafsson
Aalto University, Department of Mathematics and Systems Analysis
P.O. Box 11100
FI-00076 Aalto
Finland
`firstname.lastname@aalto.fi`