A Universal Predictor-Corrector Type Incremental Algorithm for the Construction of Weighted Straight Skeletons Based on the Notion of Deforming Polygon

Barış İrhan[1] (ID)

[1]Arkitech Advanced Construction Technologies, barisirhan@gmail.com

Corresponding author: Barış İrhan, barisirhan@gmail.com

Abstract. A new predictor-corrector type incremental algorithm is proposed for the construction of weighted straight skeletons of 2D general planar polygons of arbitrary complexity based on the notion of deforming polygon. Algorithm is general, sound, simple and easy to implement. In the proposed algorithm, the raw input provided by the polygon itself is enough to resolve edge collapse and edge split events. Neither the construction of a kinetic triangulation nor the computation of a motorcycle graph, or any other means, is required. Due to its incremental nature, there is always a room in the algorithm for the interactive construction of the straight skeleton. For example, the edges which start to move at different times could easily be handled by adjusting the weights manually. The proposed algorithm is of predictor-corrector type. In the algorithm, the edge collapse and edge split events are tackled by a completely different novel original approach. In the predictor step, the position of the vertices is advanced in time by direct integration assuming no event. Then predicted positions are corrected by using linear interpolation if there are edge collapse or edge split events within the same increment. In the algorithm edge collapse and edge split events are detected by, respectively, edge swap and edge penetration. Both edge swap and edge penetration have counterparts in computational solid mechanics. Former statement is motivated by impenetrability of matter. Whereas, the second statement is motivated by kinematical constraint posed when two deformable body, or body with itself, come into contact. It simply states that bodies cannot penetrate into each other during contact. The new algorithm has been used to construct roof topology starting from a floor plan of various complexity ranging from simple convex to highly nonconvex with holes. In order to construct, improve and test the building blocks of the underlying algorithm, a graphical user interface, Straight Skeleton Development Kit, has also been developed in parallel by the author using C++ programming language.

1 INTRODUCTION

Consider the closed planar polygon given in Figure 1, where its configuration at different time instances is depicted. In classical straight skeleton problem, each edge of the polygon moves inwards with a constant unit velocity in such a way that it remains essentially parallel to itself throughout the shrinking process, which continues until the polygon collapses down to a segment or a point. The regions swept by the polygon edges throughout the shrinking process construct faces of the straight skeleton structure of the underlying polygon. During the shrinking process, polygon topology might change due to edge collapse and edge split events. The edge collapse event takes place at an instance when at least one edge of the polygon collapses down to a point, whereas the edge split event comes into play as at least one of the vertices hits an edge or another vertex of the polygon. Velocity of the vertices involved changes as these events happen and change the topology. Therefore, these events must be detected and handled properly in order to end up with correct straight skeleton structure.
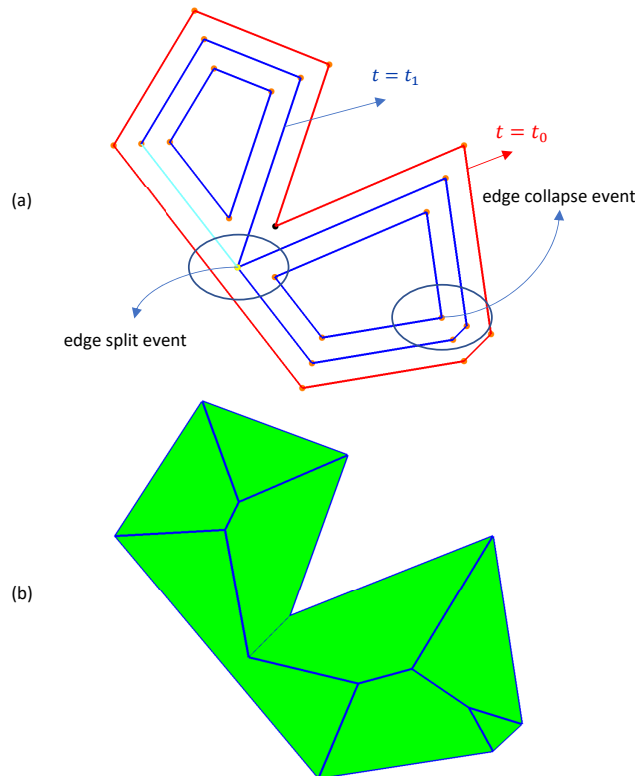

Figure 1: (a) Deforming polygon. (b) Straight skeleton structure.

First algorithm to construct straight skeleton of polygons was appeared in [3]. It is based on wavefront propagation. In the algorithm, a priority queue with respect to the occurrence time of edge events is constructed and processed. To determine split events, it is supplemented by a triangulation algorithm. In [23, 10], an algorithm similar to one given in [3] was presented with a different strategy to handle reflex vertices. In [4], use of kinetic triangulation is made to detect edge collapse and edge split events. Events are signaled as soon as a triangle collapses down to a segment or a point. In case of triangle flip, however, triangle collapse is not always

associated with an edge collapse or edge split event. Therefore, edge flip events must be detected and filtered out which brings additional overburden and shadows the simplicity of the underlying idea. Also note that in the algorithm solution of a quadratic function is required to determine the collapse time of a triangle which brings additional cost and possible loss of accuracy together. In [27, 25], an approach to eliminate flip events based on the Steiner triangulations was proposed and based on this observation a straight skeleton algorithm for arbitrary planar straight line graphs based on generalized motorcycle graph was presented. This work was later extended in [17] to construct straight skeletons of positively weighted polygons based on bisector arrangement.

In [21], the classical algorithm was enhanced by enabling the propagation of the edges at different speeds and hence the weighted version of the straight skeleton algorithm was emerged. The classical and positively weighted version of the algorithm are similar. However, there is a price to pay for the additional flexibility offered by weighted version. Due to additional degree of freedom provided, some ambiguities might arise as the edge collapse event happens. After processing such an event, one might end up with vertices which are colinear. For a colinear vertex, the set of equations governing its propagation velocity turns out to be linearly dependent and hence the solution is undefined. Therefore, after processing edge collapse events, colinear vertices must be found and removed from the forefront polygon. However, removing the colinear vertices is not enough. One has to take another step and decide how the region, line segment, associated with a colinear vertex will continue its propagation [8, 30]. Note also that both the classical and the weighted version of the algorithm might suffer from the ambiguities which arise after detecting and processing multiple vertex events [8, 30, 9]. Before passing note that it has been shown systematically in [8] that weighted straight skeleton has different geometrical characteristics than its unweighted counterpart.

Motorcycle graph was first introduced in [21] to address the essential problems related with reflex vertices while constructing the straight skeletons. The relationship between motorcycle graph and construction of the straight skeletons was first exploited in [13]. Later on several straight skeleton algorithms based on the motorcycle graph have been published [27, 17, 12, 14]. There are also some other algorithms in the literature for the construction of straight skeletons. In [7], an algorithm for monotone polygons is proposed, whereas in [2, 19], algorithms for simple rectilinear polygons are presented.

Weighted straight skeleton algorithm has a wide range of application area. It has been used, (i) to construct roof and terrains [21, 30, 24, 2, 45, 44, 3, 4], (ii) to compute mitered offset curves [5, 37], (iii) in mathematical origami and the fold-and-cut problem [43, 16], (iv) for contour interpolation [31], (v) for polygon decomposition [46], (vi) for digital pathology to improve image segmentation and image analysis [35].

Straight skeleton algorithm has been implemented into several software packages [10, 20, 26]. Of course, without such a tool at hand it is very unlikely to end up with a reliable straight skeleton algorithm.

In this paper a new predictor-corrector type incremental algorithm for the construction of weighted straight skeletons is presented. It is organized as follows. Motivation based on the notion of deforming polygon is given in Section 2, which is followed by the Section 3 where the most striking points of the new algorithm are highlighted. In Section 4, algorthmic details are explained and some useful examples are presented. Application of the new algorithm to roof modelling is given in Section 5. Straight Skeleton Development Kit program developed during the construction of new algorithm is briefly explained in Section 6 and finally conclusions are given in Section 7.

2   MOTIVATION

Computational solid mechanics is a subdiscipline of computational mechanics where the partial differential equations governing the evolution of physical quantities, like linear momentum, energy, mass, etc., are integrated, in space and time, by resorting to vast amount of numerical techniques available [49, 28, 6].

For subsequent developments, we will investigate set of evolution equations for linear momentum which is governed by a second order hyperbolic partial differential equation and has the form

$$\rho\dot{\boldsymbol{v}} = \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} + \rho\boldsymbol{b} \tag{1}$$

where ρ, $\boldsymbol{v}$, $\boldsymbol{\nabla}$, $\boldsymbol{\sigma}$ and $\boldsymbol{b}$ are, respectively, mass density, velocity, gradient vector, stress tensor and body force associated with the material point. Eq. (1) simply states that temporal evolution of linear momentum at a material point is equal to sum of the net momentum flux (force acting at the vicinity like contact) and the source term (force acting at a distance like gravity) associated with the material point. This equation is called as balance equation for linear momentum, momentum equation or wave equation in the literature and in general terms it governs the propagation of stress waves in inelastic media. Note that momentum equation is nothing else but the continuum generalization of Newton's second law of motion, $f = ma$. By using constitutive (stress-strain) and kinematical (strain-position) operators [36], momentum equation can be rewritten in terms of partial derivatives of particle position with respect to space and time.

In computational solid mechanics, Lagrangian description of the motion is generally adapted in order to track the evolution of physical processes. In Lagrangian framework, the motion of each material particle is tracked throughout the course of the deformation process and the existence of a motion function of the form

$$\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{X}, t) \tag{2}$$

is postulated. In Eq. (2), $\boldsymbol{x}$ and $\boldsymbol{X}$ are, respectively, the position of a typical material point at initial and current configuration of the body. In order to be able to define the deformation of a solid, the motion function $\boldsymbol{\phi}$ must possess some properties. To analyze the deformation locally around the material point, use of the gradient of the motion function, also called as deformation gradient, is made. It has the following standard form:

$$\boldsymbol{F}(\boldsymbol{X}, t) = \frac{\partial\boldsymbol{\phi}}{\partial\boldsymbol{X}} = \boldsymbol{\nabla}\boldsymbol{\phi} \tag{3}$$

In order to define the solid deformation, the following conditions must hold for $\boldsymbol{F}$ [36]:

$$\begin{aligned}\boldsymbol{F}d\boldsymbol{X} \neq \boldsymbol{0} \quad &\forall \quad d\boldsymbol{X} \neq \boldsymbol{0} \\ \boldsymbol{F}d\boldsymbol{X} = \boldsymbol{0} \quad &\leftrightarrow \quad d\boldsymbol{X} = \boldsymbol{0}\end{aligned} \tag{4}$$

From physical standpoint above conditions state that an infinitesimal material line element with non-zero length cannot be reduced down to a zero-length line element during deformation. This constraint implies that $\boldsymbol{F}$ must be non-singular and therefore has non-zero determinant, i.e., $det\boldsymbol{F} \neq 0$. The volume of a typical material point at current configuration is given by the expression

$$dv = det\boldsymbol{F}\ dV \tag{5}$$

where dV is its volume at initial configuration. For a solid, the volume of the material point must be positive throughout the course of the deformation process. Therefore, in addition to being nonzero, the determinant of the deformation gradient must also preserve its sign, i.e.,

$$det\boldsymbol{F} > 0 \tag{6}$$

The inequality constraint given in Eq. (6) is also known as the impenetrability of matter. Inability of the motion function given by Eq. (2) to satisfy this constraint is a clear indication of severe deformations. The parts of the body experiencing severe deformations does not behave like a solid anymore but a fluid or a gas. Standard techniques fall short when a solid starts to flow or a fluid starts to deform. The possible change of the material state, which could also be called as change in topology, is the driving force behind several

discretization techniques, like ALE [47, 32], adaptive remeshing [22, 33, 11, 40], adaptive element deletion [29], coupled SPH-FEM [38, 15, 39] etc. if one insists to stay in the Lagrangian framework throughout the course of the deformation process.

Predictor-corrector type numerical algorithms are commonly used to solve problems where the solution space is constrained. For example, in multibody contact problem the solution of the momentum equation given by Eq. (1), i.e., particle postions at current configuration, must also respect the kinetical and kinematical contact constraints [48] which are defined as

$$g_n \geq 0; \quad t_n \leq 0; \quad t_n \, g_n = 0 \tag{7}$$

where, g_n, t_n are, respectively, normal component of the gap vector and normal component of the contact traction. These inequality constraints imply that during contact bodies cannot penetrate into each other, $g_n \geq 0$, and normal component of the contact traction transmitted must be compressive, $t_n \leq 0$. As mentioned before, multibody contact problem described could be solved by employing a predictor-corrector type algorithm. Employing, for example, finite element method [49, 28, 6] to discretize momentum equation in space, the discrete dynamic equilibrium equation for a typical node I is written as

$$m_I \boldsymbol{a}_I = \underbrace{\boldsymbol{f}_I^{ext} - (\boldsymbol{f}_I^{str} + \boldsymbol{f}_I^{con})}_{\boldsymbol{f}_I} \tag{8}$$

where m_I, $\boldsymbol{a}_I$, $\boldsymbol{f}_I^{ext}$, $\boldsymbol{f}_I^{str}$ and $\boldsymbol{f}_I^{con}$ are, respectively, lumped mass, acceleration, external force, internal force and contact force associated with typical node I. For high velocity short duration impact problems, Eq. (8) is usually integrated in time by resorting to an explicit time integration scheme [22, 33, 29]. Using, for example, second order central differences, nodal acceleration at time step t_n can be approximated as

$$\boldsymbol{a}_{In} \approx \frac{\Delta \boldsymbol{x}_{In} - \Delta \boldsymbol{x}_{In-1}}{\Delta t^2} \tag{9}$$

For brevity, we drop subscript I appended for the derivations that follow. Combining Eq. (8) and Eq. (9), an explicit expression for the position update is then obtained as

$$\Delta \boldsymbol{x}_n = \frac{\Delta t^2}{m} \boldsymbol{f}_n + \underbrace{\Delta \boldsymbol{x}_{n-1}}_{known} \tag{10}$$

In the presence of contact interactions, unfortunately, position update cannot be performed in single step because contact forces are not known beforehand. One can remedy such a situation by setting up a predictor-corrector type algorithm. In the algorithm, $\Delta \boldsymbol{x}_n$ is first additively decomposed as

$$\Delta \boldsymbol{x}_n = \Delta \boldsymbol{x}_n^p + \Delta \boldsymbol{x}_n^c \tag{11}$$

where $\Delta \boldsymbol{x}_n^p$, $\Delta \boldsymbol{x}_n^c$ are, respectively, predictor (known) and corrector (unknown) position increments. In the predictor step node-level update is performed assuming no contact. Therefore, for typical node I we have

$$\begin{aligned} \boldsymbol{f}_n^p &= \boldsymbol{f}_n^{ext} - \boldsymbol{f}_n^{int} \\ \boldsymbol{f}_n^{con} &= \boldsymbol{0} \end{aligned} \tag{12}$$

where $\boldsymbol{f}_n^p$ is the total predictor (known) force acting on node I. Using Eq. (10), predictor position increment and predictor position itself are obtained as

$$\Delta \boldsymbol{x}_n^p = \frac{\Delta t^2}{m} \boldsymbol{f}_n^p + \Delta \boldsymbol{x}_{n-1}$$

$$\boldsymbol{x}_{n+1}^p = \boldsymbol{x}_n + \Delta \boldsymbol{x}_n^p \tag{13}$$

Next, based on predictor position given by Eq. (13), a global contact search is performed, and it is checked if there are some nodes for which impenetrability constraint given by Eq. (7) is violated. If there is no violation, predicted positions are correct and we have

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_{n+1}^p \tag{14}$$

On the other hand, if there is violation, one has to take another step and compute corrector position increment $\Delta \boldsymbol{x}_n^c$ due to unknown contact force $\boldsymbol{f}_n^{con}$. Assuming that the contact force is known, the corrector position increment $\Delta \boldsymbol{x}_n^c$ is then obtained as

$$\Delta \boldsymbol{x}_n^c = -\frac{\Delta t^2}{m} \boldsymbol{f}_n^{con} \tag{15}$$

Unknown contact force $\boldsymbol{f}_n^{con}$ can be computed by exploiting the kinematical contact constraint given in Eq. (7), see [29] for details. Note that predictor-corrector type algorithms are also commonly used for the numerical integration of inelastic constitutive equations [41, 34, 42].

The inequality constraint given by Eq. (6), namely impenetrability of the matter, and the realization of predictor-corrector type algorithm for the solution of multibody contact problems motivate a new original algorithm for the construction of weighted straight skeletons of planar polygons.
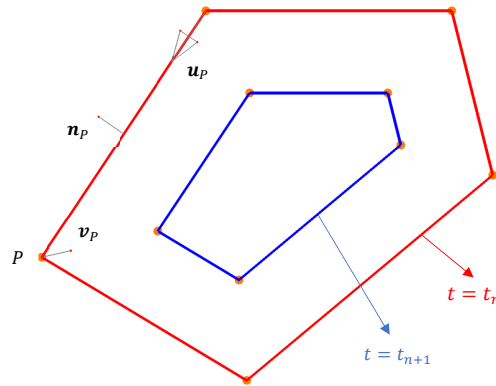

Figure 2: A polygon and a typical vertex.

3   CONTRIBUTION

To the author knowledge, a predictor-corrector type incremental algorithm has never been used before for the construction of straight skeletons. In this work an algorithm of this class is presented. In addition to being original, the algorithm enjoys some very powerful features. First of all, it is simple, intuitive and easy to implement. It is general that is it can be applied to any kind of closed planar polygons of arbitrary complexity equally well. There is no room for exceptional cases. Algorithm works with raw input provided by the underlying

polygon. Construction of a kinetic triangulation or a motorcycle graph, etc. is not necessary. Extension to general planar straight-line graphs seems to be straightforward. In this work, speed of the underlying algorithm has not been studied. In its current version it is slow. To construct, improve and test the building blocks of the algorithm, a graphical user interface, Straight Skeleton Development Kit, has been developed in parallel by the author using C++ programming language.

4 THE ALGORITHM

Consider the closed planar polygon given in Figure 2, where its configuration at time t_n and t_{n+1} is depicted. Each vertex of the polygon changes position in time with a prescribed constant velocity. Recall that velocity of the vertices has a particular form in case of straight skeleton problem such that an edge belonging to the polygon remains essentially parallel to itself during incremental motion. The weight assigned to the edge belonging to a typical vertex P is given by

$$w_P = \boldsymbol{v}_P \cdot \boldsymbol{n}_P \tag{16}$$

where $\boldsymbol{v}_P$ and $\boldsymbol{n}_P$ are, respectively, velocity and normal assigned to the vertex P, see Figure 2. Then for a given time increment

$$\Delta t_n = t_{n+1} - t_n \tag{17}$$

the position of a vertex at time t_{n+1} can exactly be computed by direct integration as

$$\boldsymbol{P}_{n+1} = \boldsymbol{P}_n + \Delta t_n \boldsymbol{v}_P \tag{18}$$
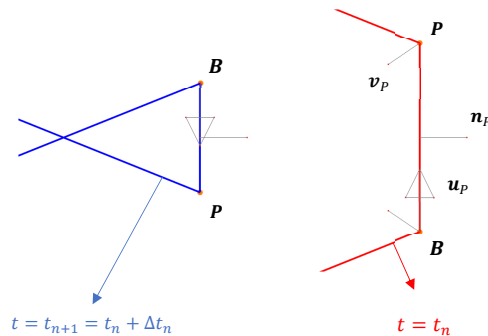

Figure 3: Edge swap.

While the polygon is deforming, the physical constraints given by Eq. (6) and Eq. (7) must be respected. According to impenetrability of matter, see Eq. (6), polygon edges cannot swap during incremental motion, see Appendix A. If edge swap happens, it indicates that there are edges which collapse within the given time increment. Second physical constraint is the so-called contact impenetrability, see Eq. (7). That is during deformation, polygon cannot penetrate into itself, from inside to outside or vice versa. If such a penetration occurs, it indicates that there are edges which split within the given time increment. Position of the vertices computed by direct time integration is then used as a predictor if edge collapse or edge split events take place during incremental motion. In such a case, predicted positions must be corrected. Correction phase requires the determination of the exact time instances of the edge collapse and edge split events.
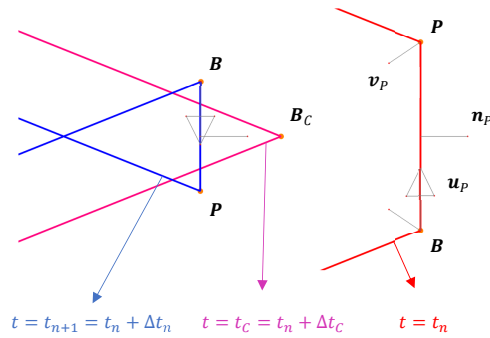
Figure 4: The instance of edge collapse event.

4.1 Edge Collapse Event

Edge collapse event within a time increment is detected by edge swap. An edge is swapped if it changes direction (see Figure 3). Since vertices move along a linear path with a constant velocity, the instance of an edge collapse event can exactly be computed by linear interpolation.

Projected lengths of a given vertex P corresponding to time steps t_n and t_{n+1} are given, respectively, by

$$l_n = (\boldsymbol{P}_n - \boldsymbol{B}_n) \cdot \boldsymbol{u}_P \tag{19}$$

and

$$l_{n+1} = (\boldsymbol{P}_{n+1} - \boldsymbol{B}_{n+1}) \cdot \boldsymbol{u}_P \tag{20}$$
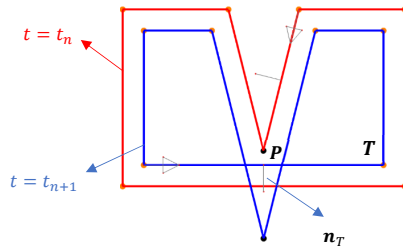
Figure 5: Edge penetration.

Then one can parameterize projected length within a time increment as

$$l = \frac{1}{2}(1 - \xi)l_n + \frac{1}{2}(1 + \xi)l_{n+1} \tag{21}$$

with $\xi \in [-1, 1]$.
At the instance of edge collapse event the following equation must hold:

$$l = 0 \tag{22}$$

Inserting Eq. (21) into Eq. (22), local coordinate corresponding to edge collapse event is then computed as

$$\xi_C = -\frac{l_M}{l_D} \tag{23}$$

where l_M and l_D are defined as

$$l_M = \frac{1}{2}(l_{n+1} + l_n)$$
$$l_D = \frac{1}{2}(l_{n+1} - l_n) \tag{24}$$

One can also parameterize the time increment, similar to Eq. (21), as

$$\Delta t = \frac{1}{2}(1 - \xi)0 + \frac{1}{2}(1 + \xi)\Delta t_n \tag{25}$$

By inserting ξ_C computed in Eq. (23), time increment corresponding to edge collapse event, see Figure 4, is obtained as

$$\Delta t_C = \frac{1}{2}(1 + \xi_C)\Delta t_n \tag{26}$$
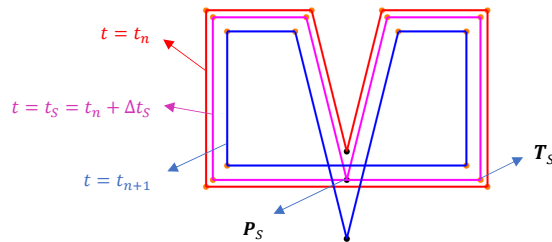
Figure 6: The instance of edge split event.

4.2 Edge Split Event

Edge split event within a time increment is detected by edge penetration (see Figure 5). Since the vertices move along a linear path with constant velocities, exact instance of edge split event can again be computed by using linear interpolation. For this purpose, we define parameterization of vertices P and T as

$$P = \frac{1}{2}(1 - \xi)P_n + \frac{1}{2}(1 + \xi)P_{n+1}$$
$$T = \frac{1}{2}(1 - \xi)T_n + \frac{1}{2}(1 + \xi)T_{n+1} \tag{27}$$

At the instance of edge split event, vertex P must be on the edge belonging to vertex T (see Figure 6). Therefore, the following relation must hold:

$$(\boldsymbol{P}_S - \boldsymbol{T}_S) \cdot \boldsymbol{n}_T = 0 \tag{28}$$

Inserting Eq. (27) into Eq. (28), local coordinate corresponding to edge split event is computed as

$$\xi_S = -\frac{(\boldsymbol{P}_M - \boldsymbol{T}_M) \cdot \boldsymbol{n}_T}{(\boldsymbol{P}_D - \boldsymbol{T}_D) \cdot \boldsymbol{n}_T} \tag{29}$$

where $\boldsymbol{P}_M$ and $\boldsymbol{P}_D$ are defined as

$$\boldsymbol{P}_M = \frac{1}{2}(\boldsymbol{P}_{n+1} + \boldsymbol{P}_n)$$
$$\boldsymbol{P}_D = \frac{1}{2}(\boldsymbol{P}_{n+1} - \boldsymbol{P}_n) \tag{30}$$

Definitions for $\boldsymbol{T}_M$ and $\boldsymbol{T}_D$ follows from Eq. (30). Using the equation for the parameterization of the time increment, Eq. (25), time increment corresponding to edge split event is obtained as

$$\Delta t_S = \frac{1}{2}(1 + \xi_S)\Delta t_n \tag{31}$$

4.3 Predictor-Corrector Algorithm

For a given time increment, predicted position of a typical vertex P is computed by direct integration as

$$\boldsymbol{P}_{n+1}^{pred} = \boldsymbol{P}_n + \Delta t_n \boldsymbol{v}_P \tag{32}$$

After predicted positions are computed, admissibility of the polygon is checked with respect to edge collapse and edge split events using Eq. (22) and Eq. (28). If there is no event taking place, final positions are equal to predicted positions, i.e.,

$$\boldsymbol{P}_{n+1} = \boldsymbol{P}_{n+1}^{pred} \tag{33}$$

If there is an edge collapse event within the time increment, predicted positions are corrected as

$$\boldsymbol{P}_{n+1} = \boldsymbol{P}_{n+1}^{pred} + \Delta \boldsymbol{P}_C \tag{34}$$

with

$$\Delta \boldsymbol{P}_C = (\Delta t_C - \Delta t_n)\boldsymbol{v}_P \tag{35}$$

Similarly, if there is an edge split event within the time increment, predicted positions are corrected as

$$\boldsymbol{P}_{n+1} = \boldsymbol{P}_{n+1}^{pred} + \Delta \boldsymbol{P}_S \tag{36}$$

with

$$\Delta \boldsymbol{P}_S = (\Delta t_S - \Delta t_n)\boldsymbol{v}_P \tag{37}$$

If both edge collapse and edge split events take place within the same increment, the one which happens earlier is considered and the other one is disregarded.
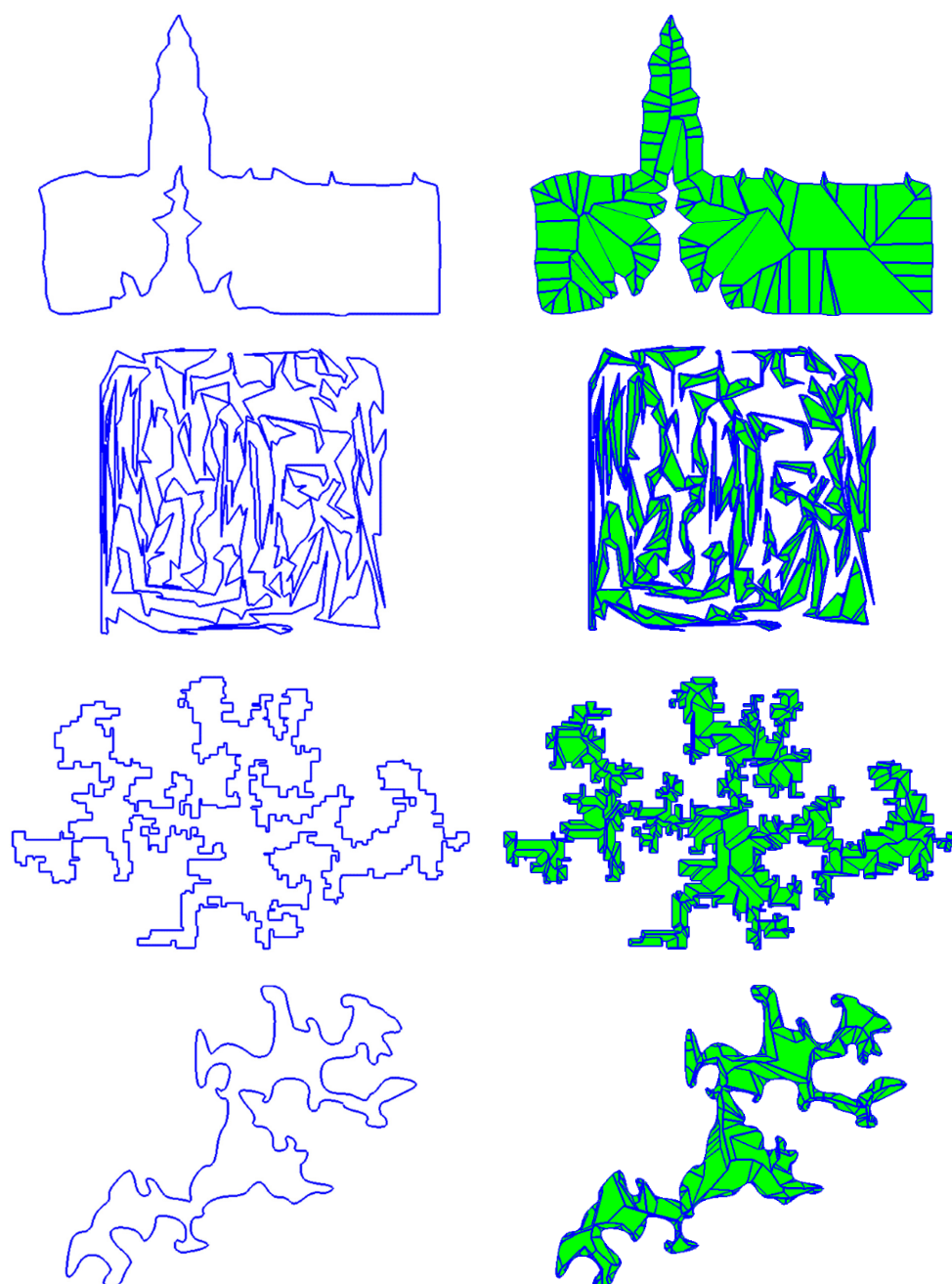

Figure 7: Selected polygons without holes from Salzburg polygon database together with total construction time [18]: Residenzplatz-123 (22s), spg-a-poly_0000500_1 (162s), srpg_iso0000693 (185s), srpg_smo0000619 (200s) (from top to bottom).
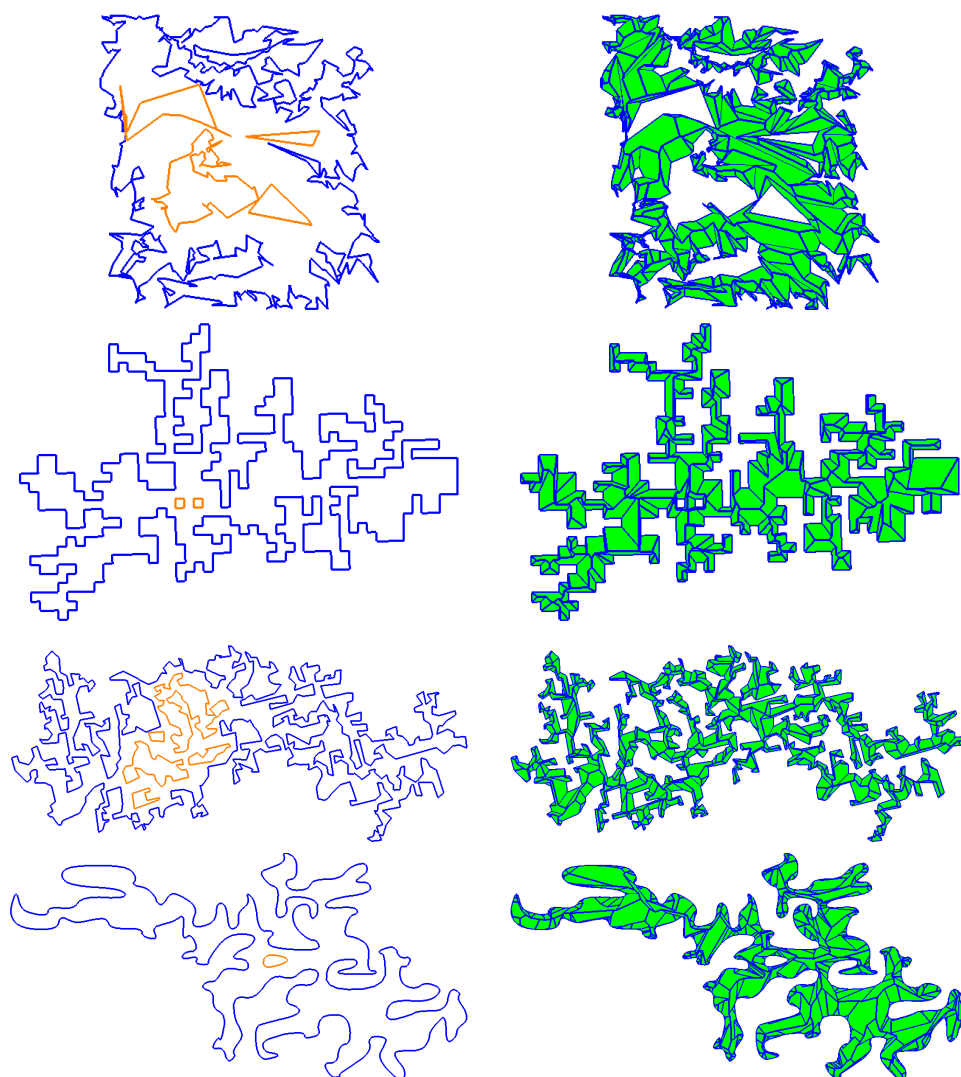
Figure 8: Selected polygons with holes from Salzburg polygon database together with total construction time [18] : fpg-poly_0000000600_h5 (319s), srpg_iso_aligned_mc0000329 (55s), srpg_mc0000782 (458s), srpg_smo_mc0000866 (433s) (from top to bottom).

4.4 Examples

The Salzburg polygon database introduced in [18] has been used to test the performance of the new algorithm. This database contains huge amount of polygon data with varying size and complexity, more than 100GB in total. Straight skeleton structures obtained using the new algorithm for randomly selected polygons, among those with a reasonable size, are presented in Figure 7 and Figure 8 together with total construction time. Videos generated, for the examples presented, can be downloaded using the link given in [1].

4.5 Data Structures

Polygons are stored as doubly linked vertex lists. A vertex stores basically information about position, velocity, unit vector to edge before it, normal vector to edge before it, pointer to vertex before it and pointer to vertex after it. To keep track of the evolution of straight skeleton structure, a vertex also stores pointers to the vertex above and the vertex below, which are associated, respectively, with the polygon above and polygon below.

Data structure described must be adapted as the edge collapse and edge split events occur during incremental motion. When an edge corresponding to a vertex is collapsed, it is simply removed from the linked list and links, namely vertex before and vertex after pointers, for the neighbor vertices are updated. When an edge split event occurs, there are two possibilities. Split event could occur from an edge, vertex-in-edge (see Figure 9), or from a vertex, vertex-on-vertex (see Figure 10). Note that in case of vertex-in-edge, a new vertex is created and added to linked list. The link update should work fine for rather extreme cases like simultaneous penetration of an edge multiple times (see Figure 11). After the links are updated, velocity of the contributing vertices must be recomputed.

In case of simultaneous multiple vertex-on-vertex collision events, there are three possible cases which must be handled properly. (i) If the vertices which collide belong to the same subpolygon, we define polygon split event. In the polygon split event, links are updated in a recursive manner until no more unprocessed event left (see Figure 12). The number of subpolygons at the end of the link update must be equal to the number of vertices which collide. (ii) If the vertices which collide belong to different subpolygons, one speaks of polygon merge event (see Figure 13). In polygon merge event, the most suitable vertex has to be found for a given vertex in order to end up with correct link update. Among all colliding vertices, the one which has the smallest relative angle with respect to given vertex is selected to perform link update. The number of subpolygons at the end of link update must be equal to one. (iii) Combination of cases (i) and (ii). In this case, first polygon split event is processed and then polygon merge event is resolved. Note that in contrary to [30], no preprocessing is necessary to order the vertices contributing to the simultaneous vertex-on-vertex collisions. Algorithm works perfectly fine when those vertices are handled in an arbitrary order. The ambiguities related with vertex-on-vertex collisions have been discussed in depth in [8, 9].

4.6 Search for Edge Penetration

The most time-consuming part the of the predictor-corrector algorithm is the construction of the vertex pairs for edge penetration within a given time increment. In the search for edge penetration, no differentiation is made between convex and reflex vertices, because depending on the weight both can cause edge split event to occur. A vertex can penetrate into the polygon both from inside and from outside, see Figure 14. One must be careful because sometimes during a given time increment a vertex could even penetrate through, see Figure 14. Criteria given by Eq. (28), holds in all possible scenarios and could safely be used. To test Eq. (28) for a given vertex P, all the vertices, except P and vertex after P, must be checked. This brute force approach, resulting in $\mathcal{O}(n^2)$ algorithm, is not very feasible especially for a polygon with many vertices and independent subpolygons. Instead of performing a brute force search, an enhancement based on bounding box of a vertex or a quadtree structure or any other similar approach, like given in [21], based on spatially grouping the vertex input could be proposed. Speed of the algorithm is not the main concern of the current study and therefore
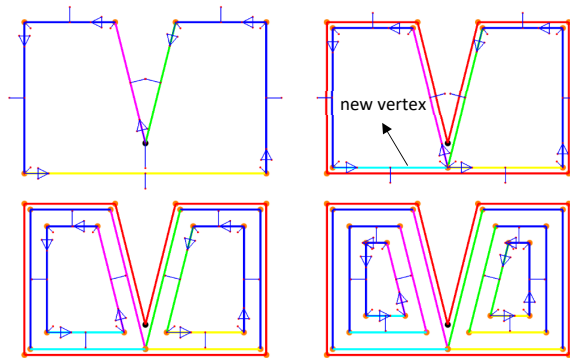
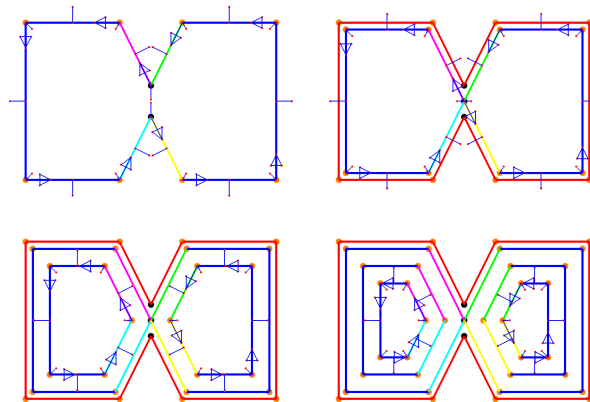**Figure 9**: Vertex-in-edge collision.

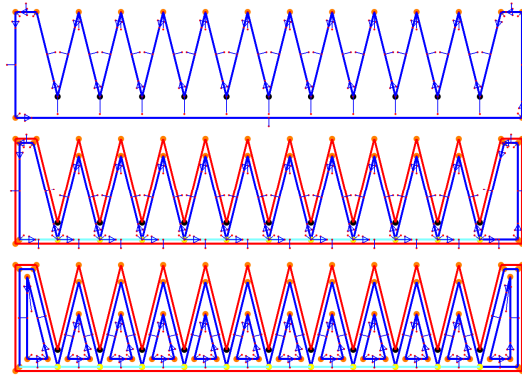**Figure 10**: Vertex-on-vertex collision.

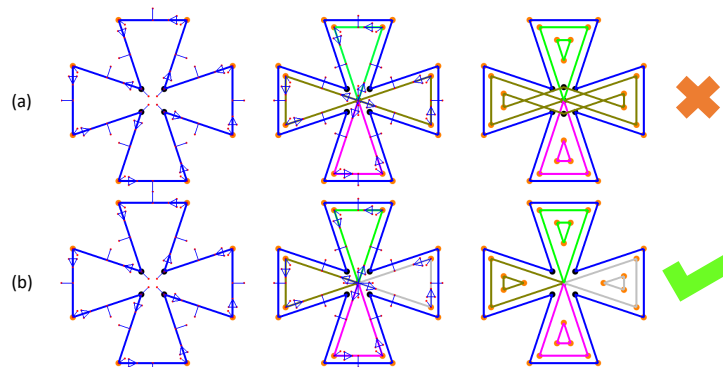**Figure 11**: Multiple simultaneous vertex-in-edge edge split event.

Figure 12: Polygon split event. (a) one-pass resolution. (b) recursive resolution.
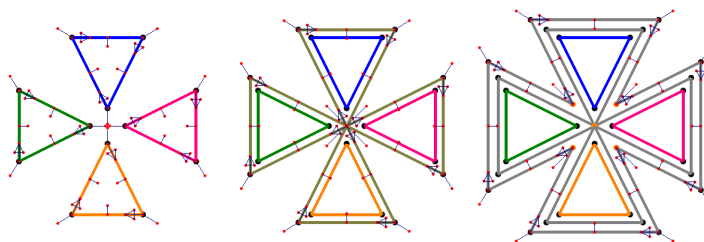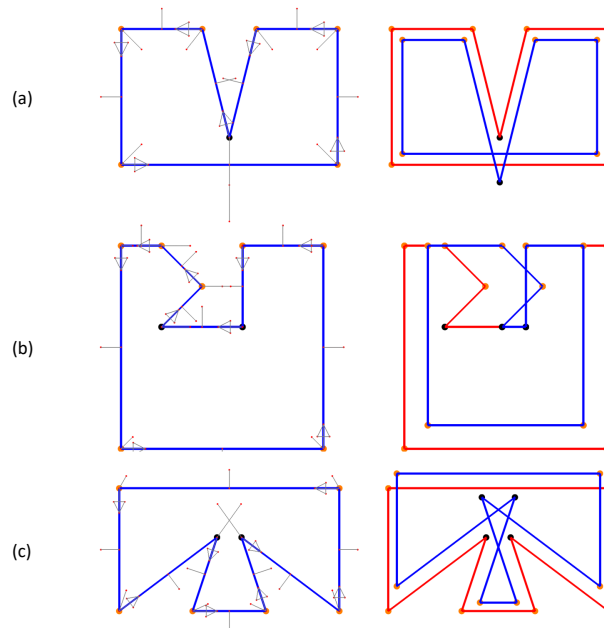

Figure 13: Polygon merge event.

Figure 14: Edge penetration. Different possibilities. Vertex penetrates (a) from inside outside, (b) from outside inside, (c) through.
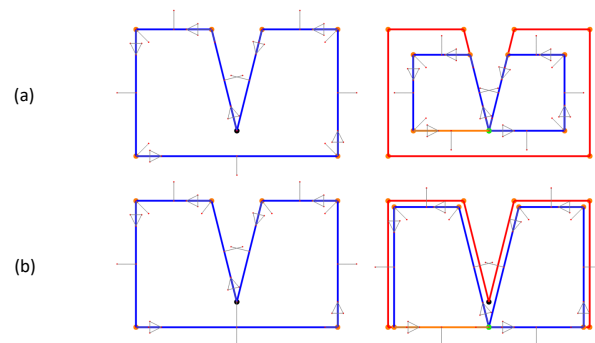

Figure 15: Edge penetration search. Extreme cases (a) stationary vertex and moving edge. (b) moving vertex and stationary edge

no attempt has been made to accelerate search for edge penetration. Proposed brute force search algorithm works also fine for rather extreme cases where penetrating vertices or penetrated edges are stationary, see Figure 15.

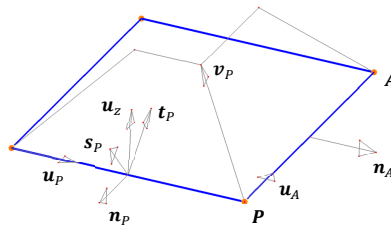**Figure 16**: Roofs with curved edges.

Figure 17: Roof planes associated with vertex P and A.

5 ROOF MODELLING

To construct a roof topology using straight skeleton algorithm proposed, a proper expression for the vertex velocity is necessary. For a roof with straight edges, velocity for a typical vertex P assumes the form

$$\boldsymbol{v}_P = \bar{\boldsymbol{v}}_P + v_z \boldsymbol{e}_z \tag{38}$$

where $\bar{\boldsymbol{v}}_P$ is the projection of the velocity vector $\boldsymbol{v}_P$ onto the polygon plane which is spanned by unit vectors $\boldsymbol{e}_x$ and $\boldsymbol{e}_y$, and, therefore, has the form

$$\bar{\boldsymbol{v}}_P = v_{Px} \boldsymbol{e}_x + v_{Py} \boldsymbol{e}_y \tag{39}$$

Note that z component of the velocity vector is same for all the vertices therefore polygon remains planar while deforming. By setting $v_z = 1$, the amount of offset applied to polygon in z direction turns out to be equal to the corresponding time increment. One can construct piecewise linear approximations to the roofs
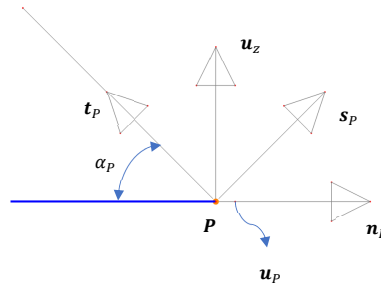
Figure 18: Construction of roof unit normal s_P.

with curved edges by connecting z component of the velocity vector to a function (see Figure 16) without introducing any additional complexity to the underlying algorithm.

For a typical vertex P, velocity components v_{Px} and v_{Py} are computed using the inclination angles α_P and α_A, with $\alpha \in (0, \pi)$, assigned respectively to the vertex P and vertex after it, which we denote as A (see Figure 17 and Figure 18). The roof plane associated with a typical vertex is oriented with respect to surface normal defined as s_P. To compute s_P consistently, we first define a unit vector t_P which lies inside the roof plane as

$$t_P = \cos{(\pi - \alpha_P)}n_P + \sin{(\pi - \alpha_P)}u_z \tag{40}$$

where u_z is unit vector normal to the polygon plane. Both t_P and u_P lie inside roof plane associated with the vertex P. Then normal to this roof plane can easily be constructed by taking the cross product as

$$s_P = u_P \times t_P \tag{41}$$

While the polygon is deforming vertices must lie inside the roof surfaces belonging to them, which can be expressed by the following set of linear equations:

$$s_P \cdot (P_{n+1} - P_n) = 0$$
$$s_A \cdot (P_{n+1} - P_n) = 0 \tag{42}$$

Note that computation of s_A follows from Eq. (40) and Eq. (41). With the insertion of Eq. (18) and Eq. (39), Eq. (42) can be rewritten as

$$\bar{s}_P \cdot \bar{v}_P + s_{Pz}v_z = 0$$
$$\bar{s}_A \cdot \bar{v}_P + s_{Az}v_z = 0 \tag{43}$$

As can be realized from Eq. (41), Eq. (42) and Eq. (43), $\bar{v}_P$ is a function of inclination angles assigned to vertices P and A. Therefore, if the roof planes associated with the vertices of underlying polygon have possibly different inclination angles, different weights are assigned to them which can be computed using Eq. (16).

In Eq. (43), expressions with bar follows the definition given in Eq. (39). If the projections $\bar{s}_P$ and $\bar{s}_A$ are not colinear, Eq. (43) defines two linearly independent equations for unknown velocity components v_{Px} and v_{Py} and hence can easily be solved. Note that if, for example, stationary vertices are present, one can easily end up with an edge collapse event where colinear vertices emerge during incremental motion, see Figure 19.

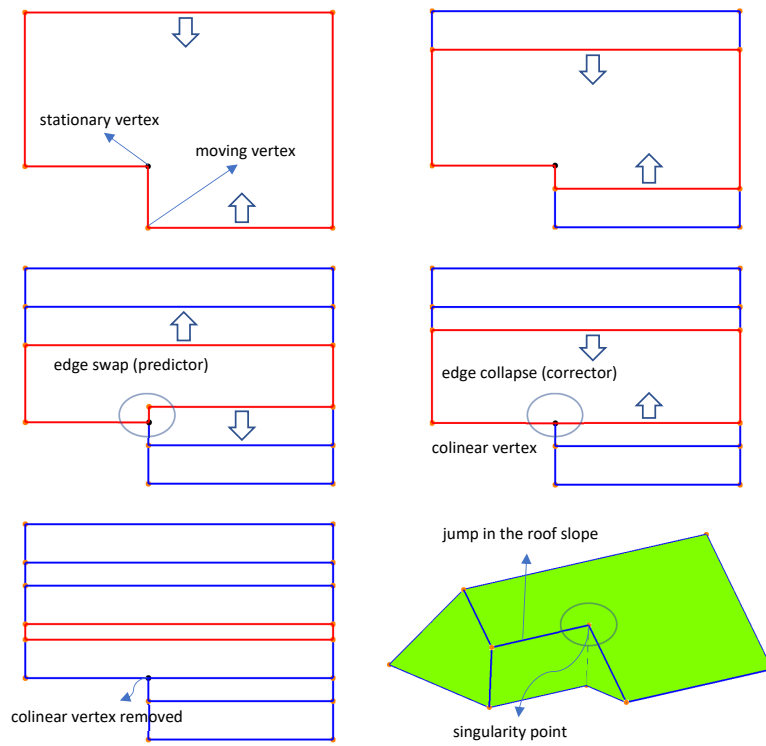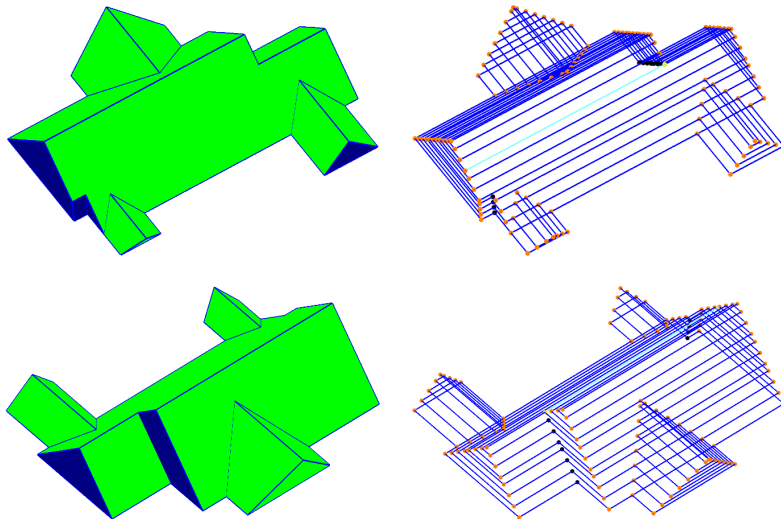**Figure 19**: Singularity due to colinear vertex.

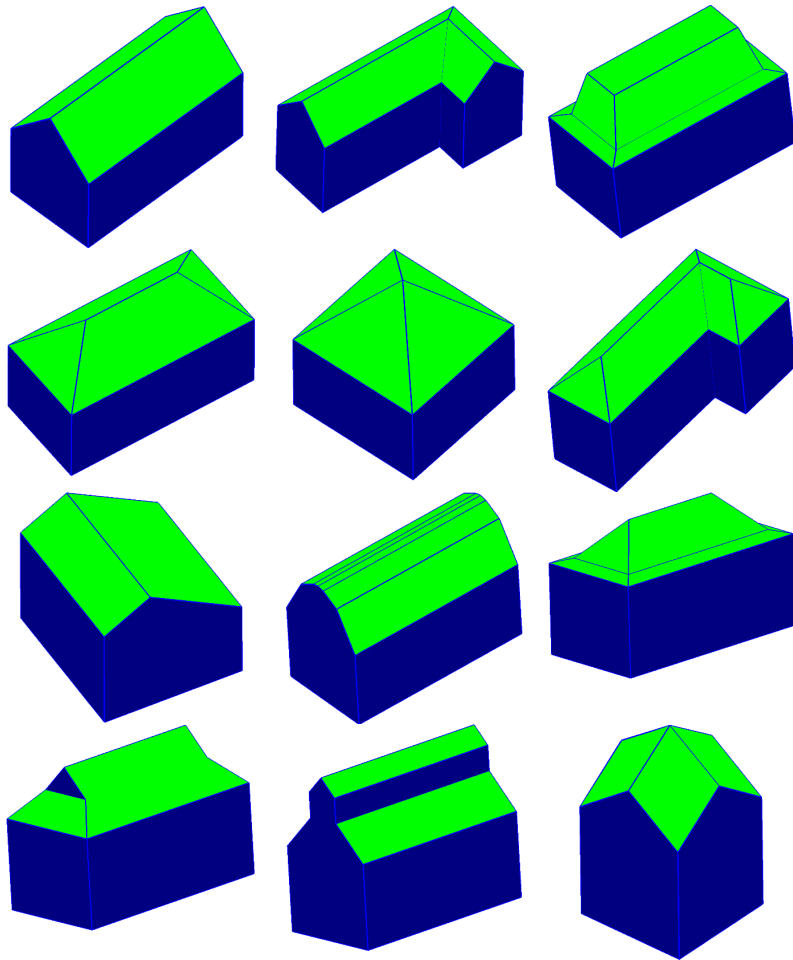**Figure 20**: A composite roof made up of a polygon with four independent subpolygons.

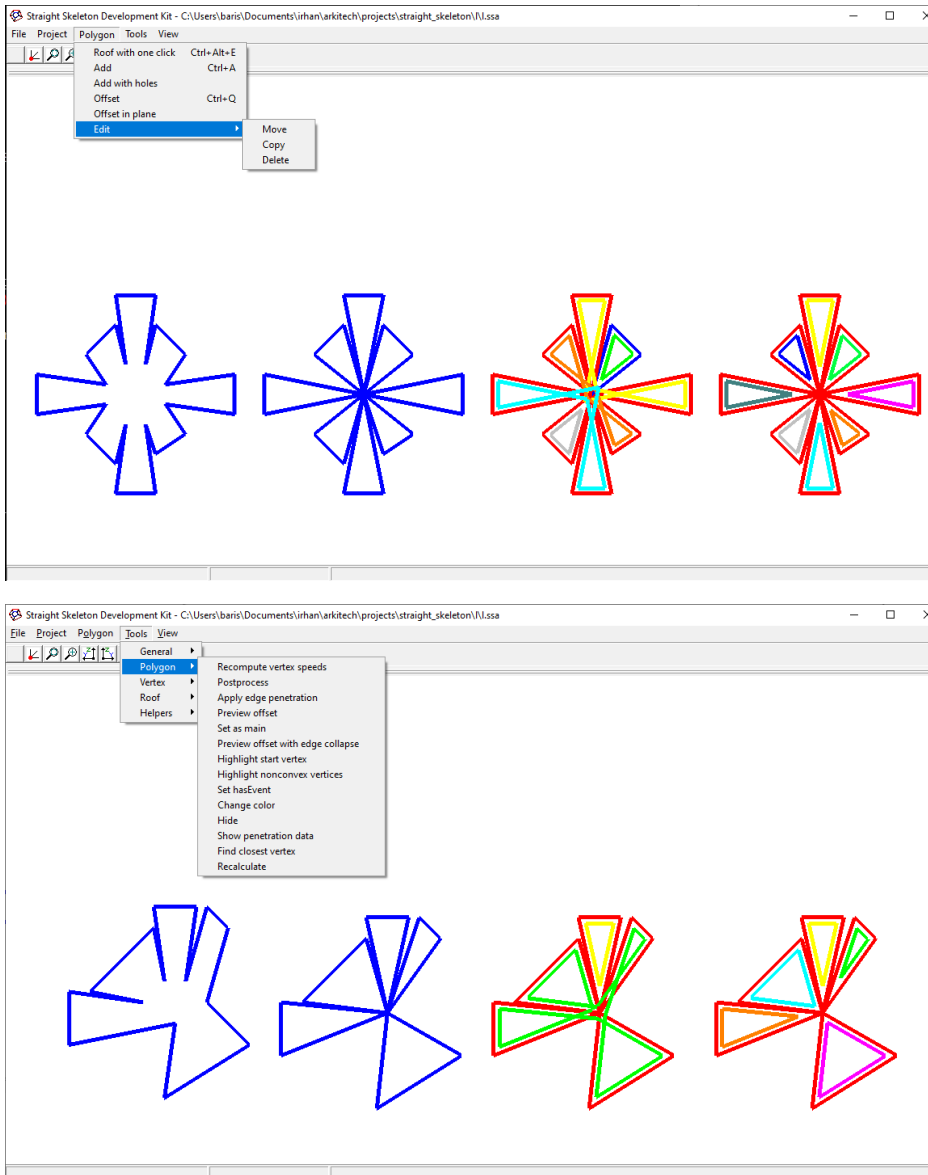**Figure 21**: Different roof types (for last row see [24])

Figure 22: Straight Skeleton Development Kit

For colinear vertices, the set of equations given by Eq. (43) becomes automatically linearly dependent and as a result solution is undefined. Therefore, after an edge collapse event, colinear vertices must be found and removed from the polygon. In the proposed algorithm, in the auto mode, the part belonging to removed vertex is added to the edge belonging to the vertex after it. However, due to underlying incremental nature, one has the capability to interrupt and make manual adjustment to decide how the base polygon will propagate further. Alternative treatments of the colinearity issue are discussed in detail in [8, 30]. Independent of the solution proposed, there will be a jump in the velocity vector along a certain part of the polygon after the colinear vertices are removed. In roof modelling, it can be interpreted as sudden change in the slope, see Figure 19, which is perfectly sound from geometrical point of view. Note that due to its incremental nature, one could interact with the algorithm at any time and , for example, change slope along some edges or insert and remove vertices in order to construct roofs with custom geometry as given in Figure 21.

In the roof construction using weighted straight skeleton algorithm, new position of the polygon is computed based on predictor-corrector algorithm for a prescribed offset in z direction. A roof is complete if there remains no possibility to further apply an offset. Recall that a polygon cannot be offset if it has collapsed down to a single vertex or a single edge, two vertices. Offset polygons are kept and used to construct the roof polygons, namely the straight skeleton structure, in a systematic way by paying special attention to the vertices which are split, collapsed or colinear. In Figure 20 and Figure 21, several representative non-exhaustive examples are given.

6 STRAIGHT SKELETON DEVELOPMENT KIT

In order to develop a reliable general purpose straight skeleton algorithm, or algorithm of any kind, it is essential to have a tool equipped with visualization capabilities. Therefore, a graphical user interface has been developed by the author in parallel for the construction of new predictor-corrector type incremental weighted straight skeleton algorithm which is based on the notion of deforming polygon using C++ programming language, see Figure 22. With the help of this tool, it was possible to test each individual building block of the algorithm separately, which gave tremendous amount of speed up to converge to the current version of the algorithm. Program was not only used to develop the algorithm but also to automize repetitive error prone and time-consuming tasks like preparing the figures for the work presented here.

7 CONCLUSIONS

A universal predictor-corrector type incremental algorithm for the construction of weighted straight skeletons of 2D general planar polygons, possibly with holes, has been developed based on the notion of deforming polygon. An algorithm of this class has never been used before for straight skeletons. Algorithm resembles the techniques which are commonly used in computational solid mechanics. The algorithm works with the raw input provided by the underlying polygon. Additional tools like kinetic triangulation or motorcycle graph are not needed. Polygon edges which move at different speeds and/or start to move at different times are supported by default. In the algorithm predicted positions of the vertices are computed by direct time integration first. Then admissibility of the polygon is checked with respect to edge swap and edge penetration. Edge swap and edge penetration are used to detect, respectively, edge collapse and edge split events within an increment. For a given event, corrected position of the vertices is computed exactly by using linear interpolation. In the algorithm, there is no special treatment for the reflex vertices. Positively, or negatively, weighted and stationary vertices are dealt within the same framework proposed. By assuming a special form for the vertex velocity, the algorithm has been used to construct roof topology starting from a floor plan given. The algorithm could also be used to generate piecewise approximations to the roofs with curved edges by simply connecting z component of the vertex velocity to a prescribed function without introducing any additional complexity. A graphical user interface, Straight Skeleton Development Kit, has been developed in parallel from scratch to

construct, improve and test the algorithm.

ORCID

Barış İrhan, http://orcid.org/0000-0003-4752-0823

Appendices

A IMPENETRABILITY OF AN EDGE

In this section deformation of an individual edge of a polygon will be analyzed and it will be shown that inpenetrability of matter constraint strongly requires projected length of an edge always be positive.
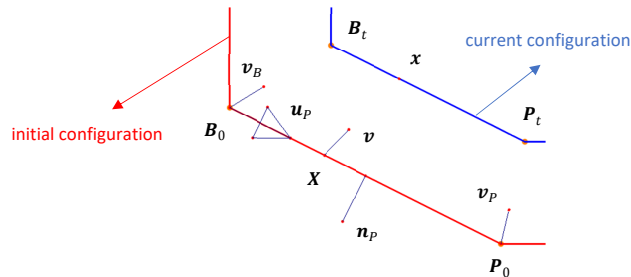

Figure 23: Deforming edge.

Consider the edge of the polygon depicted in Figure 23. The projected length of the edge is defined as

$$l_P = (\boldsymbol{P} - \boldsymbol{B}) \cdot \boldsymbol{u}_P \tag{44}$$

The projected length at initial configuration is then given by

$$l_{P0} = (\boldsymbol{P}_0 - \boldsymbol{B}_0) \cdot \boldsymbol{u}_P \tag{45}$$

By definition, initial projected length is positive, i.e.,

$$l_{P0} > 0 \tag{46}$$

Position of the vertices B and P at current configuration are given by

$$\begin{aligned} \boldsymbol{B} &= \boldsymbol{B}_0 + \boldsymbol{v}_B t \\ \boldsymbol{P} &= \boldsymbol{P}_0 + \boldsymbol{v}_P t \end{aligned} \tag{47}$$

With the insertion of Eq. (47), Eq. (44) can be rewritten as

$$l_P = l_{P0} + \Delta v_u t \tag{48}$$

with

$$\Delta v_u = v_{uP} - v_{uB} \tag{49}$$

In Eq. (49), v_{uB} and v_{uP} are given by

$$v_{uB} = \boldsymbol{v}_B \cdot \boldsymbol{u}_P$$
$$v_{uP} = \boldsymbol{v}_P \cdot \boldsymbol{u}_P \tag{50}$$

Initial position vector for a typical particle belonging to the edge can be written in parametric form as

$$\boldsymbol{X} = \frac{1}{2}(1 - \xi)\boldsymbol{B}_0 + \frac{1}{2}(1 + \xi)\boldsymbol{P}_0 \tag{51}$$

with $\xi \in [-1, 1]$.
Similar to Eq. (51), velocity for a typical particle is expressed as

$$\boldsymbol{v} = \frac{1}{2}(1 - \xi)\boldsymbol{v}_B + \frac{1}{2}(1 + \xi)\boldsymbol{v}_P \tag{52}$$

Recall that edge is to remain parallel to itself while deforming in straight skeleton problem. In order to bring this parallelism constraint into action, velocities $\boldsymbol{v}_B$ and $\boldsymbol{v}_P$ are first written in component form as

$$\boldsymbol{v}_B = v_{uB}\boldsymbol{u}_P + v_{nB}\boldsymbol{n}_P + v_z\boldsymbol{e}_z$$
$$\boldsymbol{v}_P = v_{uP}\boldsymbol{u}_P + v_{nP}\boldsymbol{n}_P + v_z\boldsymbol{e}_z \tag{53}$$

where v_{uB} and v_{uP} are given by Eq. (50), and v_{nB} and v_{nP} are similarly defined as

$$v_{nB} = \boldsymbol{v}_B \cdot \boldsymbol{n}_P$$
$$v_{nP} = \boldsymbol{v}_P \cdot \boldsymbol{n}_P \tag{54}$$

Due to parallelism constraint, normal components of $\boldsymbol{v}_B$ and $\boldsymbol{v}_P$ must be equal, i.e.,

$$v_{nB} = v_{nP} = v_n \tag{55}$$

Using equations Eq. (53), Eq. (54) and Eq. (55), final form the particle velocity is obtained as

$$\boldsymbol{v} = v_u\boldsymbol{u}_P + v_n\boldsymbol{n}_P + +v_z\boldsymbol{e}_z \tag{56}$$

where v_n is given by Eq. (55) and v_u is defined as

$$v_u = \frac{1}{2}(1 - \xi)v_{uB} + \frac{1}{2}(1 + \xi)v_{uP} \tag{57}$$

The position of the particle at current configuration is given by the motion function which is defined as

$$\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{X}, t) = \boldsymbol{X} + \boldsymbol{v}t \tag{58}$$

Using Eq. (58), the deformation gradient is then obtained as

$$\boldsymbol{F}(\boldsymbol{X},t) = \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{X}} = \boldsymbol{I} + \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{X}} t \tag{59}$$

By employing Eq. (56), gradient of particle velocity, $\partial \boldsymbol{v}/\partial \boldsymbol{X}$, turns out to be

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{X}} = \boldsymbol{u}_P \otimes \frac{\partial v_u}{\partial \boldsymbol{X}} \tag{60}$$

By applying the chain rule, $\partial v_u/\partial \boldsymbol{X}$ can be expressed as

$$\frac{\partial v_u}{\partial \boldsymbol{X}} = \frac{\partial v_u}{\partial \xi} \frac{\partial \xi}{\partial \boldsymbol{X}} \tag{61}$$

With the insertion of Eq. (57) and Eq. (49) , $\partial v_u/\partial \xi$ is obtained as

$$\frac{\partial v_u}{\partial \xi} = \frac{\Delta v_u}{2} \tag{62}$$

By employing Eq. (51) and Eq. (45), $\partial \boldsymbol{X}/\partial \xi$ can be written as

$$\frac{\partial \boldsymbol{X}}{\partial \xi} = \frac{1}{2}(\boldsymbol{P}_0 - \boldsymbol{B}_0) = \frac{l_{P0}}{2} \boldsymbol{u}_P \tag{63}$$

Using the identity

$$1 = \frac{\partial \boldsymbol{X}}{\partial \xi} \cdot \frac{\partial \xi}{\partial \boldsymbol{X}} \tag{64}$$

and Eq. (63), $\partial \xi/\partial \boldsymbol{X}$ is obtained as

$$\frac{\partial \xi}{\partial \boldsymbol{X}} = \frac{2}{l_{P0}} \boldsymbol{u}_P \tag{65}$$

Inserting Eq. (62) and Eq. (65) into Eq. (61), it is obtained that

$$\frac{\partial v_u}{\partial \boldsymbol{X}} = \frac{\Delta v_u}{l_{P0}} \boldsymbol{u}_P \tag{66}$$

Putting Eq. (66) into Eq. (60), the final form of the velocity gradient is obtained as

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{X}} = \frac{\Delta v_u}{l_{P0}} \boldsymbol{u}_P \otimes \boldsymbol{u}_P \tag{67}$$

Identity tensor $\boldsymbol{I}$ is defined as

$$\boldsymbol{I} = \boldsymbol{u}_P \otimes \boldsymbol{u}_P + \boldsymbol{n}_P \otimes \boldsymbol{n}_P + \boldsymbol{e}_z \otimes \boldsymbol{e}_z \tag{68}$$

Putting Eq. (67) and Eq. (68) into Eq. (59), final form of the deformation gradient is obtained as

$$\boldsymbol{F} = (1 + \frac{\Delta v_u t}{l_{P0}}) \boldsymbol{u}_P \otimes \boldsymbol{u}_P + \boldsymbol{n}_P \otimes \boldsymbol{n}_P + \boldsymbol{e}_z \otimes \boldsymbol{e}_z \tag{69}$$

From Eq. (69), determinant of $\boldsymbol{F}$ is obtained as

$$det\boldsymbol{F} = 1 + \frac{\Delta v_u t}{l_{P0}} \tag{70}$$

Using Eq. (48), normalized projected length of the edge can be written as

$$\frac{l_P}{l_{P0}} = 1 + \frac{\Delta v_u t}{l_{P0}} \tag{71}$$

Comparing Eq. (70) and Eq. (71), it is observed that

$$det\boldsymbol{F} = \frac{l_P}{l_{P0}} \tag{72}$$

According to impenetrability of matter, $det\boldsymbol{F} > 0$, and noting that $l_{P0} > 0$, it is finally concluded that projected length of the edge must be positive throughout the course of the deformation process, i.e.,

$$l_P > 0 \tag{73}$$

REFERENCES

[1] Videos generated for the examples selected from Salzburg Polygon Database. https://github.com/bi100477/MyDocuments/tree/master/straight_skeleton.

[2] Ahn, H.K.; Bae, S.W.; Knauer, C.; Lee, M.; Shin, C.S.; Vigneron, A.: Realistic roofs over a rectilinear polygon. Comput. Geom., 46(9), 1042–1055, 2013. http://doi.org/https://doi.org/10.1016/j.comgeo.2013.06.002.

[3] Aichholzer, O.; Alberts, D.; Aurenhammer, F.; Gärtner, B.: Straight skeletons of simple polygons. In Proc. 4th Internat. Symp. of LIESMARS, 114–124, 1995.

[4] Aichholzer, O.; Aurenhammer, F.: Straight skeletons for general polygonal figures in the plane. In International computing and combinatorics conference, 117–126. Springer, 1996. http://doi.org/https://doi.org/10.1007/3-540-61332-3_144.

[5] Barequet, G.; Goryachev, A.: Offset polygon and annulus placement problems. Comput. Geom., 47(3), 407–434, 2014. http://doi.org/https://doi.org/10.1016/j.comgeo.2013.10.003.

[6] Belytschko, T.; Moran, B.; Liu, W.K.: Nonlinear finite element analysis for continua and structures, vol. 1. Wiley, 1999.

[7] Biedl, T.; Held, M.; Huber, S.; Kaaser, D.; Palfrader, P.: A simple algorithm for computing positively weighted straight skeletons of monotone polygons. Inf. Process. Lett., 115(2), 243–247, 2015. http://doi.org/https://doi.org/10.1016/j.ipl.2014.09.021.

[8] Biedl, T.; Held, M.; Huber, S.; Kaaser, D.; Palfrader, P.: Weighted straight skeletons in the plane. Comput. Geom., 48(2), 120–133, 2015. http://doi.org/https://doi.org/10.1016/j.comgeo.2014.08.006.

[9] Biedl, T.; Huber, S.; Palfrader, P.: Planar matchings for weighted straight skeletons. Int. J. Comput. Geom. Appl., 26(3&4), 211–229, 2016. http://doi.org/https://doi.org/10.1142/S0218195916600050.

[10] Cacciola, F.: A cgal implementation of the straight skeleton of a simple 2d polygon with holes. In 2nd CGAL User Workshop, vol. 1, 2004.

[11] Camacho, G.; Ortiz, M.: Adaptive lagrangian modelling of ballistic penetration of metallic targets. Comput. Methods Appl. Mech. Eng., 142(3), 269–301, 1997. http://doi.org/https://doi.org/10.1016/S0045-7825(96)01134-6.

[12] Cheng, S.W.; Mencel, L.; Vigneron, A.: A faster algorithm for computing straight skeletons. In European Symposium on Algorithms, 272–283. Springer, 2014. http://doi.org/https://doi.org/10.1007/978-3-662-44777-2_23.

[13] Cheng, S.W.; Vigneron, A.: Motorcycle graphs and straight skeletons. In Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, 156–165. Society for Industrial and Applied Mathematics, 2002.

[14] Cheng, S.W.; Vigneron, A.: Motorcycle graphs and straight skeletons. Algorithmica, 47(2), 159–182, 2007. http://doi.org/https://doi.org/10.1007/s00453-006-1229-7.

[15] Chuzel-Marmot, Y.; Ortiz, R.; Combescure, A.: Three dimensional sph–fem gluing for simulation of fast impacts on concrete slabs. Comput. Struct., 89(23), 2484–2494, 2011. http://doi.org/https://doi.org/10.1016/j.compstruc.2011.06.002.

[16] Demaine, E.D.; Demaine, M.L.; Mitchell, J.S.: Folding flat silhouettes and wrapping polyhedral packages: New results in computational origami. Comput. Geom., 16(1), 3–21, 2000. http://doi.org/https://doi.org/10.1016/S0925-7721(99)00056-5.

[17] Eder, G.; Held, M.: Computing positively weighted straight skeletons of simple polygons based on a bisector arrangement. Inf. Process. Lett., 132, 28–32, 2018. http://doi.org/https://doi.org/10.1016/j.ipl.2017.12.001.

[18] Eder, G.; Held, M.; Jasonarson, S.; Mayer, P.; Palfrader, P.: Salzburg database of polygonal data: Polygons and their generators. Data in Brief, 31(105984), 2020. http://doi.org/https://doi.org/10.1016/j.dib.2020.105984.

[19] Eder, G.; Held, M.; Palfrader, P.: Computing the straight skeleton of an orthogonal monotone polygon in linear time. In Proc. 35th EuroCG, 2019.

[20] Eder, G.; Held, M.; Palfrader, P.: On implementing straight skeletons: Challenges and experiences. In 36th International Symposium on Computational Geometry (SoCG 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. http://doi.org/https://doi.org/10.1016/j.comgeo.2021.101760.

[21] Eppstein, D.; Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Discrete Comput. Geom., 22(4), 569–592, 1999. http://doi.org/https://doi.org/10.1007/PL00009479.

[22] Erhart, T.; Wall, W.A.; Ramm, E.: Robust adaptive remeshing strategy for large deformation, transient impact simulations. Int. J. Numer. Methods Eng., 65(13), 2139–2166, 2006. http://doi.org/https://doi.org/10.1002/nme.1531.

[23] Felkel, P.; Obdrzalek, S.: Straight skeleton implementation. In Proceedings of Spring Conference on Computer Graphics, 210–218, 1998.

[24] Held, M.; Palfrader, P.: Straight skeletons with additive and multiplicative weights and their application to the algorithmic generation of roofs and terrains. Computer-Aided Design, 92, 33–41, 2017. http://doi.org/https://doi.org/10.1016/j.cad.2017.07.003.

[25] Huber, S.: Computing straight skeletons and motorcycle graphs: theory and practice. Ph.D. thesis. University of Salzburg, 2011.

[26] Huber, S.; Held, M.: Theoretical and practical results on straight skeletons of planar straight-line graphs. In Proceedings of the twenty-seventh annual symposium on Computational geometry, 171–178, 2011. http://doi.org/https://doi.org/10.1145/1998196.1998223.

[27] Huber, S.; Held, M.: A fast straight-skeleton algorithm based on generalized motorcycle graphs. Int. J. Comput. Geom. Appl., 22(5), 471–498, 2012. http://doi.org/https://doi.org/10.1142/S0218195912500124.

[28] Hughes, T.J.: The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Dover Publications, 2012.

[29] İrhan, B.; Ožbolt, J.; Ruta, D.: 3d finite element simulations of high velocity projectile impact. Int.

J. Solids Struct., 72, 38–49, 2015. http://doi.org/https://doi.org/10.1016/j.ijsolstr.2015.07.010.

[30] Kelly, T.: Unwritten procedural modeling with the straight skeleton. Ph.D. thesis. University of Glasgow, 2014.

[31] Levi-Steiner, A.; Steinerd, D.: Contour interpolation by straight skeletons. Graphical Models, 66, 245–260, 2004. http://doi.org/https://doi.org/10.1016/j.gmod.2004.05.001.

[32] Linder, C.: An arbitrary Lagrangian-Eulerian finite element formulation for dynamics and finite strain plasticity models. Institut für Baustatik und Baudynamik (Bauwesen) der Universität Stuttgart, 2003.

[33] Marusich, T.; Ortiz, M.: Modelling and simulation of high-speed machining. Int. J. Numer. Methods Eng., 38(21), 3675–3694, 1995. http://doi.org/https://doi.org/10.1002/nme.1620382108.

[34] Miehe, C.: Exponential map algorithm for stress updates in anisotropic multiplicative elastoplasticity for single crystals. Int. J. Numer. Methods Eng., 39(19), 3367–3390, 1996. http://doi.org/https://doi.org/10.1002/(SICI)1097-0207(19961015)39:19%3C3367::AID-NME4%3E3.0.CO;2-7.

[35] Obando, D.F.G.; Olivo-Marin, J.C.; Wendling, L.; Meas-Yedid, V.: Vector-based morphological operations on polygons using straight skeletons for digital pathology. In International Conference on Discrete Geometry for Computer Imagery, 249–261. Springer, 2019. http://doi.org/https://doi.org/10.1007/978-3-030-14085-4_20.

[36] Ogden, R.: Nonlinear Elastic Deformations. Dover Publications, 1997.

[37] Palfrader, P.; Held, M.: Computing mitered offset curves based on straight skeletons. Comput Aided Des Appl, 12(4), 414–424, 2015. http://doi.org/https://doi.org/10.1080/16864360.2014.997637.

[38] Profizi, P.; Combescure, A.; Ogawa, K.: Sph modeling of adhesion in fast dynamics: Application to the cold spray process. C.R. Mec., 344(4-5), 211–224, 2016. http://doi.org/https://doi.org/10.1016/j.crme.2016.02.001.

[39] Rabczuk, T.; Xiao, S.; Sauer, M.: Coupling of mesh-free methods with finite elements: basic concepts and test results. Commun. Numer. Methods Eng., 22(10), 1031–1065, 2006. http://doi.org/https://doi.org/10.1002/cnm.871.

[40] Radovitzky, R.; Ortiz, M.: Lagrangian finite element analysis of newtonian fluid flows. Int. J. Numer. Methods Eng., 43(4), 607–619, 1998. http://doi.org/https://doi.org/10.1002/(SICI)1097-0207(19981030)43:4%3C607::AID-NME399%3E3.0.CO;2-N.

[41] Simo, J.C.; Hughes, T.J.: Computational Inelasticity. Springer, 1998.

[42] Stein, E.; Sagar, G.: Theory and finite element computation of cyclic martensitic phase transformation at finite strain. Int. J. Numer. Methods Eng., 74(1), 1–31, 2008. http://doi.org/https://doi.org/10.1002/nme.2148.

[43] Sugihara, K.: Design of pop-up cards based on weighted straight skeletons. In 2013 10th International Symposium on Voronoi Diagrams in Science and Engineering, 23–28. IEEE, 2013. http://doi.org/https://doi.org/10.1109/ISVD.2013.9.

[44] Sugihara, K.: Straight skeleton computation optimized for roof model generation. In WSCG'2019 - 27. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2019, 101–109. Václav Skala-UNION Agency, 2019. http://doi.org/https://doi.org/10.24132/CSRN.2019.2901.1.12.

[45] Sugihara, K.; Khmelevsky, Y.: Roof report from automatically generated 3d building models by straight skeleton computation. In 2018 Annual IEEE International Systems Conference (SysCon), 1–8. IEEE, 2018. http://doi.org/https://doi.org/10.1109/SYSCON.2018.8369554.

[46] Tanase, M.; Veltkamp, R.C.: Polygon decomposition based on the straight line skeleton. In Proceedings of

the nineteenth annual symposium on Computational geometry, 58–67, 2003. http://doi.org/https://doi.org/10.1145/777792.777802.

[47] Wall, W.A.: Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen. Institut für Baustatik und Baudynamik (Bauwesen) der Universität Stuttgart, 1999.

[48] Wriggers, P.: Computational Contact Mechanics. Springer, 2006.

[49] Zienkiewicz, O.; Taylor, R.; Fox, D.: The finite element method for solid and structural mechanics, 7th Edition. Butterworth-Heinemann, 7 ed., 2014.