Penalty Force for Coupling Materials with Coulomb Friction: Technical Document

1 Pseudocode

This is the technical document for [Ano18]. In this section, we will present pseudocode for computing a time step of the method along with details of the underlying Newton solver and penalty forces.

Algorithm 1 Overview of time integration.

procedure TIME_STEP
  Solve BACKWARD_EULER_EQUATIONS($\Delta\mathbf{v}$) $= \mathbf{0}$ using NEWTON_SOLVE
  $\mathbf{v}^{n+1} \leftarrow \mathbf{v}^n + \Delta\mathbf{v}$, $\mathbf{x}^{n+1} \leftarrow \mathbf{x}^n + \Delta t\mathbf{v}^{n+1}$ ▷ Update rotations for rigid bodies, etc.
  for all registered attachments **do**
    if is_active **then**
      UPDATE_ATTACHMENT_XX ▷ XX is DI, DR, RR, or DD
    else
      Discard attachment

procedure BACKWARD_EULER_EQUATIONS($\Delta\mathbf{v}$) ▷ Compute $\mathbf{g}$
  $\mathbf{a} \leftarrow$ physical forces ▷ Constitutive models, gravity
  Compute candidate configuration from $\Delta\mathbf{v}$
  Compute collisions and add to list of registered collisions
  for all registered attachments **do**
    PRECOMPUTE_XX ▷ XX is DI, DR, RR, or DD
    if is_active **then**
      $\mathbf{a} \leftarrow \mathbf{a} +$ FORCE_XX ▷ XX is DI, DR, RR, or DD
  return $\mathbf{M}\Delta\mathbf{v} - \Delta t\mathbf{a}$ ▷ $\mathbf{M}$ is the mass matrix

procedure BACKWARD_EULER_EQUATIONS_DIFF($\delta\mathbf{u}$) ▷ Compute $\mathbf{H}\delta\mathbf{u} = \nabla\mathbf{g}\delta\mathbf{u}$
  $\mathbf{a} \leftarrow$ multiply physical force derivatives by $\delta\mathbf{u}$ ▷ Constitutive models, gravity
  for all registered attachments **do**
    if is_active **then**
      $\mathbf{a} \leftarrow \mathbf{a} +$ FORCE_DIFF_XX($\delta\mathbf{u}$) ▷ XX is DI, DR, RR, or DD
  return $\mathbf{M}\delta\mathbf{u} - \Delta t^2\mathbf{a}$ ▷ $\mathbf{M}$ is the mass matrix

procedure BACKWARD_EULER_EQUATIONS_DIFF_TRANSPOSE($\delta\mathbf{u}$) ▷ Compute $\mathbf{H}^T\delta\mathbf{u} = \nabla\mathbf{g}^T\delta\mathbf{u}$
  $\mathbf{a} \leftarrow$ multiply physical force derivatives by $\delta\mathbf{u}$ ▷ Constitutive models, gravity
  for all registered attachments **do**
    if is_active **then**
      $\mathbf{a} \leftarrow \mathbf{a} +$ FORCE_DIFF_TRANSPOSE_XX($\delta\mathbf{u}$) ▷ XX is DI, DR, RR, or DD
  return $\mathbf{M}\delta\mathbf{u} - \Delta t^2\mathbf{a}$ ▷ $\mathbf{M}$ is the mass matrix

Algorithm 2 Newton-based solver for asymmetrical systems.

procedure NEWTON_SOLVE($\mathbf{z}$)
  while true **do**
    $\mathbf{b} \leftarrow$ BACKWARD_EULER_EQUATIONS($\mathbf{z}$)
    $\mathbf{a} \leftarrow$ BACKWARD_EULER_EQUATIONS_DIFF_TRANSPOSE($\mathbf{b}$)
    $m \leftarrow \|\mathbf{a}\|$
    if $\min(m, \|\mathbf{b}\|) < \tau$ **then** ▷ τ is the Newton tolerance
      return $\mathbf{z}$
    $\Delta\mathbf{z} \leftarrow \mathbf{H}^{-1}(-\mathbf{b})$ ▷ Solve with GMRES, using operator BACKWARD_EULER_EQUATIONS_DIFF
    $a \leftarrow \|\Delta\mathbf{z}\|$, $b \leftarrow \Delta\mathbf{z} \cdot \mathbf{a}$
    if $b > a\tau_a m$ **then** ▷ τ_a is a small angle tolerance; we use $\tau_a = 10^{-4}$
      $\Delta\mathbf{z} \leftarrow -\Delta\mathbf{z}$ ▷ Uphill direction; look backwards
    else if $b \geq -a\tau_a m$ **then**
      $\Delta\mathbf{z} \leftarrow -\frac{a}{m}\mathbf{a}$ ▷ Not appreciably uphill or downhill; gradient direction
    Create function: $h(t) = \frac{1}{2}\|$BACKWARD_EULER_EQUATIONS$(\mathbf{z} + t\Delta\mathbf{z})\|^2$
    Compute α by performing line search on function h with Wolfe conditions ▷ See [NW06].
    $\mathbf{z} \leftarrow \mathbf{z} + \alpha\Delta\mathbf{z}$

1.1 Time step

We use Backward Euler for our time integration scheme. In Algorithm 1, we present pseudocode for our overall time integration scheme for one time step, assuming a deformable body simulation. For rigid bodies, there is additional state that must be updated (rotations, angular velocity). For the material point method, there are particle-to-grid and grid-to-particle transfers that are required. These extra steps are not affected by our modifications.

We have chosen to parameterize our system in terms of $\Delta\mathbf{v}$. This avoids the complexities that would result from parameterizing rigid bodies in terms of positional degrees of freedom. Following this convention, `Backward_Euler_Equations_Diff` and `Backward_Euler_Equations_Diff_Transpose` are linear operators that compute matrix-vector products (without or with transpose) using the derivative of `Backward_Euler_Equations` with respect to $\Delta\mathbf{v}$. Following the usual physics conventions, the forces are always differentiated with respect to positions to construct the linear operators `Force_Diff_XX` and `Force_Diff_Transpose_XX`. This accounts for the factor of Δt that occurs in the definitions of `Backward_Euler_Equations` and the factor of Δt^2 in its derivatives.

Although our algorithm uses force derivatives, no part of our algorithm requires us to compute an explicit matrix representation. Instead, it suffices to perform matrix-vector multiplies, possibly with transpose. In this way, `Force_Diff_XX`($\delta\mathbf{u}$) computes the matrix-vector product $\nabla\mathbf{f}\delta\mathbf{u}$, and `Force_Diff_Transpose_XX`($\delta\mathbf{u}$) computes the matrix-vector product with transpose $\nabla\mathbf{f}^T\delta\mathbf{u}$. The result of these operations is a vector. The vector $\delta\mathbf{u}$ simply represents some abstract vector to which the derivatives should be multiplied; it need not have any particular physical meaning and often does not. In the case of GMRES, for example, these vectors are the Krylov basis vectors. These force derivative multiplies do not modify internal state; indeed, they would fail to be linear operators if they did. `Backward_Euler_Equations_Diff` and `Backward_Euler_Equations_Diff_Transpose` compute matrix-vector products in the same way.

1.2 Newton solver

We use a stabilized Newton-based solver designed for use with asymmetrical systems. This solver takes an initial guess $\mathbf{z}$ and seeks a solution $\mathbf{g}(\mathbf{z}) = \mathbf{0}$. Let $\mathbf{H} = \nabla\mathbf{g}$, where here $\mathbf{H}$ will generally be an asymmetric matrix. Pseudocode for this solver is given in Algorithm 2. In our parametrization, $\mathbf{z}$ corresponds to $\Delta\mathbf{v}$. Note that we do not actually compute $\mathbf{H}^{-1}$; rather we solve this system using GMRES, which only requires us to compute matrix-vector multiplications $\mathbf{H}\delta\mathbf{u} =$ `Backward_Euler_Equations_Diff`($\delta\mathbf{u}$).

1.3 Penalty forces

In this section, we present pseudocode for the relaxation procedures, forces, and force derivatives required to implement the penalty forces in that paper. Routines are provided for deformable vs level set (Algorithm 5, denoted DI), deformable vs rigid body (Algorithm 6, denoted RD), rigid body vs rigid body (Algorithm 7 and Algorithm 8, denoted RR), and deformable vs deformable (Algorithm 9, denoted DD). These algorithms rely on helper utilities for differentiating rigid body motion (Algorithm 4) as well as relaxation in a plane (Algorithm 3), or mesh (Algorithm 10, Algorithm 11, and Algorithm 12).

The pseudocode aims to strike a balance between completeness, readability, and conciseness. We use bold letters to represent vectors and matrices. Regular letters represent scalars. Note that bold capital letters are used for both vector and matrix quantities. Quantities written as partial derivatives such as $\frac{\partial \mathbf{K}}{\partial \mathbf{X}}$ represent regular variables which store the value of the partial derivative suggested by the name. Although this makes the pseudocode appear more complex and less concise, it makes it immediately clear what many of the quantities being computed represent. As a general rule (but not always), we will assign results to a variable and then return the variable to make it clear what the quantity being returned represents. When we return multiple items (such as a, b, and c), we denote the aggregate using the notation $\langle a, b, c\rangle \leftarrow \textsc{Function}(\mathbf{Y}, \mathbf{Z})$.

We often omit the identity of particles and rigid bodies for clarity. When colliding a particle with a rigid body, for instance, we are dealing with one particle and one rigid body, so this should not lead to confusion. When dealing with two rigid bodies, we give them names the names i and s. Rigid body i is the object, and s is the rigid body with the particle. That is, the level set or surface mesh of body i is colliding with a particle in the surface mesh of body s.

We use standard linear algebra notation except when dealing with rank-3 tensors, in which case we use index notation with summation convention applied; this notation is adopted for individual lines, and this is noted on those lines by a comment.

Algorithm 3 Relax a particle against a flat plane.

procedure RELAX_PLANE($\mathbf{Z}$, $\mathbf{X}$, $\mathbf{W}$, μ)
  $a \leftarrow \|\mathbf{Z} - \mathbf{W}\|, \quad b \leftarrow \|\mathbf{X} - \mathbf{W}\|, \quad q \leftarrow \mu a$
  if $b \le q$ **then** ▷ Sticking friction
    return $\langle \mathbf{X}, \mathbf{0}, \mathbf{I}, \mathbf{0}\rangle$
  $\mathbf{u} \leftarrow \dfrac{\mathbf{Z} - \mathbf{W}}{a}, \quad \mathbf{v} \leftarrow \dfrac{\mathbf{X} - \mathbf{W}}{b}, \quad \mathbf{K} \leftarrow \mathbf{W} + q\mathbf{v}$ ▷ Dynamic friction
  $\dfrac{\partial \mathbf{K}}{\partial \mathbf{X}} \leftarrow \dfrac{q}{b}(\mathbf{I} - \mathbf{v}\mathbf{v}^T), \quad \dfrac{\partial \mathbf{K}}{\partial \mathbf{Z}} \leftarrow \mu \mathbf{v}\mathbf{u}^T, \quad \dfrac{\partial \mathbf{K}}{\partial \mathbf{W}} \leftarrow \mathbf{I} - \dfrac{\partial \mathbf{K}}{\partial \mathbf{X}} - \dfrac{\partial \mathbf{K}}{\partial \mathbf{Z}}$
  return $\left\langle \mathbf{K}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{Z}}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{X}}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{W}} \right\rangle$

procedure PROJECT_ATTACHMENT_TO_MOVING_SURFACE(b, $\mathbf{X}$, ϕ, exit_early)
  $\left\langle \mathbf{U}, \dfrac{\partial \mathbf{U}}{\partial \mathbf{X}}, \dfrac{\partial \mathbf{U}}{\partial \mathbf{v}}, \dfrac{\partial \mathbf{U}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow \textsc{Frame\_Inverse\_Times}(b, \mathbf{X})$
  Compute $\left\langle d, \mathbf{N}, \dfrac{\partial \mathbf{N}}{\partial \mathbf{U}} \right\rangle$ ▷ Level set ϕ along with normal and Hessian evaluated at $\mathbf{U}$.
  if exit_early **and** $d > 0$ **then**
    return $\langle$false, $\cdots\rangle$
  $\left\langle \mathbf{n}, \dfrac{\partial \mathbf{n}}{\partial \mathbf{N}}, \dfrac{\partial \mathbf{n}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow \textsc{Rotate}(b, \mathbf{N})$
  $\mathbf{W} \leftarrow \mathbf{X} - d\mathbf{n}, \quad \dfrac{\partial \mathbf{n}}{\partial \mathbf{U}} \leftarrow \dfrac{\partial \mathbf{n}}{\partial \mathbf{N}} \dfrac{\partial \mathbf{N}}{\partial \mathbf{U}}, \quad \dfrac{\partial \mathbf{W}}{\partial \mathbf{U}} \leftarrow -d \dfrac{\partial \mathbf{n}}{\partial \mathbf{U}} - \mathbf{n}\mathbf{N}^T$
  $\dfrac{\partial \mathbf{W}}{\partial \mathbf{X}} \leftarrow \mathbf{I} + \dfrac{\partial \mathbf{W}}{\partial \mathbf{U}} \dfrac{\partial \mathbf{U}}{\partial \mathbf{X}}, \quad \dfrac{\partial \mathbf{W}}{\partial \mathbf{v}} \leftarrow \dfrac{\partial \mathbf{W}}{\partial \mathbf{U}} \dfrac{\partial \mathbf{U}}{\partial \mathbf{v}}, \quad \dfrac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}} \leftarrow \dfrac{\partial \mathbf{W}}{\partial \mathbf{U}} \dfrac{\partial \mathbf{U}}{\partial \boldsymbol{\omega}} - d \dfrac{\partial \mathbf{n}}{\partial \boldsymbol{\omega}}$
  return $\left\langle \text{true}, \mathbf{W}, \dfrac{\partial \mathbf{W}}{\partial \mathbf{X}}, \dfrac{\partial \mathbf{W}}{\partial \mathbf{v}}, \dfrac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}} \right\rangle$

Algorithm 4 Routines for applying and differentiating rigid body motion.

procedure MOVE_RIGID_BODY($\mathbf{v}$, $\boldsymbol{\omega}$, Δt)
  $\mathbf{x} \leftarrow \mathbf{x}^n + \Delta t \mathbf{v}, \quad \mathbf{u} \leftarrow \Delta t \boldsymbol{\omega}, \quad \mathbf{R} \leftarrow e^{\mathbf{u}^*}\mathbf{R}^n$
  $\theta \leftarrow \|\mathbf{u}\|, \quad \mathbf{K} \leftarrow \dfrac{\mathbf{u}^*}{\theta}, \quad \mathbf{A} \leftarrow \mathbf{I} - \mathbf{u}\mathbf{u}^T, \quad \mathbf{B} \leftarrow \cos\theta\mathbf{K} + \sin\theta\mathbf{K}^2, \quad a \leftarrow -\dfrac{\sin\theta}{\theta}, \quad b \leftarrow \dfrac{1-\cos\theta}{\theta}$
  $\mathbf{T}_{ijk} \leftarrow (\mathbf{B}_{im}\mathbf{u}_k + a\mathbf{e}_{imr}\mathbf{A}_{kr} + b\mathbf{A}_{ik}\mathbf{u}_m + b\mathbf{A}_{mk}\mathbf{u}_i)\mathbf{R}^n_{mj}$ ▷ Tensor index notation
  Store: $\mathbf{x}, \mathbf{R}, \mathbf{T}$

procedure FRAME_TIMES(body, $\mathbf{u}$)
  Lookup $\mathbf{R}, \mathbf{x}, \mathbf{T}$ for rigid body
  $\mathbf{Z} \leftarrow \mathbf{R}\mathbf{u} + \mathbf{x}, \quad \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}} \leftarrow \mathbf{R}, \quad \dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}} \leftarrow \mathbf{I}$
  $\left(\dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}}\right)_{ij} \leftarrow \mathbf{T}_{ikj}\mathbf{u}_k$ ▷ Tensor index notation
  return $\left\langle \mathbf{Z}, \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}}, \dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}}, \dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}} \right\rangle$

procedure FRAME_INVERSE_TIMES(body, $\mathbf{u}$)
  Lookup $\mathbf{R}, \mathbf{x}, \mathbf{T}$ for rigid body
  $\mathbf{w} \leftarrow \mathbf{u} - \mathbf{x}, \quad \mathbf{Z} \leftarrow \mathbf{R}^T\mathbf{w}, \quad \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}} \leftarrow \mathbf{R}^T, \quad \dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}} \leftarrow -\mathbf{R}^T$
  $\left(\dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}}\right)_{ij} \leftarrow \mathbf{T}_{kij}\mathbf{w}_k$ ▷ Tensor index notation
  return $\left\langle \mathbf{Z}, \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}}, \dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}}, \dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}} \right\rangle$

procedure ROTATE(body, $\mathbf{u}$)
  Lookup $\mathbf{R}, \mathbf{T}$ for rigid body
  $\mathbf{Z} \leftarrow \mathbf{R}\mathbf{u}, \quad \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}} \leftarrow \mathbf{R}$
  $\left(\dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}}\right)_{ij} \leftarrow \mathbf{T}_{ikj}\mathbf{u}_k$ ▷ Tensor index notation
  return $\left\langle \mathbf{Z}, \dfrac{\partial \mathbf{Z}}{\partial \mathbf{u}}, \dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}} \right\rangle$

Each of the 4 relaxation types (DI, RD, RR, and DD) have four routines, named (for example) PRECOMPUTE_DI, FORCE_DI, FORCE_DIFF_DI, and FORCE_DIFF_TRANSPOSE_DI. The precompute routine performs the attachment relaxation. It should be used to update the attachment points before any of the force routines are computed. The three force-related routines are written as though they are writing (or adding) forces to a large force vector $\mathbf{f}$, which is initially zero. The notation $\mathbf{f}[\mathbf{Z}] \leftarrow \mathbf{0}$ is used to indicate that zero is written to the entries in the large force vector $\mathbf{f}$ corresponding to the degrees of freedom of $\mathbf{Z}$. The force diff routine multiplies an input vector $\delta\mathbf{u}$ (same dimensions as $\mathbf{f}$) by the force derivatives and places the results in $\mathbf{f}$. The notation $\delta\mathbf{u}[\mathbf{Z}]$ means take the entries from $\delta\mathbf{u}$ corresponding to the degrees of freedom $\mathbf{Z}$. Since force derivative matrix is not symmetric and we do not construct the matrix explicitly, we also provide a routine to multiply by its transpose.

2 Finite termination of mesh relaxation

In this section, we show that the mesh relaxation algorithm must terminate after a finite number of steps. The algorithm has 4 cases (1=initial, 2=triangle, 3=edge, 4=point). The initial case is when the attachment starts in a triangle interior; this is the starting case and is never revisited, so we do not consider it further. The cases are associated with a individual primitives of a triangulate surface and an attachment point $\mathbf{Y}$

Algorithm 5 Routines for penalty force between a particle and a fixed level set.

procedure PROJECT_ATTACHMENT_TO_SURFACE($\mathbf{X}$, ϕ, exit_early)
  Compute $\langle d, \mathbf{n}, \mathbf{H} \rangle$ ▷ Level set ϕ along with gradient and Hessian evaluated at $\mathbf{X}$.
  if exit_early **and** $d > 0$ **then**
    return $\langle \text{false}, \cdot, \cdot \rangle$
  $\mathbf{W} \leftarrow \mathbf{X} - d\mathbf{n}, \quad \dfrac{\partial \mathbf{W}}{\partial \mathbf{X}} \leftarrow \mathbf{I} - \mathbf{n}\mathbf{n}^T - d\mathbf{H}$
  return $\left\langle d \le 0, \mathbf{W}, \dfrac{\partial \mathbf{W}}{\partial \mathbf{X}} \right\rangle$

procedure PRECOMPUTE_DI($\mathbf{Z}$, ϕ)
  $\left\langle \text{is\_active}, \mathbf{W}, \dfrac{\partial \mathbf{W}}{\partial \mathbf{Z}} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_SURFACE($\mathbf{Z}, \phi, \text{true}$)
  Store: is_active
  if is_active **then** ▷ Check if intersecting
    $\left\langle \mathbf{K}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{Z}}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{X}}, \dfrac{\partial \mathbf{K}}{\partial \mathbf{W}} \right\rangle \leftarrow$ RELAX_PLANE($\mathbf{Z}, \mathbf{X}, \mathbf{W}, \mu$)
    $\left\langle \cdot, \mathbf{Y}, \dfrac{\partial \mathbf{Y}}{\partial \mathbf{K}} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_SURFACE($\mathbf{K}, \phi, \text{false}$)
    $\dfrac{\partial \mathbf{Y}}{\partial \mathbf{Z}} \leftarrow \dfrac{\partial \mathbf{Y}}{\partial \mathbf{K}} \left(\dfrac{\partial \mathbf{K}}{\partial \mathbf{Z}} + \dfrac{\partial \mathbf{K}}{\partial \mathbf{W}} \dfrac{\partial \mathbf{W}}{\partial \mathbf{Z}} \right)$
    Store: $\mathbf{Y}, \dfrac{\partial \mathbf{Y}}{\partial \mathbf{Z}}$

procedure FORCE_DI
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{f}[\mathbf{Z}] \leftarrow -k(\mathbf{Z} - \mathbf{Y}), \quad$ **return f**

procedure FORCE_DIFF_DI($\delta\mathbf{u}$)
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{f}[\mathbf{Z}] \leftarrow k \left(\dfrac{\partial \mathbf{Y}}{\partial \mathbf{Z}} - \mathbf{I} \right) \delta\mathbf{u}[\mathbf{Z}], \quad$ **return f**

procedure FORCE_DIFF_TRANSPOSE_DI($\delta\mathbf{u}$)
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{f}[\mathbf{Z}] \leftarrow k \left(\dfrac{\partial \mathbf{Y}}{\partial \mathbf{Z}} - \mathbf{I} \right)^T \delta\mathbf{u}[\mathbf{Z}], \quad$ **return f**

procedure UPDATE_ATTACHMENT_DI
  $\mathbf{X} \leftarrow \mathbf{Y}$; **Store:** $\mathbf{X}$

on the primitive. The attachment point moves in the direction of fastest decrease in distance to $\mathbf{Z}$, so the distance $\|\mathbf{Y} - \mathbf{Z}\|$ decreases whenever the attachment $\mathbf{Y}$ moves. We show that the attachment can visit any primitive (vertex, triangle, edge) at most a finite number of times; this guarantees termination.

Vertices and case 4. A vertex can be visited in case 4 (vertex case). It can also be visited in case 2 (on an edge) or in case 3 (on a triangle) as a degeneracy. The triangle case checks for this and transitions to case 4. The edge case always terminates or transitions to case 4. Thus, we can assume we are visiting the vertex in case 4. Case 4 checks to see whether progress can be made from a triangle or an edge; if progress can be made, it transitions to the triangle (case 2) or edge (case 3). Progress will be made in that case, and the attachment point will move. Since the attachment will then be closer to $\mathbf{Z}$ than the vertex, the vertex can never be revisited. If no progress can be made from case 4, the algorithm terminates. Thus, vertices can be visited at most twice (once in case 2 or 3, then once in case 4).

Case 3. When case 3 is visited, we are on an edge. If the closest point on the line to $\mathbf{Z}$ is in the interior of the edge, the algorithm terminates from this case. If the algorithm does not terminate, one of the endpoints must be closer to $\mathbf{Z}$, and the algorithm visits that endpoint in case 4. At this point, the attachment has moved to the endpoint of the edge, which is closer to $\mathbf{Z}$ than any other point on the edge. If the algorithm

Algorithm 6 Routines for penalty force between a particle and a rigid body.

procedure PRECOMPUTE_RD($\mathbf{Z}$)
  if Use level set **then** ▷ Check if intersecting
    $\left\langle \hat{\mathbf{X}}, \cdot, \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{v}}, \frac{\partial \hat{\mathbf{X}}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow$ FRAME_TIMES(rigid_body, $\mathbf{X}$)
    $\left\langle \text{is\_active}, \mathbf{W}, \frac{\partial \mathbf{W}}{\partial \mathbf{Z}}, \frac{\partial \mathbf{W}}{\partial \mathbf{v}}, \frac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_MOVING_SURFACE(rigid_body, $\mathbf{Z}, \phi$, true)
    if is_active **then**
      $\left\langle \mathbf{K}, \frac{\partial \mathbf{K}}{\partial \mathbf{Z}}, \frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}}, \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \right\rangle \leftarrow$ RELAX_PLANE($\mathbf{Z}, \hat{\mathbf{X}}, \mathbf{W}, \mu$)
      $\left\langle \cdot, \mathbf{V}, \frac{\partial \mathbf{V}}{\partial \mathbf{K}}, \frac{\partial \mathbf{V}}{\partial \mathbf{v}}, \frac{\partial \mathbf{V}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_MOVING_SURFACE(rigid_body, $\mathbf{K}, \phi$, true)
      $\mathbf{Y} \leftarrow \mathbf{V}, \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{Z}} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \mathbf{Z}} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathbf{Z}} \right)$
      $\frac{\partial \mathbf{Y}}{\partial \mathbf{v}} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}} \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{v}} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathbf{v}} \right) + \frac{\partial \mathbf{V}}{\partial \mathbf{v}}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}} \frac{\partial \hat{\mathbf{X}}}{\partial \boldsymbol{\omega}} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}} \right) + \frac{\partial \mathbf{V}}{\partial \boldsymbol{\omega}}$
  else
    $\left\langle \hat{\mathbf{Z}}, \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{Z}}, \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{v}}, \frac{\partial \hat{\mathbf{Z}}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow$ FRAME_TIMES(rigid_body, $\mathbf{Z}$)
    $\left\langle \text{is\_active}, e, w, \hat{\mathbf{Y}}, \text{diff\_list} \right\rangle \leftarrow$ RELAX_MESH($e_0, w_0, \hat{\mathbf{Z}}$, none, μ)
    if is_active **then**
      $\left\langle \mathbf{Y}, \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}}, \frac{\partial \mathbf{Y}}{\partial \mathbf{v}}, \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}} \right\rangle \leftarrow$ FRAME_TIMES(rigid_body, $\hat{\mathbf{Y}}$)
      $\frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{Z}} \leftarrow \mathbf{0}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}} \leftarrow \mathbf{0}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}} \leftarrow \mathbf{0}$
      for all $\langle \mathbf{D_Y}, \mathbf{D_Z}, \cdot \rangle \in$ diff_list **do** ▷ Visit in the order they were added to the list
        $\frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{Z}} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{Z}} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{Z}}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{v}}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \boldsymbol{\omega}}$
      $\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{Z}}, \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{v}} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}} + \frac{\partial \mathbf{Y}}{\partial \mathbf{v}}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}} + \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}}$
  Store: $\mathbf{Y}, e, w, \frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}, \frac{\partial \mathbf{Y}}{\partial \mathbf{v}}, \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}}$, is_active

procedure FORCE_RD
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \mathbf{f}[\mathbf{Z}] \leftarrow -\mathbf{j}, \quad \mathbf{f}[\mathbf{v}] \leftarrow \mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}] \leftarrow (\mathbf{Y} - \mathbf{x}_b) \times \mathbf{j}, \quad$ **return f**

procedure FORCE_DIFF_RD($\delta \mathbf{u}$)
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \delta \mathbf{Y} \leftarrow \frac{\partial \mathbf{Y}}{\partial \mathbf{Z}} \delta \mathbf{u}[\mathbf{Z}] + \frac{\partial \mathbf{Y}}{\partial \mathbf{v}} \delta \mathbf{u}[\mathbf{v}] + \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}} \delta \mathbf{u}[\boldsymbol{\omega}], \quad \delta \mathbf{j} \leftarrow k(\delta \mathbf{u}[\mathbf{Z}] - \delta \mathbf{Y})$
  $\mathbf{f}[\mathbf{Z}] \leftarrow -\delta \mathbf{j}, \quad \mathbf{f}[\mathbf{v}] \leftarrow \delta \mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}] \leftarrow (\mathbf{Y} - \mathbf{x}_b) \times \delta \mathbf{j} + (\delta \mathbf{Y} - \delta \mathbf{u}[\mathbf{v}]) \times \mathbf{j}, \quad$ **return f**

procedure FORCE_DIFF_TRANSPOSE_RD($\delta \mathbf{u}$)
  $\mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \delta \mathbf{v} \leftarrow \delta \mathbf{u}[\mathbf{v}] + \delta \mathbf{u}[\boldsymbol{\omega}] \times (\mathbf{Y} - \mathbf{x}), \quad \delta \boldsymbol{\omega} \leftarrow \delta \mathbf{u}[\boldsymbol{\omega}], \quad \delta \mathbf{j} \leftarrow \delta \mathbf{v} - \delta \mathbf{u}[\mathbf{Z}]$
  $\delta \mathbf{w} \leftarrow \mathbf{j} \times \delta \boldsymbol{\omega}, \quad \delta \mathbf{Z} \leftarrow k \delta \mathbf{j}, \quad \delta \mathbf{Y} \leftarrow \delta \mathbf{w} - \delta \mathbf{Z}, \quad \mathbf{f} \leftarrow \mathbf{0}$
  $\mathbf{f}[\mathbf{Z}] \leftarrow \delta \mathbf{Z} + \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}} \right)^T \delta \mathbf{Y}, \quad \mathbf{f}[\mathbf{v}] \leftarrow \left(\frac{\partial \mathbf{Y}}{\partial \mathbf{v}} \right)^T \delta \mathbf{Y} - \delta \mathbf{w}, \quad \mathbf{f}[\boldsymbol{\omega}] \leftarrow \left(\frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}} \right)^T \delta \mathbf{Y}, \quad$ **return f**

procedure UPDATE_ATTACHMENT_RD($\delta \mathbf{u}$)
  $e_0 \leftarrow e, w_0 \leftarrow w, \mathbf{X} \leftarrow$ FRAME_INVERSE_TIMES(rigid_body, $\mathbf{Y}$); **Store:** $e_0, w_0, \mathbf{X}$

Algorithm 7 Relaxation routine for contact between two rigid bodies.

procedure PRECOMPUTE_RR(s, i)
  if Use level set **then** ▷ Check if intersecting
    $\left\langle \mathbf{Z}, \cdot, \frac{\partial \mathbf{Z}}{\partial \mathbf{v}_s}, \frac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}_s} \right\rangle \leftarrow$ FRAME_TIMES$(s, \overline{\mathbf{Z}})$
    $\left\langle \hat{\mathbf{X}}, \cdot, \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{v}_i}, \frac{\partial \hat{\mathbf{X}}}{\partial \boldsymbol{\omega}_i} \right\rangle \leftarrow$ FRAME_TIMES$(i, \mathbf{X})$
    $\left\langle \text{is\_active}, \mathbf{W}, \frac{\partial \mathbf{W}}{\partial \mathbf{Z}}, \frac{\partial \mathbf{W}}{\partial \mathbf{v}_i}, \frac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}_i} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_MOVING_SURFACE$(i, \mathbf{Z}, \phi, \text{true})$
    if is_active **then**
      $\left\langle \mathbf{K}, \frac{\partial \mathbf{K}}{\partial \mathbf{Z}}, \frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}}, \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \right\rangle \leftarrow$ RELAX_PLANE$(\mathbf{Z}, \hat{\mathbf{X}}, \mathbf{W}, \mu)$
      $\left\langle \cdot, \mathbf{V}, \frac{\partial \mathbf{V}}{\partial \mathbf{K}}, \frac{\partial \mathbf{V}}{\partial \mathbf{v}_i}, \frac{\partial \mathbf{V}}{\partial \boldsymbol{\omega}_i} \right\rangle \leftarrow$ PROJECT_ATTACHMENT_TO_MOVING_SURFACE$(i, \mathbf{K}, \phi, \text{true})$
      $\mathbf{Y} \leftarrow \mathbf{V}, \quad \frac{\partial \mathbf{V}}{\partial \mathbf{Z}} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \mathbf{Z}} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathbf{Z}} \right), \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_s} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{Z}} \frac{\partial \mathbf{Z}}{\partial \mathbf{v}_s}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_s} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{Z}} \frac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}_s}$
      $\frac{\partial \mathbf{Y}}{\partial \mathbf{v}_i} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}} \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{v}_i} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathbf{v}_i} \right) + \frac{\partial \mathbf{V}}{\partial \mathbf{v}_i}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i} \leftarrow \frac{\partial \mathbf{V}}{\partial \mathbf{K}} \left(\frac{\partial \mathbf{K}}{\partial \hat{\mathbf{X}}} \frac{\partial \hat{\mathbf{X}}}{\partial \boldsymbol{\omega}_i} + \frac{\partial \mathbf{K}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \boldsymbol{\omega}_i} \right) + \frac{\partial \mathbf{V}}{\partial \boldsymbol{\omega}_i}$
  else
    $\left\langle \mathbf{Z}, \cdot, \frac{\partial \mathbf{Z}}{\partial \mathbf{v}_s}, \frac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}_s} \right\rangle \leftarrow$ FRAME_TIMES$(s, \overline{\mathbf{Z}})$
    $\left\langle \hat{\mathbf{Z}}, \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{Z}}, \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{v}_i}, \frac{\partial \hat{\mathbf{Z}}}{\partial \boldsymbol{\omega}_i} \right\rangle \leftarrow$ FRAME_INVERSE_TIMES$(i, \mathbf{Z})$
    $\left\langle \text{is\_active}, e, w, \hat{\mathbf{Y}}, \text{diff\_list} \right\rangle \leftarrow$ RELAX_MESH$(e_0, w_0, \hat{\mathbf{Z}}, \text{none}, \mu)$
    if is_active **then**
      $\left\langle \mathbf{Y}, \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}}, \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_i}, \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i} \right\rangle \leftarrow$ FRAME_TIMES$(i, \hat{\mathbf{Y}})$
      $\frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_s} \leftarrow \mathbf{0}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_s} \leftarrow \mathbf{0}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_i} \leftarrow \mathbf{0}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_i} \leftarrow \mathbf{0}$
      for all $\langle \mathbf{D_Y}, \mathbf{D_Z}, \cdot \rangle \in$ diff_list **do** ▷ Visit in the order they were added to the list
        $\frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_s} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_s} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{v}_s}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_s} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_s} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \boldsymbol{\omega}_s}$
        $\frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_i} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_i} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \mathbf{v}_i}, \quad \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_i} \leftarrow \mathbf{D_Y} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_i} + \mathbf{D_Z} \frac{\partial \hat{\mathbf{Z}}}{\partial \boldsymbol{\omega}_i}$
      $\frac{\partial \mathbf{Y}}{\partial \mathbf{v}_s} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_s}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_s} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_s}, \quad \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_i} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{v}_i} + \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_i}, \quad \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i} \leftarrow \frac{\partial \mathbf{Y}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \boldsymbol{\omega}_i} + \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i}$
  Store: $\mathbf{Z}, \mathbf{Y}, e, w, \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_s}, \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_s}, \frac{\partial \mathbf{Y}}{\partial \mathbf{v}_i}, \frac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i}$, is_active

does not terminate from case 4, the following step must move $\mathbf{Y}$. The attachment is now closer to $\mathbf{Z}$ than *any* point on the edge; the edge can never be revisited. Thus, an edge can be visited in case 3 at most once.

At this point, the only case we must consider is when edges are visited in case 2 as the edges of triangles. This is by far the most common case. The three edges of a triangle can be labeled as *entering* or *leaving* edges. If the projection of $\mathbf{Z}$ into the plane of the triangle lies on the other side of the edge from the interior of the triangle, the edge is a leaving edge. Otherwise it is an entering edge.

When the attachment transitions from case 2, it must do so through a vertex or through a leaving edge. If the attachment point enters through a leaving edge, it immediately transitions to the edge in case 3; this ensures that the edge is never visited again. Since vertices can only be visited a finite number of times, the

Algorithm 8 Force routines for penalty force between two rigid bodies.

procedure FORCE_RR(s, i)
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y})$
  $\mathbf{f}[\mathbf{v}_s] \leftarrow -\mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}_s] \leftarrow (\mathbf{Z} - \mathbf{x}_s) \times -\mathbf{j}, \quad \mathbf{f}[\mathbf{v}_i] \leftarrow \mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}_i] \leftarrow (\mathbf{Y} - \mathbf{x}_i) \times \mathbf{j}, \quad$ **return f**

procedure FORCE_DIFF_RR(s, i, $\delta\mathbf{u}$)
  $\mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \delta\mathbf{Z} \leftarrow \dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}_s}\delta\mathbf{u}[\mathbf{v}_s] + \dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}_s}\delta\mathbf{u}[\boldsymbol{\omega}_s]$
  $\delta\mathbf{Y} \leftarrow \dfrac{\partial \mathbf{Y}}{\partial \mathbf{v}_s}\delta\mathbf{u}[\mathbf{v}_s] + \dfrac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_s}\delta\mathbf{u}[\boldsymbol{\omega}_s] + \dfrac{\partial \mathbf{Y}}{\partial \mathbf{v}_i}\delta\mathbf{u}[\mathbf{v}_i] + \dfrac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i}\delta\mathbf{u}[\boldsymbol{\omega}_i], \quad \delta\mathbf{j} \leftarrow k(\delta\mathbf{Z} - \delta\mathbf{Y})$
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{f}[\mathbf{v}_s] \leftarrow \delta\mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}_s] \leftarrow (\mathbf{Y} - \mathbf{x}_b) \times \delta\mathbf{j} + (\delta\mathbf{Y} - \delta\mathbf{u}[\mathbf{v}]) \times \mathbf{j}$
  $\mathbf{f}[\mathbf{v}_i] \leftarrow -\delta\mathbf{j}, \quad \mathbf{f}[\boldsymbol{\omega}_i] \leftarrow (\mathbf{Y} - \mathbf{x}_b) \times -\delta\mathbf{j} + (\delta\mathbf{Y} - \delta\mathbf{u}[\mathbf{v}]) \times -\mathbf{j}, \quad$ **return f**

procedure FORCE_DIFF_TRANSPOSE_RR(s, i, $\delta\mathbf{u}$)
  $\mathbf{j} \leftarrow -k(\mathbf{Z}-\mathbf{Y}), \quad \delta\mathbf{v}_s \leftarrow \delta\mathbf{u}[\mathbf{v}_s]+\delta\mathbf{u}[\boldsymbol{\omega}_s]\times(\mathbf{Z}-\mathbf{x}_s), \quad \delta\boldsymbol{\omega}_s \leftarrow \delta\mathbf{u}[\boldsymbol{\omega}_s], \quad \delta\mathbf{v}_i \leftarrow \delta\mathbf{u}[\mathbf{v}_i]+\delta\mathbf{u}[\boldsymbol{\omega}_i]\times(\mathbf{Y}-\mathbf{x}_i)$
  $\delta\boldsymbol{\omega}_i \leftarrow \delta\mathbf{u}[\boldsymbol{\omega}_i], \quad \delta\mathbf{j} \leftarrow \delta\mathbf{v}_i-\delta\mathbf{v}_s, \quad \delta\mathbf{w}_s \leftarrow \mathbf{j}\times\delta\boldsymbol{\omega}_s, \quad \delta\mathbf{w}_i \leftarrow \mathbf{j}\times\delta\boldsymbol{\omega}_i, \quad \delta\mathbf{Z} \leftarrow \delta\mathbf{j}-\delta\mathbf{w}_s, \quad \delta\mathbf{Y} \leftarrow \delta\mathbf{w}_i-\delta\mathbf{j}$
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{f}[\mathbf{v}_i] \leftarrow \left(\dfrac{\partial \mathbf{Y}}{\partial \mathbf{v}_i}\right)^T \delta\mathbf{Y} - \delta\mathbf{w}_i, \quad \mathbf{f}[\boldsymbol{\omega}_i] \leftarrow \left(\dfrac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_i}\right)^T \delta\mathbf{Y}$
  $\mathbf{f}[\mathbf{v}_s] \leftarrow \left(\dfrac{\partial \mathbf{Y}}{\partial \mathbf{v}_s}\right)^T \delta\mathbf{Y} + \left(\dfrac{\partial \mathbf{Z}}{\partial \mathbf{v}_s}\right)^T \delta\mathbf{Z} + \delta\mathbf{w}_s, \quad \mathbf{f}[\boldsymbol{\omega}_s] \leftarrow \left(\dfrac{\partial \mathbf{Y}}{\partial \boldsymbol{\omega}_s}\right)^T \delta\mathbf{Y} + \left(\dfrac{\partial \mathbf{Z}}{\partial \boldsymbol{\omega}_s}\right)^T \delta\mathbf{Z}, \quad$ **return f**

procedure UPDATE_ATTACHMENT_RR($\delta\mathbf{u}$)
  $e_0 \leftarrow e, w_0 \leftarrow w, \mathbf{X} \leftarrow$ FRAME_INVERSE_TIMES$(i, \mathbf{Y})$; **Store:** $e_0, w_0, \mathbf{X}$

only case that needs to be considered is that one or more triangles are being visited multiple times from entering edges in case 2. Each of these edges is a leaving edge to one adjacent triangle and an entering edge to the other. When an attachment enters through an entering edge and leaves through a leaving edge, the attachment has necessarily moved.

Next, consider the accumulation points of the path that attachment traverses if this algorithm does not terminate and is left to run forever. The distance to $\mathbf{Z}$ is a monotonically decreasing nonnegative function, so it must converge to some distance. The accumulation points of the attachment path are all at this distance to the point $\mathbf{Z}$. Note that within each triangle, the accumulation points must form a line; since it is not possible for a line to contain points all of the same distance to $\mathbf{Z}$, there cannot be accumulation points in the interiors of the triangles. The same observation excludes them from the interiors of the edges. Thus, the accumulation point must be a single vertex. Thus, after a finite number of steps, the attachment point will be restricted to the triangles and edges adjacent to a single vertex.

The attachment point must visit the triangles in order (since for each triangle the attachment must enter through the entering edge and leave through the leaving edge). Thus, we must have a spiral motion of the attachment point. In the language of differential equations, this motion is a gradient system (the motion is locally gradient descent on distance to $\mathbf{Z}$). Equilibrium points of such systems must be nodes or saddle points; they can never be spirals (See [Dob17], page 526). The spiral path around the vertex that must exist if the algorithm fails to terminate is therefore impossible. It follows that the algorithm must eventually terminate.

The proof presented is not constructive, and we do not have a proven bound on the number of times that a triangle can be visited. In practice, we only observe primitives to be visited once in any particular state.

References

[Ano18] ANONYMOUS: Penalty force for coupling materials with coulomb friction. In *Proceedings of the 2018 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2018), ACM, p. 0.

Algorithm 9 Routines for penalty force between deformable bodies.

procedure PRECOMPUTE_DD
  $\langle \text{is\_active}, e, \mathbf{w}, \mathbf{Y}, \text{diff\_list}, \text{weight\_diff} \rangle \leftarrow \text{RELAX\_MESH}(e_0, w_0, \mathbf{Z}, p, \mu)$
  Store: is_active, $e, \mathbf{w}, \mathbf{Y}$, diff_list, weight_diff

procedure FORCE_DD
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \mathbf{f}[\mathbf{Z}] \leftarrow -\mathbf{j}$
  for all $i \in \{0, 1, 2\}$ **do**
    $\mathbf{f}[\text{vertices}[e][i]] \leftarrow \mathbf{w}[i]\mathbf{j}$
  return f

procedure FORCE_DIFF_DD($\delta\mathbf{u}$)
  $\delta\mathbf{Y} \leftarrow \sum_{i=0}^{2} \mathbf{w}_0[i]\delta\mathbf{u}[\text{vertices}[e_0][i]], \quad \delta\hat{\mathbf{Y}} \leftarrow \mathbf{0}, \quad \delta\mathbf{w} \leftarrow \mathbf{0}$
  for all $\langle \mathbf{D_Y}, \mathbf{D_Z}, \mathbf{D_A}, \mathbf{D_B}, \mathbf{D_C}, \hat{e} \rangle \in$ diff_list **do** ▷ Visit in the order they were added to the list
    $\delta\hat{\mathbf{Y}} \leftarrow \delta\mathbf{Y}$
    $\delta\mathbf{Y} \leftarrow \mathbf{D_Y}\delta\mathbf{Y} + \mathbf{D_Z}\delta\mathbf{u}[\mathbf{Z}] + \mathbf{D_A}\delta\mathbf{u}[\text{vertices}[\hat{e}][0]] + \mathbf{D_B}\delta\mathbf{u}[\text{vertices}[\hat{e}][1]] + \mathbf{D_C}\delta\mathbf{u}[\text{vertices}[\hat{e}][2]]$
  if diff_list is not empty **then**
    $\langle \mathbf{E_Y}, \mathbf{E_Z}, \mathbf{E_A}, \mathbf{E_B}, \mathbf{E_C} \rangle \leftarrow$ weight_diff
    $\delta\mathbf{w} \leftarrow \mathbf{E_Y}\delta\hat{\mathbf{Y}} + \mathbf{E_Z}\delta\mathbf{u}[\mathbf{Z}] + \mathbf{E_A}\delta\mathbf{u}[\text{vertices}[e][0]] + \mathbf{E_B}\delta\mathbf{u}[\text{vertices}[e][1]] + \mathbf{E_C}\delta\mathbf{u}[\text{vertices}[e][2]]$
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \delta\mathbf{j} \leftarrow k(\delta\mathbf{u}[\mathbf{Z}] - \delta\mathbf{Y}), \quad \mathbf{f}[\mathbf{Z}] \leftarrow -\delta\mathbf{j}$
  for all $i \in \{0, 1, 2\}$ **do**
    $\mathbf{f}[\text{vertices}[e][i]] \leftarrow \mathbf{w}[i]\delta\mathbf{j} + \delta\mathbf{w}[i]\mathbf{j}$
  return f

procedure FORCE_DIFF_TRANSPOSE_DD
  $\mathbf{j} \leftarrow k(\mathbf{Z} - \mathbf{Y}), \quad \delta\mathbf{j} \leftarrow -\delta\mathbf{u}[\mathbf{Z}] + \sum_{i=0}^{2} \mathbf{w}_0[i]\delta\mathbf{u}[\text{vertices}[e_0][i]]$
  for all $i \in \{0, 1, 2\}$ **do**
    $\delta\mathbf{w}[i] \leftarrow \mathbf{j} \cdot \delta\mathbf{u}[\text{vertices}[e_0][i]]$
  $\mathbf{f} \leftarrow \mathbf{0}, \quad \delta\mathbf{Y} \leftarrow -k\delta\mathbf{j}, \quad \mathbf{f}[\mathbf{Z}] \leftarrow -\delta\mathbf{Y}$
  if diff_list is empty **then**
    $\delta\mathbf{Z} \leftarrow \mathbf{0}, \quad \delta\hat{\mathbf{Y}} \leftarrow \mathbf{0}$
  else
    $\langle \mathbf{E_Y}, \mathbf{E_Z}, \mathbf{E_A}, \mathbf{E_B}, \mathbf{E_C} \rangle \leftarrow$ weight_diff
    $\delta\mathbf{f}[\text{vertices}[e][0]] += \mathbf{E_A}^T\delta\mathbf{w}, \quad \delta\mathbf{f}[\text{vertices}[e][1]] += \mathbf{E_B}^T\delta\mathbf{w}, \quad \delta\mathbf{f}[\text{vertices}[e][2]] += \mathbf{E_C}^T\delta\mathbf{w}$
    $\delta\hat{\mathbf{Y}} \leftarrow \mathbf{E_Y}^T\delta\mathbf{w}, \quad \delta\mathbf{Z} \leftarrow \mathbf{E_Z}^T\delta\mathbf{w}$
  for all $\langle \mathbf{D_Y}, \mathbf{D_Z}, \mathbf{D_A}, \mathbf{D_B}, \mathbf{D_C}, \hat{e} \rangle \in$ diff_list **do** ▷ Visit in the order they were added to the list
    $\delta\mathbf{f}[\text{vertices}[\hat{e}][0]] += \mathbf{D_A}^T\delta\mathbf{Y}, \quad \delta\mathbf{f}[\text{vertices}[\hat{e}][1]] += \mathbf{D_B}^T\delta\mathbf{Y}, \quad \delta\mathbf{f}[\text{vertices}[\hat{e}][2]] += \mathbf{D_C}^T\delta\mathbf{Y}$
    $\delta\mathbf{Z} \leftarrow \mathbf{D_Z}^T\delta\mathbf{Y}, \quad \delta\mathbf{Y} \leftarrow \mathbf{D_Y}^T\delta\mathbf{Y} + \delta\hat{\mathbf{Y}}, \quad \delta\hat{\mathbf{Y}} \leftarrow \mathbf{0}$
  for all $i \in \{0, 1, 2\}$ **do**
    $\mathbf{f}[\text{vertices}[e_0][i]] += \mathbf{w}_0[i]\delta\mathbf{Y}$
  $\mathbf{f}[\mathbf{Z}] += \delta\mathbf{Z}$
  return f

[Dob17] DOBRUSHKIN V.: *Applied Differential Equations with Boundary Value Problems.* CRC Press, 2017.

[NW06] NOCEDAL J., WRIGHT S.: *Numerical Optimization.* Springer series in operations research and financial engineering. Springer, 2006.

Algorithm 10 Mesh relaxation routines (Part I).

procedure RELAX_MESH(e_o, w_0, $\mathbf{Z}$, ρ, μ)
  $\mathbf{X} \leftarrow$ point on mesh element e_0 with barycentric weights w_0
  $\langle c, \mathbf{Y}, \mathbf{w}, e\rangle \leftarrow$ CASE_TRIANGLE($\mathbf{X}, \mathbf{Z}, w_0, e_0, \rho, \mu,$ true)
  if c **then**
    return $\langle$true$, e, \mathbf{w}, \mathbf{Y},$ diff_list, weight_diff$\rangle$
  else
    return $\langle$false$, \cdot\rangle$

procedure CASE_TRIANGLE($\mathbf{X}$, $\mathbf{Z}$, $\mathbf{w}_0$, e, ρ, μ, first)
  if $\rho \in$ vertices$[e]$ **then**
    return $\langle$false$, \cdot, \cdot, \cdot\rangle$
  $\langle \mathbf{A}, \mathbf{B}, \mathbf{C}\rangle \leftarrow$ vertices of triangle e
  $\mathbf{u} \leftarrow \mathbf{A} - \mathbf{C}, \quad \mathbf{v} \leftarrow \mathbf{A} - \mathbf{C}, \quad \mathbf{x} \leftarrow \mathbf{X} - \mathbf{C}, \quad \mathbf{z} \leftarrow \mathbf{Z} - \mathbf{C}, \quad \mathbf{N} \leftarrow \mathbf{u} \times \mathbf{v}, \quad \mathbf{W} \leftarrow \mathbf{z} - \dfrac{\mathbf{z} \cdot \mathbf{N}}{\mathbf{N} \cdot \mathbf{N}}\mathbf{N}$
  inside $\leftarrow (z \cdot N \le 0), \quad \alpha \leftarrow |d|\mu, \quad \beta \leftarrow \|\mathbf{x} - \mathbf{W}\|^2$
  if $\beta < \alpha^2$ **then**
    $\mathbf{Y} \leftarrow \mathbf{X}$
    return $\langle$inside$, \mathbf{Y}, \mathbf{w}_0, e\rangle$
  $a \leftarrow \dfrac{\alpha}{\sqrt{\beta}}, \quad \mathbf{y} \leftarrow \mathbf{W} + a(\mathbf{x} - \mathbf{W})$
  $\mathbf{M} \leftarrow \begin{pmatrix}\mathbf{u} & \mathbf{v}\end{pmatrix}, \quad \mathbf{q} \leftarrow (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{y}, \quad \mathbf{r} \leftarrow (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{x}, \quad \mathbf{Y} \leftarrow \mathbf{y} + \mathbf{C}$
  if Segment $\mathbf{q}, \mathbf{r}$ intersects triangle $(0,0), (1,0), (0,1)$ **then**
    $\mathbf{q} \leftarrow$ intersection point
    $\mathbf{Y} \leftarrow \mathbf{q}[0]\mathbf{u} + \mathbf{q}[1]\mathbf{v} + \mathbf{C}$
    Record which triangle edge was intersected
  $\mathbf{w} \leftarrow \big(\mathbf{q}[0], \mathbf{q}[1], 1 - \mathbf{q}[0] - \mathbf{q}[1]\big)$
  Compute $\mathbf{D_Y} = \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}, \mathbf{D_Z} = \frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}, \mathbf{D_A} = \frac{\partial \mathbf{Y}}{\partial \mathbf{A}}, \mathbf{D_B} = \frac{\partial \mathbf{Y}}{\partial \mathbf{B}}, \mathbf{D_C} = \frac{\partial \mathbf{Y}}{\partial \mathbf{C}}$ by autodifferentiation
  Compute $\mathbf{E_Y} = \frac{\partial \mathbf{w}}{\partial \mathbf{X}}, \mathbf{E_Z} = \frac{\partial \mathbf{w}}{\partial \mathbf{Z}}, \mathbf{E_A} = \frac{\partial \mathbf{w}}{\partial \mathbf{A}}, \mathbf{E_B} = \frac{\partial \mathbf{w}}{\partial \mathbf{B}}, \mathbf{E_C} = \frac{\partial \mathbf{w}}{\partial \mathbf{C}}$ by autodifferentiation
  Overwrite weight_diff $\leftarrow \langle \mathbf{E_Y}, \mathbf{E_Z}, \mathbf{E_A}, \mathbf{E_B}, \mathbf{E_C}\rangle$
  Append $\langle \mathbf{D_Y}, \mathbf{D_Z}, \mathbf{D_A}, \mathbf{D_B}, \mathbf{D_C}, e\rangle$ to diff_list
  if Segment intersected triangle **then**
    $\mathbf{q} \leftarrow$ intersection point
    for all $i \in \{0, 1, 2\}$ **do**
      if $\mathbf{w}[i] > 1 - 128\epsilon$ **then**
        return CASE_VERTEX($\mathbf{Z}$, vertices$[e][i], \rho, \mu$)
    if not first **and** $\mathbf{X} = \mathbf{Y}$ **then** ▷ Attachment leaves on same edge it entered
      Segment intersected triangle edge (a, b)
      Use $\mathbf{w}$ to compute barycentric weights $\mathbf{w}_e$ on edge
      return CASE_EDGE($\mathbf{Y}, \mathbf{Z}, \mathbf{w}_e, (a, b), \rho, \mu$)
    else
      Segment intersected triangle edge; let $\hat{e}$ be the element on the other side
      Use $\mathbf{w}$ to compute barycentric weights $\hat{\mathbf{w}}$ in triangle $\hat{e}$
      return CASE_TRIANGLE($\mathbf{Y}, \mathbf{Z}, \hat{\mathbf{w}}, \hat{e}, \rho, \mu,$ false)
  else
    return $\langle$inside$, \mathbf{Y}, \mathbf{w}, e\rangle$

Algorithm 11 Mesh relaxation routines (Part II).

procedure CASE_EDGE($\mathbf{X}$, $\mathbf{Z}$, $\mathbf{w}_0$, edge, ρ, μ)
  if $\rho \in$ vertices of edge **then**
    return $\langle \text{false}, \cdot, \cdot, \cdot \rangle$
  $\langle \mathbf{A}, \mathbf{B} \rangle \leftarrow$ vertices of edge
  $\mathbf{u} \leftarrow \mathbf{A} - \mathbf{C}, \mathbf{x} \leftarrow \mathbf{X} - \mathbf{C}, \mathbf{z} \leftarrow \mathbf{Z} - \mathbf{C}, \mathbf{W} \leftarrow \dfrac{\mathbf{z} \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}} \mathbf{u}, \alpha \leftarrow \|\mathbf{z} - \mathbf{W}\|^2 \mu^2, \beta \leftarrow \|\mathbf{x} - \mathbf{W}\|^2$
  if $\beta < \alpha$ **then**
    Select a triangle e containing edge, compute triangle barycentric weights $\mathbf{w}$ from edge weights $\mathbf{w}_0$
    return $\langle$INSIDE_EDGE$(\mathbf{Z}, \text{edge}), \mathbf{X}, \mathbf{w}, e \rangle$
  $a \leftarrow \sqrt{\dfrac{\alpha}{\beta}}, \mathbf{y} \leftarrow \mathbf{W} + a(\mathbf{x} - \mathbf{W}), q \leftarrow \dfrac{\mathbf{y} \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}}$
  if $q < 0$ **then**
    $\mathbf{Y} \leftarrow \mathbf{B}, \quad f \leftarrow \text{true}, \quad w \leftarrow 0$
  else if $q > 1$ **then**
    $\mathbf{Y} \leftarrow \mathbf{A}, \quad f \leftarrow \text{true}, \quad w \leftarrow 1$
  else
    $\mathbf{Y} \leftarrow q\mathbf{u} + \mathbf{B}, \quad f \leftarrow \text{false}, \quad w \leftarrow q$
  Compute $\mathbf{D_Y} = \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}, \mathbf{D_Z} = \frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}, \mathbf{D_A} = \frac{\partial \mathbf{Y}}{\partial \mathbf{A}}, \mathbf{D_B} = \frac{\partial \mathbf{Y}}{\partial \mathbf{B}}$ by autodifferentiation
  Compute $\mathbf{E_Y} = \frac{\partial \mathbf{w}}{\partial \mathbf{X}}, \mathbf{E_Z} = \frac{\partial \mathbf{w}}{\partial \mathbf{Z}}, \mathbf{E_A} = \frac{\partial \mathbf{w}}{\partial \mathbf{A}}, \mathbf{E_B} = \frac{\partial \mathbf{w}}{\partial \mathbf{B}}$ by autodifferentiation
  Choose a triangle e containing this edge, compute barycentric weights $\mathbf{w}$ from edge weight w
  Relabel $\mathbf{D_A}, \mathbf{D_B}, \mathbf{D_C}, \mathbf{E_A}, \mathbf{E_B}, \mathbf{E_C}$ to correspond to the vertices of this triangle (two will be zero)
  Overwrite weight_diff $\leftarrow \langle \mathbf{E_Y}, \mathbf{E_Z}, \mathbf{E_A}, \mathbf{E_B}, \mathbf{E_C} \rangle$
  Append $\langle \mathbf{D_Y}, \mathbf{D_Z}, \mathbf{D_A}, \mathbf{D_B}, \mathbf{D_C}, e \rangle$ to diff_list
  if f **then**
    return CASE_VERTEX$(\mathbf{Z}, \bar{v}, \rho, \mu)$
  else
    return $\langle$INSIDE_EDGE$(\mathbf{Z}, \text{edge}), \mathbf{Y}, \mathbf{w}, e \rangle$

procedure CASE_VERTEX($\mathbf{Z}$, p, ρ, μ)
  if $\rho = p$ **then**
    return $\langle \text{false}, \cdot, \cdot, \cdot \rangle$
  $\mathbf{C} \leftarrow$ position of particle $p, \quad \mathbf{z} = \mathbf{Z} - \mathbf{C}$
  for all triangles e containing point p **do**
    $\langle \mathbf{A}, \mathbf{B} \rangle \leftarrow$ other two vertex positions
    $\mathbf{u} = \mathbf{A} - \mathbf{C}, \quad \mathbf{v} = \mathbf{B} - \mathbf{C}, \quad \mathbf{n} = \mathbf{u} \times \mathbf{v}, \quad a = \dfrac{\mathbf{z} \cdot (\mathbf{v} \times \mathbf{n})}{\mathbf{n} \cdot \mathbf{n}}, \quad b = -\dfrac{\mathbf{z} \cdot (\mathbf{u} \times \mathbf{n})}{\mathbf{n} \cdot \mathbf{n}}$
    if $a \geq 128\epsilon$ **and** $b \geq 128\epsilon$ **then**
      Compute barycentric weights $\mathbf{w}$ of vertex $\mathbf{C}$ in e
      return CASE_TRIANGLE$(\mathbf{C}, \mathbf{Z}, \mathbf{w}, e, \rho, \mu, \text{false})$
  for all edges g containing point p **do**
    $\mathbf{A} \leftarrow$ other vertex position
    if $(\mathbf{A} - \mathbf{C}) \cdot \mathbf{z} \geq 128\epsilon$ **then**
      Compute barycentric weights $\mathbf{w}_e$ of vertex $\mathbf{C}$ in g
      return CASE_EDGE$(\mathbf{C}, \mathbf{Z}, \mathbf{w}_e, g, \rho, \mu)$
  Select a triangle e containing p, compute barycentric weights $\mathbf{w}$ of p in e
  return $\langle$INSIDE_VERTEX$(\mathbf{Z}, p), \mathbf{C}, \mathbf{w}, e \rangle$

Algorithm 12 Mesh relaxation routines (Part III).

procedure INSIDE_EDGE($\mathbf{Z}$, edge)
  Let t_0 and t_1 be the triangles adjacent to edge
  $i_0 \leftarrow \mathrm{Vol}(t_0, \mathbf{Z}) \leq 0$, $i_1 \leftarrow \mathrm{Vol}(t_1, \mathbf{Z}) \leq 0$ ▷ determine inside/outside per triangle via tet volumes
  if $i_0 = i_1$ **then**
    return i_0
  Let q be vertex of t_0 not on edge
  return $\mathrm{Vol}(t_1, q) > 0$ ▷ Resolve by testing convexity

procedure INSIDE_VERTEX($\mathbf{Z}$, p)
  $s \leftarrow 0$ ▷ Solid angle centered at $\mathbf{Z}$ of triangle fan around p
  for all triangles (p, a, b) containing p **do** ▷ Triangle vertices are oriented
    Get triangle vertices: $\mathbf{P}, \mathbf{A}, \mathbf{B}$
    $\mathbf{w} \leftarrow \mathbf{P} - \mathbf{Z}, \quad \mathbf{u} \leftarrow \mathbf{A} - \mathbf{P}, \quad \mathbf{v} \leftarrow \mathbf{B} - \mathbf{P}$, $a \leftarrow \|\mathbf{w}\|, \quad b \leftarrow \|\mathbf{u}\|, \quad c \leftarrow \|\mathbf{v}\|$
    $s \leftarrow s + 2\,\mathrm{atan2}(\mathbf{w} \cdot (\mathbf{u} \times \mathbf{v}), abc + a(\mathbf{u} \cdot \mathbf{v}) + b(\mathbf{v} \cdot \mathbf{w}) + c(\mathbf{w} \cdot \mathbf{u}))$ ▷ Solid angle for triangle
  return $s > 0$