# SOFA Tearing Plugin

Tearing plugin is a simulation tool designed to replicate the tearing process, offering users the flexibility to control and observe fracture scenarios. Whether you prefer specifying a fracture scenario with precision or allowing fractures to occur dynamically due to simulated forces, this plugin empowers you with versatile tearing simulations.

## Features

- Low-level API (C++ methods) is accessible through the **TearingAlgorithms** objects for surface tissue, enabling the computation of the fracture path from a specified point in a particular direction. Similarly, **VolumeTearingAlgorithms** facilitates the computation of plane intersections from elongated tetrahedra.
- A high-level API is also accessible through SOFA components, namely the **TearingEngine** and **VolumeTearingEngine**. These components can be employed in a SOFA scene to execute the fracture operation, allowing for either the specification of a starting point and fracture direction or a force-driven fracture.
- Several data elements are available to control the fracture, such as:

  - stressThreshold: Threshold value for element stress which will determine candidate elements (triangles) to initiate the fracture.
  - fractureMaxLength: Maximum length of the fracture segment per simulation step.
  - nbFractureMax: Maximum number of allowed fracture occurrences for the simulation.

## Installation

This plugin should be added as an external plugin of SOFA using the CMAKE_EXTERNAL_DIRECTORIES CMake variable of SOFA. See SOFA documentation for more information.

## Getting Started with TearingEngine API

### Flexible fracture scenarios

Several scenarios through SOFA scene files are available in *scenes* directory. Corresponding screenshots of the results can be found in the *doc* directory.

To specify a fracture within Tearing plugin, you need to provide the following details:

1. Staring Point. Indicate the initial location where the fracture will initiate.

   Example

   ```
   <TearingEngine name="TearingEngine" input_position="@Mo.position""
   startVertexId="421"  startDirection="1.0 0.0 0.0" startLength="5"/>
   ```

2. Fracture Direction. Define the direction in which the fracture will propagate.

   Example

   ```
   <TearingEngine name="TearingEngine" input_position="@Mo.position""
   startVertexId="421"  startDirection="1.0 0.0 0.0" startLength="5"/>
   ```

3. Fracture Length. Specify the length of the fracture to control its extent.

   Example

   ```
   <TearingEngine name="TearingEngine" input_position="@Mo.position""
   startVertexId="421"  startDirection="1.0 0.0 0.0" startLength="5"/>
   ```

### Force-Driven Tearing

Tearing plugin provides the capability to simulate tearing dynamically, allowing fractures to occur naturally in response to applied forces without

the need to explicitly specify a fracture scenario. This feature enables users to observe the real-time effects of forces on the simulated material.

Example

External force →

Internal force →

```
<ConstantForceField name="CFF" indices="462" forces="0 -300 0"
showArrowSize=".01" />

<TriangularFEMForceField name="FEM" youngModulus="100"
poissonRatio="0.3" method="large"/>

<TearingEngine name="TearingEngine" input_position="@Mo.position"
stressThreshold="0.1" step="20" nbFractureMax="4" />
```

**Remark.** it's important to note that the simulation's behaviour is contingent upon both the properties of the applied forces, such as magnitude, and the specified material property, particularly the stress threshold. For example, changing the data parameters for a same mesh cause different result:

```
<ConstantForceField name="CFF" indices="462"
forces="0 -300 0" showArrowSize=".01" />

<TriangularFEMForceField name="FEM"
youngModulus="100" poissonRatio="0.3"
method="large"/>

<TearingEngine name="TearingEngine"
input_position="@Mo.position"
stressThreshold="5.0" step="20"
nbFractureMax="4" />
```

```
<ConstantForceField name="CFF" indices="462"
forces="0 -300 0" showArrowSize=".01" />

<TriangularFEMForceField name="FEM"
youngModulus="60" poissonRatio="0.3"
method="large"/>

<TearingEngine name="TearingEngine"
input_position="@Mo.position"
stressThreshold="1.0" step="20"
nbFractureMax="4" />
```


## Principal steps of force-driven fracture simulation

Our fracture simulation follows four principal steps:

1. It generates a list of candidate triangles capable of experiencing fractures. This list is determined based on stressThreshold. Each triangle with a principal stress value exceeding this specified threshold is included in the candidate list.
2. Among all the triangles included in the list, the one with the highest value is considered as the triangle to perform the fracture.
3. To determine the starting vertex, there are three available options. Let $\sigma_i$ indicates the principal stress values of triangles adjacent to

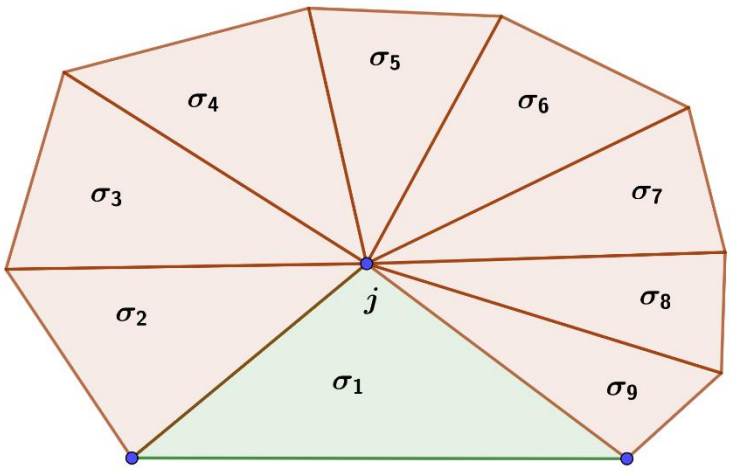


*Figure 1: A value is attributed to vertex j based on the principal values of the triangles adjacent to it.*

vertex j in the candidate triangle. In each method, the vertex with maximum $s_j$ value is selected as the starting point for fracture, see Figure 1.

Using Area-weighted average, we define $s_j$:

$$s_j = \frac{\sum_i |\sigma_{T_i}| A_{T_i}}{\sum_i A_{T_i}},$$

where $A_{T_i}$ indicates the area of the i-th triangle adjacent to vertex j. To activate this option, you need to pass `WeightedAverageArea` to `ComputeVertexStressMethod`.

```
<TearingEngine name="TearingEngine" input_position="@Mo.position"
ComputeVertexStressMethod= “WeightedAverageArea” />
```

Using unweighted average, we define $s_j$:

$$s_j = \sum |\sigma_{T_i}|.$$

To activate this option, you need to pass `UnweightedAverageArea` to `ComputeVertexStressMethod`.

```
<TearingEngine name="TearingEngine" input_position="@Mo.position"
ComputeVertexStressMethod= “UnweightedAverageArea” />
```

Using reciprocal-distance weighted average,

$$s_j = \sum_i \frac{|\sigma_{T_i}|}{d_{ij}}.$$

To activate this option, you need to pass `WeightedAverageInverseDistance` to `ComputeVertexStressMethod`.

```
<TearingEngine name="TearingEngine" input_position="@Mo.position"
ComputeVertexStressMethod= “WeightedAverageInverseDistance” />
```

4. To initiate the fracture, it is necessary to determine both its direction and length.
   The direction of the fracture is calculated to be perpendicular to the first dominant principal stress direction.
   The fracture length is specified by the user. You can define this parameter by passing value to fractureMaxLength.

   Example

```
<TearingEngine name="TearingEngine"
input_position="@Mo.position" fractureMaxLength= ="5"
```

   Figure 2 encapsulates the fracture simulation process within the TearingEngine API through a comprehensive flowchart, providing a global overview of the interconnected components and their sequential interactions.

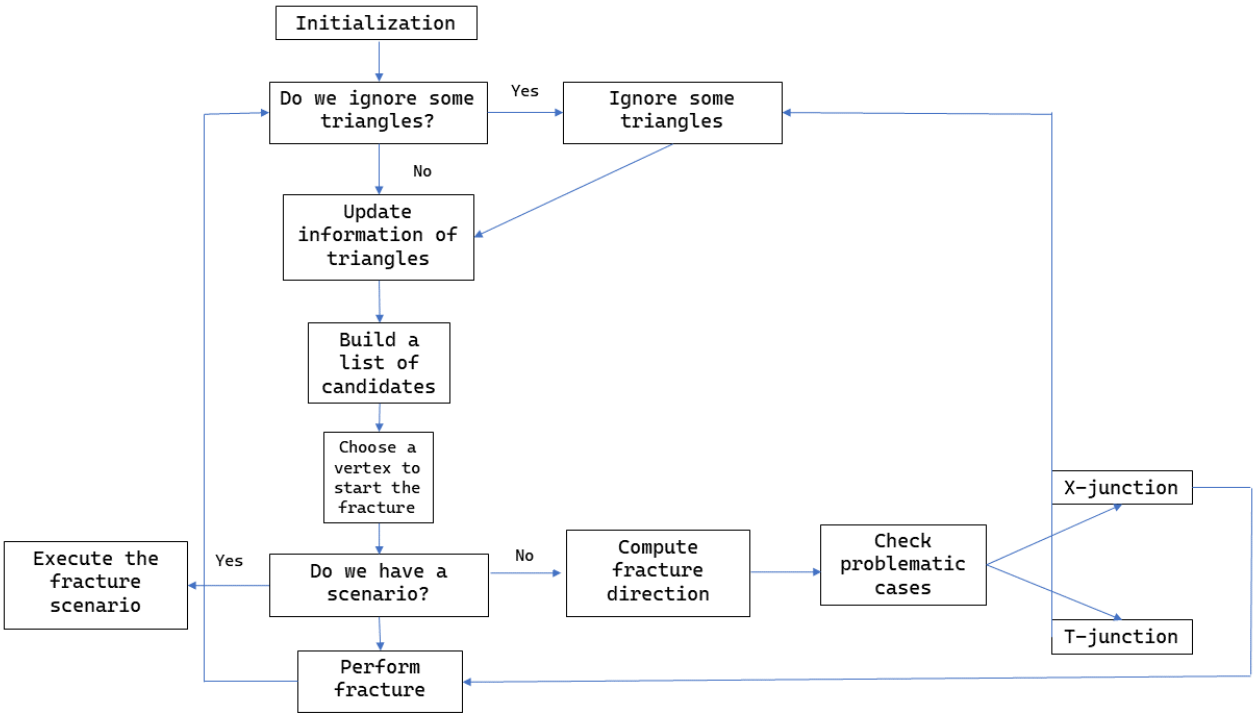


*Figure 2: An overview of implemented functions in TearingEngine plugin.*

## Technical look inside the TearingEngine plugin

This section provides a comprehensive overview of the seamlessly integrated functions designed to simulate tearing.

```
void TearingEngine<DataTypes>::triangleOverThresholdPrincipalStress();
```

**Task.**

- A list of triangles is generated based on the first dominant principal stress value of each triangle. This list stores the index of triangles with stress values exceeding the specified threshold.
- From the triangles in this list, the one with the highest principal stress value is selected as the potential candidate for fracture initiation.
- Among the vertices of the selected triangle, the starting point for the fracture is determined based on the chosen option for ComputeVertexStressMethod.

```
void TearingEngine<DataTypes>::updateTriangleInformation();
```

**Task.** For each triangle in the mesh, essential information—including *stress* ($stress = K * strain$), *maxStress* (the first dominant principal stress value), and *principalStressDirection* (the corresponding principal stress direction)—is initialized.

```
void TearingEngine<DataTypes>::doUpdate();
```

**Task.**

- To enhance simulation performance, certain triangles are excluded from the candidate selection process, and, consequently, they do not contribute to the simulation. Additionally, during the simulation, two specific cases, namely X-junction and T-junction, may arise based on the potential triangle and its fracture direction. In these instances, distinct actions are taken. This filtering process is enabled by default.
- The information for all eligible triangles is initialized by `updateTriangleInformation`.
- The potential triangle is selected by triangleOverThresholdPrincipalStress.

```
void TearingEngine<DataTypes>::computeEndPoints();
```

**Task.** Using the fracture direction and the user-defined maximum length, the coordinates of the two endpoints of the fracture segment are calculated.

```
void TearingAlgorithms<DataTypes>::algoFracturePath (…);
```

**Task.**

- Given the starting point, the function computes the intersection of the fracture segment with the mesh.
- If any intersection exists:

    - All intersection objects are converted to meaningful mesh elements by TearingAlgorithms<DataTypes>::pathAdaptationObject.
    - Utilizing the topological information from the previous step, a remeshing operation is executed using TearingAlgorithms<DataTypes>::splitting.
- If no intersection is found, two special cases are checked: X-junction and T-junction.
    - In the case of an X-junction, specific action is performed to proceed with the fracture.
    - For a T-junction, the index of this potential triangle is added to the list of triangles to be ignored.

```
void TearingEngine<DataTypes>::algoFracturePath();
```

**Task.**

- The fracture segment is calculated by computeEndPoints.
- The fracture is performed by TearingAlgorithms<DataTypes>::algoFracturePath.