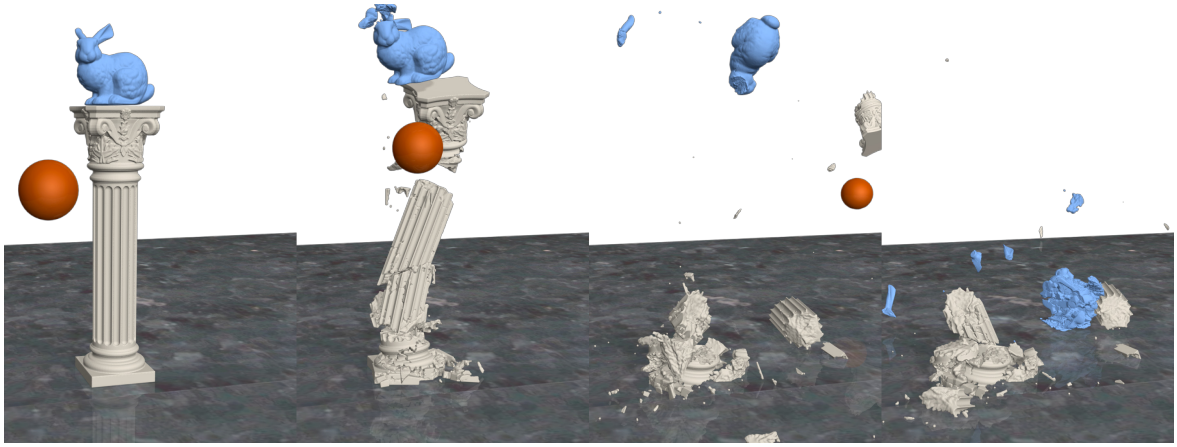
Brittle Fracture Simulation with Boundary Elements for Computer Graphics



by

David Hahn

June, 2017

A thesis presented to the

Graduate School

of the

Institute of Science and Technology Austria, Klosterneuburg, Austria

in partial fulfilment of the requirements

for the degree of

Doctor of Philosophy

IST AUSTRIA

Institute of Science and Technology

The thesis by

David Hahn

titled

Brittle Fracture Simulation with Boundary Elements for Computer Graphics

is approved by (all signatures omitted in online version):

Supervisor: Chris Wojtan, IST Austria

Committee member: Thomas Schrefl, Danube University Krems, Austria

Committee member: Herbert Edelsbrunner, IST Austria

Defense chair: Björn Hof, IST Austria

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

David Hahn

June, 2017

Abstract

This thesis describes a brittle fracture simulation method for visual effects applications. Building upon a symmetric Galerkin boundary element method, we first compute stress intensity factors following the theory of linear elastic fracture mechanics. We then use these stress intensities to simulate the motion of a propagating crack front at a significantly higher resolution than the overall deformation of the breaking object. Allowing for spatial variations of the material's toughness during crack propagation produces visually realistic, highly-detailed fracture surfaces.

Furthermore, we introduce approximations for stress intensities and crack opening displacements, resulting in both practical speed-up and theoretically superior runtime complexity compared to previous methods. While we choose a quasi-static approach to fracture mechanics, ignoring dynamic deformations, we also couple our fracture simulation framework to a standard rigid-body dynamics solver, enabling visual effects artists to simulate both large scale motion, as well as fracturing due to collision forces in a combined system.

As fractures inside of an object grow, their geometry must be represented both in the coarse boundary element mesh, as well as at the desired fine output resolution. Using a boundary element method, we avoid complicated volumetric meshing operations. Instead we describe a simple set of surface meshing operations that allow us to progressively add cracks to the mesh of an object and still re-use all previously computed entries of the linear boundary element system matrix. On the high resolution level, we opt for an implicit surface representation. We then describe how to capture fracture surfaces during crack propagation, as well as separate the individual fragments resulting from the fracture process, based on this implicit representation.

We show results obtained with our method, either solving the full boundary element system in every time step, or alternatively using our fast approximations. These results demonstrate that both of these methods perform well in basic test cases and produce realistic fracture surfaces. Furthermore we show that our fast approximations substantially out-perform the standard approach in more demanding scenarios. Finally, these two methods naturally combine, using the full solution while the problem size is manageably small and switching to the fast approximations later on. The resulting hybrid method gives the user a direct way to choose between speed and accuracy of the simulation.

Acknowledgements

First of all, let me thank my committee members, especially my supervisor, Chris Wojtan, for supporting me throughout my PhD. Obviously, none of this work would have been possible without you.

Furthermore, Thank You to all the people who have contributed to this work in various ways, in particular Martin Schanz and his group for providing and supporting the HyENA boundary element library, as well as Eder Miguel and Morten Bojsen-Hansen for (repeatedly) proof reading and providing valuable suggestions during the writing of this thesis.

I would also like to thank Bernd Bickel, and all the members – past and present – of his and Chris' research groups at IST Austria for always providing honest and insightful feedback throughout many joint group meetings, as well as Christopher Batty, Eitan Grinspun, and Fang Da for many insights into boundary element methods during our collaboration.

As only virtual objects have been harmed in the process of creating this work, I would like to acknowledge the Stanford scanning repository for providing the "Bunny" and "Armadillo" models, the AIM@SHAPE repository for "Pierre's hand, watertight", and S. Gainsbourg for the "Column" via Archive3D.net. Sorry for breaking these models in many different ways.

Finally, a big Thank You to my entire family for always supporting me.

This work has received funding from IST Austria and the European Research Council (ERC) under the European Union's Horizon 2020 programme (grant agreement no. 638176).

List of Publications appearing in this thesis

Hahn, D. & Wojtan, C.
High-resolution Brittle Fracture Simulation with Boundary Elements
ACM Transactions on Graphics, 2015, 34, 151:1-151:12

Hahn, D. & Wojtan, C.
Fast Approximations for Boundary Element Based Brittle Fracture Simulation
ACM Transactions on Graphics, 2016, 35, 104:1-104:11

Accompanying source codes and videos available online.

Contents

List of Tables

List of Figures

1 Introduction

Computer generated fractures have become a popular visual effect in movies and video games, from thin cracks in ice or glass to violent explosions of buildings or entire planets. Consequently, visual effect artists face an ever increasing demand for dramatic and convincingly realistic fracture effects.

Animating these fracture effects by hand can be challenging for a number of reasons, such as scale, surface detail, and complex fragment motion. Consequently, computer graphics researchers have developed diverse methods for fracturing *virtual objects*. The majority of these fracture methods focuses on *fragmentation*, i.e. breaking a (large) object into many smaller pieces (fragments). The main challenge for fragmentation stems from the various scales involved in the problem: when a big object shatters into a great number of pieces, the size of these fragments may range from tiny specks of dust to chunks that are almost as big as the original object. Purely geometric approaches use either pre-fractured models or pre-defined fracture patterns and focus on how to apply these fractures to produce the desired output geometry. On the other hand, *simulation methods* (such as mass-spring systems, finite and boundary element methods, or meshless continuum mechanics discretizations) approximate the underlying physics of the fracture process.

Apart from computing how an objects breaks into many pieces, the *fracture surfaces* themselves can be of interest for visual effects (even if the object does not yet break apart). In many materials, fracture surfaces exhibit very distinctive small-scale details. These *surface details* not only reveal information about how the object broke, but also create a very distinctive visual impression making fractures easily distinguishable from (for example) clean cuts. The visual appearance of fractures strongly depends on how the material deforms just before it breaks. We primarily distinguish *elastic* deformation, where the object returns to its original shape once all forces are removed, from *plastic* deformation, where the object remains permanently deformed in the absence of external forces. Most materials behave elastically as long as the deformation is small enough and plastically afterwards. The question then is whether the material fails before or after a plastic response happens: if it breaks while only deforming elastically, the fracture process is called *brittle*, otherwise the fracture is referred to as *ductile*. Also note that in some materials the rate at which they undergo plastic deformation may be quite limited, such that deforming an object slowly is more likely to produce plastic deformation and eventually ductile fracture, while a sudden deformation may lead to brittle fracture.

Interestingly, in materials that have a high *stiffness*, such as steel, glass, rock, or concrete, brittle fractures can grow very quickly: the crack propagation speed is roughly on the order of one kilometre per second; see also eq. (2.31). Consequently, in order to simulate this fast fracture process with sufficient accuracy to produce small-scale surface details we need a very *high temporal resolution* for such materials. Fully *elastodynamic* simulations are able to handle global motion, as well as local deformation

and fracture in a unified framework. However, these methods become inefficient for very stiff materials: as the speed of sound (i.e. the travelling speed of a pressure wave) within the material increases, the required temporal resolution restricts the maximal time step size. Consequently, many previous methods combine *elastostatic* continuum mechanical models with rigid-body methods to simulate elastic fracture mechanics in a *quasi-static* sense: the rigid-body system handles the global motion of objects, while the elastostatic model treats the deformation in a time-independent (average or maximal) sense, and dynamic wave propagation in the material is ignored.

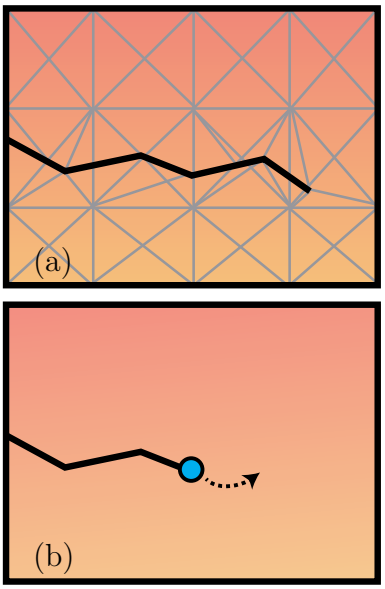


Figure 1: Eulerian (a) vs. Lagrangian (b) view of crack propagation.

One important feature of brittle fractures is that creating a (small) crack in an undamaged region typically requires a higher load than extending an already existing crack. Phenomenologically, this behaviour is described by two distinct material parameters: *strength* for the former case and *toughness* for the latter. Many computer graphics approaches neglect this distinction, resulting in excessive fracturing sometimes called *artificial shattering*. Furthermore, most materials are not homogeneous continua: material parameters (such as stiffness, strength, or toughness) may vary at relatively high spatial frequencies. For example small inclusions can locally reduce the toughness. Such variations influence the fracture process and give rise to visually interesting fracture surface details. Therefore, we also require a *high spatial resolution* in the fracture simulation; see figure 2 for some examples. Unfortunately, simulation-based methods become inefficient if all the detail of the fracture surfaces is present in the computational deformation model, effectively limiting the amount of visual detail that can be captured. While adaptive re-meshing (or re-sampling in mesh-less methods) is typically used to mitigate this limitation, the resolution of fractures is still bounded by the resolution of the deformation model. Some methods also use heuristics to add more visual detail to fracture surfaces as a post-process, but they do not influence the simulation in any way.

Our main goal is therefore to break free of this resolution constraint and simulate the crack propagation process at a significantly higher resolution than the deformation. Furthermore, we introduce acceptable approximations to the deformation model leading to a fast and efficient fracture simulation method. The standard approach in fracture simulation for computer graphics is to use a volumetric deformation method, most commonly the finite element method (FEM), and then cut or re-mesh one element at a time as the crack propagates through the material. This procedure requires small time steps and is analogous to an *Eulerian* reference frame as the crack advances through space. Our approach departs from this traditional viewpoint by adopting a *Lagrangian* reference frame for crack propagation, as illustrated in figure 1. This point

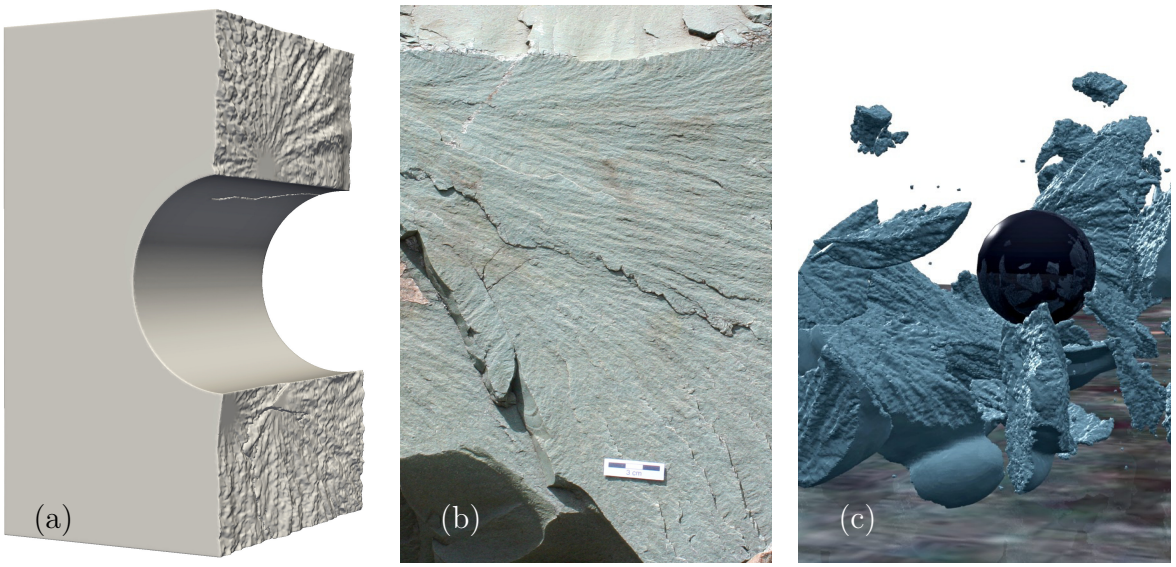


Figure 2: Comparison of our results (a, c) to a real-world photograph (b) courtesy of M. C. Rygel via Wikimedia Commons (cropped).

of view allows us to utilize techniques from front tracking, improves the achievable resolution of fracture surfaces, and still treats the underlying physics with acceptable accuracy. Instead of using FEM, we build our method upon the symmetric Galerkin boundary element method (SGBEM).

In the remainder of this chapter, we first present an overview of related work, introduce some terminology, especially relating to the geometry of cracked objects, and finally state the problem we aim to solve in this thesis more precisely. The following chapters will then introduce the required background in detail, before describing our high-resolution fracture simulation method, including our main contributions:

- An efficient symmetric Galerkin BEM implementation for quasi-static brittle fracture simulation, where each entry in the system matrix is computed only once (ch. 4.2).
- Fracture criteria treating crack initiation and propagation separately using the appropriate material parameters (strength and toughness) in the context of linear elastic fracture mechanics (ch. 4.4 and 4.5.1).
- An interpolation scheme that allows us to simulate crack propagation at a resolution much higher than that of the BEM mesh, including a new, fast, and simple treatment of spatially varying toughness, yielding realistic fracture patterns (ch. 4.5.2 and 4.5.3).
- Fast approximations of fracture mechanical quantities based on our boundary element method, resulting in a linear-time crack propagation algorithm (ch. 5).
- A simple and efficient way to handle the generated high-resolution surface geometry, detect fragments, and construct corresponding coarse triangle meshes (ch. 6).
- And a two-way coupling scheme between our fracture method and a rigid-body system, carefully treating the resulting Neumann boundary value problem (ch. 7).

Finally we present our results and conclusions in chapters 8 and 9 respectively.

1.1 Related work

Elastic deformation
There is a wide range of methods available for simulating elastic deformations, including mass-spring networks [53,67], finite element methods (FEM) [41,54,55,76,78] and extended FEM (X-FEM) [1,20,45,48], mesh-less or particle-based methods [59] including the material point method (MPM) [39,68,69], and boundary element methods (BEM) [2,16,34,38,44,65,71,77]. While mass-spring systems discretize deformable objects as a collection of point-masses with forces between them (usually confined to a small neighbourhood of each point), all other methods listed above are based on continuum mechanics. The main differences between FEM, X-FEM, MPM, and BEM can be characterized by the distribution of degrees of freedom and the resulting approximation spaces. Degrees of freedom in FEM and X-FEM are stored on (the vertices of) a volumetric mesh, spanning a piecewise polynomial function space (FEM), which can be enriched by additional, specialized basis functions (X-FEM). The MPM usually stores mass and velocity on particles, while forces are computed on a regular grid, and data is interpolated between the grid and the particles. The degrees of freedom in a BEM, however, are stored on a surface mesh and the boundary integral form of the governing equations is used instead of (volumetric) partial differential equations. While this limits the possibility to treat spatially varying elasticity parameters efficiently, it reduces the required number of degrees of freedom (compared to a volumetric discretization of the same resolution). Furthermore, only the boundary data is approximated, but the governing equations within the domain are satisfied exactly using a *fundamental solution* (also called *Green's function*). In contrast, a FEM approximates the solution in a (typically) piecewise polynomial space.

All of these continuum mechanics based methods are capable of producing equivalent results when applied to linear elasticity, which is the application we are interested in because brittle fractures are most challenging to simulate in very stiff materials (i.e. even very small deformations cause the material to break). In terms of further applications, (X-)FEM provides a flexible framework to model a wide range of materials, especially soft or non-linear ones, while MPM handles materials of granular nature very well, such as snow or sand [39,68], where the material behaves very differently under compression as opposed to tension. On the other hand, a BEM quickly becomes computationally expensive if the fundamental solution cannot be evaluated in closed form, making it less attractive for modelling non-linear or inhomogeneous materials. Nevertheless, using a BEM and a homogeneous linear elastic material allows us to simulate fractures in the presence of *inhomogeneous* strength and toughness efficiently, because these *fracture* parameters do not influence the *elastic* behaviour of the material directly. In fact, the resulting fracture surfaces modify the BEM mesh, introducing new boundary conditions and additional degrees of freedom for the material to deform, while the material's elastic behaviour remains unchanged. As boundary element methods operate on surface meshes rather than volumetric ones, the required meshing operations are much simpler and easier to implement.

(Brittle) fracture simulation
Purely geometric methods, such as [49, 70], are popular in computer graphics as they are typically much faster than any simulation. Fractures are either pre-defined during modelling, or generated by applying a pre-defined fracture pattern and then updating the topology of the object. Some methods also combine physics-based deformation models with geometric fracturing [30, 66]. Furthermore, additional visual detail can be generated for coarse simulation results by modifying the fracture surfaces on a smaller scale [7]. Such methods could in principle be applied to our results as a post-process to increase the perceived visual detail even further. For the remainder of this discussion we focus on continuum mechanical models of fracture. These physics-based fracture simulations typically consist of three steps: first deformations and internal forces are computed, then some fracture criterion is evaluated, and finally the computational model is updated to account for newly formed cracks. Evaluating a fracture criterion means deciding whether a material fails at a certain location under a given load. Terzopoulos and Fleischer [72] use a maximal strain threshold, while O'Brien and colleagues [54, 55] use a stress-based "separation tensor" at the nodes of a tetrahedral FEM mesh. This method has been combined with dynamic mesh improvement [76] and also modified by Pfaff et al. [60] to fracture triangulated thin shells.

In the context of linear elastic fracture mechanics, it is important to distinguish between the material parameters *strength* and *toughness*. Strength describes how much load is required to form a new fracture, whereas toughness determines whether an existing crack will (continue to) propagate [17, 63]. A similar, but simplified, concept can be found in [67]. For example, a strong but fragile specimen will withstand a large amount of load as long as it contains no cracks, but once it starts to fracture, cracks will quickly cut completely through it. On the other hand, in a weak but tough specimen small cracks will appear at relatively low loads, but only few of them will propagate (where the load is sufficiently high). Our examples in chapter 8.1 demonstrate the desired behaviour; see also fig. 36. Consequently, working only with maximal stress criteria for brittle fractures can lead to artificial shattering, as many nodes in a high-stress region may fulfil the criterion at the same time. Pfaff et al. [60] propose a local update procedure to mitigate this issue, while Koschier et al. [41] first find such regions and then pick only the point of maximal stress within each region to initiate a fracture.

We believe that the proper way to deal with brittle fractures is to treat crack initiation and propagation separately. Even nominally brittle materials exhibit a very small plastic zone around crack fronts, which is not captured by the linear elastic fracture mechanics model [22]. Consequently, stress is not the proper measure for crack propagation, as it is singular at the crack front. We use stress intensity factors (SIF) instead, which describe the magnitude of the singular stress field in the vicinity of crack fronts [17, 33].

Implicit and explicit surfaces
The most common way to approximate a (2D) surface (embedded in a 3D space) is a triangle mesh: each *element* references three *nodes* (vertices), whose positions are given by three Cartesian coordinates each. As both the connectivity, as well as the

location, of the surface are directly given, this form is referred to as an *explicit* surface representation. An *implicit* surface instead defines the surface as the (zero) level-set of a real-valued function in 3D space. Typically the value of this function is the (signed) distance from the surface. If the surface bounds a volume and is orientable (i.e. it is possible to distinguish whether any point in space is inside of, outside of, or exactly on the surface), the common convention is to use negative distance function values inside and positive values outside of the surface [56]. Computationally, one convenient way to approximate an implicit surface is storing distance function samples on a regular spatial grid and (piecewise polynomially) interpolating the function values between these samples. The OpenVDB library [11, 52] provides tools to store and process implicit surfaces on a sparse grid data structure, as well as convert between explicit and implicit surface representations. For further details, the reader is referred to chapter 6, as well as [52, 56].

While explicit surface representations are more flexible and allow for efficient storage and processing of the geometry, testing for important properties of the surface, such as manifoldness or orientability, requires additional computation. An implicit surface based on a signed distance function on the other hand is guaranteed to have these two properties [56]. Furthermore, implicit surfaces naturally handle topology changes, which is useful when treating growing fractures. Consequently, we use a (coarse) triangle mesh for our BEM computations, but rely on a signed distance function to describe the object's high-resolution geometry (including cracks) in our fracture simulation.

The X-FEM fracture simulation of Belytschko and colleagues [20,45] describes fractures with implicit surfaces, which are also used to build the enrichment functions providing new degrees of freedom to the displacement field. They use two level-set functions per fracture: the first one is a signed distance function of the actual fracture surface. The second function describes a "virtual" surface that intersects the first one exactly at the crack front in order to define the extent of the fracture. While this approach yields a very precise description of the fracture, we present a simplified approach (chapter 6) that allows us to quickly cut an object by multiple fractures and represent the resulting geometry in a single level-set function.

Rigid body dynamics

O'Brien and Hodgins [55] use a fully dynamic FEM to simulate brittle fractures, which means that both the overall motion as well as the deformation is treated in the FEM framework. However, as the material approaches rigidity, the speed of sound in the material increases, demanding very small time steps or risking artefacts from under-resolved pressure waves. Consequently, many methods, such as [19,51], use *quasi-static* formulations of fracture mechanics to avoid this issue, while objects are treated as rigid bodies when simulating their dynamic motion. Using a quasi-static approach, we iterate between first solving for an elastostatic equilibrium deformation of a given object (possibly including fractures), and then adding new fractures to the object according to the chosen fracture criteria. This way, only the fracture process is resolved in time, but elastodynamic effects, such as pressure waves, are ignored as we "jump" from one elastostatic solution to the next.

The challenging part of coupling rigid-body dynamics to a (quasi-static) elastic fracture simulation is the transition from the rigid to the deformable model. While elastic models consider *forces*, rigid models typically operate with *impulses* to resolve collisions. These impulses are independent of the (rigid body) time step size as collisions are assumed to be resolved instantaneously in a rigid model. See [4] for a detailed review of rigid-body dynamics. In a deformable model, collisions are resolved gradually within a finite time interval. Converting impulses to forces consequently requires scaling by the collision duration, which cannot be determined from the rigid-body step size. Both Glondu et al. [19] and Koschier et al. [41] use a Hertzian contact model to estimate the collision duration, which we also employ (see [61] for further details).

Another interesting aspect of the coupling between rigid bodies and deformable models is how to apply boundary conditions to the deformable model. The main idea is to let the rigid-body simulation handle translations and rotations, while treating deformations separately. Once collision impulses have been converted to forces, the problem that needs to be solved in the deformable model has only Neumann boundary conditions. In this case, the elastostatic solution is only defined up to arbitrary translation and rotation, leading to a rank deficient linear system after discretization. This issue can be avoided by either using a more expensive elastodynamic model [41], adding artificial Dirichlet boundary conditions [51], or carefully treating the null-space of the linear system [78]. The method described in [78] deals with FEM discretizations and consequently with sparse matrices and iterative solvers. In contrast, our method is based on a boundary element method, resulting in a dense linear system, which we solve by factorization. Zhu et al. [79] use an indirect boundary element method, which implicitly removes the null-space by solving for a potential function first and representing displacements as boundary integrals of this potential. Since we use a direct BEM, our treatment of the elastostatic Neumann problem's null-space follows the same principle as [78]. They, however, operate on the nodes of a tetrahedral mesh, while we work on the elements of a triangular surface mesh.

1.2 Terminology

As indicated in the introduction, one key idea of this work is to choose different resolution levels to simulate different phenomena, in particular we distinguish three main components: (a) the boundary element method, which handles deformation and quasi-static fracture at a coarse level, (b) our Lagrangian crack propagation method, which simulates the fracture process at a fine level, and finally (c) the rigid-body engine, which takes care of large, global motion of objects. We allow the user to choose a different spatial resolution in each of these components to approximate the object's surface. Typically, the BEM mesh is the coarsest representation (specified either directly by the target number of elements, or similarly by the *average edge length* in the mesh), while the fracture simulation uses an implicit surface representation at the finest resolution (defined by the grid spacing, or *voxel size*). The rigid-body engine uses either a convex hull or a triangle mesh of intermediate resolution.

In terms of temporal resolution, the situation is slightly different: here the rigid-body engine is expected to take the largest time steps; we also refer to these as "*rigid-body steps*". The BEM mesh will be updated at intermediate intervals during the fracture simulation, and we refer to these update intervals simply as *time steps*. Finally, the crack propagation, of course, uses the finest temporal resolution, and we refer to each small propagation (time) step as a "*sub step*".

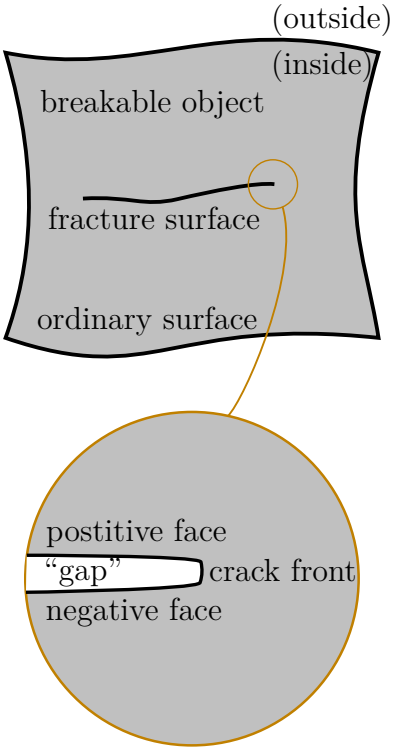


Figure 3: Overview of surface terminology.

When describing the geometry of a breaking object, we distinguish the object's surface in its unfractured state (*ordinary surface* or *outside surface*) from (internal) *fracture surfaces* (also *fractures* or equivalently *cracks*). Each crack, in this sense, consists of two *crack faces*, which are geometrically coincident (if the object is not deformed), but have oppositely oriented surface normals. The *crack front* is the common bounding curve of these faces. (The 2D equivalent to the crack front is usually called *crack tip*.) When referring to a fracture surface, we regularly think of just one surface with arbitrary (but consistent) orientation of normal vectors, rather than distinguish both crack faces. On occasions when this distinction is necessary, we refer to one face as the "*positive*" and the other as the "*negative*" side of the crack.

Processing the high-resolution geometry of fractured objects consists of two major components. First we need to store the fracture surfaces produced by the fracture simulation (*surface recording*): as the crack front propagates it cuts through an object like a wire cutting through cheese. Afterwards we need to detect topology changes, i.e. find out where an object breaks into separate pieces (*fragmentation*). We choose an implicit surface representation on the finest spatial resolution, allowing for an easier implementation of the fragmentation step. In order to do so reliably, however, we need to slightly separate the two crack faces, creating a small "*gap*" in between them, such that they bound a thin "*fracture volume*", within which we consider the material to be broken (see figure 3). These gaps can of course be removed from the final output once we have identified all the fragments. Note that geometrically coincident surfaces may produce undesirable visual artefacts when rendering the final output images, in which case it might be preferable to keep (some of) these gaps. Finally, due to numerical inaccuracies during the fracture simulation, two regions of the breaking object that are almost separate fragments sometimes remain connected by very thin pieces of material we call "*spindles*". Our fragmentation method allows to remove such spindles if they are small enough, as shown in fig. 28.

1.3 Problem statement

The goal of the this thesis is to develop a novel way of simulating *brittle fracture* for visual effects applications. Contrary to ductile ones, brittle fractures happen very quickly making it difficult to resolve the fracture process temporally; they also produce finely detailed surface patterns and consequently require a high spatial resolution as well. While geometry-based approaches for fracture animation have become very fast recently, the apparent realism of such pre-defined fractures or fracture patterns depends entirely on the artist. Therefore we believe it is preferable to use physics-based simulation techniques to create visually attractive and convincingly realistic fracture effects.

Because brittle fracture occurs most commonly in very stiff materials, we assume *infinitesimally small* deformations, which allows us to employ *linearized* versions of the governing equations of elasticity and fracture mechanics. Furthermore, we do not aim to handle rigid motion of objects in the same framework as our fracture simulation; instead such rigid motions are treated separately by coupling our fracture method to a specialized, third-party rigid-body-dynamics engine. In order to build a stable and efficient fracture algorithm, we choose a quasi-static approach, ignoring dynamic effects, such as stress waves, which would impose stringent time step limits on the elastic deformation simulation. We also believe that the visually apparent influence of these effects is hardly perceptible (in terms of a computer graphics application), especially when spatial toughness variations are present in the material.

As our target audience is the computer graphics community, we aim to provide the user with a choice to trade-off accuracy of the simulation for shorter runtime. With respect to fracture effects, high accuracy is not critical because fractures behave (somewhat) chaotically, making it difficult to distinguish (approximate) solutions of varying accuracy, as long as qualitative features are equivalent. In this sense, it is more important that the algorithm remains stable, even under large time steps, and qualitative visual features of the generated fracture surfaces appear realistic. In order to capture these visual features we do, however, need high (temporal and spatial) resolution during the crack propagation simulation. At the same time, our method must be fast enough to be useful to visual effects artists. To this end, we present approximations of fracture mechanical quantities, improving the asymptotic scaling of our algorithm's runtime.

In particular, all previous fracture simulation methods suffer from a fundamental limitation: the resolution of the generated fracture surfaces must be the same as the resolution of the underlying deformation model. Our primary goal is to break free of this constraint and instead allow for crack propagation (which generates the fracture surfaces) to be simulated at a much higher resolution than the elastostatic deformation.

2 Background

Having laid out the problem we wish to solve, as well as given an overview of related work above, this chapter discusses the required background more formally. We start by introducing some basic notation and theorems of vector calculus, ch. 2.1. We then give a short overview of continuum mechanics and derive the governing equations of linear elasticity in differential form, ch. 2.2, as well as their finite element discretization, ch. 2.3.1. Furthermore, we present the boundary integral form of linear elasticity and discretize it using the symmetric Galerkin boundary element method (SGBEM), ch. 2.3.2. Finally, we introduce the key ideas of linear elastic fracture mechanics (LEFM), ch. 2.4, and show how fractures are treated as new boundary conditions and additional degrees of freedom in the BEM formulation, ch. 2.5.

2.1 Notation and vector calculus

Green's first identity serves as a starting point for the weak form used in FEM, as well as boundary representation formulae used in BEM. In this chapter, we briefly derive this important theorem and introduce some common notation in the process.

For now, assume $\mathbf{A}$ and $\mathbf{v}$ are arbitrary, sufficiently smooth tensor and vector fields in 3D space respectively, i.e. $\mathbf{A}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}^{3\times3}$ and $\mathbf{v}(\mathbf{x}) : \mathbb{R}^3 \to \mathbb{R}^3$. The components of the *gradient* of a vector field and similarly the components of the *divergence* of a tensor field are defined as

$$(\nabla\mathbf{v})_{ij} := \partial\mathbf{v}_i/\partial\mathbf{x}_j \quad \text{and} \quad (\nabla\cdot\mathbf{A})_i := \textstyle\sum_j \partial\mathbf{A}_{ij}/\partial\mathbf{x}_j \quad \text{respectively.} \tag{2.1}$$

Using the identity tensor $\mathbf{I}$ we can write the divergence of a vector field as $\nabla\cdot\mathbf{v} = \operatorname{tr}(\nabla\mathbf{v}) = \mathbf{I} : (\nabla\mathbf{v})$, where the *scalar product* of tensors $\mathbf{A}$ and $\mathbf{B}$ is defined as

$$(\mathbf{A} : \mathbf{B}) := \textstyle\sum_i \sum_j \mathbf{A}_{ij}\mathbf{B}_{ij}, \tag{2.2}$$

i.e. the sum of the products of corresponding coefficients. Finally, we use

$$\operatorname{sym}(\mathbf{A}) := (\mathbf{A} + \mathbf{A}^\mathsf{T})/2 \quad \text{and} \quad \operatorname{asym}(\mathbf{A}) := (\mathbf{A} - \mathbf{A}^\mathsf{T})/2 \tag{2.3}$$

to denote the symmetric and anti-symmetric parts of $\mathbf{A}$, such that $\mathbf{A} = \operatorname{sym}(\mathbf{A}) + \operatorname{asym}(\mathbf{A})$. As a direct consequence we see that $\operatorname{sym}(\mathbf{A}) : \operatorname{asym}(\mathbf{B}) = 0$.

Now consider a closed connected volume (i.e. a connected 3-manifold with boundary) $\Omega \subset \mathbb{R}^3$ (representing our deformable object) and its boundary surface $\Gamma := \partial\Omega$. The divergence theorem and the divergence product rule (eq. (1.293) and (1.289) in [27] respectively) are

$$\int_\Omega \nabla\cdot\mathbf{v}(\mathbf{x})\; dv_\mathbf{x} = \int_\Gamma \mathbf{v}(\tilde{\mathbf{x}})\cdot\mathbf{n}(\tilde{\mathbf{x}})\; ds_{\tilde{\mathbf{x}}} \text{ and} \tag{2.4}$$

$$\nabla \cdot \left(\mathbf{A}^\mathsf{T}\mathbf{v}\right) = (\nabla \cdot \mathbf{A}) \cdot \mathbf{v} + \mathbf{A} : (\nabla \mathbf{v}), \tag{2.5}$$

where $\mathbf{n}(\tilde{\mathbf{x}})$ is the outward unit surface normal at $\tilde{\mathbf{x}} \in \Gamma$. Integrating (2.5) over Ω, applying the divergence theorem, and rearranging terms yields *Green's first identity*:

$$\int_\Omega (\nabla \cdot \mathbf{A}) \cdot \mathbf{v} \; dv_\mathbf{x} = \int_\Gamma (\mathbf{An}) \cdot \mathbf{v} \; ds_{\tilde{\mathbf{x}}} - \int_\Omega \mathbf{A} : (\nabla \mathbf{v}) \; dv_\mathbf{x}, \tag{2.6}$$

where we use $(\mathbf{A}^\mathsf{T}\mathbf{v}) \cdot \mathbf{n} = (\mathbf{A}^\mathsf{T}\mathbf{v})^\mathsf{T}\mathbf{n} = \mathbf{v}^\mathsf{T}\mathbf{An} = (\mathbf{An}) \cdot \mathbf{v}$ to rewrite the surface integral on the right-hand side.

2.2 Continuum mechanics

Now that we have established some basic notation, we can turn our attention to describing the properties of solid objects in the context of continuum mechanics. The basic assumptions of continuum mechanics are that (a) the material is infinitely divisible, and (b) *locally* homogeneous, i.e. the material can be subdivided into ever smaller regions, and at some point all subdivisions (at least in some local region) have exactly the same properties [6]. These assumptions make sense whenever the object in question is much larger than atomistic length scales, also see chapters 38 and 39 in [15].

As motivated in the introduction, we focus on materials that behave elastically up to the point of failure, i.e. as long as an object is not broken, it will recover its original shape once all forces causing deformation are removed. A *constitutive model*, which might not be linear in general, describes the elastic response of a material by relating deformations (strains) to internal forces (stresses). Because the physical behaviour of the material is independent of rigid motion (i.e. global translation or rotation), constitutive models must reflect these properties as well. For further details, the reader is referred to chapter 5.4 in the textbook by Holzapfel [27].

We now use $\Omega \subset \mathbb{R}^3$ to denote the undeformed *reference configuration* of an object (our computational domain) and again write $\Gamma := \partial\Omega$ for its surface. From now on we always assume that Ω has some positive volume and only one connected component. Later on, in chapters 6.3 and 7.3 we show how to find disconnected fragments created by our fracture simulation and treat each fragment as a new individual object. In the following derivations we work with the reference configuration (i.e. in *material space* or *rest space*), because it will be more convenient in terms of formulating a linear elastostatic BEM later. Consequently, we define a deformed configuration (at time t) by mapping any point $\mathbf{x} \in \Omega$ to its corresponding position $\mathbf{X}(\mathbf{x}, t) \in \Omega_t$ in the deformed object. If the deformation changes over time, we denote velocity as $\dot{\mathbf{X}}(\mathbf{x}, t) := d\mathbf{X}(\mathbf{x}, t)/dt$ and acceleration as $\ddot{\mathbf{X}}(\mathbf{x}, t) := d^2\mathbf{X}(\mathbf{x}, t)/dt^2$. Finally, we define the *displacement* $\mathbf{u}$ as the difference between the reference and the deformed configuration: $\mathbf{u}(\mathbf{x}, t) := \mathbf{X}(\mathbf{x}, t) - \mathbf{x}$.

When a distributed external force acts on some part of the surface of a deformable object, the (nominal) *traction vector* $\mathbf{q}(\tilde{\mathbf{x}})$ denotes the force measured per unit surface

area at the point $\tilde{\mathbf{x}} \in \Gamma$. *Cauchy's stress theorem* (eq. (3.3) in [27]) states that there exists a tensor field $\boldsymbol{\sigma}$ such that

$$\boldsymbol{\sigma}\mathbf{n} = \mathbf{q} \tag{2.7}$$

holds on the object's surface (where $\mathbf{n}$ is the outward unit surface normal). Similarly, forces acting inside of the object (internal forces) can be characterized by arbitrarily choosing three orthogonal, infinitesimally small surface patches located at some point inside of the object and measuring traction with respect to each of these surfaces. If the (arbitrary) orientation of these surfaces is described by three orthogonal unit vectors $(\mathbf{n}_1,\ \mathbf{n}_2,\ \mathbf{n}_3)$ and the measured traction vectors are $(\mathbf{q}_1,\ \mathbf{q}_2,\ \mathbf{q}_3)$, then the stress tensor can be defined as $\boldsymbol{\sigma} := \sum_i \mathbf{q}_i \mathbf{n}_i^\mathsf{T}$, where $i = 1..3$. By construction, this stress tensor satisfies Cauchy's stress theorem also in the interior of the object. Integrating eq. (2.7) over the surface Γ and applying the divergence theorem (2.4) yields

$$\int_\Gamma \mathbf{q}\ ds = \int_\Gamma \boldsymbol{\sigma}\mathbf{n}\ ds = \int_\Omega \nabla \cdot \boldsymbol{\sigma}\ dv. \tag{2.8}$$

Equation (2.8) relates total force (traction integrated over the surface) to a volume integral of internal force density (on the right-hand side). This force accelerates the object and consequently causes a change of linear momentum. Similarly, we may consider distributed body forces, such as gravity, collectively denoted by $\mathbf{g}$. *Cauchy's first equation of motion* expresses conservation of linear momentum in the presence of these force terms (in global form): $\int_\Omega (\nabla \cdot \boldsymbol{\sigma} + \mathbf{g} - \rho\ddot{\mathbf{X}})dv = 0$, where ρ is the mass density. This equation must equally hold for all sub-volumes of Ω, which means that we can it write in the more convenient local form (see also eq. (4.63) in [27]):

$$\rho(\mathbf{x})\ddot{\mathbf{X}}(\mathbf{x},t) = \nabla \cdot \boldsymbol{\sigma}(\mathbf{x},t) + \mathbf{g}(\mathbf{x},t). \tag{2.9}$$

In order to satisfy conservation of angular momentum, $\boldsymbol{\sigma}$ must also be symmetric, i.e. $\boldsymbol{\sigma} = \boldsymbol{\sigma}^\mathsf{T}$; this is *Cauchy's second equation of motion* (see also eq. (4.68) in [27]). Note that we work in material space, so technically $\boldsymbol{\sigma}$ denotes the second Piola-Kirchhoff stress, which is identical to the Cauchy stress under infinitesimal deformations.

Equation (2.7) relates the stress acting on a surface of a particular orientation to the traction this surface experiences. Sometimes it is useful to find the orientation such that the resulting traction is maximal (or minimal) along the surface normal (for example when determining the most likely orientation of a fracture surface). These directions can be found by solving the eigenvalue problem $\boldsymbol{\sigma}\mathbf{n}_i = \lambda_i \mathbf{n}_i$, see eq. (3.36) in [27]. As a consequence of the stress tensor's symmetry, there are three (real) eigenvalues λ_i associated with orthogonal eigenvectors $\mathbf{n}_i$. These eigenvalues are called *principal (normal) stresses*, and their corresponding eigenvectors are called *principal stress directions*, while the planes orthogonal to the principal stress directions are referred to as *principal planes*. As the eigenvectors form an orthogonal basis, the stress tensor can be written as $\boldsymbol{\sigma} = \sum_i \lambda_i (\mathbf{n}_i \mathbf{n}_i^\mathsf{T})$, eq. (3.42) in [27]. Note that this *spectral representation* also implies that shear stresses vanish along the principal planes.

Now that we have established some general properties of elastic deformation, we turn our attention to the case of infinitesimally small displacements, which admits a linearization of the constitutive model. In particular, if the material is stiff enough relative to its strength, it will fail and exhibit brittle fracture before undergoing large deformations. In order to derive the equations of linear elasticity, we start with a truncated Taylor expansion of the displacement field $\mathbf{u}$:

$$\begin{aligned} \mathbf{u}(\mathbf{x}+\boldsymbol{\Delta}\mathbf{x}) &\approx \mathbf{u}(\mathbf{x}) + (\nabla\mathbf{u})(\mathbf{x})\ \boldsymbol{\Delta}\mathbf{x}, \\ \mathbf{u} &\approx \mathbf{u}_0 + \operatorname{asym}(\nabla\mathbf{u})\ \boldsymbol{\Delta}\mathbf{x} + \operatorname{sym}(\nabla\mathbf{u})\ \boldsymbol{\Delta}\mathbf{x}, \\ \mathbf{u} &\approx \mathbf{u}_0 + \boldsymbol{\omega}\ \boldsymbol{\Delta}\mathbf{x} + \boldsymbol{\varepsilon}\ \boldsymbol{\Delta}\mathbf{x}, \end{aligned} \tag{2.10}$$

where $\boldsymbol{\omega} := \operatorname{asym}(\nabla\mathbf{u})$ is an *infinitesimal rotation* and $\boldsymbol{\varepsilon} := \operatorname{sym}(\nabla\mathbf{u})$ is the *linearized strain* tensor. We then ignore the rotational part and apply a general *linear* constitutive model relating strain to stress: $\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\varepsilon}$, where $\mathbf{C}$ is a fourth-order elasticity tensor. One important distinction is the difference between infinitesimal rotations, such as $\boldsymbol{\omega}$ above, which satisfy $\boldsymbol{\omega} - \boldsymbol{\omega}^\mathsf{T} = \mathbf{0}$, and true rotations $\mathbf{R}$ which satisfy $\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}$. Consequently, the linearized constitutive model is only invariant to infinitesimal rotations and no longer applicable to large displacements (in particular large rotations).

We further assume that the material is *isotropic*, i.e. the strain-stress relation has no directional dependence. It then follows by symmetry that $\mathbf{C}$ has only 2 independent components and we can write the constitutive model using the *Lamé parameters* λ and μ (see also eq. (39.20) in [15]) as:

$$\boldsymbol{\sigma}(\mathbf{u}) = 2\mu\boldsymbol{\varepsilon}(\mathbf{u}) + \lambda\operatorname{tr}(\boldsymbol{\varepsilon}(\mathbf{u}))\mathbf{I} = 2\mu\operatorname{sym}(\nabla\mathbf{u}) + \lambda(\nabla\cdot\mathbf{u})\mathbf{I}. \tag{2.11}$$

In a *globally homogeneous* material λ and μ are constant. Similarly, we can write the same constitutive model using Young's modulus E and Poisson's ratio ν instead, which are related to the Lamé parameters by $E = (\mu(3\lambda+2\mu)) \,/\, (\lambda+\mu)$ and $\nu = \lambda/\,(2(\lambda+\mu))$.

If we now ignore time-dependent dynamic motion, i.e. $\ddot{\mathbf{X}} = \dot{\mathbf{X}} = 0$, we finally arrive at the governing equations for linear elastostatics:

$$\begin{aligned} -\nabla\cdot\boldsymbol{\sigma} &= \mathbf{g} && \text{in } \Omega, \\ \boldsymbol{\sigma}\mathbf{n} &= \mathbf{q}_\Gamma && \text{on } \Gamma_N, \text{ and} \\ \mathbf{u} &= \mathbf{u}_\Gamma && \text{on } \Gamma_D, \end{aligned} \tag{2.12}$$

where we apply Neumann boundary conditions, i.e. prescribed surface tractions $\mathbf{q}_\Gamma(\tilde{\mathbf{x}})$, on some part of the surface Γ_N, as well as Dirichlet boundary conditions, i.e. prescribed displacements $\mathbf{u}_\Gamma(\tilde{\mathbf{x}})$, on the other part of the surface Γ_D. In order to get a well-defined solution, every point $\tilde{\mathbf{x}}$ on the surface must be either in Γ_N or in Γ_D. In chapter 7.1 we also treat the special case of pure Neumann boundary conditions, i.e. $\Gamma_N = \Gamma$ and $\Gamma_D = \emptyset$.

2.3 Finite and boundary element methods

Simulating elastostatic deformations requires numerical methods capable of finding approximate solutions to eq. (2.12). We first briefly discuss the basics of the finite element method (FEM), because it is probably the most widely used method in related work, and also because our surface stress evaluation in chapter 4.3 follows the same concept, albeit only in a two-dimensional space. At the same time, we also introduce the basic concept of interpolating fields over a (simplicial) mesh using nodal basis functions and per-node degrees of freedom.

In the second part of this chapter, we then derive the boundary element formulation that serves as the starting point for our fracture simulations. While we aim to give a basic introduction to BEM here, many technical details in terms of actually implementing a numerical scheme to evaluate the required surface integrals will be omitted for the sake of brevity. Existing BEM libraries readily provide reliable implementations of this functionality. We build our implementation upon the HyENA library [29]; for further details, the interested reader is referred to the book by Sauter and Schwab [65], as well as the thesis by Kielhorn [38].

2.3.1 Weak form and FEM

The following derivation of the finite element formulation of linear elastostatics summarizes chapter 11 of [42]. We start by writing the governing differential equation (2.12) in a weak (or variational) form. To this end, we dot-multiply (2.12) by some arbitrary (square-integrable) test function $\mathbf{v}$ and integrate over the entire domain Ω, which yields

$$-\int_\Omega (\nabla \cdot \boldsymbol{\sigma})(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x})\, dv_{\mathbf{x}} = \int_\Omega \mathbf{g}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x})\, dv_{\mathbf{x}}. \tag{2.13}$$

We omit function arguments for the sake of simplified notation, apply Green's first identity (2.6) to the integral on the left-hand side, and move the resulting surface integral to the right-hand side to get

$$\int_\Omega \boldsymbol{\sigma} : (\nabla \mathbf{v})\, dv_{\mathbf{x}} = \int_\Omega \mathbf{g} \cdot \mathbf{v}\, dv_{\mathbf{x}} + \int_\Gamma \boldsymbol{\sigma}\mathbf{n} \cdot \mathbf{v}\, ds_{\tilde{\mathbf{x}}}. \tag{2.14}$$

Finally, we expand $\boldsymbol{\sigma}$ using the constitutive model (2.11), substitute the known boundary tractions, and rearrange terms using $\mathbf{I} : (\nabla \mathbf{v}) = \nabla \cdot \mathbf{v}$ and $\operatorname{sym}(\mathbf{u}) : \operatorname{asym}(\mathbf{v}) = 0$ to arrive at

$$\int_\Omega \left[2\mu \operatorname{sym}(\nabla \mathbf{u}) : \operatorname{sym}(\nabla \mathbf{v}) + \lambda(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v})\right] dv_{\mathbf{x}} = \int_\Omega \mathbf{g} \cdot \mathbf{v}\, dv_{\mathbf{x}} + \int_\Gamma \mathbf{q}_\Gamma \cdot \mathbf{v}\, ds_{\tilde{\mathbf{x}}}. \tag{2.15}$$

The problem now reads: find $\mathbf{u}$ (whose derivatives are square-integrable) such that the weak form (2.15) is satisfied for all $\mathbf{v}$ (with the additional constraint that $\mathbf{v} = 0$

on Γ_D in order to evaluate the surface integral on the right-hand side). The following discussion focuses on the most common approach to solve this problem numerically: using a tetrahedral mesh to approximate the computational domain and a piecewise-linear displacement interpolation. In this setting, the unknown displacement field $\mathbf{u}$ is defined via a finite set of shape functions ψ_k and corresponding unknown coefficients $\mathbf{u}_k$, such that $\mathbf{u}(\mathbf{x}) \approx \sum_k \mathbf{u}_k \psi_k(\mathbf{x})$. Each shape function corresponds to a node of the mesh and satisfies the Kronecker property $\psi_k(\mathbf{x}_l) = \delta_{kl}$ (i.e. ψ_k is one at node k, zero at all other nodes, and piecewise linear in between). As a direct consequence, the shape functions form a *partition of unity* over the mesh: $\sum_k \psi_k(\mathbf{x}) = 1$.

When using a Galerkin finite element formulation, we choose each shape function ψ_l that does not correspond to a node on Γ_D as a test function (for each coordinate axis in turn): $\mathbf{v} = \mathbf{e}_j \psi_l$, where $\mathbf{e}_j$ is the standard basis unit vector along the j-th axis. Substitution of the displacement approximation and test functions into eq. (2.15) results in the discretized form of linear elastostatics: $\mathbf{K}\hat{\mathbf{u}} = \mathbf{f}$, see also eq. (11.70) in [42]. Here the discretized displacement $\hat{\mathbf{u}}$ is a vector collecting all nodal displacements and $\mathbf{K}$ is the *stiffness matrix* built from the integral on the left-hand side of eq. (2.15). From this integral we can easily see that the stiffness matrix must be symmetric. Furthermore, only shape and test function pairs corresponding to the same or neighbouring nodes result in a non-zero contribution to the stiffness matrix entry of these nodes, which means that $\mathbf{K}$ is sparse. Finally, $\mathbf{f}$ is the load vector built from the integrals on the right-hand side of eq. (2.15), collecting both body forces and surface tractions due to boundary conditions. Inhomogeneous Dirichlet boundary conditions contribute an extra term to the right-hand side by splitting the displacement vector into two parts: $\hat{\mathbf{u}} = \hat{\mathbf{u}}_0 + \hat{\mathbf{u}}_D$, where $\hat{\mathbf{u}}_0$ is a solution with homogeneous Dirichlet boundary conditions and $\hat{\mathbf{u}}_D$ contains the actual boundary data and is zero on all non-Dirichlet nodes.

Of course, we must pay attention to correctly enumerating all degrees of freedom and their corresponding shape and test functions while building the discrete linear FEM system. This careful bookkeeping of indices results in the following notation often found in engineering and computer graphics applications: we first look at each (tetrahedral) element individually, where the displacement field inside of the element is a linear interpolation of the nodal values. We then get a constant matrix of basis function derivatives $\mathbf{B}_e$ for each element e, such that the linear strain (in vector form) becomes $\boldsymbol{\varepsilon}_e = \mathbf{B}_e \hat{\mathbf{u}}_e$. Here $\hat{\mathbf{u}}_e$ collects the displacement vectors of all four nodes of the tetrahedron into a vector with 12 entries, see also [50]. Similarly, we can compute the (per-element) deformation gradient, following [32], as $\mathbf{F}_e = \mathbf{D}_{se}\mathbf{D}_{me}^{-1}$, where $\mathbf{D}_{me}$ is a matrix whose columns are the edge vectors $\mathbf{x}_{e,i} - \mathbf{x}_{e,4},\ i = 1..3$ of tetrahedron e in the material space (reference) configuration ($\mathbf{x}_{e,i}$ being the coordinates of the tetrahedron's nodes), and similarly $\mathbf{D}_{se}$ contains the edge vectors in the deformed configuration. We then find the linearized strain (in tensor form) $\boldsymbol{\varepsilon}_e = (\mathbf{F}_e + \mathbf{F}_e^\mathsf{T})/2 - \mathbf{I}$. Applying the constitutive model (2.11) to this strain and integrating over the element results in a constant stress matrix per element. As all quantities are constant (assuming homogeneous material) within the element, the integration reduces to simply multiplying by the tetrahedron's volume V_e.

When working with a linear constitutive model (2.11), it can be written as $\mathbf{S}$ in the matrix form given in eq. (11.56) of [42], such that $\boldsymbol{\sigma} = \mathbf{S}\boldsymbol{\varepsilon}_e = \mathbf{S}\mathbf{B}_e\hat{\mathbf{u}}_e$. Consequently we find the *element stiffness matrix* $\mathbf{K}_e = V_e\mathbf{B}_e^\mathsf{T}\mathbf{S}\mathbf{B}_e$. We can then assemble the global stiffness matrix $\mathbf{K}$ from all the per-element stiffness blocks $\mathbf{K}_e$ to obtain the same discrete linear elastostatic system $\mathbf{K}\hat{\mathbf{u}} = \mathbf{f}$.

2.3.2 Boundary integral equations and BEM

While the finite element formulation, introduced above, effectively describes a balance of internal forces between every pair of neighbouring nodes in a volumetric mesh, the basic idea of a boundary element method is to reduce the problem in such a way that there are no degrees of freedom remaining in the interior of the computational domain. Consequently, the (approximate) solution must be described solely in terms of unknown fields on the boundary (surface) of the domain; such a description is commonly referred to as a *boundary representation formula*. When moving the evaluation point of the representation formula from the interior of the domain to the boundary as well, and equating the result to the known boundary data, we obtain a *boundary integral equation*. Finally, this integral equation is then discretized using boundary elements, leading once again to a linear system of equations, albeit with slightly different properties than the FEM version.

To derive a boundary integral formulation of eq. (2.12), we first consider the divergence product rule, eq. (2.5), and substitute $\mathbf{A} = \mathbf{A}^\mathsf{T} = \boldsymbol{\sigma}(\mathbf{u})$. Here we use $\boldsymbol{\sigma}(\mathbf{u})$ to denote the stress caused by some (as of yet unknown) displacement $\mathbf{u}$ according to the constitutive model, eq. (2.11). The product rule now reads $\nabla \cdot (\boldsymbol{\sigma}(\mathbf{u})\mathbf{v}) = (\nabla \cdot \boldsymbol{\sigma}(\mathbf{u})) \cdot \mathbf{v} + \boldsymbol{\sigma}(\mathbf{u}) : (\nabla\mathbf{v})$. Interchanging $\mathbf{u}$ and $\mathbf{v}$ similarly results in $\nabla \cdot (\boldsymbol{\sigma}(\mathbf{v})\mathbf{u}) = (\nabla \cdot \boldsymbol{\sigma}(\mathbf{v})) \cdot \mathbf{u} + \boldsymbol{\sigma}(\mathbf{v}) : (\nabla\mathbf{u})$. Taking the difference of these two equations and integrating over the domain Ω results in

$$\int_\Omega \left[\nabla \cdot (\boldsymbol{\sigma}(\mathbf{u})\mathbf{v}) - \nabla \cdot (\boldsymbol{\sigma}(\mathbf{v})\mathbf{u})\right] \, dv_\mathbf{x} = \int_\Omega \left[(\nabla \cdot \boldsymbol{\sigma}(\mathbf{u})) \cdot \mathbf{v} - (\nabla \cdot \boldsymbol{\sigma}(\mathbf{v})) \cdot \mathbf{u}\right] \, dv_\mathbf{x}. \tag{2.16}$$

Note that the difference of the terms containing the gradient of $\mathbf{u}$ and $\mathbf{v}$ respectively vanishes once we expand the stress tensors using the constitutive model. When writing this expansion in the same form as it appears in eq. (2.15), this difference simplifies as follows:

$$\begin{aligned}
&\boldsymbol{\sigma}(\mathbf{u}) : (\nabla\mathbf{v}) - \boldsymbol{\sigma}(\mathbf{v}) : (\nabla\mathbf{u}) \\
&\quad = 2\mu\,\mathrm{sym}(\nabla\mathbf{u}) : \mathrm{sym}(\nabla\mathbf{v}) + \lambda(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v}) \\
&\quad - [2\mu\,\mathrm{sym}(\nabla\mathbf{v}) : \mathrm{sym}(\nabla\mathbf{u}) + \lambda(\nabla \cdot \mathbf{v})(\nabla \cdot \mathbf{u})] \\
&\quad = 2\mu\,\mathrm{sym}(\nabla\mathbf{u}) : \mathrm{sym}(\nabla\mathbf{v}) - 2\mu\,\mathrm{sym}(\nabla\mathbf{v}) : \mathrm{sym}(\nabla\mathbf{u}) \\
&\quad + \lambda(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v}) - \lambda(\nabla \cdot \mathbf{v})(\nabla \cdot \mathbf{u}) \qquad = 0.
\end{aligned}$$

Applying the divergence theorem (2.4) to the left-hand side of (2.16) yields a generalization of *Green's second identity* also known as the *Maxwell-Betti reciprocal theorem*:

$$\int_\Gamma \left[\boldsymbol{\sigma}(\mathbf{u})\mathbf{n}\cdot\mathbf{v} - \boldsymbol{\sigma}(\mathbf{v})\mathbf{n}\cdot\mathbf{u}\right]\, ds_{\tilde{\mathbf{x}}} = \int_\Omega \left[(\nabla\cdot\boldsymbol{\sigma}(\mathbf{u}))\cdot\mathbf{v} - (\nabla\cdot\boldsymbol{\sigma}(\mathbf{v}))\cdot\mathbf{u}\right]\, dv_{\mathbf{x}}. \tag{2.17}$$

So far, $\mathbf{v}$ has played a similar role to the test function in the FEM derivation, in the sense that it can be any arbitrary vector field. Now we use a particular choice for $\mathbf{v}$ in order to simplify the volume integral on the right-hand side, namely $\mathbf{v} = \mathcal{U}_i(\mathbf{y}-\mathbf{x})$. Here $\mathcal{U}_i$ is the i-th column of the elastostatic *fundamental solution*, also known as elastostatic *Green's function* (2.21), which has the following special property: $-\nabla_{\mathbf{x}}\cdot\boldsymbol{\sigma}(\mathcal{U}_i) = \delta(\mathbf{y}-\mathbf{x})\mathbf{e}_i$, where δ is the Dirac delta function and $\mathbf{e}_i$ is the i-th standard basis vector (i.e. the unit vector along the i-th Cartesian coordinate axis). Using this property simplifies the second term on the right-hand side of (2.17) accordingly, while we substitute $-\mathbf{g}$ for $\nabla\cdot\boldsymbol{\sigma}(\mathbf{u})$ according to the governing equation (2.12). Consequently, eq. (2.17) becomes

$$\int_\Gamma \left[\boldsymbol{\sigma}(\mathbf{u})\mathbf{n}\cdot\mathcal{U}_i - \boldsymbol{\sigma}(\mathcal{U}_i)\mathbf{n}\cdot\mathbf{u}\right]\, ds_{\tilde{\mathbf{x}}} = -\int_\Omega \mathbf{g}\cdot\mathcal{U}_i\, dv_{\mathbf{x}} + \int_\Omega \delta(\mathbf{y}-\mathbf{x})\mathbf{e}_i\cdot\mathbf{u}(\mathbf{x})\, dv_{\mathbf{x}}. \tag{2.18}$$

Using the screening property of the Dirac-δ function and rearranging terms results in the following representation formula for the i-th component of the displacement field:

$$\mathbf{u}_i(\mathbf{y}) = \int_\Gamma \left[\boldsymbol{\sigma}(\mathbf{u})\mathbf{n}\cdot\mathcal{U}_i - \boldsymbol{\sigma}(\mathcal{U}_i)\mathbf{n}\cdot\mathbf{u}\right]\, ds_{\tilde{\mathbf{x}}} + \int_\Omega \mathbf{g}\cdot\mathcal{U}_i\, dv_{\mathbf{x}}. \tag{2.19}$$

We now adopt the more convenient notation of [38] to denote surface tractions by the *traction operator* $\mathcal{T}(\circ) := \boldsymbol{\sigma}(\circ)\mathbf{n} = 2\mu\,\mathrm{sym}(\nabla\circ)\mathbf{n} + \lambda(\nabla\cdot\circ)\mathbf{n}$, and consequently $\mathbf{q} = \boldsymbol{\sigma}(\mathbf{u})\mathbf{n} = \mathcal{T}\mathbf{u}$. Furthermore, we collect all three components of vectors by writing a column-vector of the form $(\mathbf{q}\cdot\mathcal{U}_1\ \mathbf{q}\cdot\mathcal{U}_2\ \mathbf{q}\cdot\mathcal{U}_3)^\mathsf{T}$ as a product $\mathcal{U}^\mathsf{T}\mathbf{q}$. Once we move the evaluation point to the boundary ($\mathbf{y}\to\tilde{\mathbf{y}}$), we obtain the following displacement boundary integral equation (BIE) of linear elastostatics:

$$\mathbf{u}(\tilde{\mathbf{y}}) = \int_\Gamma \mathcal{U}\mathbf{q}\, ds_{\tilde{\mathbf{x}}} - \int_\Gamma (\mathcal{T}\mathcal{U})^\mathsf{T}\mathbf{u}\, ds_{\tilde{\mathbf{x}}} + \int_\Omega \mathcal{U}\mathbf{g}\, dv_{\mathbf{x}}. \tag{2.20}$$

Note that the fundamental solution $\mathcal{U}$ is a symmetric 3×3 tensor field, which is why transpositions have been omitted. Similarly, $\mathcal{T}\mathcal{U}$ is a 3×3 tensor field, where each column is $\boldsymbol{\sigma}(\mathcal{U}_i)\mathbf{n}$, which is generally not symmetric. A more detailed derivation of the fundamental solution can be found in [38]; it is usually written in the form found in [9]:

$$\mathcal{U}_{ij}(\mathbf{y}-\mathbf{x}) = \frac{1+\nu}{8\pi E(1-\nu)r}\left((3-4\nu)\delta_{ij} + \partial_i r\ \partial_j r\right), \tag{2.21}$$

where $r := \|\mathbf{y}-\mathbf{x}\|$ and $\partial_i r := \partial r/\partial \mathbf{y}_i = (\mathbf{y}_i - \mathbf{x}_i)/r$. Intuitively each coefficient of the fundamental solution describes the displacement along the j-th axis at $\mathbf{y}$ caused by a point force applied along the i-th direction at $\mathbf{x}$, see also [16].

In addition to the BIE (2.20), applying the traction operator with respect to $\tilde{\mathbf{y}}$ yields the following boundary integral equation for tractions (subscripts indicate that tractions are evaluated with respect to either source or field point):

$$\mathcal{T}_{\tilde{\mathbf{y}}}\mathbf{u}(\tilde{\mathbf{y}}) = \mathbf{q}(\tilde{\mathbf{y}}) = \mathcal{T}_{\tilde{\mathbf{y}}} \int_\Gamma \mathcal{U}\mathbf{q} \; ds_{\tilde{\mathbf{x}}} - \mathcal{T}_{\tilde{\mathbf{y}}} \int_\Gamma (\mathcal{T}_{\tilde{\mathbf{x}}}\mathcal{U})^\mathsf{T}\mathbf{u} \; ds_{\tilde{\mathbf{x}}} + \mathcal{T}_{\tilde{\mathbf{y}}} \int_\Omega \mathcal{U}\mathbf{g} \; dv_{\mathbf{x}}. \tag{2.22}$$

The fundamental solution (2.21) exhibits a singularity of the form $1/r$ as $\mathbf{y} \to \mathbf{x}$. Each application of the traction operator introduces an additional spatial derivative, and consequently increases the order of the singularity, which is why eq. (2.22) is sometimes referred to as a hypersingular boundary integral equation (HBIE).

Furthermore, the remaining volume integral on the right-hand side can be transformed to a surface integral in certain cases. Details can be found in [22, 38]; this discussion is, however, omitted here as we do not treat any body forces in our BEM formulation, and we assume $\mathbf{g} = 0$ from now on. In particular, we treat gravity via a rigid-body system instead (see chapter 7). Figure 30 shows an example of a solid bar crumbling under its own weight, demonstrating that handling gravity in this way is sufficiently accurate for our purposes.

Similarly to the displacement representation formula, internal stresses can be evaluated via the *Kelvin stress kernels* as follows, see also [16, 38] for further details:

$$\boldsymbol{\sigma}_{ij}(\mathbf{y}) = \int_\Gamma \mathbf{S}_1(\mathbf{y} - \tilde{\mathbf{x}})\mathbf{q}(\tilde{\mathbf{x}}) \; ds_{\tilde{\mathbf{x}}} - \int_\Gamma \mathbf{S}_2(\mathbf{y} - \tilde{\mathbf{x}})\mathbf{u}(\tilde{\mathbf{x}}) \; ds_{\tilde{\mathbf{x}}} \tag{2.23}$$

$$\begin{aligned}
(\mathbf{S}_1)_{kij} :=& \frac{1}{8\pi(1-\nu)r^2} \; [(1-2\nu)(\delta_{kj}\partial_i r + \delta_{ki}\partial_j r - \delta_{ij}\partial_k r) + 3\partial_i r \partial_j r \partial_k r] \,, \\
(\mathbf{S}_2)_{kij} :=& \frac{E}{8\pi(1-\nu^2)r^3} [3\partial_{\mathbf{n}} r((1-2\nu)\delta_{ij}\partial_k r + \nu\delta_{jk}\partial_i r + \nu\delta_{ik}\partial_j r - 5\partial_i r \partial_j r \partial_k r) \\
&+ 3n_k(1-2\nu)\partial_i r \partial_j r + n_i((1-2\nu)\delta_{jk} + 3\nu\partial_j r \partial_k r) \\
&+ n_j((1-2\nu)\delta_{ik} + 3\nu\partial_i r \partial_k r) - n_k\delta_{ij}(1-4\nu)] \,,
\end{aligned} \tag{2.24}$$

where r and its derivatives are defined in the same way as in eq. (2.21), and additionally the normal derivative is $\partial_{\mathbf{n}} r := (\mathbf{y} - \tilde{\mathbf{x}}) \cdot \mathbf{n}/r$ and $\mathbf{n}$ is the outward surface normal at $\tilde{\mathbf{x}}$.

In the boundary integral formulation, both surface displacements $\mathbf{u}$ and tractions $\mathbf{q}$ are required to describe the elastostatic solution. As for the differential form, eq. (2.12), boundary conditions must be given in such a way that for any point on the surface either the displacement or the traction is known and the other unknown. In the remainder of this chapter, we focus on the common case of mixed boundary conditions, while treating the pure Neumann problem (where all surface tractions are given, but all displacements are unknown) in chapter 7.1.

At this point we now move to the discrete setting: we represent the surface of the deformable object with a triangle mesh and define approximation spaces for both bound-

ary displacements and tractions. Similar to the FEM formulation in the previous chapter, we choose piecewise-linear shape functions to interpolate per-node displacements as $\mathbf{u}(\tilde{\mathbf{x}}) \approx \sum_k \mathbf{u}_k \psi_k(\tilde{\mathbf{x}})$. Note that in the BEM context the shape functions are only defined on the triangular surface mesh rather than a volumetric mesh as in FEM. Furthermore, we also need to deal with surface tractions explicitly in a BEM approach, and we choose piecewise-constant shape functions to interpolate per-element tractions: $\mathbf{q}(\tilde{\mathbf{x}}) \approx \sum_j \mathbf{q}_j \varphi_j(\tilde{\mathbf{x}})$. We have now discretized the geometry, as well as the boundary data. However, in order to fully discretize the problem, we also need to choose how the governing equations are enforced, such that we arrive at a linear system that contains the same number of equations as (unknown) degrees of freedom. We choose a (symmetric) Galerkin BEM formulation, which works in a similar way to the Galerkin FEM discussed earlier: instead of enforcing the boundary integral equations at certain locations, we multiply them by an arbitrary test function and integrate over the whole surface. We then choose each of the shape functions in turn as the test function, basically enforcing the integral equation in a weighted average sense. Note that in our case, displacement shape and test functions are non-zero in the one-ring neighbourhood of their corresponding node, while traction shape functions are non-zero only in their corresponding element.

When using the *symmetric* Galerkin boundary element method (SGBEM), see also [71], we apply the displacement boundary integral equation (2.20) on the part of the boundary where Dirichlet boundary conditions are given and the traction boundary integral equation (2.22) on the part where Neumann boundary conditions are given. Applying this discretization scheme, as well as all boundary conditions, and moving all known degrees of freedom to the right-hand side, results in the discrete linear system

$$\begin{pmatrix} \mathbf{V} & -\mathbf{K} \\ \mathbf{K}^\mathsf{T} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{q}} \\ \hat{\mathbf{u}} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_D \\ \mathbf{f}_N \end{pmatrix}, \tag{2.25}$$

where the matrix blocks are defined according to eq. (2.26), see also eq. (3.33) and (5.17) in [38], for a more rigorous derivation. Once again, the vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{q}}$ collect unknown per-node displacements and per-element tractions respectively. (From now on we drop the "hat" from the discretized unknown vectors for convenience of notation.) Each of the four blocks of the system matrix discretizes one of the boundary integral operators found in eq. (2.20) and (2.22) respectively, where each pair of shape functions contributes a 3×3 block to the global system matrix according to:

$$\begin{aligned} \mathbf{V}_{ik} &= \int_{\Gamma_D} \varphi_i \int_{\Gamma_D} \mathcal{U} \varphi_k \, ds_{\tilde{\mathbf{x}}} ds_{\tilde{\mathbf{y}}}, \\ \mathbf{K}_{il} &= \int_{\Gamma_D} \varphi_i \int_{\Gamma_N} (\mathcal{T}_{\tilde{\mathbf{x}}} \mathcal{U})^\mathsf{T} \psi_l \, ds_{\tilde{\mathbf{x}}} ds_{\tilde{\mathbf{y}}}, \\ \mathbf{D}_{jl} &= -\int_{\Gamma_N} \psi_j \mathcal{T}_{\tilde{\mathbf{y}}} \int_{\Gamma_N} (\mathcal{T}_{\tilde{\mathbf{x}}} \mathcal{U})^\mathsf{T} \psi_l \, ds_{\tilde{\mathbf{x}}} ds_{\tilde{\mathbf{y}}}. \end{aligned} \tag{2.26}$$

Throughout the derivation of the boundary integral equations, as well as the discrete BEM system, we have not denoted restrictions of functions from the domain Ω to its boundary Γ explicitly. This restriction is usually written using the *boundary trace operator* Tr, meaning the limit of a function as the evaluation point approaches the boundary. In order to evaluate the integrals in eq. (2.26) this limiting process must be carried out carefully, and the resulting singularities must be handled appropriately. This process is commonly referred to as *regularization* and depends to some extent on the employed discretization. Details on the regularization and quadrature methods required to assemble the system matrix blocks in (2.25) can be found in [16, 38, 65]. We present a short overview in the following, but skip the mathematical details for the sake of brevity, as all these methods are readily available in BEM libraries, such as HyENA [29].

As the singularity order increases each time we apply a traction operator to the fundamental solution (2.21), we need to handle three types of kernel singularities in eq. (2.26), resulting in *weakly* singular ($\mathbf{V}$), *strongly* singular ($\mathbf{K}$) and *hyper*singular integrals ($\mathbf{D}$). In the presence of a weak singularity, the definite integral still exists, but standard numerical quadrature schemes converge slowly; typically a Duffy transform [13,37] is used to remove the singularity and hence improve the convergence of quadrature rules. For strong singularities, the integral only exists in the sense of a *Cauchy principal value* [24]: the basic idea is to exclude a small circular region around the singularity from the integration, and then evaluate the limit as the radius of this region approaches zero. A result of this procedure is an additional *integral-free term* that depends only on the geometry of the surface at the location of the singularity; if the surface is smooth at this point, the value of the integral-free term is $1/2$, which is how it appears in the first term of the right-hand side vectors defined in eq. (2.27). Finally, for hypersingular kernels, the integral exists only in the finite-part sense of Hadamard [47]. The main idea for evaluating these integrals in the Galerkin BEM approach is to use a strategy similar to integration by parts, where one derivative moves from the kernel to the test function, thereby reducing the singularity order. Please refer to chapter 4 in [38] for a detailed derivation of these regularization techniques. Considering the additional term resulting from the Cauchy principal value integrals, we can now define the right-hand side vectors of eq. (2.25):

$$\begin{aligned}
\mathbf{f}_{Di} &= \int_{\Gamma_D} \varphi_i \left[\mathbf{u}_\Gamma/2 + \int_{\Gamma_D} (\mathcal{T}_{\tilde{\mathbf{x}}}\mathcal{U})^\mathsf{T} \mathbf{u}_\Gamma \, ds_{\tilde{\mathbf{x}}} - \int_{\Gamma_N} \mathcal{U}\mathbf{q}_\Gamma \, ds_{\tilde{\mathbf{x}}} \right] ds_{\tilde{\mathbf{y}}}, \\
\mathbf{f}_{Nj} &= \int_{\Gamma_N} \psi_j \left[\mathbf{q}_\Gamma/2 - \int_{\Gamma_N} (\mathcal{T}_{\tilde{\mathbf{y}}}\mathcal{U})\mathbf{q}_\Gamma \, ds_{\tilde{\mathbf{x}}} + \mathcal{T}_{\tilde{\mathbf{y}}} \int_{\Gamma_D} (\mathcal{T}_{\tilde{\mathbf{x}}}\mathcal{U})^\mathsf{T} \mathbf{u}_\Gamma \, ds_{\tilde{\mathbf{x}}} \right] ds_{\tilde{\mathbf{y}}}.
\end{aligned} \tag{2.27}$$

Note that in our implementation the *known* boundary displacement and traction fields $\mathbf{u}_\Gamma$ and $\mathbf{q}_\Gamma$ are also given as piecewise-linear and piecewise-constant interpolations of per-node and per-element data respectively.

According to the definition of the system matrix blocks (2.26) we immediately see that both $\mathbf{V}$ and $\mathbf{D}$ are symmetric, and consequently the linear system discretizing the mixed boundary value problem of eq. (2.12) is block-skew symmetric. Unlike the

discrete FEM system, however, the BEM system matrix is dense, i.e. every unknown degree of freedom influences all the other ones. Consequently, the runtime required to assemble this dense linear system, as well as the memory required to store it, scale quadratically in the number of unknowns. The runtime required to solve a dense linear system generally scales cubically, although in our implementation we have observed that the assembly time dominates the overall runtime for the system size we typically use.

There exist a couple of methods that can provide improved scaling properties over the standard BEM approach by only approximately assembling the linear system (such as the fast multipole method [37, 79], or adaptive cross approximation [44]). Typically, these methods only result in a practical speed-up if the BEM system is sufficiently large. Instead, we aim to keep the system size manageably small by using a coarse surface mesh to simulate the elastostatic deformation and simulate only crack propagation at a higher resolution (as discussed in chapter 4.5.2). Consequently, we will not consider these fast BEM approaches in this work.

2.4 Linear elastic fracture mechanics

Now that we have established the governing equations of linear elasticity and introduced numerical methods to solve for elastostatic deformations due to given surface loads, we are ready to turn our attention to *linear elastic fracture mechanics* (LEFM). In particular we aim for a description of how materials fracture under the assumption that they behave linearly elastic until they fail. Obviously, the same assumptions used in the derivation of linear elasticity are still in place, which means that in terms of real materials, we ignore plasticity and large strains. Consequently, the LEFM model works for materials that fail under small, elastic deformations and exhibit brittle fracture.

First, consider a very basic fracture experiment: we take a cylinder made of some ideal, brittle material with a constant radius and no pre-made initial cuts or factures (or any other imperfections of the material). We then start to pull both ends of the cylinder apart along the cylinder's axis. As we increase this uniaxial tension, at some point the cylinder will break apart into two separate pieces, and we take note of the amount of tension at the breaking point. Similarly, we could push the cylinder, rather than pull it apart, to find out how much compression the material can withstand.

These basic experiments give rise to *Rankine's principal stress fracture hypothesis*: a material fails if either the maximal (tensile) or the minimal (compressive) principal stress (magnitude) exceeds some critical value referred to as *tensile* or *compressive strength* respectively. This hypothesis describes a "global" view of the fracture experiment in the sense that the strength is defined as the stress magnitude at which the entire specimen breaks. As discussed in chapter 2.2 the (normal) stress is maximal or minimal along one of its principal directions. Consequently, we can expect the fracture surface to be (roughly) orthogonal to that direction, i.e. the material breaks along a principal plane (see also fig. 2.10a in [22]).

Whenever we actually perform such a fracture experiment, real samples of materials will never be as perfect as we have assumed so far. This circumstance reveals a major problem for Rankine-type fracture criteria, namely that "*brittle materials do not have a well-defined tensile strength*" (chapter 3.3 in [26]), because small imperfections always randomly reduce the experimentally observed result. Usually averages of strength values obtained in multiple tests are reported. Griffith [21] pointed out that this effect leads to a discrepancy between the experimentally observed strength of materials and predictions from molecular theory. In particular, small scratches on the surface suffice to effectively reduce the apparent strength of some brittle materials.

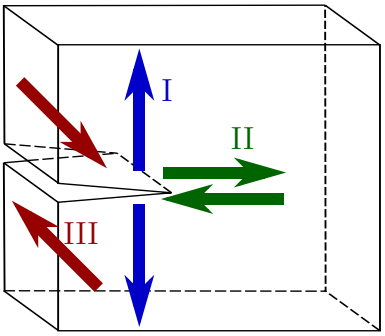


Figure 4: Loading modes.

This observation motivates focusing on cases where some initial cracks (or similarly small notches or grooves) are already present in the material. We can then turn our attention to what happens locally at the crack front (i.e. the bounding curve of a fracture surface). Consequently we consider a local coordinate system at the crack front ($\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$), where we choose $\mathbf{n}_1$ to be the fracture surface normal, $\mathbf{n}_2$ the crack-front normal (in the plane of the fracture), and $\mathbf{n}_3$ the crack-front tangent. Depending on how the material deforms locally around the crack front, we then define the three *loading modes* as illustrated in fig. 4, see also fig. 1.4 in [17]: (I) opening parallel to the surface normal, (II) shearing parallel to the crack-front normal, and (III) shearing parallel to the crack-front tangent.

We first consider pure *mode-I loading*: we take (for example) our cylinder and add a (small) planar crack (or cut) orthogonal to the axis of tension. If we now repeat the fracture experiment described earlier, the observed load at which the cylinder breaks apart completely will be lower than before due to the presence of the crack. Adopting a more localized point of view leads to *Griffith's failure hypothesis*, which states that the load must be (locally) maximal at the crack front and the crack extends "*if the system can pass from the unbroken to the broken condition by a process involving a continuous decrease in potential energy*" [21], see also eq. (10)–(13) there.

Based on the analysis of Westergaard [75], Irwin [33] gives a precise description of the stress state in the vicinity of a crack tip (which is the 2D analogue of a crack front). Imagine a straight crack contained in an otherwise infinite 2D plate (or similarly a cross-section of a 3D block), where the crack is aligned with the x-axis, and we apply a uniaxial tension along the y-axis (far away from the crack). According to eq. (10) in [33] the tension observed at a distance r away from the crack tip, at an angle θ with respect to the x-axis, is of the form:

$$\boldsymbol{\sigma}_{yy} = K_{\mathrm{I}} \cos(\theta/2) \left[1 + \sin(\theta/2) \sin(3\theta/2)\right] / \sqrt{2\pi r}, \tag{2.28}$$

introducing the *mode-I stress intensity factor* K_{I}. The exact value of K_{I} depends on the magnitude of the applied far-field tension, as well as on the geometry of both the

object and the crack. Some example results for relatively simple situations are listed in chapter 5.1, as well as in table 4.1 of [22]. Irwin [33] also describes how to measure the stress intensity experimentally using strain gauges ahead of the crack tip.

Note that the tension in eq. (2.28) has a $1/\sqrt{r}$ singularity as $r \to 0$. Figure 5 illustrates this stress component: the red curve plots the tension along the x-axis (white line), while the background colour shows its distribution in the x-y-plane. In real materials, infinite stresses can obviously not occur, therefore crack tips are always slightly blunt, rather than perfectly sharp, due to some plastic deformation in a small region around the crack tip. Nevertheless, the influence of this "plastic zone" (where the assumptions of linear elasticity break down) is small if it "*does not extend away from the crack by more than a small fraction of the crack length*" [33]. Brittle fracture can then be described phenomenologically by *Griffith's energy balance*, which in a general form reads

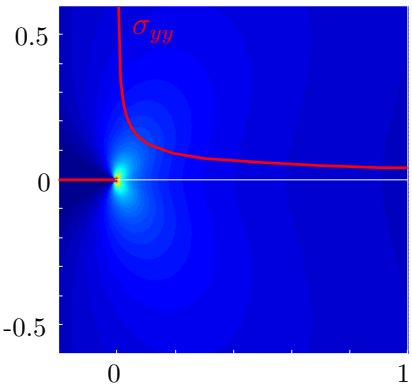


Figure 5: Singular stress field at a crack tip.

$$\frac{dW_e}{dA} + \frac{dW_s}{dA} = 0, \tag{2.29}$$

stating that during the fracture process, elastic strain energy W_e is converted to fracture surface energy W_s, where A is the crack area (or similarly for the crack length a in 2D). Introducing the *strain energy release rate* G for a small (straight) crack advance and the *critical energy release rate* G_c, we can write this criterion as $G = G_c$. According to eq. (4.84) and eq. (4.98) in [22] respectively, G (for the pure mode-I case) and G_c are:

$$G := -\frac{dW_e}{dA} = K_{\mathrm{I}}^2 \frac{1-\nu^2}{E} \quad \text{and} \quad G_c := \frac{dW_s}{dA} = 2\gamma, \tag{2.30}$$

where (E, ν) are Young's modulus and Poisson's ratio respectively, and γ is the material's *surface energy density*, i.e. the energy required to form a new fracture surface of infinitesimal area dA in the material. (Recall that every crack has a positive and a negative face, hence the factor 2.) Using the relation between G and K_{I} (2.30), we can also write the energy balance criterion in the equivalent form $K_{\mathrm{I}}^2 = K_c^2$, introducing the material's *toughness* $K_c^2 := 2\gamma E/(1-\nu^2)$.

So far, we have considered a statically loaded crack, where Griffith's energy balance provides a criterion for the onset on crack propagation. In order to determine the speed at which the crack is then expected to propagate, we also need to consider the elastodynamic equivalent of K_{I}: the dynamic stress intensity factor $K_{\mathrm{I}}^{\mathrm{dyn}}(v)$, depending on the crack propagation speed v, as well as the resulting dynamic energy release rate $G^{\mathrm{dyn}}(v)$. According to eq. (5.3.11) and (6.4.26) in [17], the relation between the static and dynamic quantities can be written in terms of universal functions of the crack speed, $A_{\mathrm{I}}(v)$ and $k(v)$, as $K_{\mathrm{I}}^{\mathrm{dyn}}(v) = K_{\mathrm{I}} k(v)$ and

$G^{\text{dyn}}(v) = A_{\text{I}}(v)K_{\text{I}}^{\text{dyn}}(v)^2(1-\nu^2)/E = A_{\text{I}}(v)k(v)^2G$ respectively. Substitution of these relations into Griffith's energy balance yields the dynamic crack propagation criterion (eq. (7.4.4) in [17]): $[K_{\text{I}}k(v)]^2A_{\text{I}}(v) = K_c^2$.

Applying an affine approximation to the factor on the left-hand side: $k(v)^2A_{\text{I}}(v) \approx 1 - v/c_R$, eq. (7.4.5) in [17], where c_R is the Rayleigh wave speed in the material, yields an estimate of the crack propagation speed (if $|K_{\text{I}}| \geq K_c$):

$$v(K_{\text{I}}) \approx c_R\left(1 - K_c^2/K_{\text{I}}^2\right). \tag{2.31}$$

Rayleigh waves are displacement waves that propagate along free surfaces, and their speed imposes an upper bound on the crack propagation speed. According to Gross and Selig [22], c_R is typically within 85% to 95% of the shear wave speed, and we choose the approximate value $c_R \approx 0.9\sqrt{\mu/\rho}$ for all our examples.

All the results so far describe only the case of pure mode-I loading of a straight crack propagating in a straight line. Unfortunately, in the more general case of *mixed mode loading*, it is not immediately clear in which direction a crack should propagate, and how to derive the energy release rate in that direction. Furthermore, due to the difference between pressure- and shear wave speeds, it is also not clear how the dynamic SIF and energy release rate depend on the propagation speed. However, in terms of developing a fracture model suitably fast and stable for computer graphics applications, we ignore these issues as much as possible and opt for simple approximations that we consider good enough for our purposes. In particular, having already assumed a linear elastic behaviour, this linearity allows for a *superposition* of the three distinct loading modes, and following eq. (4.86) in [22], we can extend the energy release rate from eq. (2.30) to

$$G(K_{\text{I}}, K_{\text{II}}, K_{\text{III}}) = (K_{\text{I}}^2 + K_{\text{II}}^2)\frac{1-\nu^2}{E} + K_{\text{III}}^2\frac{1+\nu}{E}. \tag{2.32}$$

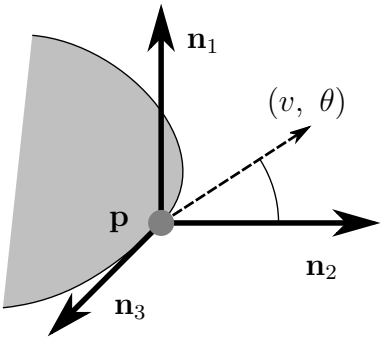


Figure 6: Local crack-front coordinates and propagation angle.

Note that this energy release rate still assumes that the crack propagates straight ahead. In order to determine the direction of crack propagation, we first need to describe how the energy release rate changes with respect to a rotation by θ around the crack-front tangent. Given a local coordinate system at a point $\mathbf{p}$ on the crack front $(\mathbf{n}_1,\ \mathbf{n}_2,\ \mathbf{n}_3)$, as shown in fig. 6, we can define a stress intensity factor (SIF) associated with each loading mode: $(K_{\text{I}},\ K_{\text{II}},\ K_{\text{III}})$. We can then figure out the propagation direction θ in the $(\mathbf{n}_1 \times \mathbf{n}_2)$-plane at $\mathbf{p}$ due to the given SIFs.

While there exist various approaches to derive the propagation angle θ, we find the *kink model*, given in eq. (4.147) of [22], appropriate for our application. This model basically states that

the crack should propagate in the direction of maximal energy release rate with respect to the crack-front near-field stress (i.e. the stress field in a small region around the crack front, where the diverging stress field described by the SIFs dominates all other contributions). The underlying assumption is that the crack already has an infinitesimally small "kink" in that direction. In this case the energy release rate for a crack propagating along a direction given by θ can be calculated by replacing the SIFs in eq. (2.32) with expressions for the *local* SIFs in a coordinate system rotated by θ around the crack-front tangent.

In order to derive these local SIFs (given in eq. (2.35)) we use a generalization of Irwin's result (2.28) by superposition of loading modes. The near-field stress (see eq. (4.26) in [22]) now reads

$$\sqrt{2\pi r}\tilde{\boldsymbol{\sigma}}_{ij} = K_{\mathrm{I}} f^{\mathrm{I}}_{ij}(\theta) + K_{\mathrm{II}} f^{\mathrm{II}}_{ij}(\theta) + K_{\mathrm{III}} f^{\mathrm{III}}_{ij}(\theta), \tag{2.33}$$

where the angular dependence is given by the following functions (note that the stress tensor must be symmetric, so $f_{ij} = f_{ji}$):

$$\begin{aligned}
f^{\mathrm{I}}_{11}(\theta) &= \quad \cos(\theta/2)(1 - \sin(\theta/2)\sin(3\theta/2)),\\
f^{\mathrm{I}}_{22}(\theta) &= \quad \cos(\theta/2)(1 + \sin(\theta/2)\sin(3\theta/2)),\\
f^{\mathrm{I}}_{12}(\theta) &= \quad \cos(\theta/2)\sin(\theta/2)\cos(3\theta/2),\\
f^{\mathrm{II}}_{11}(\theta) &= -\sin(\theta/2)(2 + \cos(\theta/2)\cos(3\theta/2)),\\
f^{\mathrm{II}}_{22}(\theta) &= \quad \cos(\theta/2)\sin(\theta/2)\cos(3\theta/2),\\
f^{\mathrm{II}}_{12}(\theta) &= \quad \cos(\theta/2)(1 - \sin(\theta/2)\sin(3\theta/2)),\\
f^{\mathrm{III}}_{13}(\theta) &= -\sin(\theta/2),\\
f^{\mathrm{III}}_{23}(\theta) &= \quad \cos(\theta/2).
\end{aligned} \tag{2.34}$$

Following the derivation of Patricio and Mattheij [58], we then apply a rotation by θ around the crack-front tangent, encoded in the rotation matrix $\mathbf{R}_\theta$, to the near-field stress $\tilde{\boldsymbol{\sigma}}$ of eq. (2.33), which results in $\hat{\boldsymbol{\sigma}} = \mathbf{R}_\theta^{\mathsf{T}} \tilde{\boldsymbol{\sigma}} \mathbf{R}_\theta$. Finally, we define the rotated stress intensities $K_\theta := \lim_{r\to 0} \sqrt{2\pi r}\hat{\boldsymbol{\sigma}}_{22}$, $K_r := \lim_{r\to 0} \sqrt{2\pi r}\hat{\boldsymbol{\sigma}}_{12}$, and $K_3 := \lim_{r\to 0} \sqrt{2\pi r}\hat{\boldsymbol{\sigma}}_{23}$ describing the *local* mode-I, II, and III loading in the rotated coordinate system (which is aligned with the direction the crack is going to propagate in). Note that the limits are easily evaluated because the input stress field is already defined in terms of SIFs according to eq. (2.33). Expanding these expressions produces the local SIFs in the form found in [22, 58]:

$$\begin{aligned}
K_\theta &= K_{\mathrm{I}}\cos^3(\theta/2) - 3K_{\mathrm{II}}\cos^2(\theta/2)\sin(\theta/2),\\
K_r &= K_{\mathrm{I}}\cos^2(\theta/2)\sin(\theta/2) + 3K_{\mathrm{II}}\cos^3(\theta/2) - 2K_{\mathrm{II}}\cos(\theta/2),\\
K_3 &= K_{\mathrm{III}}\cos(\theta/2).
\end{aligned} \tag{2.35}$$

These rotated SIFs do not depend on stresses tangential to the crack surface ($\hat{\boldsymbol{\sigma}}_{11}$, $\hat{\boldsymbol{\sigma}}_{13}$, $\hat{\boldsymbol{\sigma}}_{33}$), because the singular SIF field dominates for small distances r. We can

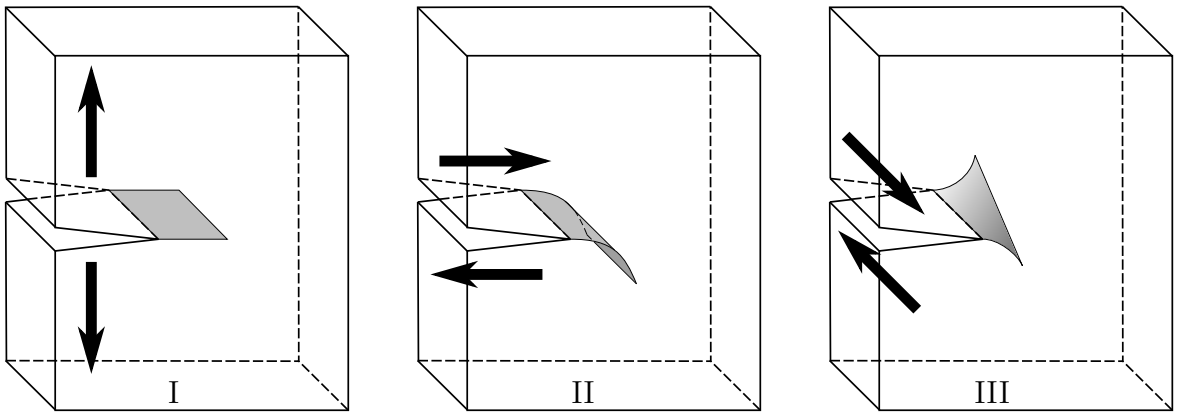


Figure 7: Expected fracture shape per loading mode.

now use these SIFs to evaluate the energy release rate G with respect to an assumed kink-angle θ, according to eq. (2.32): $G(\theta) = (K_\theta^2 + K_r^2)(1-\nu^2)/E + K_3^2(1+\nu)/E$. (This equation also appears as (4.9) when we present our crack propagation method.)

To the best of our knowledge, there is no analytical expression for the angle θ that maximizes this energy release rate in general. However, assuming $K_{\text{III}} = 0$ and consequently $K_3 = 0$ regardless of θ, and solving $dG/d\theta = 0$ for θ, the directions of maximum energy release rate for the mode-I-II loaded case are

$$\theta^*(K_{\text{I}},\ K_{\text{II}}) = 2\arctan\frac{K_{\text{I}} \mp \sqrt{K_{\text{I}}^2 + 8K_{\text{II}}^2}}{4K_{\text{II}}}, \tag{2.36}$$

if $K_{\text{II}} \neq 0$, otherwise the maximum is $\theta^* = 0$ in the pure mode-I case. Assuming $K_{\text{III}} = 0$ implies that a direction that maximizes the energy release rate also maximizes (or minimizes) the local mode-I SIF K_θ, i.e. $(dK_\theta/d\theta)(\theta^*) = 0$, and the local mode-II SIF K_r vanishes, i.e. $K_r(\theta^*) = 0$, see [58]. One could of course apply numerical methods to find the maximal energy release rate in the more general case where $K_3 \neq 0$, but for the sake of computational efficiency we use an approximate solution for our crack propagation method instead, see also fig. 12.

As a consequence of the above analysis, we expect a crack to propagate in such a way that the local mode-I loading is maximal. In terms of the global shape of a fracture due to mixed mode loading, we arrive at a similar conclusion as for the Rankine hypothesis, which is that the fracture surface should be (locally) orthogonal to the maximal principal stress direction. Figure 7 illustrates the expected crack propagation behaviour for an initially flat crack for each (global) deformation mode. Equation (2.36) together with (2.31) (using K_θ instead of K_{I}) model this behaviour and specify the crack propagation speed and direction respectively for a mixed mode-I-II loaded point on the crack front. We consider general mixed mode loading in chapter 4.5.1.

2.5 Fracture simulation

In order to simulate brittle fracture based on the LEFM model, derived in the previous chapter, we need to evaluate the stress intensities along the crack front, which allow us to determine the crack front's propagation velocity. We can then integrate the crack front motion over time resulting in a new fracture surface. The fracture process releases elastic strain energy and consumes energy required to form the new fracture. Consequently, as the crack grows, it introduces additional degrees of freedom for the object to deform, which in turn affects the stress intensities observed at the crack front. While the Griffith criterion models the crack propagation process, we still rely on a Rankine criterion to initiate new fractures. We choose to make these initial fractures very small, however, and immediately switch to the Griffith criterion once the initial crack front is known.

Contrary to our approach, many state-of-the-art fracture simulations in computer graphics (such as [55, 57, 60, 79]) still use mostly stress-based (Rankine-type) fracture criteria, ignoring (which means numerically smoothing) the stress singularity around the crack front. Smoothing out the singular stress field might alleviate the divergent part directly at the crack front, but in turn increases the (finite) stress nearby; this erroneous stress increase typically results in excessive fracturing (artificial shattering), as seen for example in fig. 11 of [57]. One strategy to alleviate this problem is to choose a different fracture threshold in the vicinity of existing cracks, as done in [67].

Whichever way one chooses to simulate the fracture process, the degrees of freedom in the elastic deformation model must be updated when adding new fractures, such as to reflect the changing geometry and topology of the breaking object. In terms of finite element methods, the straightforward approach is to split the mesh along existing faces (duplicating nodes in the process) [57]. However, this basic approach cannot model fractures that do not align with faces already present in the initial FEM mesh. A more general approach is to cut through the elements as required and inserting new nodes where appropriate [55, 60]. Cutting an element with an arbitrary plane might result in badly-shaped pieces and degrade the mesh quality over time. An alternative idea is to duplicate the elements instead (and marking which part actually represents intact material) and inserting "virtual" nodes to keep the elements well shaped [46, 74]. All of these methods directly represent the fractures in the mesh geometry.

Alternatively, an extended FEM [1, 20, 45] enriches the displacement approximation space with additional basis functions to represent both the crack surfaces, as well as the crack front, providing better resolution of the singular stress field, and at the same time avoiding the sometimes tedious re-meshing operations required by standard FEM. Especially in computer graphics applications, the additional crack tip functions, modelling the divergent part of the stress field, are regularly ignored in favour of simplicity and speed [36, 48].

When using a boundary element method, fractures are in principle additional boundaries that are added to the mesh of the object's surface. Recall, however, that every

crack has two faces. Consequently, there exist two approaches to represent fractures in a BEM mesh: the first option is to mesh both faces of a crack explicitly (placed geometrically coincident, but with opposite normals), such that each face can deform independently (DualBEM) [2, 77]. The DualBEM approach leads to a mesh connectivity that is locally manifold (without boundary) everywhere, at the cost of slightly more complicated meshing operations. In order to build a well-posed discretization, the displacement boundary integral equation (BIE), eq. (2.20), is applied on one face of the crack, while the traction boundary integral equation (HBIE), eq. 2.22, is applied on the opposite face, as illustrated in fig. 8. The second, simpler option to add fractures to a BEM mesh is to represent each crack by a single surface (e.g. a triangle sheet), which is a manifold with boundary and not connected to other surfaces in the mesh, as introduced in [16]. In this case, the unknown data on the fracture surface is the difference between the displacement of the positive and the negative face, referred to as the *crack opening displacement* (COD). The COD can be interpreted as a jump-discontinuity in the displacement field (defined in rest space) across the fracture surface.

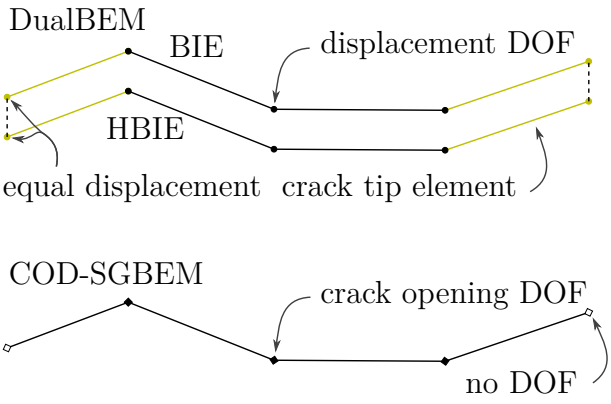


Figure 8: Comparison of DualBEM and COD-SGBEM.

While the DualBEM naturally provides more accurate results in regions where cracks intersect other boundaries (as these intersections are explicitly resolved in the mesh), it also requires special *crack-tip elements* to resolve the singular stress field, see also [62]. On the other hand, the crack opening displacement variant is less accurate when fractures exists in a small boundary layer (see [16] for a more detailed analysis), but does not require any special treatment of the crack tip (or crack front in 3D), when using a symmetric Galerkin BEM. Obviously, at the same mesh resolution, the DualBEM uses roughly twice as many degrees of freedom to represent the unknown displacement of fracture surfaces compared to the COD-SGBEM approach. Note that in the case of infinitesimal displacements (where linear elasticity is valid) and therefore negligible self-collision forces, crack surfaces are traction free (which means they are basically a special case of homogeneous Neumann boundaries). Figure 8 illustrates the two different BEM approaches for a single fracture.

The COD-SGBEM formulation incorporates the crack opening displacements directly into the linear system as new unknowns, which is advantageous for our purposes, because there is a direct relation between the COD and the SIFs we are looking for. In the remainder of this chapter, we derive the COD-SGBEM using eq. (2.25) as a starting point, and introduce a simple and efficient way to compute the stress intensities. First, we conceptually assume that the unknown displacement vector $\mathbf{u}$

can be split into three parts: the displacements of the object's ordinary surface, the displacements of the positive crack face $\mathbf{u}^+$, and the displacements of the negative crack face $\mathbf{u}^-$. Splitting the matrix blocks $\mathbf{D}$ and $\mathbf{K}$ in (2.25) accordingly results in the following SGBEM system:

$$\begin{pmatrix} \mathbf{V} & -\mathbf{K} & -\mathbf{K}^+ & -\mathbf{K}^- \\ \mathbf{K}^\mathsf{T} & \mathbf{D} & \mathbf{D}^+ & \mathbf{D}^- \\ \mathbf{K}^{+\mathsf{T}} & \mathbf{D}^{+\mathsf{T}} & \mathbf{D}^{++} & \mathbf{D}^{-+} \\ \mathbf{K}^{-\mathsf{T}} & \mathbf{D}^{-\mathsf{T}} & \mathbf{D}^{+-} & \mathbf{D}^{--} \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \mathbf{u} \\ \mathbf{u}^+ \\ \mathbf{u}^- \end{pmatrix} = \begin{pmatrix} \mathbf{f}_D \\ \mathbf{f}_N \\ \mathbf{f}^+ \\ \mathbf{f}^- \end{pmatrix}. \tag{2.37}$$

Unfortunately, this system is rank deficient because the third and fourth rows in the matrix are linearly dependent. In order to remove this null-space from the system, we define the *crack opening displacement* as $\boldsymbol{\Delta}\mathbf{u} := \mathbf{u}^+ - \mathbf{u}^-$ and then subtract the fourth row from the third in (2.37). Similarly, we refer to $\mathbf{u}_c := (\mathbf{u}^+ + \mathbf{u}^-)/2$ as the *crack's average displacement*, such that $\mathbf{u}^\pm = \mathbf{u}_c \pm \boldsymbol{\Delta}\mathbf{u}/2$. Due to the symmetric Galerkin discretization, eq. (2.26), the following relations hold: $\mathbf{K}^+ = -\mathbf{K}^-$, $\mathbf{D}^+ = -\mathbf{D}^-$, and $\mathbf{D}^{++} = \mathbf{D}^{--} = -\mathbf{D}^{+-} = -\mathbf{D}^{-+}$. Finally, in order to simplify the notation we define $\mathbf{W} := \mathbf{K}^+$, $\mathbf{X} := \mathbf{D}^+$, and $\mathbf{Y} := \mathbf{D}^{++}$ to get the COD-SGBEM system representing discrete elastostatics in the presence of fractures:

$$\begin{pmatrix} \mathbf{V} & -\mathbf{K} & -\mathbf{W} \\ \mathbf{K}^\mathsf{T} & \mathbf{D} & \mathbf{X} \\ \mathbf{W}^\mathsf{T} & \mathbf{X}^\mathsf{T} & \mathbf{Y} \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \mathbf{u} \\ \boldsymbol{\Delta}\mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_D \\ \mathbf{f}_N \\ \mathbf{0} \end{pmatrix}. \tag{2.38}$$

Note that the third row of the right-hand side is $\mathbf{0}$ due to the traction-free crack assumption. Formally, when assembling the matrix blocks $\mathbf{W}$, $\mathbf{X}$, and $\mathbf{Y}$, the Neumann surface Γ_N is split into two parts as well, one covering the ordinary surface and the other covering fracture surfaces. The integrals in (2.26) are then carried out over the part of Γ_N where the corresponding degrees of freedom reside. Note that the actual assembly implementation is exactly the same for both surface types, so any BEM implementation that is capable of assembling eq. (2.25) can immediately be used to also assemble the additional matrix blocks in eq. (2.38) as well.

Finally, we must avoid numerical issues due to the stress singularity at the crack front: we must make sure that there are no degrees of freedom located directly at the crack front. Recall that in our BEM formulation, we use per-element tractions (which are known as we assume traction-free cracks) and per-node displacements. Consequently, we enforce the following condition for any point $\mathbf{p}$ on the crack front: $\mathbf{u}^+(\mathbf{p}) = \mathbf{u}^-(\mathbf{p})$, or equivalently $\boldsymbol{\Delta}\mathbf{u}(\mathbf{p}) = 0$. While this condition is perfectly appropriate at "real" crack fronts inside of the material, it also restricts the opening displacements where cracks meet other surfaces, which explains the loss of accuracy in a small boundary layer; see [16] for an evaluation of this method.

Now that we have a linear system of equations that we can solve for the crack opening displacements (on the fracture surface), we want to evaluate the stress intensity factors (at the crack front), which in turn reveal how the crack grows. Rather

independently of the chosen simulation method, there exist various ways to compute these SIFs; Ingraffea and Wawrzynek [31] provide a good overview. While more accurate solutions may be obtained using the J-integral [64], the simplest way for our application is the *displacement correlation technique*, eq. (25) in [31], which relates opening displacements at a small distance behind the crack front to SIFs as follows:

$$K_i(\mathbf{p}) = \mu\sqrt{\pi}(\mathbf{\Delta u}_d \cdot \mathbf{n}_i)/(c\sqrt{2d}), \tag{2.39}$$

where $\mathbf{n}_i$ is the i-th unit normal vector of the local coordinate system at $\mathbf{p}$, $\mathbf{\Delta u}_d$ is the crack opening displacement evaluated at some distance d behind the crack front, i.e. $\mathbf{\Delta u}_d := \mathbf{\Delta u}(\mathbf{p} - d\mathbf{n}_2)$, and the coefficient $c = 1$ if $i = 3$ and $c = (1 - \nu)$ otherwise.

In summary, in this chapter we have introduced the governing equations of linear elasticity, as well as the basic principles of linear elastic fracture mechanics (LEFM). We have then outlined numerical methods to solve for elastostatic deformations in the presence of fractures, in particular a boundary element method including the crack opening displacements. These results combined provide a framework for simulating quasi-static crack propagation in the context of LEFM: we solve the elastostatic deformation problem, use the resulting crack opening displacements to propagate the cracks, and then update the elastostatic solution due to the newly formed fracture surfaces, and repeat.

3 Design choices and overview

Having outlined the problem we want to address in this work, as well as introduced related work and background information in the previous chapters, we now summarize the design choices we make in building our fracture simulation method and give a brief overview of the remainder of this thesis.

We work with a boundary element method because it avoids tedious volumetric meshing operations and allows us to concentrate all the computational effort on boundary surfaces. As computer graphics applications traditionally use surface meshes to represent and render the final output geometry, the required surface meshing techniques are well studied in this field. More precisely, we choose a *direct* BEM as it allows us to immediately compute the surface data we are interested in – in particular crack opening displacements – without the need for further integral evaluations. Conversely, we do *not* use a fast multipole method (or similar acceleration techniques) because we aim to work with a coarse deformation solution and keep the number of degrees of freedom in the BEM system below the cross-over point, where these theoretically fast methods become advantageous in practice. Consequently, we simulate the crack propagation process at a much higher resolution than the deformation, producing detailed and visually realistic fracture surfaces. We also choose to represent fractures as triangle sheets in the BEM mesh, following the COD-SGBEM approach instead of the DualBEM, because it further simplifies both the surface meshing operations as well as the BEM formulation and implementation. Similarly, the number of degrees of freedom in the linear system is lower than in a DualBEM, while still providing sufficient accuracy.

For our fracture simulation, we use a basic Rankine criterion to model crack initiation, and a Griffith criterion to model crack propagation. We evaluate stress intensity factors using the displacement correlation technique in order to determine both speed and direction of crack growth. As we aim for a higher resolution of the crack propagation method, compared to the deformation simulation, we need a separate geometry representation at each resolution. On the coarse level the BEM formulation requires a triangulation of the object's (ordinary) surface and similarly a sheet of triangles for each crack. On the fine level, however, we choose an implicit surface representation, implemented via the OpenVDB library. We store each fracture, as well as the object's ordinary surface, on a separate sparse grid during the fracture simulation. Afterwards, we combine all of these grids to build a single implicit surface of the fractured object, and use this representation to determine how the object separates into individual fragments.

Finally, we choose a quasi-static approach to fracture simulation, which allows us to use significantly larger time steps in our fracture simulation compared to fully dynamic methods. In doing so, we ignore small-scale dynamics (such as stress waves), whose influence on the visual appearance of fracture surfaces is relatively small compared to the influence of material inhomogeneities (which we do model). Later on we also

reduce this quasi-static model by approximating the direct influence between fracture surfaces on their opening displacements, which allows us to speed up our simulation considerably, both in terms of theoretical complexity, as well as practical runtime. Conversely, we treat large-scale dynamics in a rigid-body model (where each connected object can move around freely, but not deform). For fairly simple scenes, we first run the fracture simulation (with manually specified boundary conditions) and then simulate rigid-body dynamics of the resulting fragments as a post-process. For more complex scenes, we integrate our fracture method into the rigid-body system, such that a collision can cause an object to break, and the resulting fragments are added back into the rigid-body scene.

Our simulation method obviously requires various input parameters; we give a brief summary of the most important ones, along with a rough categorization below, but defer further details to the following chapters, as well as Appendix A.

These common simulation parameters are:

- The problem definition, including the geometry of the object, possibly existing fractures at the start of the simulation, and boundary conditions.
- Material properties, such as density, Young's modulus, Poisson's ratio, strength, and toughness.
- Resolution parameters, in particular the target edge length when meshing new fracture surfaces, and the grid spacing of the implicit surface data. (Because the maximal crack propagation speed is limited by the material parameters, these resolution targets also determine the corresponding time step and sub-step sizes.)
- Control parameters, allowing the user to influence the simulation, including limiting the maximal number of time steps, choosing between the full BEM solution, our fast approximate method, or a combination of both, and selecting among various ways to produce the final output geometry.
- If we use our fracture method coupled to a rigid-body dynamics simulation, the initial shape, position, velocity and rigid-body properties (such as friction and restitution coefficients) are defined in an input rigid-body scene, while (most of) the other parameters listed above can be specified independently for each breakable object in the scene. In this case, the problem definition for each fracture simulation triggered by a rigid-body collision is computed automatically.

Following these design choices, we describe our boundary element based fracture simulation method in ch. 4, before introducing approximations for improving the runtime (and memory consumption) of this method in ch. 5. Chapter 6 then discusses how we store the growing fracture surfaces, detect fragments, and build the final output geometry (which can then be rendered via third-party tools), as well as how we construct triangle meshes for both BEM calculations and collision detection. Finally, we present our rigid-body coupling in ch. 7, and results produced by our method in ch. 8, before closing with some concluding remarks in ch. 9.

4 High-resolution fracture simulation

In this chapter we present the core idea of our fracture method: how to simulate quasi-static brittle fracture at a significantly higher resolution than the elastostatic deformation. As introduced in ch. 1.3, previous fracture simulations in computer graphics only produce fractures of the same resolution as the deformation model, and mostly rely on adaptive meshing to mitigate this limitation. Consequently, the ability to use a coarse deformation solution and still generate realistic, highly detailed fracture surfaces constitutes a significant efficiency improvement over previous work.

We build our method on the approaches introduced in chapters 2.4 and 2.5: a symmetric Galerkin boundary element method that includes unknown crack opening degrees of freedom (COD-SGBEM), and a crack propagation criterion based on stress intensity factors (SIF), following Griffith's energy balance principle. All fracture surfaces are represented as triangle sheets in the BEM mesh, as in [16]. Each such sheet is a manifold with boundary and is not connected to other surfaces; please also refer to figures 3 and 26a. Consequently, we never need to change any existing mesh elements during the simulation and the only remaining meshing operation is to add more elements as fractures grow. The high-resolution geometry of the object (including all fractures) is stored as an implicit surface; please refer to ch. 6 for details. We consider two input parameters that allow the user to control the resolution of these two surface representations independently: r_c (coarse) defines the target (average) edge-length in the BEM mesh, and r_f (fine) defines the grid spacing of the implicit surface, which is also the resolution of the final output geometry.

We first give an overview of our algorithm and introduce how we solve the linear elastostatic COD-SGBEM system before turning our attention to crack initiation. We choose to limit crack initiation to surfaces (generally including fracture surfaces). Consequently, we need to evaluate surface stresses due to both boundary displacements and tractions. We then introduce our crack initiation criterion, which is based on the Rankine hypothesis, but adds additional constraints to avoid any artificial shattering problems, and also allows the user some control over the simulation. Note that we only initiate small cracks of a fixed size due to this criterion, and switch to a Griffith-type crack propagation criterion as soon as the initial position and orientation of a new crack are defined. Our crack propagation method is based on the results presented in ch. 2.4, and we add simplifying assumptions that allow us to handle a mixture of all three loading modes. We then present a SIF interpolation scheme that enables us to increase the resolution of our crack propagation simulation beyond the resolution of the BEM mesh. In order to take advantage of this increased resolution, we model inhomogeneous materials with a spatially varying toughness field, and heuristically formulate how the resulting toughness gradients influence the crack propagation behaviour. This approach allows us to model various materials, control the fracture behaviour, and produce visually detailed fracture surfaces, as shown for example in fig. 18, 19, and 35.

4.1 Algorithm overview

Our basic fracture simulation method takes as input the geometry of an object (either a detailed surface mesh or a coarse BEM mesh) along with material parameters (most importantly Young's modulus and Poisson's ratio), fracture parameters (strength and toughness, possibly spatially varying), and boundary conditions (displacements and/or tractions). Additionally we use a few control parameters (most notably the desired resolution of both the BEM mesh and the output geometry), which will be described in the following chapters as required. Appendix A provides a list of all fracture simulation parameters.

For now, we consider a single fracture simulation, in the sense that the boundary conditions are fixed, and we simulate until the fracture process is completed (or up to a user-specified number of time steps). Note, however, that multiple fractures can occur during one such simulation. In chapter 7 we then run one fracture simulation for each object involved in a (sufficiently forceful) collision after each rigid-body step, where the time limit is then determined by the duration of the collision.

Our fracture algorithm proceeds as follows:

(1) Load the input geometry. The user may provide either a pre-defined (coarse) BEM mesh (possibly including initial cracks), or alternatively a detailed surface mesh (without any fractures).
(2) Convert the input geometry to an implicit surface; see ch. 6 for details.
(3) If the BEM mesh has not been specified in step (1), we construct a (new) triangle mesh from the implicit surface and apply quadric simplification [18] until a user-specified number of triangles remain; see ch. 6.1 for further details. This procedure ensures a closed orientable manifold BEM mesh. The user may also manually specify small initial cracks, which we add after constructing the mesh of the object's ordinary surface.
(4) Apply boundary conditions, assemble the BEM system matrix, as well as the right-hand side vector of eq. (2.38), and solve for the initial elastostatic deformation (see ch. 4.2).
(5) The *simulation loop*:
In each BEM time step, initiate new cracks as described in ch. 4.4, then propagate all crack fronts according to ch. 4.5 (performing a fixed number of propagation "sub steps"), finally add the newly created fracture surfaces to the BEM mesh, and update the elastostatic deformation. Repeat until no more fracture elements were added to the BEM mesh (or the time limit has been reached).
(6) Post processing: compute the topology of the broken object (i.e. find fragments, see ch. 6) and output a (detailed) triangle mesh for each fragment, which can be used for rendering or subsequent animation. As per the user's choice, we output either undeformed fragments (most useful for animating their motion after the fracture – we also use this option for our rigid-body coupling), or deformed ones (possibly also for intermediate time steps) allowing for a "slow motion" animation of the fracture process; see ch. 6.4 and fig. 29 for details.

In the remainder of this chapter, we describe each step of the main simulation loop in detail, starting with the elastostatic deformation. We then present our surface stress evaluation, which forms the basis of our crack initiation method. Finally, we turn our attention to our high-resolution crack propagation method, capable of handling general 3D loading situations, tensile and compressive fracture, as well as materials with spatially inhomogeneous toughness fields.

4.2 Elastostatics with fractures

In the absence of fractures, the SGBEM block-matrix system, eq. (2.25), can be solved using Schur complements following [38]: first compute the inverse of $\mathbf{V}$, then solve $(\mathbf{K}^\mathsf{T}\mathbf{V}^{-1}\mathbf{K} + \mathbf{D})\mathbf{u} = \mathbf{f}_N - \mathbf{V}^{-1}\mathbf{f}_D$ for $\mathbf{u}$, and finally compute $\mathbf{q} = \mathbf{V}^{-1}(\mathbf{f}_D + \mathbf{K}\mathbf{u})$.

We use a similar strategy to solve eq. (2.38) including fracture surfaces. However, the matrix blocks $\mathbf{V}$, $\mathbf{K}$, and $\mathbf{D}$ still describe the interaction between pairs of (unknown) degrees of freedom located on the object's ordinary surface. These DOFs, as well as their pairwise interaction, will never change as we add new fracture surfaces. Consequently, we combine their corresponding matrix blocks into a larger block $\mathbf{A}$ and compute its inverse at the beginning of the simulation. The inverse of this block-skew-symmetric matrix is also block-skew-symmetric, see eq. (2.8.18) in [5], so we need to compute and store only one of its off-diagonal blocks.

$$\mathbf{A}^{-1} := \begin{pmatrix} \mathbf{V} & -\mathbf{K} \\ \mathbf{K}^\mathsf{T} & \mathbf{D} \end{pmatrix}^{-1} = \begin{pmatrix} (\mathbf{V} + \mathbf{K}\mathbf{D}^{-1}\mathbf{K}^\mathsf{T})^{-1} & (\mathbf{V} + \mathbf{K}\mathbf{D}^{-1}\mathbf{K}^\mathsf{T})^{-1}\mathbf{K}\mathbf{D}^{-1} \\ -(\mathbf{D} + \mathbf{K}^\mathsf{T}\mathbf{V}^{-1}\mathbf{K})^{-1}\mathbf{K}^\mathsf{T}\mathbf{V}^{-1} & (\mathbf{D} + \mathbf{K}^\mathsf{T}\mathbf{V}^{-1}\mathbf{K})^{-1} \end{pmatrix}. \tag{4.1}$$

Nevertheless, introducing new fractures will change the overall deformation of the object, requiring us to solve (2.38) repeatedly during the fracture simulation. In order to simplify the notation, we also combine the other matrix blocks accordingly: $-\mathbf{W}$ and $\mathbf{X}$ into $\mathbf{B}$, as well as $\mathbf{W}^\mathsf{T}$ and $\mathbf{X}^\mathsf{T}$ into $\mathbf{C}$. Finally, we combine the unknown surface tractions $\mathbf{q}$ and displacements $\mathbf{u}$ into a vector of unknown surface data $\mathbf{s}$, and similarly, we combine the right-hand side vectors $\mathbf{f}_D$ and $\mathbf{f}_N$ into $\mathbf{f}$. The COD-SGBEM system (2.38) now reads

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{Y} \end{pmatrix} \begin{pmatrix} \mathbf{s} \\ \boldsymbol{\Delta}\mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}. \tag{4.2}$$

Using the Schur complement method again (having already pre-computed $\mathbf{A}^{-1}$) results in the following equation for the unknown crack opening displacements $\boldsymbol{\Delta}\mathbf{u}$:

$$(\mathbf{Y} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})(\boldsymbol{\Delta}\mathbf{u}) = -\mathbf{C}\mathbf{A}^{-1}\mathbf{f}. \tag{4.3}$$

Consequently, the ordinary surface data (consisting of both tractions and displacements) becomes $\mathbf{s} = \mathbf{A}^{-1}(\mathbf{f} - \mathbf{B}\,\boldsymbol{\Delta}\mathbf{u})$, where the first term on the right-hand side

($\mathbf{A}^{-1}\mathbf{f}$) corresponds to the solution of the unfractured system, and the second term ($-\mathbf{A}^{-1}\mathbf{B}\,\boldsymbol{\Delta}\mathbf{u}$) describes the influence of the fractures onto the overall deformation. We use the *Eigen* library [23] to solve these linear systems. In particular we use their partial-pivot L-U decomposition for inverting matrices and solving all linear systems except (4.3). When solving (4.3) for the crack opening displacements, we have observed better performance when using Eigen's Cholesky factorization with pivoting.

Recall that nodes on the crack front are *not* degrees of freedom in the BEM system, as their crack opening displacement is constrained to $\mathbf{0}$ (see ch. 2.5). Consequently, our crack initiation method must generate at least one interior node for each new crack. Furthermore, as the crack front propagates, nodes that were previously located on the crack front become interior nodes. At this point, they now contribute a crack opening DOF, but their one-ring neighbourhood in the BEM mesh is already fixed. Note that this one-ring neighbourhood is the region within which the DOF's corresponding shape function is non-zero; as only the crack front can move, this region will not change anymore in subsequent time steps. Therefore, the *fracture* blocks $\mathbf{B}$ and $\mathbf{C}$ in eq. (4.2), consisting of $\mathbf{W}$ and $\mathbf{X}$ in (2.38), as well as $\mathbf{Y}$, *grow* along with the vector of opening displacements $\boldsymbol{\Delta}\mathbf{u}$ whenever new fracture surfaces are created by either crack initiation or propagation. However, because the shape-function support regions of *existing* degrees of freedom do *not* change during crack propagation, no matrix entry ever changes once it has been computed. Consequently, we compute each entry in the entire BEM system matrix exactly *once*, amortizing the matrix assembly cost over the course of the fracture simulation.

4.3 Surface stress evaluation

In order to facilitate crack initiation at surfaces, we now describe how to evaluate surface stress from the elastostatic solution described in the previous chapter. In particular, we compute an accurate 3D stress tensor for a surface triangle given the displacements of its nodes $\mathbf{u}_i$ (interpolated linearly over the element), as well as the traction $\mathbf{q}_e$ (constant over the element). This surface data is either given as a boundary condition, or found by solving eq. (4.2).

In theory, we could use the boundary representation of the stress field, eq. (2.23), to evaluate a stress sample anywhere inside the object. As the evaluation point approaches the boundary, however, the resulting singularities in the surface integrals would again require regularization. Without regularization, the interior evaluation suffers from numerical noise close to the surface. Instead, we aim to evaluate surface stress on a per-element basis and avoid costly integrations over the entire surface. Figure 10 shows that our surface stress evaluation agrees well with the interior stress according to eq. (2.23).

In order to determine the surface stress $\boldsymbol{\sigma}_e$ on each element, recall the following conditions introduced in ch. 2.2: $\boldsymbol{\sigma}_e$ must satisfy eq. (2.7), i.e. $\boldsymbol{\sigma}_e\mathbf{n}_1 = \mathbf{q}_e$, where $\mathbf{n}_1$ is the triangle's face normal. The surface stress must also be symmetric ($\boldsymbol{\sigma}_e = \boldsymbol{\sigma}_e^\mathsf{T}$).

Finally, the (in-plane) deformation of the element must be related to its stress according to the constitutive model, eq. (2.11). However, when converting from a plane stress on a triangle to a 3D stress, we need to account for the Poisson contraction in the plane of this triangle caused by the given out-of-plane traction $\mathbf{q}_e$. As illustrated in fig. 9, a traction (blue arrows) causes the object to deform from its rest state (transparent) to a stretched state (brown), and the triangle on the top face (yellow) contracts as indicated by the yellow arrows.

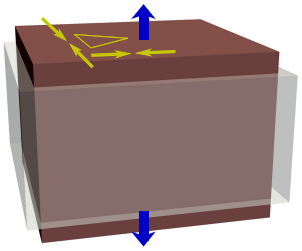
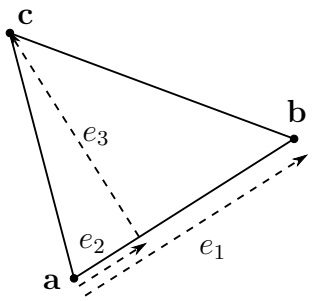


Figure 9: Illustration of Poisson contraction.

For each surface triangle, we first define a local coordinate system by the vectors $\mathbf{n}_1$ (the triangle's surface normal), $\mathbf{n}_2$ (the unit vector along the first edge of the triangle), and $\mathbf{n}_3 := \mathbf{n}_1 \times \mathbf{n}_2$. We then compute a 2D stress $\boldsymbol{\sigma}_{2d}$ in the plane spanned by $\mathbf{n}_2$ and $\mathbf{n}_3$, correct it for Poisson contraction, and finally combine it with the out-of-plane traction to build the 3D stress.

In order to compute $\boldsymbol{\sigma}_{2d}$, we map the nodal displacements into the $(\mathbf{n}_2 \times \mathbf{n}_3)$-plane and collect them in the vector $\mathbf{u}_{2d}$, as illustrated in the inset figure. Let $(\mathbf{a},\ \mathbf{b},\ \mathbf{c})$ be the coordinates of the triangle's corners in rest space, and $(\mathbf{u}_a,\ \mathbf{u}_b,\ \mathbf{u}_c)$ the nodal displacement at each corner. We then map the triangle into the 2D plane such that $\mathbf{a} \to (0,0)$, $\mathbf{b} \to (e_1, 0)$, and $\mathbf{c} \to (e_2, e_3)$, where $e_1 = \|\mathbf{b} - \mathbf{a}\|$ is the length of the edge $\mathbf{ab}$, $e_2 = (\mathbf{c} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a})/e_1$ is the projected length of the edge $\mathbf{ac}$ in the direction of $\mathbf{ab}$, and $e_3 = \|(\mathbf{c} - \mathbf{a}) - (\mathbf{b} - \mathbf{a})e_2/e_1\|$ is the orthogonal distance from the edge $\mathbf{ab}$ to node $\mathbf{c}$. We then apply the same transformation to the deformed configuration of the triangle $(\mathbf{a} + \mathbf{u}_a,\ \mathbf{b} + \mathbf{u}_b,\ \mathbf{c} + \mathbf{u}_c)$ and obtain the deformed 2D coordinates $(f_1,\ f_2,\ f_3)$. The combined 2D displacement vector is then $\mathbf{u}_{2d} := (0,\ 0,\ e_1 - f_1,\ 0,\ e_2 - f_2,\ e_3 - f_3)^\mathsf{T}$.



The in-plane stress computation follows the principle of a standard 2D FEM discretization, as introduced in ch. 2.3.1. In particular, the three independent components of $\boldsymbol{\sigma}_{2d}$ are $(\sigma_{\mathbf{ab}}, \sigma_{\perp\mathbf{ab}}, \tau)^\mathsf{T} := \mathbf{S}\mathbf{B}_e\mathbf{u}_{2d}$, i.e. the normal stresses along, and orthogonal to the edge $\mathbf{ab}$ respectively, as well as the in-plane shear. The matrix form $\boldsymbol{\sigma}_{2d}$ is then the symmetric (2×2)-matrix built from these components. Here $\mathbf{S}$ encodes the Hookean stress-strain relation and $\mathbf{B}_e$ contains the derivatives of the linear shape functions within the triangle (whose area is A_e) as follows (see also eq. (11.60) in [42]):

$$\mathbf{S} = \begin{pmatrix} \lambda + 2\mu & \lambda & 0 \\ \lambda & \lambda + 2\mu & 0 \\ 0 & 0 & \mu \end{pmatrix} \quad \text{and} \tag{4.4}$$

$$\mathbf{B}_e = \begin{pmatrix} e_3 & 0 & -e_3 & 0 & 0 & 0 \\ 0 & e_1 - e_2 & 0 & e_2 & 0 & -e_1 \\ e_1 - e_2 & e_3 & e_2 & -e_3 & -e_1 & 0 \end{pmatrix} \frac{1}{2A_e}. \tag{4.5}$$

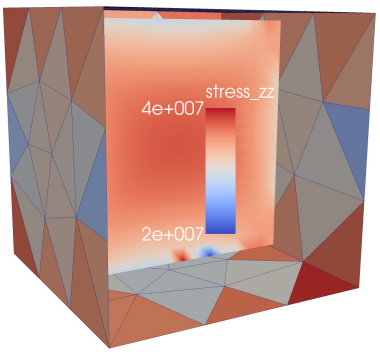


Figure 10: Comparison of interior stress (2.23) and surface stress (4.6).

We then rotate $\boldsymbol{\sigma}_{2d}$ into the global 3D coordinate system to obtain traction vectors $\mathbf{q}_2^*$ and $\mathbf{q}_3^*$. These traction vectors are $(\mathbf{q}_2^*\ \mathbf{q}_3^*) := (\mathbf{n}_2\ \mathbf{n}_3)\ \boldsymbol{\sigma}_{2d}$, where $(\mathbf{n}_2\ \mathbf{n}_3)$ is a (3×2)-matrix containing the local in-plane coordinate axes as columns. In other words, $\mathbf{q}_2^*$ and $\mathbf{q}_3^*$ are defined as the first and second column of the rotated 2D stress respectively.

Given three traction vectors with corresponding orthogonal normals, the 3D stress could be constructed by dyadic summation, as in eq. (4.6). However, summation over $(\mathbf{q}_e, \mathbf{q}_2^*, \mathbf{q}_3^*)$ will in general produce an *asymmetric* matrix, because the tractions built from the in-plane stresses might not be consistent with the given out-of-plane traction $\mathbf{q}_e$. In order to obtain a symmetric result, the traction vectors need to satisfy $\mathbf{q}_2 \cdot \mathbf{n}_1 = \mathbf{q}_e \cdot \mathbf{n}_2$ and $\mathbf{q}_3 \cdot \mathbf{n}_1 = \mathbf{q}_e \cdot \mathbf{n}_3$. Furthermore, the presence of an out-of-plane traction will modify the in-plane tractions due to Poisson's effect. Consequently, we define the *corrected in-plane tractions* as $\mathbf{q}_i := \mathbf{q}_i^* + \mathbf{n}_1(\mathbf{q}_e \cdot \mathbf{n}_i) + \mathbf{n}_i(\mathbf{q}_e \cdot \mathbf{n}_1)\nu/(1-\nu)$, where $i \in 2, 3$ and ν is Poisson's ratio. Finally, we compute the 3D per-element stress by dyadic summation:

$$\boldsymbol{\sigma}_e := (\mathbf{q}_e\ \mathbf{q}_2\ \mathbf{q}_3)\,(\mathbf{n}_1\ \mathbf{n}_2\ \mathbf{n}_3)^\mathsf{T}. \tag{4.6}$$

Verifying that $\boldsymbol{\sigma}_e$ is symmetric and satisfies $\boldsymbol{\sigma}_e \mathbf{n}_1 = \mathbf{q}_e$ is straightforward. In order to derive the factor $\nu/(1-\nu)$ for the Poisson correction, consider a cube under uniaxial tension σ_z, as illustrated in fig. 9: the strain along the z-axis ε_z satisfies $\sigma_z = E\varepsilon_z$, where E is Young's modulus. The Poisson contraction along both x- and y-axis is $\varepsilon_x = \varepsilon_y = -\nu\varepsilon_z$. Now consider a triangle on the top face of this cube: the in-plane stress due to Poisson's effect is $\sigma_x = E^*(\varepsilon_x + \nu\varepsilon_y) = -E^*(1+\nu)\nu\varepsilon_z$ with $E^* := E/(1-\nu^2)$ (and analogously for σ_y), which is exactly the result we get for $\boldsymbol{\sigma}_{2d}$. However, this stress is caused by the out-of-plane tension σ_z; there is no external force along either x- or y-axis, so we know that the correct result should be $\sigma_x = \sigma_y = 0$. Consequently, we need to add $E^*(1+\nu)\nu\varepsilon_z = \nu E\varepsilon_z(1+\nu)/(1-\nu^2) = \sigma_z\nu/(1-\nu)$ to both σ_x and σ_y.

4.4 Crack initiation

In principle, it is possible to initiate cracks anywhere inside the object or on its surface, based on any criterion due to interior or surface stress or strain, or at any location specified by the user. In some of our test cases, we also start the simulation with an object already containing cracks of a particular shape and size. For efficiency reasons, we use per-element *surface* stress according to eq. (4.6) as our main criterion for crack initiation. Conversely, evaluating interior stress using eq. (2.23) would require integration over the entire surface in a BEM framework. Another reason to work with

surface stress is that the interior stress field is divergence free in the absence of body forces according to eq. (2.12) (i.e. there are no internal sources of stress), so we expect its maxima to be located at the boundary surface. Our surface-based crack initiation method also works on previously existing fractures, thus allowing for branching cracks.

We start a new crack if all of the following conditions hold:

(a) a surface element's principal stress exceeds the local material strength,
(b) this element has not initiated a crack before,
(c) this element is farther than the average BEM edge-length (r_c) away from any crack front, and
(d) the user-specified limit for the maximum number of cracks in the simulation has not been reached yet.

Condition (a) is our main (Rankine-type) criterion for crack initiation, and we use the maximal principal stress to account for tensile fracture, as well as the minimal principal stress for compressive fracture. The element where the stress magnitude exceeds the strength the most will be fractured first. Note that the material's strength can vary in space, and we evaluate it at the centroid of each element. Many materials are easier to break under tension than under compression, i.e. their compressive strength is higher than their tensile strength. We allow the user to specify the (constant) ratio of compressive- to tensile strength to account for this effect.

In the COD-SGBEM approach, fractures are not connected to other surfaces in the BEM mesh, which means that we do not cut any elements in the mesh when a crack reaches another surface. Consequently, we keep a list of "fractured" elements to test condition (b), such that every element can initiate at most one crack. Otherwise, we risk creating overlapping elements in the mesh, which would introduce untreated singularities in the BEM integrals. Recall that due to the stress singularity around a crack front, using a Rankine-type fracture criterion in its vicinity is not appropriate. Condition (c) ensures that our Griffith-type crack propagation criterion takes precedence in this situation. Intuitively, (c) means that starting a new crack too close to a crack front is probably erroneous and also becomes redundant as soon as that crack front propagates. Both conditions (c) and (d) increase the efficiency of our method by reducing, or limiting, the total number of cracks. We find that condition (d) is also an easy way to allow the user some control over the simulation.

When we evaluate condition (a), we compute the surface stress as described in ch. 4.3. For each (ordinary) surface element e we use the corresponding entries from the BEM solution vectors ($\mathbf{q}_e$, $\mathbf{u}_i$) (where i iterates over the nodes of triangle e) to obtain the stress tensor $\boldsymbol{\sigma}_e$ according to eq. (4.6). We then compute the eigenvalues and eigenvectors of $\boldsymbol{\sigma}_e$ to find the maximal and minimal principal stress and their corresponding principal planes. The same process also allows us to handle new cracks branching off of existing fracture elements with minor modifications: in this case, due to the traction-free assumption, $\mathbf{q}_e$ is always zero, while the nodal displacement is composed of the crack's average displacement $\mathbf{u}_c$ and the crack opening displacement. Consequently, we find the displacement of the positive or negative crack face as $\mathbf{u}_i^{\pm} = \mathbf{u}_c(\mathbf{x}_i) \pm \boldsymbol{\Delta}\mathbf{u}_i/2$.

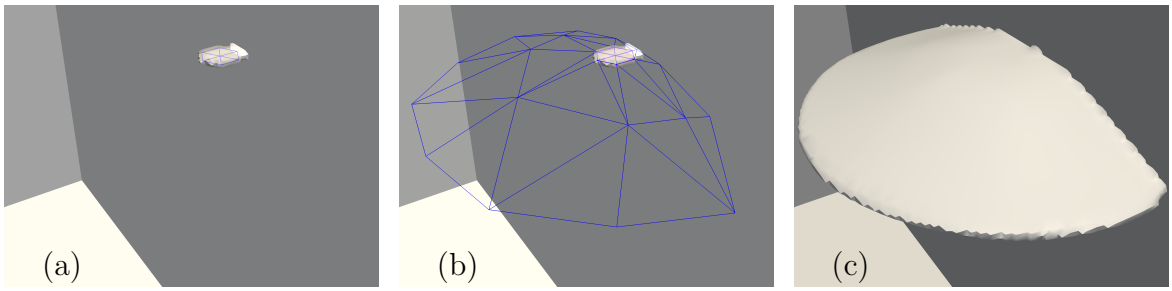


Figure 11: A newly created crack (a), its BEM mesh after 2 propagation steps (b) and the corresponding fracture surface (c). Note that the initially hexagonal shape is not visible after crack propagation. Figure 18 shows a complete view of this example, including the toughness field used to bias crack propagation.

In order to approximate the average displacement, we evaluate the representation formula, eq. (2.20), at the nodes $\mathbf{x}_i$ of the fracture element e by integrating over the *ordinary* surface (excluding fracture surfaces). We then initiate the branching crack on the positive or negative side of the fracture element depending on which side suffers the higher principal stress magnitude.

We place every new crack such that it lies in the principal plane corresponding to the principal stress that lead to crack initiation, while the centre of the new crack is aligned with the centroid of the generating surface element and offset along that element's normal vector (such as to avoid any intersections between the generating element and the new crack). We choose a simple planar hexagon of radius $0.2\ r_c$ as our initial crack shape, which is triangulated with one central node, providing the first crack opening degree of freedom of the new crack in the BEM mesh. In order to avoid visually distracting artefacts due to this very simple initial crack shape, we constrain the crack propagation direction to be planar in the first propagation sub step. This constraint ensures that the crack front turns away from the crack's initial plane smoothly, and the initially hexagonal shape is not visually apparent in the output (assuming that the new crack propagates at least once), as shown in fig. 11.

4.5 Crack propagation

So far, we have discussed how to solve the elastostatic problem in the presence of fractures, and how to initiate new cracks due to surface stress. We are now ready to formulate a crack propagation criterion that describes how fractures grow due to arbitrary loads. We first extend the analytical results presented in ch. 2.4 from mode-I-II loading to arbitrary 3D loading, and formulate an equation of motion for the crack front. We then show how to treat tensile and compressive fracture in the same framework. Most importantly, we present how to simulate crack propagation at a much higher resolution than the elastostatic deformation by interpolating stress intensities in ch. 4.5.2, and finally we discuss our treatment of spatially varying toughness and its influence on the crack propagation behaviour.

4.5.1 Mixed mode crack front motion

We start our derivation of an equation of motion for a point $\mathbf{p}$ on the crack front from eq. (2.31) and (2.36). Given the stress intensities $(K_{\mathrm{I}},\ K_{\mathrm{II}},\ K_{\mathrm{III}})$ describing the singular stress field in the vicinity of $\mathbf{p}$ with respect to the local coordinate system $(\mathbf{n}_1,\ \mathbf{n}_2,\ \mathbf{n}_3)$, we want to model the motion of this point as the crack front propagates. The local coordinate axes, as defined in ch. 2.4 (fig. 6), are the fracture surface normal, the crack-front (forward facing) normal, and the crack-front tangent respectively.

First, we calculate the (approximate) propagation speed, based on the idea of [17], for mode-I fracture, eq. (2.31). We want to extend this propagation speed to a mixed-mode loading situation, while ignoring issues arising from the elastodynamic behaviour of the material as much as possible. In order to do so, we define the *overall effective SIF* K_{eff} by combining all three SIFs, and accounting for the pre-factors with which they contribute to the energy release rate in eq. (2.32):

$$K_{\mathrm{eff}}^2 := K_{\mathrm{I}}^2 + K_{\mathrm{II}}^2 + K_{\mathrm{III}}^2/(1-\nu). \tag{4.7}$$

We then substitute K_{eff} for K_{I} in eq. (2.31) to find the approximate propagation speed

$$v(K_{\mathrm{eff}}) = c_R(1 - K_c^2/K_{\mathrm{eff}}^2) \quad \text{if } K_{\mathrm{eff}} > K_c, \qquad 0 \text{ otherwise.} \tag{4.8}$$

Note that in order to simplify the implementation, we do not consider the angular variation of the singular stress field when determining the propagation speed, and we only continue with the more involved direction calculations, described in the remainder of this chapter, if the effective SIF exceeds the material's toughness: $K_{\mathrm{eff}} > K_c$. In other words, we only allow a fracture to propagate (in any direction) if the load is sufficiently high for it to propagate straight ahead. As fractures tend to align orthogonally to a principal stress direction (in particular we enforce this alignment when initiating new cracks), and propagate mostly under mode-I dominated loading (see also [58]), this simplification is reasonable.

Once we have determined the propagation speed, we also need to find the propagation direction. Any point on the crack front may only move in the plane locally orthogonal to the crack-front tangent, i.e. the $(\mathbf{n}_1 \times \mathbf{n}_2)$-plane. We then define the propagation direction in terms of the angle θ, such that the corresponding unit vector in 3D is $\sin(\theta)\mathbf{n}_1 + \cos(\theta)\mathbf{n}_2$, see also fig. 6. Assuming the *kink model*, as introduced in ch. 2.4, we find the stress intensities $(K_\theta,\ K_r,\ K_3)$ in a coordinate frame rotated by θ according to eq. (2.35) and the corresponding energy release rate is

$$G(\theta) = \left[(K_\theta^2 + K_r^2)(1-\nu^2) + K_3^2(1+\nu)\right]/E. \tag{4.9}$$

Under the condition that $K_3 = 0$ (i.e. under mode-I-II loading conditions), the angle θ^* that maximizes this energy release rate is given by eq. (2.36). However, under mixed mode loading, where all three SIFs are non-zero, a closed-form expression for the optimal angle is not known. While we could, of course, find the optimal angle numerically,

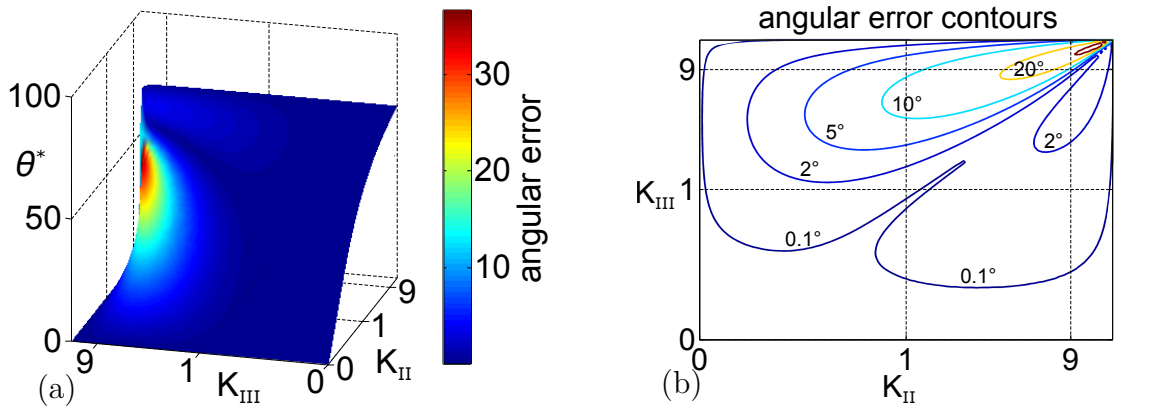


Figure 12: The crack propagation angle according to eq. (4.10) (a) due to given mode-II and III SIFs in multiples of K_I; color shows the absolute angular error compared to numerically optimizing $G(\theta)$ defined in eq. (4.9). Image (b) shows contour lines of this error.

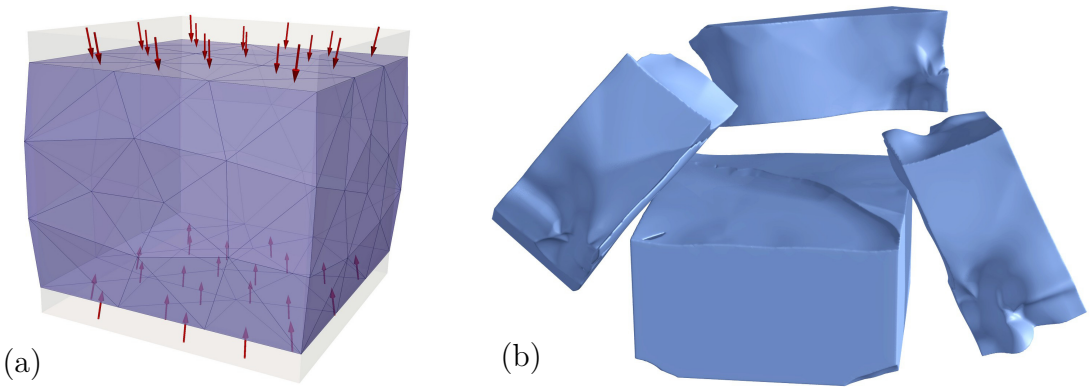
there is another problem in the rare case that mode-I loading is small and mode-II and III are both large, but almost equal to each other, i.e. $|K_\mathrm{I}/K_\mathrm{II}| \approx |K_\mathrm{I}/K_\mathrm{III}| << 1$ and $|K_\mathrm{II}/K_\mathrm{III}| \approx 1$: in this case the optimal angle θ^* is close to 0° when $K_\mathrm{III} > K_\mathrm{II}$, but about 70° when $K_\mathrm{II} > K_\mathrm{III}$, with a steep transition between the two cases. This sudden change of the optimal angle under small variations of the loading conditions can lead to unpleasant visual artefacts.

Consequently, we use a different approach: intuitively, both pure mode-I and mode-III loading conditions drive the crack "forward" (locally), i.e. $\theta^* = 0$ in both cases. (This fact can easily be seen when setting $K_\mathrm{I} \neq 0,\ K_\mathrm{II} = K_\mathrm{III} = 0$ or $K_\mathrm{III} \neq 0,\ K_\mathrm{I} = K_\mathrm{II} = 0$ in eq. (2.35): only the cosine-terms remain non-zero.) We now ignore the different angular variations of the mode-I and III SIF fields and group the two together into one *effective mode-I-III SIF* $K_\mathrm{I,III}$, while again accounting for the different pre-factors with which they contribute to the energy release rate: $K^2_\mathrm{I,III} := K^2_\mathrm{I} + K^2_\mathrm{III}/(1-\nu)$. Substituting $K_\mathrm{I,III}$ for K_I (and setting $K_\mathrm{III} = 0$), we can now again compute the optimal propagation angle using eq. (2.36), which now reads:

$$\theta^*(K_\mathrm{I,III},\ K_\mathrm{II}) = 2\,\mathrm{atan}\,\frac{K_\mathrm{I,III} - \sqrt{K^2_\mathrm{I,III} + 8K^2_\mathrm{II}}}{4K_\mathrm{II}}. \tag{4.10}$$

The error due to this simplifying assumption, in terms of absolute angle difference, is shown in fig. 12; in particular, this error is less than 2° whenever $|K_\mathrm{III}| < |K_\mathrm{I}|$, which is very likely as cracks tend to propagate in such a way as to maximize local mode-I loading. In summary, we define the 3D motion of a point $\mathbf{p}$ on the crack front according to eq. (4.8) and (4.10). The crack propagation velocity vector at $\mathbf{p}$ is then:

$$\dot{\mathbf{p}} = v(K_\mathrm{eff})\,\sin[\theta^*(K_\mathrm{I,III},\ K_\mathrm{II})]\ \mathbf{n}_1 + v(K_\mathrm{eff})\,\cos[\theta^*(K_\mathrm{I,III},\ K_\mathrm{II})]\ \mathbf{n}_2. \tag{4.11}$$



(a) (b)

Figure 13: A cube breaks under compression into multiple fragments (b). Displacements in image (a) are magnified; arrows show tractions resulting from Dirichlet boundary conditions. Note that the input mesh is not symmetric, which causes slight differences in the surface stresses, resulting in asymmetric crack initiation, even though the material properties are homogeneous.

Compressive crack propagation

Note that the sign of K_{I} (and similarly K_θ) indicates whether the crack front is currently under tension (positive) or compression (negative), while $K_{\mathrm{I,III}}$ is always positive. In the case of compressive fracture, this drawback needs to be accounted for. We focus on tensile fracture during the derivation of our crack propagation method throughout this chapter, i.e. assuming that $K_\theta > 0$. The main point of this section is to introduce a fast and simple approach that allows us to treat compressive fracture within the same framework.

First, recall that due to the assumptions of linear elasticity, in particular working with infinitesimally small deformations, self-collision forces (where the two faces of a crack are compressed together) are negligible. Instead, we opt for a phenomenological description, similar to the approach we use for crack initiation, based on the observation that most materials are more difficult to break under compression than under tension. As input parameters we consider the tensile toughness K_c and the ratio of compressive- to tensile toughness $f_c := K_c^{\mathrm{comp}}/K_c$.

In the tensile case we seek to propagate the crack in a direction where $K_\theta > K_c$, conversely, under compression we need to find a direction where $K_\theta < 0$ and $-K_\theta > f_c K_c$. In order to keep the remaining implementation of our crack propagation method unchanged, we first decide whether the crack propagates under tension or under compression: we choose compressive fracture if $\max K_\theta < -\min K_\theta / f_c$, where K_θ is evaluated according to eq. (2.35). In the case of compressive fracture we then replace all three stress intensity factors by $K_i \leftarrow -K_i/f_c$ and proceed with the calculations of propagation speed, direction, and influence of the toughness gradient as in the tensile case.

Figure 13 shows an example of a cube under compression (using Dirichlet boundary conditions on the top and bottom faces, red arrows show resulting surface tractions): due to Poisson's effect the cube also bulges out orthogonally to the axis of compression.

Consequently, both a (horizontal) compressive fracture and secondary (vertical) tensile fractures appear in the result, cutting the cube into multiple fragments (only the four largest fragments are shown).

4.5.2 High-resolution crack propagation

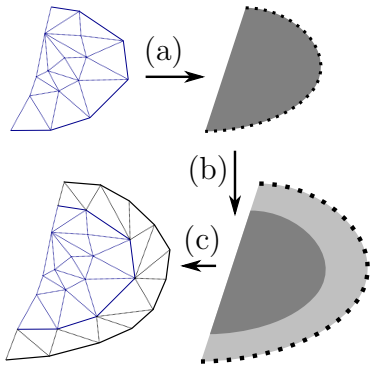


Figure 14: Overview of our crack-propagation method: we sample the crack front at a high resolution (a), propagate each crack-front marker (b), and then update the BEM mesh (c).

Now that we have an equation of motion for the crack front, we are ready to simulate the crack propagation process. The crack front motion, eq. (4.11), is determined by the stress intensity factors and their corresponding local coordinate axes. Our crack propagation method consequently proceeds as follows: we first solve the BEM system for per-node crack opening displacements on a coarse triangle mesh, eq. (4.3), and compute (coarse) stress intensities for each crack-front edge. We then sample each edge by placing a fixed number of crack-front *markers* on it and interpolate the SIFs from the mesh to the markers. Finally, we integrate (4.11) over time, for each marker independently, using a simple forward Euler scheme. Once each marker has propagated for a fixed number of small time steps (which we call "sub steps"), we complete the (full) time step by adding new elements to the BEM mesh representing the newly created fracture surface. Figure 14 illustrates this process. Any crack-front marker that reaches another surface is "deactivated" and not allowed to propagate any further; see ch. 6.2 for details on how we detect these cases.

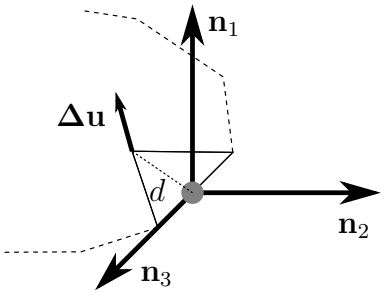


Figure 15: We use crack opening displacement at the interior node of the triangle containing a crack-front edge to evaluate stress intensities.

Evaluating stress intensities

We first evaluate the stress intensity factors for each crack-front edge in the BEM mesh using the displacement correlation technique, eq. (2.39). Note that each crack-front edge is part of exactly one triangle in the mesh. This triangle must also have a third node that does *not* lie on the crack front. (Otherwise all three edges of the triangle would be crack fronts, and the triangle does not contribute a crack opening degree of freedom to the BEM system. Such configurations can never occur due to the way we construct the mesh during both crack initiation and propagation.) Having found this third, interior node within the triangle containing the crack-front edge of interest, we then fetch the crack opening displacement at this node from the solution vector of (4.3). The correlation distance d is then the orthogonal distance from the crack-front edge to the interior node, as illustrated in fig. 15. Of course, the local

crack-front coordinate system is easily defined in terms of the triangle's face normal and the crack-front edge's tangent. Evaluating per-edge stress intensities according to (2.39) is now straightforward. Especially in our case of piecewise-linear (opening) displacement interpolation over the BEM mesh, the displacement correlation technique is not particularly accurate; nevertheless we believe this simple approach to be sufficient for our needs.

Finally, we average both the stress intensities, as well as the local coordinate frames, from the crack-front edges to the nodes, effectively smoothing them. We use an unweighted average regardless of the crack front's geometry: each node receives a 50% contribution from both its adjacent edges. Afterwards, we re-normalize and re-orthogonalize the local coordinate axes at each node.

Motion of crack-front markers
Recall the user-defined resolution parameters r_c (coarse) and r_f (fine): we aim to keep the distance between any two crack-front markers close to r_f and the edge length in the BEM mesh close to r_c. Consequently, we sample each crack-front edge with n equally spaced markers (when the crack first occurs), where $n = \lceil r_c/r_f \rceil$. The piecewise-linear curve connecting these markers (in fixed order) defines the high-resolution geometry of the crack front. We maintain equidistant spacing between the markers by sliding them along the high-resolution crack front after each propagation sub step.

Ensuring equal spacing among the markers allows us to linearly interpolate stress intensities (and local coordinate axes) from an edge in the BEM mesh to the crack-front markers in a geometry-agnostic fashion: let $\mathbf{p}_k^{ij}$, $k = 1..n$, denote the markers along the edge ij (connecting the mesh nodes $\mathbf{x}_i$ and $\mathbf{x}_j$), then the mode-I SIF for each marker is $K_\mathrm{I}(\mathbf{p}_k^{ij}) = (k-1)K_\mathrm{I}(\mathbf{x}_j)/n + (n+1-k)K_\mathrm{I}(\mathbf{x}_i)/n$ (and analogously for the remaining stress intensity factors). We use the same interpolation scheme for the local coordinate axes $\mathbf{n}_1$ (the surface normal) and $\mathbf{n}_3$ (the crack-front tangent), then normalize the resulting vectors, and finally compute $\mathbf{n}_2 = \mathbf{n}_3 \times \mathbf{n}_1$, which implies $\mathbf{n}_1 \cdot \mathbf{n}_2 = 0$. This way, we ensure that $\mathbf{n}_1$ and $\mathbf{n}_2$ (which we use to define the motion of each marker) are always orthogonal. Using this interpolation scheme, along with a piecewise-constant interpolation over time, we can evaluate the crack propagation velocity at every crack-front marker $\mathbf{p}$ using eq. (4.11) and integrate the crack-front motion over time.

Note that the SIFs describe the (singular) stress field near the crack front by splitting it into analytically known singular functions and their magnitude. Consequently, the SIFs vary much less dramatically than the stress field itself. By interpolating coarse SIFs onto (the updated position of) the crack-front markers in every sub step, we capture the rapid change in the (near) stress field at a high resolution as the crack front moves. We do not need to update the BEM result after every sub step, because the far-field stress changes much more slowly. Similarly, we can safely use a low-resolution BEM, because far-field stress gradients are comparatively small. Applying the SIFs at the current marker location in every sub step automatically models the fast shifting of the stress singularity as the crack front advances.

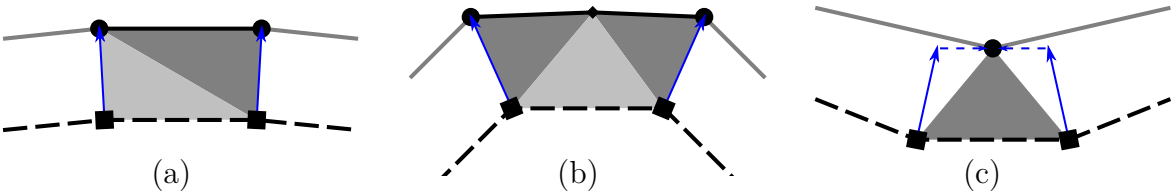


Figure 16: When the crack front propagates, the length of an edge in the BEM mesh either remains within $(0.2r_c,\ 1.4r_c)$ (a), grows too long (b), or too short (c). We subdivide (b) or contract (c) the edge to ensure the new fracture surface elements (grey triangles) fit the user-specified resolution.

We choose the duration of a sub step Δt_s such that $c_R \Delta t_s = r_f$, which means that in every sub step each marker moves no further than r_f. As the markers move, we record the fracture surface generated by this crack-front motion as an implicit surface on a sparse volumetric grid, see also ch. 6.2. Once we have completed n forward Euler sub steps, i.e. the maximal distance any marker may have moved is r_c, we update the BEM mesh by generating new elements for the additional crack surface spanned by the motion of the high-resolution crack front. Again, we aim to create new crack-front edges whose length is approximately r_c, so we place a mesh node at the average position of every n crack-front markers.

If the markers have spread out, or moved closer towards each other, during the crack propagation time step, we apply edge contraction or subdivision operations in order to keep the resulting crack-front edge lengths within the interval $(0.2r_c,\ 1.4r_c)$ (measured with respect to the resulting mesh node positions). Consequently, there are only three different ways how to triangulate the surface spanned by the crack-front motion, as illustrated in figure 16: when two adjacent crack-front nodes (black squares) propagate to new positions (along the blue arrows) their new distance is either (a) still within the allowed range, (b) too long, or (c) too short. In case (a) we add two new triangles (grey) to the BEM mesh, connecting the previous and the new nodes (black circles). In case (b) we double the sampling density along the high-resolution crack front to find the position where we insert a new mesh node, and consequently add three triangles to the mesh. Having increased the number of markers, each of the resulting two new crack-front edges is already sampled by n markers after the subdivision. Finally, in case (c) we replace the crack-front edge by a single node and add only one triangle. We delete the markers along the contracted edge, and re-distribute markers of the adjacent edges, such that the crack front remains uniformly sampled. Note that as a special case, it may happen that only one of the two nodes of an edge propagates and accordingly one of the grey triangles in fig. 16 does not exist (this exception only applies to cases (a) and (b), as we do not allow edge contraction unless both nodes have propagated). In summary, we add new elements to the BEM mesh according to the motion of the high-resolution crack front, while making sure that the size of these elements meets the coarse resolution target. We never modify elements that are already present in the BEM mesh during this process, which allows us to re-use all previously computed blocks of the BEM system matrix in eq. (4.2). After each crack propagation

step, those mesh nodes that represented the previous crack-front state now become new crack opening degrees of freedom, because they are now interior nodes on the fracture surface due to the additional triangles created during the propagation step. Consequently, we extend the BEM system matrix in (4.2) accordingly and compute the corresponding new matrix entries following eq. (2.26).

In most of our examples (see ch. 8), we choose a resolution ratio ($r_f/r_c \approx n$) in the range of 20 to 50 (please also refer to table 1), meaning the resolution of the crack front (and the implicit fracture surfaces) is up to 50 times higher than the resolution of the BEM mesh. Consequently, the number of degrees of freedom required in the deformation model (compared to a uniform surface mesh at full resolution) is reduced by a factor of up to $50^2 = 2.500$. Recall that when using a standard BEM approach, the matrix assembly time scales quadratically in the number of degrees of freedom, meaning that it would take roughly 6 million times longer to assemble the linear system for a full-resolution elastostatic BEM (a similar argument applies to the memory requirement). Clearly, using a standard full-resolution BEM is too expensive and fast approximate methods (such as a fast multipole method) would be required. According to fig. 7 in [79], their fast multipole implementation results in a practical speed-up if the BEM mesh contains more than 20.000 nodes. Using our high-resolution crack propagation method and a *coarse* BEM, we can easily operate below this threshold; most of our examples use less than 5.000 nodes (or roughly 10.000 triangles following Euler's formula), see tables 1 and 3.

Smooth fracture surfaces

During the forward Euler time stepping we always use SIFs from the preceding BEM step. Due to this (temporally) piecewise-constant SIF interpolation, the optimal propagation direction θ^* given by eq. (4.10) would jump in between BEM steps and remain constant during the crack propagation simulation, causing visually unpleasant kinks in the resulting fracture surface. This issue could be solved by using a higher-order time integration, but as re-computing θ^* requires a BEM update, it would be expensive to do so. If we were using a higher order interpolation of SIFs over time (for example a piecewise-linear one), θ^* would become (at least) a continuous function over time, and integrating (4.11) would move each marker along a smooth curve. Consequently, fractures in homogeneous materials form smooth surfaces. Instead of investing computation time in a more accurate time integration scheme, we achieve a visually similar result by enforcing a smooth transition from the previous propagation direction to the new one after each BEM update. To this end, we store each crack-front marker's last propagation direction at the end of a (full) time step and linearly blend to the new direction over the course of the following time step. We define this blended direction $\tilde{\theta}^*$ as

$$\tilde{\theta}^* := w_t \theta^* + \theta^*_{\text{old}}(1 - w_t), \tag{4.12}$$

where θ^* results from eq. (4.10) using the current SIF values, while θ^*_{old} uses the SIF values from the previous BEM step (or is set to zero if the crack has just been initiated). Furthermore, w_t is a weight that linearly transitions from 0 in the first crack

propagation sub step after the BEM update to 1 in the last sub step before the next BEM update.

In summary, our crack propagation method computes an approximate solution to the equations of crack-front motion, producing smooth high-resolution fracture surfaces. The primary input to this method are stress intensities, derived from (coarse) piecewise-linear crack-opening displacements. While we use a quasi-static deformation method, jumping from one elastostatic solution to the next as we add new (coarsely triangulated) fractures, our crack-front sampling and direction blending approach effectively avoid visually distracting artefacts on the resulting (finely resolved) fracture surfaces.

4.5.3 Inhomogeneous materials

While being able to produce smooth fracture surfaces for perfectly homogeneous materials is an important feature of a brittle fracture simulation, most fractures we observe in real-world materials have a distinctive visual quality, such as the photograph shown in fig. 2b; see also the images collected by Becker and Lampman [3], in particular fig. 5 and 41 there. The goal of this chapter is to build a simple model resulting in realistic and visually interesting surface patterns when applied to *inhomogeneous* materials. Note that our underlying BEM elastostatics solver, however, *cannot* handle spatially varying material properties in terms of *elasticity* parameters (i.e. Young's modulus and Poisson's ratio, or equivalently the Lamé parameters) – it is only applicable to homogeneous isotropic linear elastic materials. The main idea to side-step this limitation is to allow spatially varying *fracture* parameters (strength and toughness). In particular, during crack propagation we assume that the presence of a *toughness gradient* at the crack front causes a deviation from its optimal propagation path in favour of moving towards a less tough region nearby. With this approach, we can exploit the fact that these fracture-related material parameters do *not* appear in any of the elastostatic BEM calculations, and consequently we are free to choose spatially varying fields for these parameters. Our crack initiation method, as described in ch. 4.4, already handles spatially varying strength: we add new cracks in decreasing order of how much the principal stress magnitude exceeds the material's local strength at each surface element. Consequently, a spatially varying strength field automatically biases crack initiation towards weaker regions. This effect can also be used as a tool to control the simulation, see for example fig. 18.

During crack propagation, however, the local toughness only affects the propagation speed, eq. (4.8), but not the (optimal) direction (4.10). Intuitively, we expect cracks to propagate towards regions where the material is less tough and therefore the crack can propagate faster, and possibly further, and thereby achieve a higher overall strain energy release compared to trying to break through a tougher region (where it could possibly get stuck due to insufficient stress intensity). We model this behaviour heuristically by modifying the propagation direction using the (spatial) toughness gradient ∇K_c. The first step is to find not only the optimal propagation direction (where the rotated mode-I SIF K_θ is maximal), but also an *interval of valid directions*, within

which the stress intensity exceeds the local toughness, as illustrated in fig. 17. We can then pick the direction out of this interval that best matches the (negative) toughness gradient, leading the crack front towards more fragile regions.

Recall that for a crack-front marker located at $\mathbf{p}$ we have defined local surface and crack-front normals ($\mathbf{n}_1$, $\mathbf{n}_2$) by interpolating the corresponding normals from the BEM mesh. In order to determine the propagation direction in an inhomogeneous material $\hat{\theta}$, we project the toughness gradient ∇K_c (evaluated at $\mathbf{p}$) into the $(\mathbf{n}_1 \times \mathbf{n}_2)$-plane and convert the result to polar coordinates (θ_∇, r_∇) within this plane. In other words, we compute $\theta_\nabla = \tan^{-1}\left[(\nabla K_c \cdot \mathbf{n}_2)/(\nabla K_c \cdot \mathbf{n}_1)\right]$ and $r_\nabla^2 = (\nabla K_c \cdot \mathbf{n}_1)^2 + (\nabla K_c \cdot \mathbf{n}_2)^2$. In our implementation we apply the standard `atan2` function to the negative gradient to make sure that θ_∇ is in the correct quadrant, pointing in the direction of steepest toughness *decrease*.

Obviously, there is no guarantee that the energy release rate in the direction of the toughness gradient, $G(\theta_\nabla)$ according to eq. (4.9), will be sufficient to propagate the crack in this direction. Using again the grouping of mode-I and III SIFs, similar to eq. (4.10), and evaluating eq. (2.35), the effective stress intensity when rotating the crack locally by θ around the crack-front tangent becomes

$$K_\theta = K_{\mathrm{I,III}} \cos^3(\theta/2) - 3K_{\mathrm{II}} \cos^2(\theta/2) \sin(\theta/2). \tag{4.13}$$

Because the optimal direction θ^* in (4.10) maximizes this rotated mode-I SIF and the corresponding rotated mode-II SIF in this direction is $K_r(\theta^*) = 0$, we conservatively define the interval of valid propagation directions solely based on K_θ. More formally, the valid interval (highlighted region in fig. 17) is $(\theta_{\min}, \theta_{\max})$ such that $K_\theta > K_c$ if $\theta_{\min} \leq \theta \leq \theta_{\max}$, where K_c is evaluated at the current marker position $\mathbf{p}$.

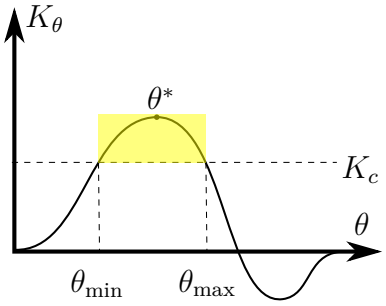


Figure 17: Valid crack propagation directions.

Analogous to the stress intensities (K_{I}, K_{II}, K_{III}), as well as the optimal propagation direction θ^*, this valid interval also changes discontinuously from one BEM step to the next. Following the same approach as our direction blending for homogeneous materials, $\tilde{\theta}^*$ in eq. (4.12), we also interpolate the valid interval linearly over time (during sub-stepping) between the current and previous BEM results: $\tilde{\theta}_{\min} := w_t \theta_{\min} + \theta_{\min}^{\mathrm{old}}(1 - w_t)$ (and analogously for $\tilde{\theta}_{\max}$). This process ensures a continuous transition of the allowed propagation directions over time. We are now ready to compute the valid direction closest to the toughness gradient in each sub step: $\tilde{\theta}_\nabla = \arg\min_{\theta \in I} |\theta - \theta_\nabla|$, where $I := (\tilde{\theta}_{\min}, \tilde{\theta}_{\max})$. In order to avoid visual artefacts from numerical errors when the toughness gradient has either a small magnitude, or is almost parallel to the crack-front tangent, we compute the final propagation direction $\hat{\theta}$ in an inhomogeneous material as a weighted average between the (smoothed) optimal direction $\tilde{\theta}^*$ and the toughness minimizing (valid) direction $\tilde{\theta}_\nabla$:

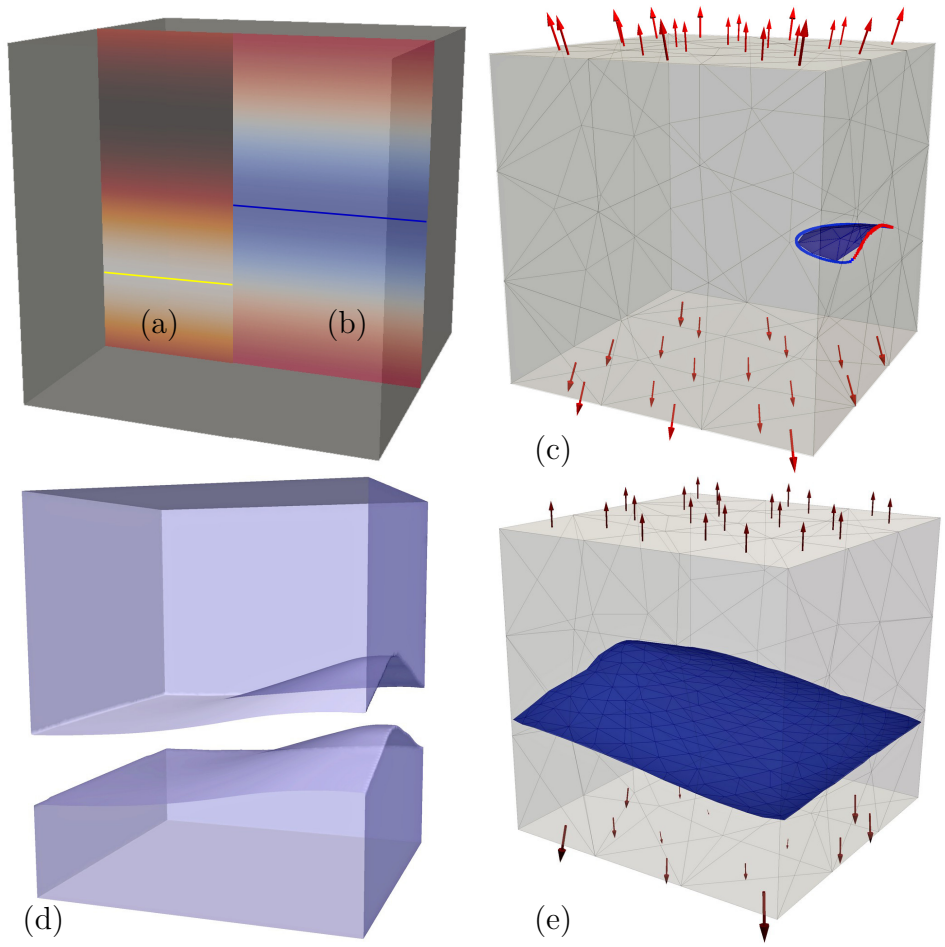


Figure 18: Cube under tension: (a) toughness (yellow line indicates minimum), (b) strength (blue line indicates minimum), BEM mesh (c) after two fracture simulation steps and (e) at the end of the simulation (red arrows show resulting surface tractions). Generated fragments (d) show a smooth fracture surface originating near the strength minimum and propagating towards the toughness minimum (front face cut away).

$$\begin{aligned} \hat{\theta} =& w\tilde{\theta}^* + \tilde{\theta}_\nabla(1-w), \qquad \text{where} \\ w :=& 1/\left(1 + r_\nabla r_c / K_c\right). \end{aligned} \tag{4.14}$$

Note that the magnitude of the projected toughness gradient has units of toughness over length, so the resulting weight w is dimensionless. The derivation of the inhomogeneous crack-propagation direction does not impose any restrictions on how to define a spatially varying toughness field in our implementation, as long as we can query both the toughness value and gradient at any point inside of the breaking object. For most of our results we use the OpenVDB library [11] to implement a tileable toughness field, in particular to model granular materials described in the following section. Nevertheless, even simple toughness fields, defined in closed form, can create visually interesting fracture surface details. For example, the results shown in fig. 19 use a

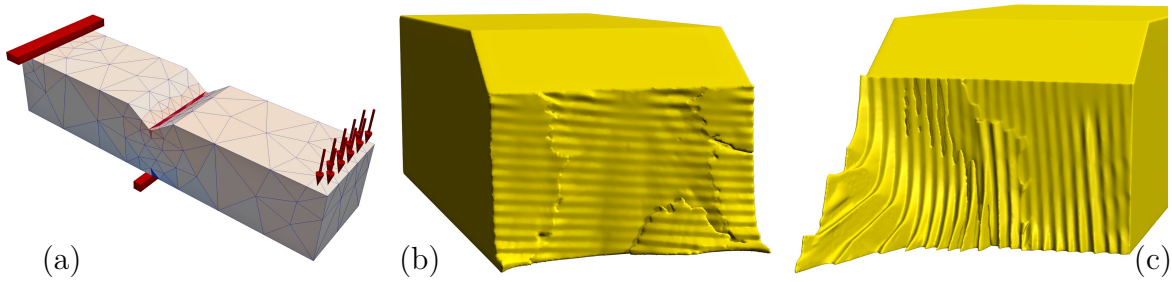


Figure 19: A notched bar in a 3-point-bending test: (a) initial BEM mesh and loading, as well as example results (b, c) using closed form toughness fields. See also fig. 35 for further results.

toughness function of the form $K_c = a - b\cos(f\ \mathbf{p} \cdot \mathbf{l})$, modelling a "layered" material, where the unit vector $\mathbf{l}$ defines the orientation of the layers, a and b specify the toughness magnitude, and the spatial frequency f controls the layer "thickness". The advantage of these simple toughness fields is that gradients can be easily computed analytically.

Granular materials

Another interesting subject are materials with a granular structure. Such grains appear on a micron scale in many metals, but they also appear on larger scales in rock compounds or concrete, for example. We do not make any assumptions on the scale at which we run our simulations (except that continuum mechanics must still be a valid model), as both cases may be of interest and their respective fracture patterns exhibit many similarities. We use the freely available tool DREAM3D [10] to generate a realistic, periodic grain structure. DREAM3D outputs a tileable surface mesh of the grain boundaries. We then compute an unsigned distance function to the grain boundary surfaces. We store this unsigned distance function on a sparse volumetric grid, using OpenVDB [11], see also ch. 6.1 for details. Finally, we map every point in 3D space into this periodic tiling of grains, and assign a toughness value proportional to the distance to the nearest grain boundary. That is, toughness is minimal at grain boundaries and maximal deep inside each grain, such that the material is more likely to fracture near grain boundaries rather than through the grains. Figure 35 shows results for such a granular material model with the same initial geometry and boundary conditions as depicted in fig. 19.

5 Linear-runtime approximations

In chapter 4 we have presented our high-resolution fracture simulation method based on a coarse linear BEM elastostatics solver. Due to the lower resolution of the simulation mesh, the resulting BEM system has significantly fewer degrees of freedom compared to a full-resolution deformation method. However, the matrix blocks corresponding to fracture surfaces, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{Y}$ in eq. (4.2), still grow as the fracture simulation proceeds. In order to achieve good fracture behaviour, we need to update the BEM solution after a fixed number of crack propagation sub steps (and then check for new crack initiation events). This means that the time required to assemble the additional matrix entries and then solve eq. (4.3), as well as the memory required to store the linear system, grows with the fractures.

In particular, the assembly time and memory requirement of a BEM system scale quadratically in the number of DOFs, while solving this dense linear system (with standard factorization methods) scales cubically. Note that in our results we have observed that for a moderate number of DOFs the assembly time dominates the total runtime of the solver. As discussed in ch. 4.5.2, we do not use enough DOFs to merit a fast approximate BEM implementation (which would reduce the assembly time of larger systems). Similarly, because the assembly time is the bottleneck in most of our examples, it would not pay off to use a more elaborate linear solver than the factorization methods provided by *Eigen* [23]. From eq. (4.2) we can easily see that the matrix blocks $\mathbf{B}$ and $\mathbf{C}$ relate crack-opening degrees of freedom to ordinary surface degrees of freedom, whereas $\mathbf{Y}$ encodes direct crack-crack interactions. Consequently, $\mathbf{B}$ and $\mathbf{C}$ only grow linearly with the number of crack-opening DOFs, while the ordinary surface DOFs remain unchanged. However, $\mathbf{Y}$ grows quadratically with the fractures.

The goal of this chapter is to introduce approximations that allow us to avoid updating $\mathbf{Y}$. Instead, we want to work only with the ordinary surface and the crack *fronts* when simulating crack propagation. The advantages of these approximations are a reduced theoretical complexity, as well as a practical speed-up, and a significant reduction in memory consumption. On the other hand, we will lose the ability to treat crack-crack interactions accurately. In particular, the method for initiating branching cracks as described in ch. 4.4 will no longer work. While we could in theory formulate different branching criteria in the context of our approximations, we choose not to do so in this work. The main reason is that the improved complexity of our approximate fracture method allows us to generate such a large number of fractures that the absence of branching (or other crack-crack interactions) is no longer visually noticeable in the results. Because we aim to speed up the computation on the coarse resolution, all calculations in this chapter focus on data stored on the BEM mesh, in particular per-node stress intensities and crack opening displacements near crack fronts. The methods for propagating a crack front, recording the high-resolution fracture surface, and constructing the additional elements for the BEM mesh remain the same as described in the previous chapter.

Furthermore, we formulate our approximations in such a way that we can switch from the *full solution* (as in ch. 4) to the *approximate solution* (described in this chapter) after each time step, even if there are already fractures present in the object. This approach leads to a simple but effective *hybrid method*, where we use the full BEM solution as long as the number of elements in the BEM mesh remains below a user-specified threshold, and we switch to the fast approximations as soon as that threshold is exceeded. Consequently, we can maintain the ability to handle crack-crack interactions and branching as long as there are sufficiently few fractures (where these effects may be perceivable in the result) and seamlessly transition to the faster, less accurate method later on.

Our approximate fracture simulation method consist of two main components: a stress intensity estimator (ch. 5.1), which we use to drive the crack propagation, and a crack opening displacement estimator (ch. 5.2), which contributes an update to the right-hand side of eq. (4.2). We estimate the stress intensity factors on the crack front from the ordinary surface data, and similarly, once we have estimated the new crack opening displacements behind the crack front, we only update the first row (corresponding to the ordinary surface data) in (4.2). In this way, our fast approximations circumnavigate all computations involving the problematic matrix block $\mathbf{Y}$. Conceptually, when using the hybrid method, all fractures present before we switch to the faster method are treated as if they were ordinary surfaces (except that their corresponding right-hand side vector remains $\mathbf{0}$ at all times).

5.1 Stress intensity factors

Having outlined the main idea for our fast approximate fracture simulation, we now focus on estimating stress intensity factors from ordinary surface data. As a starting point serve the analytic results obtained for simple object and fracture geometries, as summarized in [22]. In particular, in 2D plane strain or plane stress problems including a single straight crack, the stress fields that are either symmetric or anti-symmetric about the crack's axis (i.e. pure mode-I or mode-II loaded situations respectively) can be represented by a complex stress function according to Westergaard [75], see also eq. (1.122) and (1.123) in [22].

This approach was then used by Irwin [33] to derive the stress field around a mode-I loaded straight crack, stated in eq. (2.28), which relates the stress in the vicinity of a crack tip to the stress intensity factor K_I. In the context of linear elasticity, one can assume the elastostatic solution to a problem containing a crack under external (far-field) loading to be a superposition of two partial problems, see ch. 4.4.1 in [22]. The first part deals with the external loads, but ignores the presence of the crack, while the second part accounts only for the crack under no external loads. Note that in the final solution, we know that the crack must be traction-free. Consequently, the boundary conditions on the crack faces in the second partial problem must be chosen such as to negate the traction observed at the location of the crack in the first one. Combining

Westergaard's approach with the near-field stress based on Irwin's results, eq. (2.33), and separating near- and far-field loading by superposition produces a relationship between the far-field loading and the stress intensity factors depending on the geometry of both the object and the crack. For further details, see eq. (4.30)–(4.41) in [22].

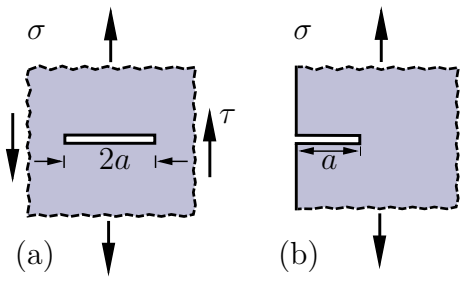


Figure 20: Planar crack problems with analytical stress intensity solutions, adapted from table 4.1 in [22].

Obviously, the geometries (and boundary conditions) for which such closed-form results can be derived are somewhat limited, and most cases assume an infinite extent of the object in question in at least one dimension. Nevertheless, assuming a sufficiently simple geometry, this method allows deriving stress intensity factors from (constant) stress far away from the crack (*far-field stress*), or point forces acting on the crack faces. A summary of various geometries and resulting stress intensities can be found in [22], table 4.1. For our purposes, considering the two basic cases shown in fig. 20 is sufficient. In the first case, an infinite plate under far-field tension σ and shear τ contains a straight centre crack of length $2a$. In the second one, a half-space under far-field tension contains an edge crack of length a. The stress intensities due to these far-field loadings are $K_\mathrm{I} = \sigma\sqrt{\pi a}$, $K_\mathrm{II} = \tau\sqrt{\pi a}$, or $K_\mathrm{I} = 1.1215\ \sigma\sqrt{\pi a}$ respectively.

Due to the infinite extent of the computational domain and the constant far-field conditions, the elastostatic stress ($\nabla \cdot \boldsymbol{\sigma} = 0$) in the absence of a crack is constant throughout the object. This observation motivates the main idea of our SIF estimator: given a point $\mathbf{p}$ on the crack front, we first measure the stress due to the ordinary surface displacements and tractions (ignoring all fractures) according to eq. (2.23). We then conceptually replace the actual geometry of the object with the straight edge crack of fig. 20b, setting the far-field stress to the traction caused by the observed stress at the crack front. Based on the analytical results summarized above, we then formulate the following *basic SIF estimator* in 3D for a point on the crack front:

$$K_i = 1.1215\ \left[\mathbf{n}_i^\mathsf{T} \boldsymbol{\sigma}(\mathbf{p}) \mathbf{n}_1\right] \sqrt[4]{\pi A}, \tag{5.1}$$

where $\boldsymbol{\sigma}(\mathbf{p})$ is the stress at $\mathbf{p}$ evaluated in the absence of any cracks. The local coordinates and stress intensities are defined as usual (see fig. 4), so $\mathbf{n}_1$ is the crack's surface normal (and consequently $\boldsymbol{\sigma}(\mathbf{p})\mathbf{n}_1$ is the traction observed at the location of the crack front with respect to the crack's tangent plane).

In order to avoid numerical issues if the crack front is too close to the ordinary surface (where the integral kernels in eq. (2.23) approach a singularity), we transition to a surface stress evaluation, as described in ch. 4.3, if $\mathbf{p}$ is closer to the ordinary surface than 10% of the closest surface triangle's size. More formally, we use surface stress whenever $\min_{\tilde{\mathbf{x}} \in \Gamma} \|\tilde{\mathbf{x}} - \mathbf{p}\| < 0.1\ l$, where l is the length of the longest edge of the triangle in the BEM mesh closest to $\mathbf{p}$. We then project the traction vector $\boldsymbol{\sigma}(\mathbf{p})\mathbf{n}_1$

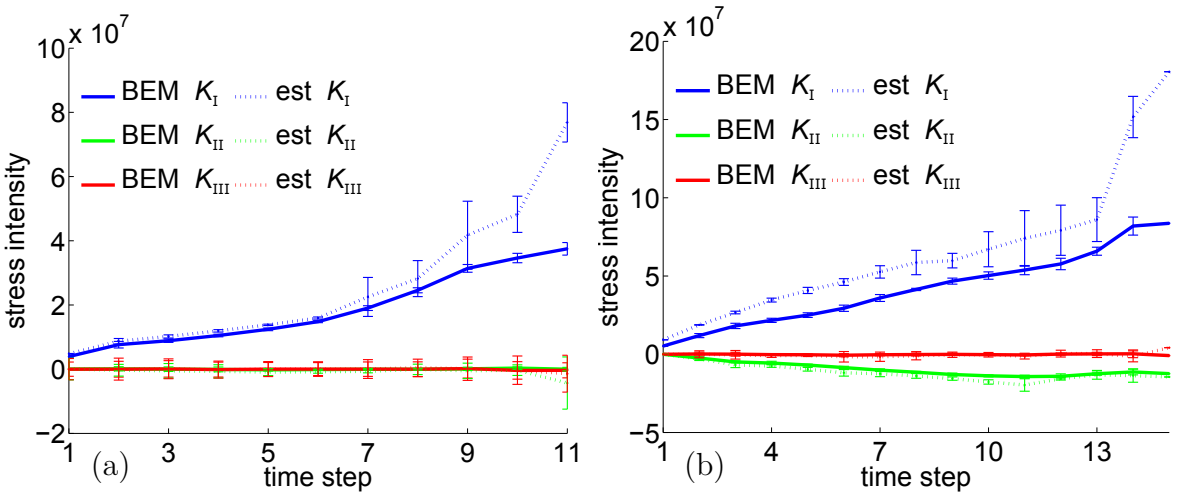
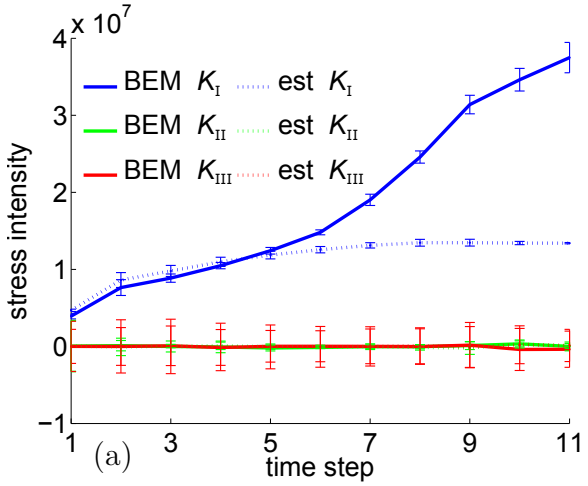


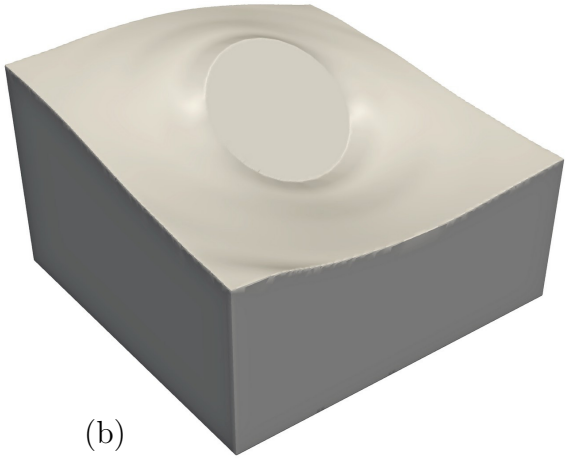
Figure 21: SIF estimates using up-to-date BEM surface data: (a) cube with 45° centre crack (as in fig. 34) and (b) initially unfractured cube (as in fig. 18). Both examples use mixed boundary conditions: fixed displacement on the base and constant traction on the top of the cube.

onto each of the local coordinate axes to find the associated stress intensity factor. Finally, we measure the total surface area A of the crack (in the coarse BEM mesh) and assume a circular shape, such that the effective crack length (cross-section radius) is $a = \sqrt{A/\pi}$. We use the constant factor 1.1215 corresponding to the edge crack case because our crack initiation method always starts cracks at another surface.

We now want to test this basic estimator against the displacement correlation technique we have used so far. For this test, we run the full fracture simulation, solving for the elastostatic deformation including crack opening displacements and computing SIFs as described in ch. 4. After each BEM update, we also approximate SIFs according to eq. (5.1) and compare the results in fig. 21. The estimated SIFs increase more than the COD-based ones as the crack reaches the surface towards the end of the simulation (where the BEM formulation underestimates the COD as discussed in ch. 2.5). In general, this comparison shows that eq. (5.1) can produce fairly accurate SIFs if we use a BEM solution that already accounts for the presence of fractures as an input. However, we do not gain any practical advantage by doing so, as we would still need to solve the entire BEM system including all fractures. In order to build a faster method, we need to estimate the stress intensities based on a solution to the *unfractured* BEM system, eq. (2.25). (If we use our hybrid method, the BEM system does include some fractures, but it will not be updated with newly generated ones anymore once we switch to the fast estimators.)

Unfortunately, simply ignoring (new) fractures and estimating the stress intensities according to eq. (5.1) based on "outdated" surface data causes two serious issues, as shown in fig. 22: first, the SIFs (a) are severely underestimated (even though the increasing crack area is accounted for in the estimator), making the material appear artificially tougher. Furthermore, when running the crack propagation simulation as





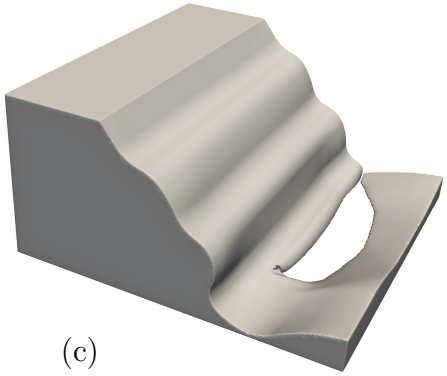


Figure 22: The basic SIF estimator fails to capture the increasing stress intensity as the crack grows (a) and also causes oscillation artefacts on the fracture surface (b, c). Images (a) and (b) show the same test case as fig. 21a and 34; image (c) shows a mode-II loaded edge-cracked cube with fixed displacement on the base and constant traction on the top as in fig. 32II.

in ch. 4.5 the propagation path, and consequently the resulting fracture surface (b), exhibits unacceptable oscillation artefacts. Especially in mode-II dominated cases (c) these oscillations can become quite severe.

We address the issue of underestimated SIFs (a) in ch. 5.2 and focus on avoiding oscillating crack propagation paths (b, c) here. We believe that these oscillation artefacts occur because our assumption of a planar circular crack in an infinitely large object under constant far-field stress never holds in practice. Consequently, even small errors in the ratio of mode-I to mode-II loading result in a crack propagation behaviour that fails to align the fracture surface normal with a principal stress direction. Once the crack has propagated in a slightly wrong direction for one time step, the local coordinate axes at the crack front change accordingly, and the propagation path over-compensates for the error in the next time step.

We aim to rectify this behaviour by choosing the ratio $\eta := K_{\mathrm{II}}/K_{\mathrm{I,III}}$ such that the resulting propagation angle θ^* aligns the crack's surface normal with an eigenvector of the stress $\boldsymbol{\sigma}$. Again, we find $\boldsymbol{\sigma}$ using eq. (2.23) in the interior of the object, or eq. (4.6) on (or very near) the ordinary surface. The propagation angle θ^* in the crack front's local coordinate system is defined in eq. (4.10). Substituting $K_{\mathrm{II}} = \eta K_{\mathrm{I,III}}$ there and solving for the ratio η that produces a desired angle θ^* yields

$$\eta(\theta^*) = \tan(\theta^*/2)/(2\tan^2(\theta^*/2) - 1), \tag{5.2}$$

if $\cos(\theta^*) \geq 1/3$, i.e. θ^* is less than approximately 70°. We do not explicitly handle larger turning angles, which can only occur under compression. Having computed the desired ratio, we set the stress intensities accordingly. Following the idea of eq. (5.1), the combined magnitude of the stress intensities is supposed to be $K_{\mathrm{I,III}}^2 + K_{\mathrm{II}}^2 = (1.1215\lambda\sqrt[4]{\pi A})^2$, where λ is the eigenvalue (principal stress magnitude) corresponding to the eigenvector (principal stress direction) we are trying to align the fracture with. Consequently, we obtain the following estimate for $K_{\mathrm{I,III}}$:

$$K_{\mathrm{I,III}} = s\lambda, \quad s := \sqrt{1/\left(1 + \eta(\theta^*)^2\right)}\, 1.1215\sqrt[4]{\pi A}. \tag{5.3}$$

For the purpose of simulating crack propagation, it would suffice to estimate $K_{\mathrm{I,III}}$ and K_{II} in this way. In the following chapter, however, we are going to need all three SIFs separately. Consequently, we split $K_{\mathrm{I,III}}$ into K_{I} and K_{III} based on the angle α between the chosen principal stress direction and the crack-front tangent ($\mathbf{n}_3$) in the ($\mathbf{n}_1 \times \mathbf{n}_3$)-plane. Recalling that $K_{\mathrm{I,III}}^2 := K_{\mathrm{I}}^2 + K_{\mathrm{III}}^2/(1-\nu)$, we finally arrive at our improved 3D *SIF estimator*:

$$\begin{aligned} K_{\mathrm{I}} &= s\lambda\ \sqrt{1 - |\cos\alpha|}, \\ K_{\mathrm{II}} &= s\lambda\ \eta(\theta^*), \text{and} \\ K_{\mathrm{III}} &= s\lambda\ \operatorname{sign}(-\cos(\alpha))\sqrt{|\cos\alpha|(1-\nu)}. \end{aligned} \tag{5.4}$$

These estimates depend directly on the angles (α, θ) between the local crack front coordinate axes and a principal component of the local stress $\boldsymbol{\sigma}$ (measured in the absence of fractures). The final step then is to choose the appropriate principal direction (eigenvector) of $\boldsymbol{\sigma}$. The fracture process should release as much elastic energy as possible; therefore the default choice is the eigenvector corresponding to the largest eigenvalue. However, this eigenvector might not be sufficiently close to the fracture surface normal to satisfy $\cos(\theta^*) \geq 1/3$. In this case, we choose either the second largest or the most negative eigenvalue, depending on which one has the higher magnitude (while also accounting for the material's ratio of compressive to tensile toughness for negative eigenvalues). If two eigenvectors are not sufficiently aligned with the surface normal, we choose the remaining one. Because $\boldsymbol{\sigma}$ is symmetric, it has orthogonal eigenvectors with real eigenvalues, so at least one of them must be close enough to the surface normal to be a valid choice (as any eigenvector can be multiplied by -1, we have a possible choice every 90°).

The SIF estimator (5.4) aligns the crack propagation angle with an eigenvector of the local stress and successfully avoids oscillation artefacts on the fracture surface. Final results corresponding to the failure cases above (fig. 22b and c) are shown in fig. 34c and 33II respectively. Figure 23 shows a similar mode-II loaded edge-cracked cube using pure Neumann boundary conditions (constant, opposite tractions on the upper

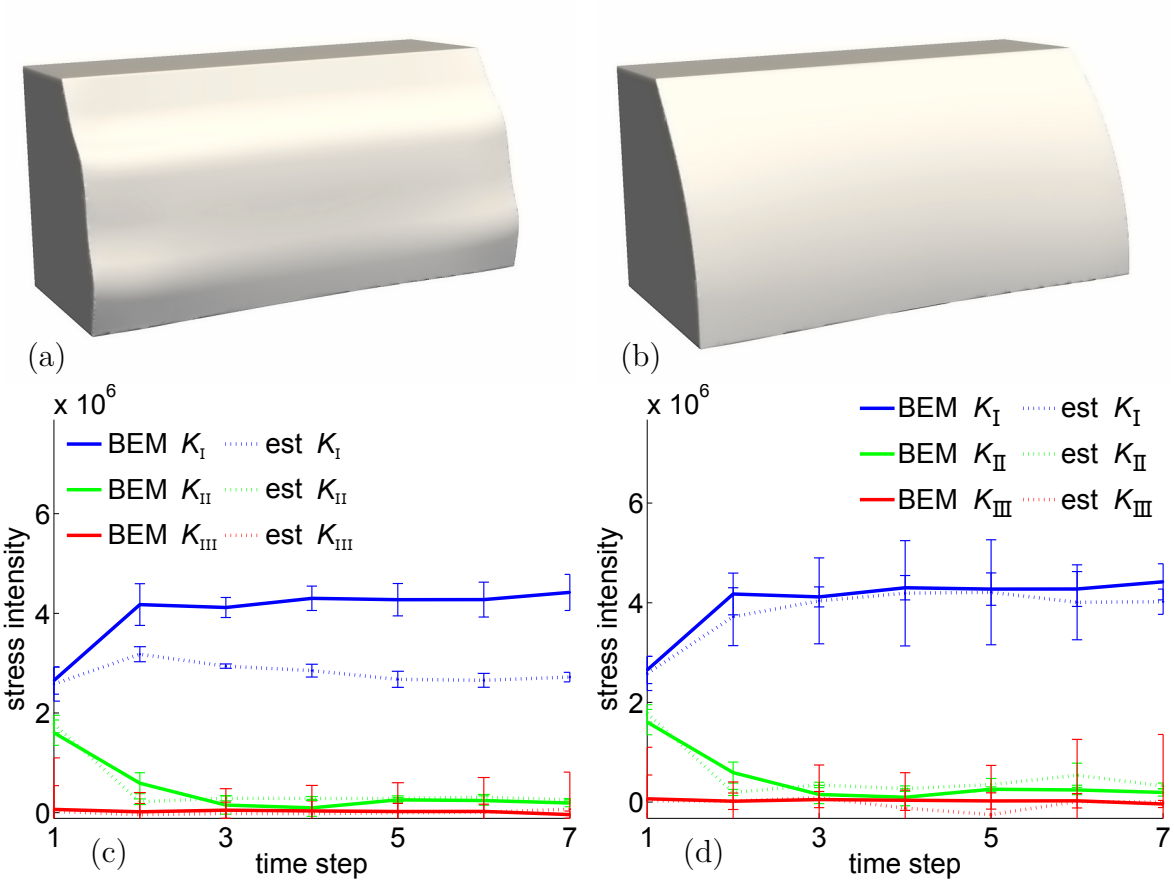


Figure 23: A mode-II loaded edge-cracked cube with pure Neumann boundary conditions: (a) oscillation artefacts caused by the basic SIF estimator, (b) aligning the propagation angle with a principal stress direction removes these problems, but the SIFs (c) are still underestimated. The method presented in ch. 5.2 yields improved SIF estimates (d).

and lower half of the cracked side face). Similar to the other examples, the oscillations caused by the basic estimator (a) are effectively removed (b), but the SIFs (c) are still too low compared to the full BEM solution; we now turn our attention to obtaining improved SIF estimates (d).

5.2 Crack opening displacements

In the previous chapter we have described how to estimate stress intensities based on the object's ordinary surface displacements and tractions. At the beginning of each fracture simulation, we obtain this information by solving a boundary element system. As the object fractures, however, it effectively weakens and deforms more easily overall. In other words, adding crack opening degrees of freedom due to fracturing influences the resulting displacements and tractions on the object's ordinary surface. Consequently, in order to get more accurate SIF estimates, it is necessary to update

the surface displacements after each crack propagation time step. A full BEM solve, eq. (2.38) and (4.2), would give the correct new displacements, but doing so would be too costly. As our main goal is to avoid this procedure, this chapter describes how to approximate the change of the ordinary surface data ($\mathbf{u},\ \mathbf{q}$) due to *estimated* crack opening displacements $\boldsymbol{\Delta\tilde{u}}$.

Given the (estimated) stress intensities K_i at the beginning of any one time step, and the new crack-front position at the end of the same (full) time step, we can estimate the crack-opening displacement $\boldsymbol{\Delta\tilde{u}}$ at the previous crack-front position by inverting the displacement correlation technique, eq. (2.39), resulting in the following components of the COD with respect to the local crack-front coordinates ($\mathbf{n}_1,\ \mathbf{n}_2,\ \mathbf{n}_3$):

$$(\boldsymbol{\Delta\hat{u}})_i = K_i c\sqrt{2d}/(\mu\sqrt{\pi}), \tag{5.5}$$

where μ is the shear modulus, d is the distance the crack has propagated during the time step, and $c = 1$ if $i = 3$ and $c = (1 - \nu)$ otherwise. Transforming back to standard Cartesian coordinates, we find the estimated opening displacements:

$$\boldsymbol{\Delta\tilde{u}} = \textstyle\sum_i \mathbf{n}_i(\boldsymbol{\Delta\hat{u}})_i. \tag{5.6}$$

We can now substitute the estimated new opening displacements for the unknown ones in eq. (4.2) and move them to the right-hand side. Remember that we only aim to update the object's ordinary surface data, so we consider only the first row in eq. (4.2), which now reads $\mathbf{As} + \mathbf{B}\ \boldsymbol{\Delta\tilde{u}} = \mathbf{f}$. Consequently, we update the right-hand side by $\mathbf{f} \leftarrow \mathbf{f} - \mathbf{B}\ \boldsymbol{\Delta\tilde{u}}$ after each BEM step and then compute the new surface data. The system matrix $\mathbf{A}$ itself does *not* change due to these updates. Recall that we invert $\mathbf{A}$ at the beginning of the simulation, and consequently, finding the updated surface data means computing only a single matrix-vector product. The matrix $\mathbf{B}$ describes the influence of (new) estimated crack opening displacements on the surface degrees of freedom; as we only need the result of the product $\mathbf{B}\ \boldsymbol{\Delta\tilde{u}}$, we never explicitly assemble this matrix. Similarly, for each time step we only need to compute $\mathbf{B}\ \boldsymbol{\Delta\tilde{u}}$ for those crack-front nodes (in the BEM mesh) that have actually propagated during this time step (i.e. nodes that were on the crack front at the start of the time step, but are now on the interior of the fracture surface as the crack front has moved on).

As shown in fig. 24, updating the surface data in this manner allows the local stress to increase, yielding good stress intensity estimates for basic test cases. The resulting high-resolution fracture surface for the initially unfractured cube (SIFs shown in fig. 24b) is visually indistinguishable from the full BEM result shown in fig. 18. Please refer to fig. 34 for the resulting fracture surface of the centre-cracked case (fig. 24a).

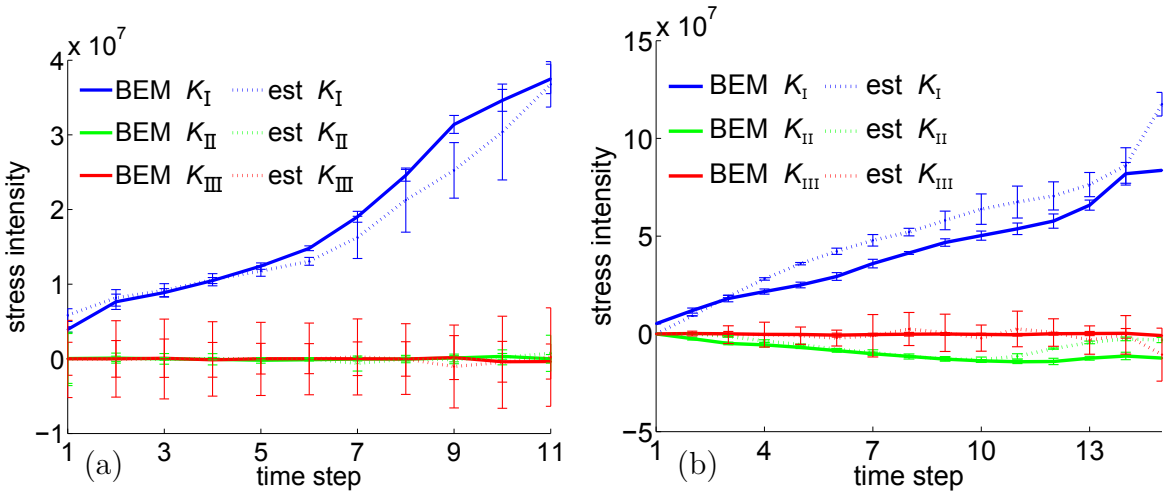


Figure 24: Comparison of stress intensity factors obtained with the full BEM solution (solid lines) and our fast estimator (dashed lines) for (a) the 45° centre-cracked cube (as in fig. 34) and (b) the cube with varying strength and toughness fields (as in fig. 18).

5.3 Scaling and speed up

In order to analyse the computational cost of our approximations, let l be the number of mesh nodes on the crack front. Recall that during crack propagation we ensure that the edge lengths between two adjacent crack front nodes is close to the coarse resolution parameter r_c, so l corresponds to the crack front *length*. In order to perform one crack propagation time step, we first need to compute the local stress at l different locations, which allows us to estimate stress intensities. In the interior of the object each stress evaluation requires integration over the object's ordinary surface, but excluding fracture surfaces. (Surface stress evaluations are faster, but only apply to crack front nodes close to the ordinary surface.) Assuming the number of surface degrees of freedom is n, the runtime required for evaluating all per-node stress intensities in one time step is $O(ln)$. Similarly, the right-hand side update ($\mathbf{B}\,\boldsymbol{\Delta}\tilde{\mathbf{u}}$) is only computed for those crack-front nodes that have actually propagated during the time step. Again, we need to integrate over the ordinary surface (but not over other fracture surfaces), therefore this calculation also takes at most $O(ln)$ time. Recall that the mesh of the object's ordinary surface *never* changes during the fracture simulation, which means n remains *constant* and the cost per time step scales linearly with the *length* of the crack front. As our approximations remove the need to assemble and store the dense matrix $\mathbf{Y}$ altogether, it also reduces the required memory considerably.

Conversely, computing the full BEM solution after every time step scales (at least) quadratically with the current surface area (including fractures). In principle, a fast multipole method could reduce the runtime to (almost) linear in the (fracture) surface *area* per time step. Consequently, our linear scaling in the crack front length represents a significant improvement over previous methods. The resulting practical speed-up compared to our full BEM solution depends strongly on the number of fracture elements in the mesh. For simple cases with less than 1000 elements, such as those shown in

fig. 37, the total runtime is dominated by the high-resolution crack propagation and fracture surface recording, and is therefore almost equal for both the full BEM and the approximate solution. On more complex examples, however, our approximations are considerably faster (see also ch. 8.2): for example, we obtained a speed-up of approximately 35x for the scene shown in fig. 41, using about 3x less memory. Figure 25 shows the CPU time required to compute each crack propagation time step for the first two collisions happening in that scene. For such complex scenes, the two methods do not proceed in exactly the same way, which results in a different amount of elements for each run. Nevertheless, our estimate solution is consistently faster, especially of course for large problems (note the logarithmic scale on the runtime axis in fig. 25). Figure 40c shows a similar result for our hybrid method, where we switch from the full BEM solution to the fast approximate one when the mesh size exceeds 3000 elements.

Having established that the runtime cost for a single time step scales linearly in the crack-front length, it follows that the computational cost of the entire fracture simulation is linear in the generated fracture *surface area*, because the required number of time steps is directly related to the distance the crack front needs to traverse in order to span the fracture surface (assuming a positive average propagation speed). As the cost of producing *any* explicit surface must at least scale with the surface area, we conclude that this crack propagation method provides optimal scaling.

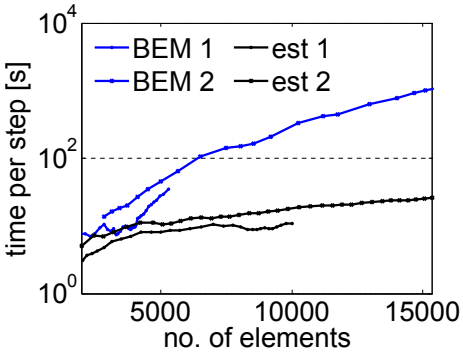


Figure 25: Runtime per full time step: comparison of the BEM solution and fast estimators (for the scene shown in fig. 41).

6 Geometry and topology handling

The goal of this chapter is to capture the high-resolution geometry of the fracture surfaces created by our crack propagation simulation and then analyse the resulting topology of the broken object, i.e. find out where the object has broken into separate fragments. While most of this chapter deals with computations involving high-resolution surface data, we also need to take the requirements of the boundary element method, used for the coarse deformation simulation, into account in this process.

On the coarse resolution level, we use a triangle mesh to describe the surface of a breakable object. As described in earlier chapters, fracture surfaces are not connected to any other surface in this mesh representation. Instead we use one triangle sheet for each fracture, which is bounded by the (coarse) crack front. Fortunately, constructing these meshes during the fracture simulation is fairly straightforward, as described in ch. 4.4 and 4.5. Consequently, the only task left to do on the coarse resolution level is to construct a suitable mesh of the object's ordinary surface from the input geometry at the start of the simulation, as described in ch. 6.1.

One important condition for the BEM to work properly is that there must not be any (self-)intersections in the mesh, otherwise the integrals in eq. (2.26) would contain singularities that could not be treated using standard regularization methods. Consequently, even though we generate each fracture mesh independently, we must ensure that cracks do not intersect one another. Similarly, crack fronts cannot physically propagate through any other surface: if a point on the crack front reaches another surface, the material is broken all the way up to the surface, which means that the stress singularity at that point disappears as the material breaks apart. In order to avoid visual artefacts in the output geometry when fractures meet, we need to handle these cases on the fine resolution level.

We choose an *implicit surface* representation on the fine resolution level, which we introduce in ch. 6.1. The main motivation for this choice (as opposed to an explicit mesh) is to avoid complicated meshing operations on the high-resolution geometry. Chapter 6.2 describes how we use this approach to handle fractures, and how we detect when they reach other surfaces. We then present how we find fragments in the presence of small numerical errors robustly in ch. 6.3. Finally, we describe various possibilities for constructing the output geometry in order to visualize the results of our fracture simulation in ch. 6.4.

Figure 26 shows a 2D illustration of our approach: in the coarse BEM mesh, the fractures are not connected to one another, nor to the object's surface. The high-resolution geometry of the object, as well as all fracture surfaces, are stored as individual implicit surfaces. Fractures are slightly "thickened" (dark grey regions in fig. 26b) in order to simplify the calculations. Stopping crack propagation when a crack front reaches another fracture ensures that they slightly overlap in this representation, but do *not* cut through one another. Similarly, cracks can propagate up to the object's ordinary surface, but not outside of it. Finally, we "cut" the object with all fractures using

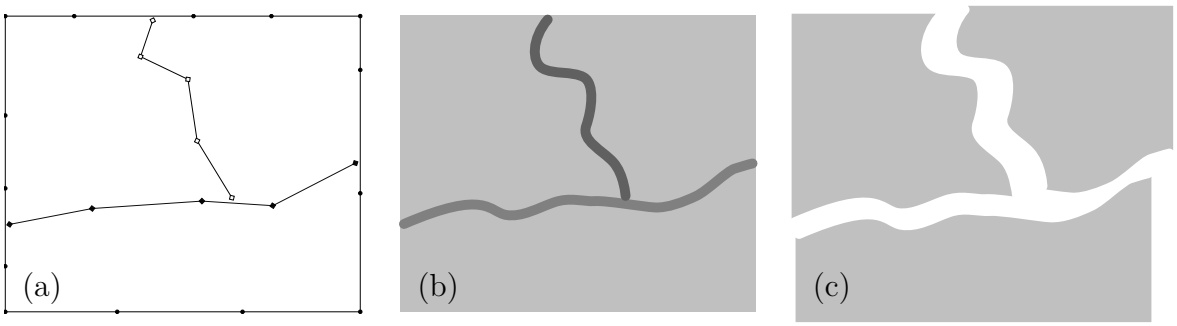


Figure 26: Overview of geometry representations: the coarse BEM mesh (a) and level-set surfaces of individual fractures (b). Cutting the object with all fractures reveals three separate fragments (c).

the set-difference operation according to eq. (6.1), producing a single implicit surface for the entire broken object. We then use this surface representation to separate the resulting fragments, as illustrated in fig. 26c.

6.1 Level-set surfaces and mesh conversion

An implicit representation of a surface Γ is the (zero-)level-set of a scalar function $\Phi : \mathbb{R}^3 \to \mathbb{R}$, i.e. $\Gamma = \{\mathbf{x} : \Phi(\mathbf{x}) = 0\}$. A convenient choice for $\Phi(\mathbf{x})$ is the distance from $\mathbf{x}$ to the point on Γ closest to $\mathbf{x}$, or more formally $\Phi(\mathbf{x}) = \min_{\mathbf{y}\in\Gamma} \|\mathbf{x} - \mathbf{y}\|$. In this case, we call Φ an *unsigned distance function*. If Γ is a closed orientable manifold (such as the surface of an object), meaning that Γ is the boundary of a (positive) volume, we can convert Φ to a *signed distance function* Φ_s (SDF), where we set $\Phi_s(\mathbf{x}) := -\Phi(\mathbf{x})$ if $\mathbf{x}$ is inside of the volume bounded by Γ and $\Phi_s(\mathbf{x}) := \Phi(\mathbf{x})$ otherwise. We drop the subscript for convenience of notation from now on and denote both signed and unsigned distance functions as Φ. Unless stated otherwise, we work with SDFs. One major advantage of SDFs over triangle meshes is that set operations, such as union, intersection, and difference of two volumes Ω_1 and Ω_2, represented by SDFs $\Phi_1 \leq 0$ and $\Phi_2 \leq 0$ respectively, can be computed conveniently as follows, see also [40]:

$$\begin{aligned} (\Omega_1 \cap \Omega_2) &= \{\mathbf{x} : \max[\Phi_1(\mathbf{x}),\ \Phi_2(\mathbf{x})] \leq 0\} \\ (\Omega_1 \cup \Omega_2) &= \{\mathbf{x} : \min[\Phi_1(\mathbf{x}),\ \Phi_2(\mathbf{x})] \leq 0\} \\ (\Omega_1 \backslash \Omega_2) &= \{\mathbf{x} : \max[\Phi_1(\mathbf{x}),\ -\Phi_2(\mathbf{x})] \leq 0\} \end{aligned} \tag{6.1}$$

In order to store an implicit surface, the values of Φ are typically sampled on a regular grid and interpolated (tri-linearly) if required. In our application, the grid spacing, i.e. the distance between two neighbouring sampling points, equals the user-specified (fine) resolution parameter r_f. However, storing a *dense* grid would cause unnecessary memory usage if the surface needs to be represented at a high resolution, while the exact SDF values are not interesting further away from the surface. In particular, we

only want to keep the exact values of $\Phi(\mathbf{x})$ if $|\Phi(\mathbf{x})| \leq h\ r_f$, where h is the *half-width* of a *narrow band* of samples around the surface. Outside of this narrow band, we only need to know the sign of Φ. We use $h = 3$ for all our results.

The OpenVDB library [11, 52] implements an efficient data structure to store such a sparse volumetric distance function. According to Museth [52], the idea of OpenVDB is to "*dynamically arrange blocks (or tiles) of a grid in a hierarchical data structure resembling a B+tree*". Although not required by OpenVDB, we use a "node-centric" view of the grid in this work, which means that the distance function samples are stored at the grid nodes, also referred to as *voxels*. In terms of memory access, voxel values are stored in the leaves of the tree data structure. Additionally, each voxel stores a state (active or inactive). Only voxels in the narrow band around the surface will be active. Outside of the narrow band, voxels are grouped together into small cubes, referred to as *tiles*, and then hierarchically into larger tiles until we reach the root of the tree data structure. For details see also fig. 2 and 3 in [52]. By default the OpenVDB tree structure has four levels with branching factors of $(2^5,\ 2^4,\ 2^3)$ between them respectively, which means that the smallest tiles will contain a cube of $8 \times 8 \times 8$ voxels, while the largest tiles contain a cube of $32 \times 32 \times 32$ intermediate tiles. We use this default configuration for all our results.

Converting between implicit and explicit surface representations
OpenVDB conveniently includes methods for converting a triangle mesh (explicit surface) to a (signed or unsigned) distance function (implicit surface) and vice versa. Converting a triangle mesh to an unsigned distance function is fairly straightforward: for each grid point (within the narrow band) one computes the distance to the closest point on the closest triangle. In order to construct a signed distance function (assuming the mesh bounds a volume and has consistently oriented triangles), the sign of $\Phi(\mathbf{x})$ close to the surface is found, in parallel, by scan conversion [28], basically finding the exterior contour of 2D slices first, and assigning inside values based on ray-intersection tests. The method presented by Houston et al. [28] also handles meshes that are not perfectly closed. Making sure that the signs are available outside of the narrow band requires a flood-filling operation, where the signs are propagated away from the surface in a bottom-up manner [52]. If the input surface is closed and orientable, no inconsistencies can arise during the flood-fill.

Similarly, there exist several methods for extracting a triangle mesh from a SDF, which can be classified as either primal contouring (such as *marching cubes*, see for example [40]) or dual contouring, see [35]. While the standard marching-cubes method places a mesh vertex along each edge connecting neighbouring grid points that is intersected by the surface, dual contouring places a mesh vertex inside of a grid cell intersected by the surface. OpenVDB provides an adaptive dual contouring scheme: the final position of a mesh vertex in a particular grid cell is the average of the marching-cubes locations computed for the edges of that cell. One important consequence of this approach is that in order to reliably build a mesh from the implicit surface stored in an OpenVDB grid, signed distance function values must be available at the corners of all grid cells intersected by the surface. Additionally, OpenVDB allows to adaptively

simplify the resulting mesh in areas of low curvature, reducing the overall amount of output vertices. Note that standard dual contouring schemes technically could produce non-manifold output geometry. OpenVDB, however, implements a more advanced version "that produces topologically robust two-manifold meshes" [12].

Surface meshing
At the start of a fracture simulation, the user typically provides a high-resolution surface mesh of the object. We first convert this mesh to a signed distance function using OpenVDB's conversion tool as described above. Similarly, if we use our rigid-body coupling, every new fragment (see ch. 7.3) has a high-resolution level-set representation (which we compute as described in ch. 6.3). In both cases, we need to construct a coarse triangle mesh from this implicit surface in order to simulate the elastostatic deformation with the boundary element method. For now, we only focus on meshing the object's (or fragment's) ordinary surface. We deal with pre-existing fractures inside of a fragment in ch. 7.3.

We convert the implicit surface to a high-resolution triangle mesh using OpenVDB's built-in dual contouring method described earlier in this chapter. We then apply quadric simplification [18], reducing the number of triangles to a user-specified target (our examples use up to roughly 1000 triangles for this initial mesh). We utilize the implementation available in the VCGlib library [73], which also prevents changing the local topology of the mesh during simplification. Nevertheless, due to this mesh simplification step, self-intersections may occur in the coarsened version. In this case, we detect all intersecting pairs of triangles, delete all triangles in their one-ring neighbourhood, and apply an intersection-free hole-filling step (also provided by VCGlib) to construct a mesh suitable for BEM computations. Finally, note that the BEM solution is only well-defined inside of the BEM mesh. During the simplification step, some parts of the BEM mesh might end up being slightly inside of the high-resolution surface, which may cause fractures in the vicinity to stop propagating before reaching the surface. To mitigate such issues, we allow the user to specify a small distance (typically less than the implicit surface's narrow-band width) by which the BEM mesh is inflated along the average normal direction at each mesh vertex.

When using rigid-body dynamics, all fragments are represented in the rigid-body scene by meshes as well, which are used to perform collision detection. We build these meshes in a similar fashion to the BEM mesh, constructing the high-resolution mesh first and then applying quadric simplification. However, the target number of triangles for these meshes can be specified separately and is typically higher than for the BEM meshes, but still coarser than the high-resolution output geometry. We use 5x as many triangles for collision detection meshes than for BEM meshes in most of our results.

6.2 Implicit fracture surfaces

In principle, a fracture surface could be represented either by an unsigned distance function, or by two signed distance functions: one that defines the surface itself, and a second one that intersects the surface to define the crack front, as done in [20]. We instead opt for a simpler way to represent a fracture with a single SDF as follows: we first compute an unsigned distance function $\Phi_u(\mathbf{x})$, which is 0 exactly at the fracture surface (and positive otherwise). Due to the grid based sampling, reconstructing the 0-level-set without signs would be very challenging in this situation. We could in theory assign signs based on the surface normals of the fracture, but as a fracture surface does not enclose any volume on its own, these signs would not be globally consistent, and therefore would not result in a proper SDF. In particular, combining multiple fractures and cutting the object with them would be difficult. Instead we "thicken" the crack by defining $\Phi_f(\mathbf{x}) := \Phi_u(\mathbf{x}) - \sqrt{3}r_f/2$, such that instead of a single fracture surface we now have a thin volume around the fracture where $\Phi_f < 0$. We can then take the difference of the object and this fracture volume using eq. (6.1), resulting in a small "gap" around the fracture, see also fig. 3. The "thickness" of the fracture ($\sqrt{3}r_f$) equals the length of the space-diagonal of a grid cell, which is the smallest possible choice that ensures that this "fracture volume" completely encloses the given fracture surface (without adding holes due to sampling artefacts). We can also union together two such SDFs, eq. (6.1), to incrementally extend a fracture after each crack propagation step.

Crack front surface intersections
Intuitively, when (a segment of) a crack front reaches any other surface it disappears, in the sense that instead of the two crack faces meeting at the crack front, they now connect to separate parts of the other surface. Contrary to the explicit fracture surface tracking proposed in [79], this change of topology is not represented immediately in our implementation. Instead, we keep the object's geometry unchanged until the fracture simulation is completed, and we store each crack on a separate VDB grid (and as a separate triangle sheet in the BEM mesh). Consequently, whenever a crack-front marker reaches any other surface after a crack propagation sub step, we "deactivate" (rather than delete) this marker such that it can no longer propagate any further. In particular, we distinguish three different cases as illustrated in fig. 27: when a crack-front marker propagates within one sub step from its previous position $\mathbf{p}_0$ to a new position $\mathbf{p}_1$, we need to detect whether the line connecting these two points intersects

(a) the object's (ordinary) surface,
(b) another crack, or
(c) the same crack that the marker belongs to.

In each of these cases, we treat every marker individually and independently of its neighbours, and test against the fracture surfaces that are already stored in the level-set grids (up to the crack-front location of the previous sub step, but not including the current one). In this way, we avoid dealing with degenerate cases or direct line-line collisions. We find that this simplification works sufficiently well in practice, although theoretically some collisions might be missed until the following step.

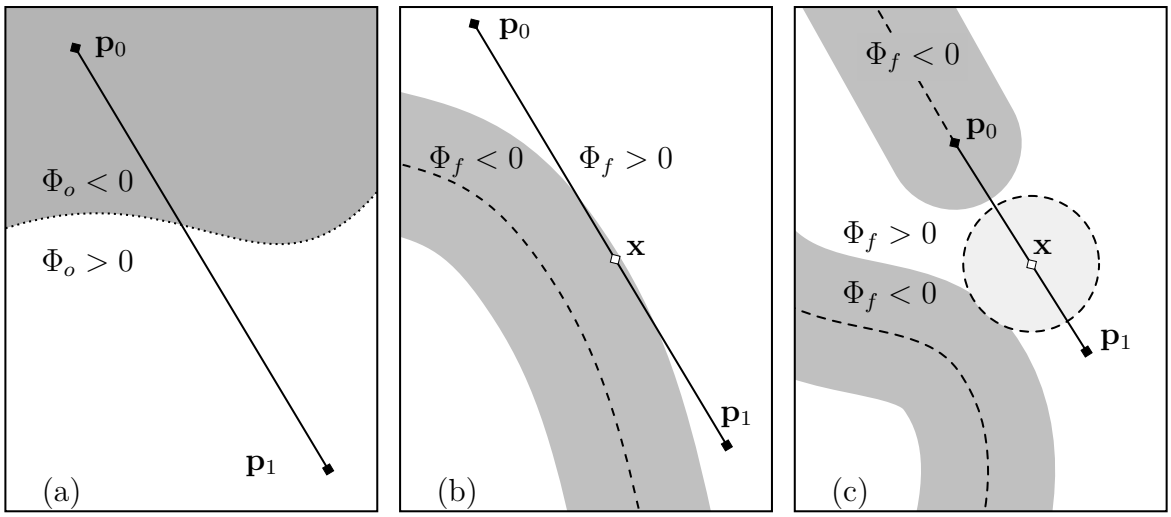


Figure 27: Three cases where a crack-front marker's propagation path in one sub step either leaves the interior of the object (a), intersects another crack (b), or self-intersects its own crack (c).

Detecting when a crack reaches the object's ordinary surface (a) is the simplest of these three cases: clearly, cracks are only allowed to propagate inside of the object, but not outside. Therefore, if Φ_o is the object's SDF representation, any marker where $\Phi_o(\mathbf{p}_1) > 0$ is deactivated. We do not need to access $\Phi_o(\mathbf{p}_0)$ here, although usually $\Phi_o(\mathbf{p}_0) < 0$ holds, because we initiate cracks inside of the object (albeit close to the surface).

Finding an intersection of a crack-front marker's propagation path with another crack (b) is slightly more involved, because the marker may have "grazed" the other crack such that both the start and end point are outside of the other crack, i.e. $\Phi_f(\mathbf{p}_0) > 0$ and $\Phi_f(\mathbf{p}_1) > 0$. (Remember that Φ_f is negative in a small region around the fracture surface and positive everywhere else.) Consequently, in order to reliably find all such intersections, we need to check if there exists a point along the line connecting $\mathbf{p}_0$ and $\mathbf{p}_1$ such that $\Phi_f(\mathbf{x}) < 0$, where $\mathbf{x} = (1-\alpha)\mathbf{p}_0 + \mathbf{p}_1\alpha$ and $\alpha \in (0,\ 1)$. Obviously, we need to check this condition independently for all cracks, except the one that the marker in question belongs to.

Finally, detecting self-intersections (c) is even more complicated. Here Φ_f now denotes the SDF of the same crack as the marker we are investigating. In this case, we must expect $\Phi_f(\mathbf{p}_0) < 0$, meaning that the marker's starting position is already inside of a broken region with respect to the crack's implicit surface representation. In order to detect self-intersections, while avoiding false-positives due to this circumstance, we limit our search to that part of the line connecting $\mathbf{p}_0$ and $\mathbf{p}_1$ where we would expect the value of Φ_f to be positive. If the crack-front marker has moved away from the crack surface, we expect the SDF value at some intermediate position $\mathbf{x}$ to be at least $\Phi_f(\mathbf{x}) \geq \varphi(\mathbf{x})$, where $\varphi(\mathbf{x}) := \Phi_f(\mathbf{p}_0) + \|\mathbf{x} - \mathbf{p}_0\| \sin(45°)$. Allowing for a "misalignment" of up to 45° produces reasonable results in practice. (Note that

Φ_f has a local minimum close to $\mathbf{p}_0$ if the crack state up to the marker's starting position is already represented in the SDF.) Intuitively, a self-intersection occurs if the marker's propagation path contains a point that is *sufficiently far away from* $\mathbf{p}_0$ (in order to avoid false-positives), but still *close enough to the crack* to cause an intersection, and the SDF gradient at this point indicates that the *propagation path moves back towards the crack*, rather than away from it. More formally, we detect a self-intersection if there exists a point $\mathbf{x} = (1-\alpha)\mathbf{p}_0 + \mathbf{p}_1\alpha$ with $\alpha \in (0,\ 1)$ such that

$$\begin{aligned} &\varphi(\mathbf{x}) > 0, \\ &\Phi_f(\mathbf{x}) < \sqrt{3}r_f/2, \text{ and} \\ &(\nabla\Phi_f|_{\mathbf{x}}) \cdot (\mathbf{p}_1 - \mathbf{p}_0) < 0. \end{aligned} \tag{6.2}$$

6.3 Finding fragments

Once the fracture simulation terminates (either due to reaching the time-step limit, or because no new fractures were created in a time step), we want to split the broken object into a collection of fragments. We first build a signed distance function representing the broken object by taking the set-difference of the object's SDF and (the set-union of) all fractures. OpenVDB already implements all the required set operations according to eq. (6.1). We then find the resulting fragments and store each one in a separate grid. Each fragment is a connected volume of the original object, again bounded by a manifold surface. We detect individual fragments using a connected component search on the subset of negative (narrow-band) voxels in the grid representation of the broken object. This procedure follows a standard breath-first-search algorithm, which is already provided in OpenVDB. However, due to the sparse grid data structure, as well as possible numerical errors during the fracture simulation, we implement two optional modifications, which are described in the remainder of this chapter: (a) we can also include tiles in the connected component search, and (b) we can remove small "spindles" connecting two otherwise separate regions.

Please note that we perform this fragmentation step on the high-resolution data, *not* on the coarse mesh used for computing the elastostatic deformation. Consequently, our method not only produces visual detail beyond the deformation resolution, but also the topology of the broken object is not limited by the coarse resolution. In particular, this allows our fracture simulation to produce debris much smaller than the BEM element size. See for example fig. 41. Once we have separated the fragments, we can then easily extract a highly detailed mesh for rendering or post-fracture animation. Furthermore, we couple our quasi-static fracture simulation with rigid-body dynamics (see ch. 7) such that objects can fracture due to collisions; their fragments then become new breakable objects and can in turn collide and break again.

Breath-first-search including tiles
In this section, we present our extension to the breath-first-search algorithm, such that the tiles of a sparse OpenVDB grid are also included. In the following discussion, the input SDF is the implicit surface representation of the object after the fracture simulation (already cut by all cracks), and the output is a collection of separate SDFs, each representing one fragment (i.e. one connected component of the input surface).

The standard breath-first-search segmentation algorithm proceeds as follows: we first deactivate all voxels in the input grid where the stored SDF value is positive. As long as there are active voxels remaining, we initialize a queue with one random active voxel and a new empty grid to store the next fragment in. While the queue is not empty, we remove the first voxel, copy it to the fragment's output grid and then deactivate it in the input grid. Finally, we add all neighbouring voxels that are still active in the input grid to the back of the queue. Recall that in order to accurately recover the surface, we need SDF values at *all* corners of each cell that is intersected by the surface, which means that if we find a neighbouring voxel with a positive value, we also copy this voxel's value to the fragment's grid, but we do not add this voxel to the queue. We ensure that all outside corner values are copied by always checking all 26 neighbours of a voxel (including diagonal neighbours, also referred to as *Moore neighbourhood*).

Our crack initiation method always places a new crack close to another surface. Therefore the standard algorithm works well in this case, because the narrow band of active voxels around each fracture overlaps with the narrow band of another surface in a small region where the fracture started. However, the user does have the option to start the simulation with pre-defined fractures that could be deep inside of the object such that their narrow band does not overlap the narrow band of any other surface. In this case, the standard segmentation would remove such interior cracks. One approach to avoid this problem would be to activate all voxels in the interior of the object first, but that would waste a lot of memory and runtime by effectively negating all the advantages of the sparse grid data structure. Instead, we extend the algorithm to also allow entire tiles to be added to the queue and process them in a similar way during the neighbourhood search: the only issue is how to define which neighbours of a tile to add to the queue. If one side face of the tile is adjacent to multiple smaller tiles, or single voxels, it could be necessary to add all those neighbours to the queue.

However, if we assume a graded adaptive grid structure (as provided by OpenVDB), starting with a narrow band of active voxels around the surface, then a region of small tiles just outside of the narrow band, and finally larger tiles far away from the surface, the situation is more manageable. In this case, it is sufficient to add only 6 neighbours of a tile to the queue: one for each face of the tile. In other words, if there is at most one tree-level between any two neighbouring tiles, all (smaller) tiles (or voxels) that are adjacent to the same face of the larger tile must be the same size. Consequently, once we process one of these smaller neighbours, we are guaranteed to also find the remaining ones in the next step. This way, we slowly work our way down from the large tiles to smaller ones and finally back to the voxel level.

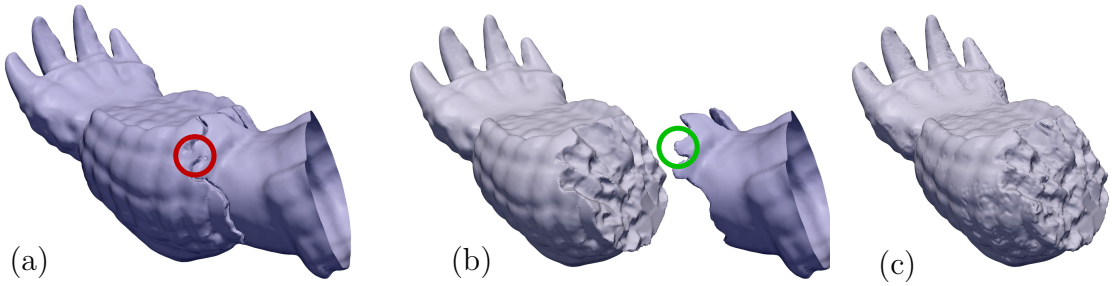


Figure 28: Breaking the armadillo's claw: a small spindle (a) prevents proper fragmentation, our over-segmentation method with $s = r_f$ removes the spindle (b), but preserves visual details of the fracture surface. Setting $s = 1.5r_f$, however, results in objectionable artefacts (c).

Detail preserving over-segmentation

The segmentation algorithm described so far is fast and reliable if the input data is accurate. Numerical errors can occur, however, because the crack front is assumed to be a polyline connecting markers that are propagated independently. Consequently, the level-set representation of the final state after the fracture simulation may contain some small erroneous gaps between fractures that should have closed up. Once we cut the object by these fractures (that do not fit together properly), a small piece of unbroken material remains, which we call a "spindle" (i.e. a small region where the broken object's SDF should be positive, but is actually negative). These spindles may end up connecting two otherwise separate fragments. Clearly, such thin connections would easily break in real objects, therefore the goal of this section is to remove these spindles (as long as they are small enough), while preserving as much of the visual detail of the fracture surfaces as possible.

The basic idea is to over-segment the input SDF: instead of running the breath-first-search algorithm on the input directly, we first add a user-defined (small, positive) threshold s to the entire input SDF (effectively "shrinking" the object and widening the cracks), then run the segmentation algorithm on this modified SDF as above, and finally subtract the threshold value from the resulting fragment SDFs to recover the original value at each voxel. Doing so, however, means that voxels that are very close to the surface are now skipped by the segmentation routine, resulting in a loss of visual detail on the surface (even if we were extrapolating the missing SDF values outwards).

Recall that, in order to recover the surface of the object, we require correct values at all corners of each cell intersected by the surface (including positive values outside of the object). If we stop the segmentation algorithm earlier, not all of the required values will be available. Consequently, we need to copy those values from the input grid to the fragment's grid afterwards. In doing so, we must avoid introducing artefacts that could occur if we, by accident, copied parts of the adjacent fragment as well. (Remember that we represent fractures by a *thin* region where the broken object's SDF is positive.) For each fragment, we copy only those voxel values that hold a positive value in the input grid and are located within the d-ring neighbourhood of a voxel that has been

copied to the fragment by the (over-)segmentation algorithm. The distance d is $1+s/r_f$ rounded to the nearest integer. Unfortunately, this procedure only works as intended if the user-specified threshold for over-segmentation satisfies $0 \leq s \leq r_f$, i.e. we can only remove spindles that are smaller than the grid spacing, otherwise we lose surface detail in the process. Of course we are always bound to lose some information around the spindles, and possibly around sharp features as well. Nevertheless, keeping the over-segmentation threshold within the grid spacing preserves almost all the surface detail, as shown in fig. 28. By default we use $s = r_f$ for our results. A summary of our final segmentation routine is shown in pseudo-code form in Appendix B, listing 1.

6.4 Visualizing results

While we store the high-resolution geometry as an implicit surface (for each fragment), we typically extract a triangle mesh (explicit surface representation) using OpenVDB's dual contouring scheme in order to visualize our results. Depending on the application, the user may choose to modify the final output in various ways, which we summarize in this chapter. In particular, we can either output the undeformed (material space) geometry, or add the interpolated elastic deformation from the BEM solution to the high-resolution output. Of course, this can also be done after every BEM time step to show intermediate states during the fracture simulation, or even after every crack-propagation sub step by interpolating the deformation in between time steps. Finally, our implicit surface representation artificially thickens fractures in order to construct a reliable signed distance function. While doing so facilitates cutting the object with cracks, intersection testing, and fragment detection, as described earlier, it does leave visually unpleasant gaps between the two faces of a crack. Consequently, we present a simple approach to remove these gaps from the final output.

Recall that the BEM solution, eq. (2.38), splits the deformation of the object into two parts: a continuous (piecewise-linear) displacement field over the ordinary surface and a displacement discontinuity across fracture surfaces (the crack opening displacement). In particular, the overall displacement of fractures is not directly available. Instead we use the representation formula, eq. (2.20), evaluating the continuous displacement field in the interior of the object, to find the crack's average displacement $\mathbf{u}_c$. We evaluate this average displacement at every node of the crack's (coarse) mesh by integrating (2.20) over the object's ordinary surface only (excluding any other cracks). We can then add the crack opening displacement $\boldsymbol{\Delta}\mathbf{u}$ to find the displacement of both the positive and the negative face of the crack as $\mathbf{u}^{\pm} = \mathbf{u}_c \pm \boldsymbol{\Delta}\mathbf{u}/2$.

Elastic deformation
While it is straightforward to build a detailed mesh in undeformed material space from the implicit surface, the elastic deformation is only available on the coarse BEM mesh. In order to deform the high-resolution output mesh accordingly, we interpolate the coarse displacements onto the vertices of the fine mesh. The first step to performing this interpolation is to find the closest point $\mathbf{y}$ on the coarse mesh for every vertex $\mathbf{x}$

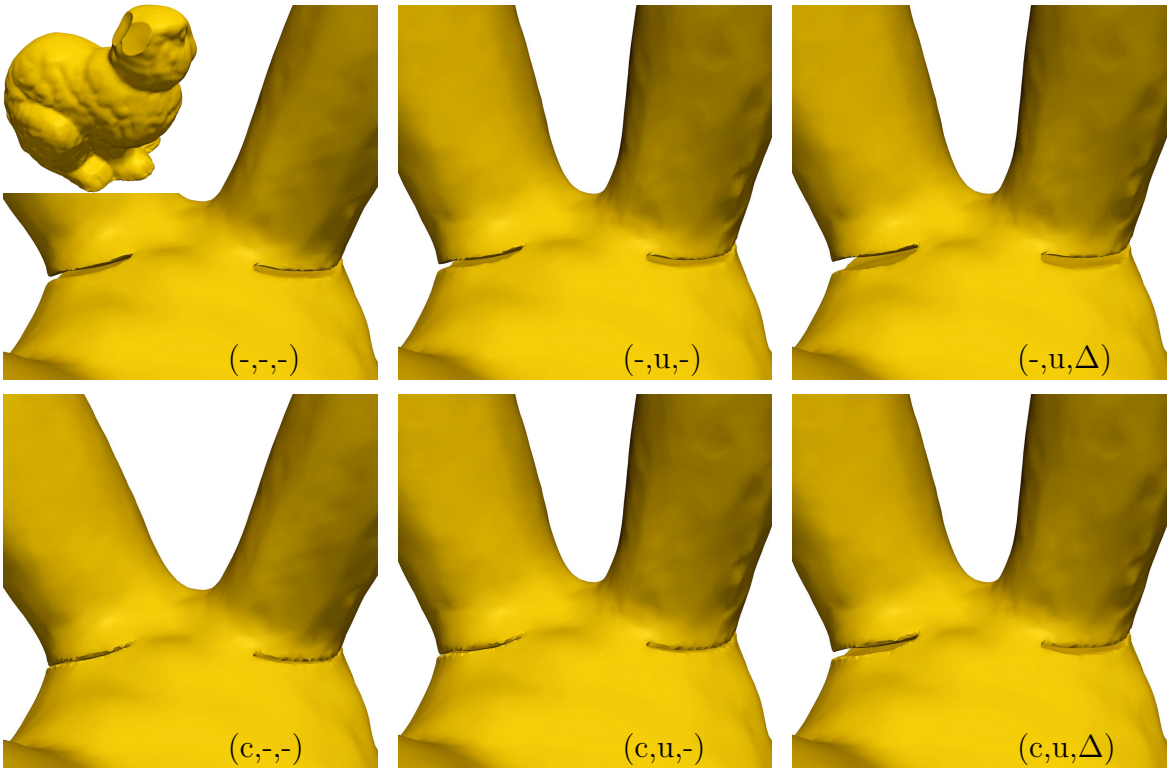


Figure 29: Breaking the ears off the Stanford bunny. Overview of various possible output modes: "c" ... close gaps due to level-set thickening, "u" ... add continuous displacement, "Δ" ... add crack opening displacement.

of the fine mesh. In order to accelerate these closest point queries, we use a coarse OpenVDB grid (with grid spacing r_c, i.e. roughly the same resolution as the BEM mesh): each grid cell stores a list of (coarse mesh) triangles intersecting that cell. For any vertex $\mathbf{x}$ within a given grid cell, we then only need to test the triangles listed in that cell, or in the cell's neighbours, to find the closest point $\mathbf{y}$ on the coarse mesh.

For stiff but weak materials that fail before undergoing large deformations, the crack opening displacements are typically small. In these situations, it is sufficient to visualize only the continuous deformation of the object, ignoring crack opening displacements. To do so, we distinguish two cases in order to find the interpolated displacement at each vertex of the output mesh $\mathbf{x}$: either the closest point $\mathbf{y}$ lies on an ordinary surface element, or on a fracture element. In the first case, we evaluate the (piecewise-linear) surface displacement field at $\mathbf{y}$ and move the vertex by this displacement: $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{u}(\mathbf{y})$.

In the second case, we first evaluate the continuous displacement field $\mathbf{u}_c$ at the coarse fracture mesh nodes, then interpolate it piecewise-linearly within the coarse element containing $\mathbf{y}$, and finally offset the vertex by $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{u}_c(\mathbf{y})$. This procedure is much faster than having to evaluate a surface integral, eq. (2.20), for every vertex in the fine output mesh.

Crack opening displacements
So far, we have added the continuous displacement field to the high-resolution output mesh, either directly from an ordinary surface element, or via the average displacement of a fracture element. Of course, for softer (or tougher) materials, the crack opening displacement will be visually interesting. In these cases, we also want to add this discontinuous deformation to the output geometry. In order to do so for each vertex of the output mesh $\mathbf{x}$, we first find the closest point on a *fracture* element $\mathbf{y}_f$ (which might not be the closest point on the coarse mesh overall), using again our acceleration grid structure. If $\mathbf{y}_f$ is closer to $\mathbf{x}$ than the narrow-band half-width of the high-resolution VDB grid, i.e. if $\left\|\mathbf{y}_f - \mathbf{x}\right\| \leq 3\,r_f$ in our implementation, we evaluate the crack opening displacement $\boldsymbol{\Delta}\mathbf{u}(\mathbf{y}_f)$ at this point, otherwise we do not proceed. As the COD specifies only the difference in the displacement field between the two faces of the crack, we first need to determine whether $\mathbf{x}$ lies on the positive or the negative face of the crack. Because we have already thickened all fractures in the implicit surface representation, the generated high-resolution mesh vertices on each crack face will be slightly offset from the *original fracture surface*, generated by the motion of the high-resolution crack front. This means that we can decide the sign of the COD based on whether this offset puts the vertex on the positive or the negative side of the original fracture surface. Of course, this test has to be performed on the high-resolution surface data. (To this end we also store the high-resolution surface normals in an auxiliary OpenVDB grid that has the same active voxel structure as the fracture surface's SDF grid.)

When interpolating the continuous deformation, as discussed earlier, we only distinguish whether the closest point on the coarse mesh $\mathbf{y}$ of any output mesh vertex $\mathbf{x}$ lies on the ordinary surface or on a fracture surface. Here, we need to consider three cases instead: if $\mathbf{y}$ lies on a fracture surface, and consequently $\mathbf{y} = \mathbf{y}_f$, we can offset the vertex by the displacement of this fracture surface (having already determined the correct sign of the COD): $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{u}_c(\mathbf{y}) \pm \boldsymbol{\Delta}\mathbf{u}(\mathbf{y})/2$. Conversely, if $\mathbf{y}$ lies on the ordinary surface, and outside of the narrow-band of the nearest fracture, we do not add any crack opening displacement. Again, this approximation is much faster than carefully evaluating boundary integrals for all output vertices. The third case, however, are vertices that are close to both the ordinary surface and a fracture. In this case, where $\mathbf{y} \neq \mathbf{y}_f$, we need to combine the continuous deformation of the ordinary surface with the crack opening displacement of the fracture. In order to create a visually pleasing, smooth transition between the two, we blend the COD with a weight w_Δ that is 1/2 at the fracture surface, 0 outside of the crack grid's narrow band, and linear within the narrow band. Finally, the high-resolution vertex is offset by $\mathbf{x}_{\text{new}} = \mathbf{x} + \mathbf{u}(\mathbf{y}) \pm \boldsymbol{\Delta}\mathbf{u}(\mathbf{y}_f)\, w_\Delta$. In this discussion we have focused on the case where $\mathbf{x}$ lies within the narrow band of *one* fracture surface. In general some vertices will lie close to intersections (and consequently within the narrow band) of more than one crack. In this situation we analogously add opening displacement offsets for each nearby fracture.

In the case of *compressive fracture* the crack opening displacement is oriented along the (outward) surface normal, which means offsetting the output mesh by this COD

could create self-intersecting geometry, which leads to unpleasant visual artefacts in many standard rendering methods. Note that this situation is not a problem for the fracture simulation, as the material's different response to compression, as opposed to tension, due to self-collision is treated phenomenologically via different (typically higher) strength and toughness values for compressive fracture. Also note that the deformation is assumed to be infinitesimally small in the linear elastic model. However, in order to produce a visually appealing output, we do provide the option to remove any compressive components from the opening displacement before we proceed with deforming the output mesh. In particular, we update the COD at each coarse mesh vertex with $\boldsymbol{\Delta u} \leftarrow \boldsymbol{\Delta u} - (\boldsymbol{\Delta u} \cdot \mathbf{n})\mathbf{n}$ if $\boldsymbol{\Delta u} \cdot \mathbf{n} > 0$, where $\mathbf{n}$ is the outward surface unit normal.

Closing gaps
The user may choose to close the gaps introduced by the level-set representation of fracture surfaces. In order to do this, we first take every vertex of the undeformed high-resolution mesh $\mathbf{x}$ and compute its closest point on the (also undeformed, high-resolution) original fracture surface $\mathbf{x}_c$. If we want to have an undeformed mesh, but close the gaps at fractures, we can move the vertex to $\mathbf{x}_c$. However, if we want to also apply some deformation, we first add the interpolated displacement to find $\mathbf{x}_{\text{new}}$ before closing the gap by moving the vertex to $\mathbf{x}_{\text{closed}} = \mathbf{x}_{\text{new}} + (\mathbf{x}_c - \mathbf{x})$. (Remember that we need to distinguish the positive and negative side of the crack when adding the COD, so the small gap is actually useful there, and closing it earlier would introduce more problems.) Of course this additional step only applies to output mesh vertices $\mathbf{x}$ whose closest point $\mathbf{y}$ lies on a fracture surface in the coarse mesh.

Summary
In summary, we first perform an interior evaluation of the continuous part of the displacement field ($\mathbf{u}_c$) at all fracture surface nodes in the BEM mesh, determining the average displacement of a crack on the coarse scale. For each vertex in the (fine) output mesh, we then find the closest triangle in the coarse mesh and perform linear interpolation of the continuous displacement field within that triangle. If the vertex is in the vicinity of a fracture, we first decide whether it lies on the positive or negative face of the crack, based on the high-resolution surface normals. We then interpolate the COD and offset the vertex to its deformed position. We allow the user to choose whether the artificial gaps due to the implicit surface representation of fractures are removed from the output geometry, and which components (if any) of the deformation are added to the output. Figure 29 shows an overview of different possible output configurations, while the implementation is detailed in pseudo-code form in Appendix B, listing 2.

Note that our implementation of the method outlined in this chapter is not perfect: in particular, numerical errors can sometimes result in choosing the wrong sign for the COD. If the COD is large compared to the narrow-band width of the OpenVDB grid, it can lead to visual artefacts on the high-resolution output mesh. Our approach of linearly fading out the COD away from the crack helps to avoid some of these issues. In order to produce visually appealing results, we clean up the deformed mesh in post-processing by applying slight mesh simplification and smoothing in some cases.

7 Coupling to rigid body dynamics

While deriving our fracture simulation method, ch. 4, as well as our fast approximations, ch. 5, we have always considered only one object breaking up into any number of fragments. This approach is useful for animations that focus on the details of the fracture process as we can specify the boundary conditions directly. Animating more complex scenes including many (possibly fracturing) objects, however, is somewhat cumbersome with this approach, where we would need to manually derive the appropriate boundary conditions for each object that we want to fracture. Consequently, the goal of this chapter is to couple our fracture simulation method to a standard impulse-based rigid-body system (*Bullet* [8]).

While modern rigid-body engines handle a great variety of interactions between the objects in a scene, such as joints, motors, springs, and generic constraints [4], we work only with the most basic features: Newtonian motion under gravity and collisions. We augment a standard rigid-body scene by marking any number of objects in the scene as *breakable* and specify all the parameters required by our fracture simulation per object. A rigid-body model resolves collisions by transferring impulses (in units of momentum, i.e. force integrated over time) between the colliding objects. In order to construct the boundary conditions for our fracture simulation, we convert these impulses to forces and apply them as piecewise-constant traction boundary conditions, see ch. 7.2.

We run a new instance of our fracture simulation method for each breakable object that experiences a collision exceeding either an impulse or a force threshold. Once a fracture simulation is completed, we replace the object that just broke by all its resulting fragments; each fragment becomes a new, independent rigid body from this point on, see ch. 7.3. As each of these fracture simulations runs independently, we limit our discussion to one single instance from now on. Note that the considerations presented in this chapter apply equally to both the full boundary element fracture simulation, ch. 4, as well as our fast approximations, ch. 5.

As input to our fracture simulation, we now consider an object and a set of collision points with impulses applied along the direction of approach. We then transition from the rigid to the elastostatic model and convert the collision impulses to surface trac-

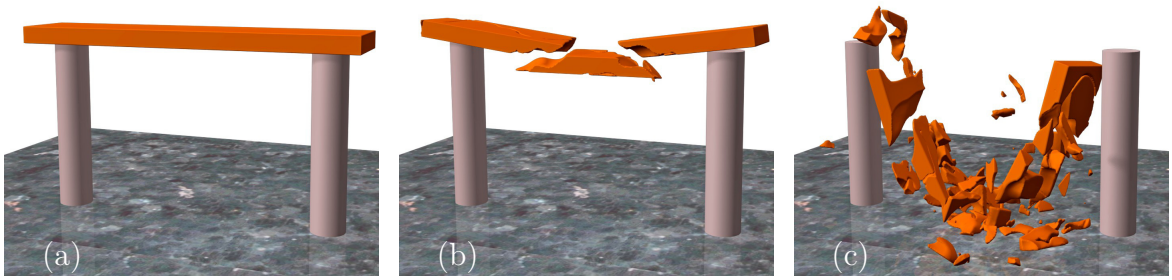


Figure 30: “Breaking II”: using a weak material, the top bar crumbles under its own weight. Note that gravity is handled by the rigid-body system and we do not need to explicitly add it to the BEM formulation.

tions. We are then left with an elastostatic problem that has pure Neumann boundary conditions; in this case the solution is only defined up to arbitrary translation and (linearized) rotation, and the resulting linear BEM system contains a null-space. We first describe how to regularize the BEM system in ch. 7.1 and defer the details on how we construct these boundary conditions to ch. 7.2. The boundary traction field generally contains both rigid and deformational components, but only the deformational part can be handled in the elastostatic model. Consequently, the surface traction field must be free of global translation and linearized rotation, otherwise the regularization we apply to the BEM system would counteract these components and lead to undesirable results. We refer to a traction field that fulfils these requirements as "*balanced*".

In summary, the main components needed to couple our fracture method to a rigid-body system are:

(a) regularizing a pure Neumann elastostatic boundary value problem,
(b) constructing a balanced surface traction field from collision impulses, yielding the right-hand-side vector $\mathbf{f}_N$ of eq. (7.4), and
(c) creating a new rigid body for each resulting fragment (including its initial position and velocity).

7.1 Regularizing the Neumann problem

In the absence of Dirichlet boundary conditions, the elastostatic SGBEM system, eq. (2.25), reduces to $\mathbf{D}\mathbf{u} = \mathbf{f}_N$, which is now rank-deficient as any (linearized) rigid motion is in the null-space of $\mathbf{D}$ (up to numerical errors). Once we add fracture surfaces (contributing unknown crack-opening degrees of freedom), following the same strategy as in eq. (2.38), we get

$$\begin{pmatrix} \mathbf{D} & \mathbf{X} \\ \mathbf{X}^\mathsf{T} & \mathbf{Y} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\Delta}\mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_N \\ \mathbf{0} \end{pmatrix}. \tag{7.1}$$

In order to reliably solve this underdetermined linear elastostatic system we construct a *Tikhonov regularizer*, which penalizes global translation and linearized rotation. This regularizer consists of two components: $\mathbf{T} := (\mathbf{T}_d^\mathsf{T}\ \mathbf{T}_r^\mathsf{T})^\mathsf{T}$, where each part is a $3 \times 3n$ block, where n is the number of nodes in the BEM mesh (excluding fracture surfaces). The first block $\mathbf{T}_d$ measures the average surface displacement $\mathbf{u}_s$ given a piecewise linear displacement field $\mathbf{u}$, and similarly, the second block $\mathbf{T}_r$ measures the global surface rotation $\mathbf{r}_s$:

$$\mathbf{u}_s = \frac{1}{A}\int_\Gamma \mathbf{u}(\mathbf{x})\ d\mathbf{x} = \frac{1}{A}\sum\nolimits_e \frac{1}{3} A_e \sum\nolimits_{i=1}^{3} \mathbf{u}_{e,i} = \mathbf{T}_d\mathbf{u}, \tag{7.2}$$

and

$$\begin{aligned} \mathbf{r}_s &= \frac{1}{A}\int_\Gamma \mathbf{u}(\mathbf{x}) \times \mathbf{x}\; d\mathbf{x} = \mathbf{T}_r\mathbf{u} \\ &= \frac{1}{A}\sum\nolimits_e \frac{1}{3}A_e \sum\nolimits_{i=1}^{3} \mathbf{u}_{e,i} \times \frac{1}{4}(\mathbf{x}_{e,i} + \mathbf{x}_{e,1} + \mathbf{x}_{e,2} + \mathbf{x}_{e,3}). \end{aligned} \tag{7.3}$$

Here $\mathbf{u}_{e,i}$ denotes the displacement of the i-th node in triangle e, while $\mathbf{x}_{e,i}$ is the material space position of this node. Furthermore, A_e is the area of triangle e, while A is the total surface area. Integration proceeds over the object's ordinary surface Γ (excluding fractures) and we assume (without loss of generality) that the object's centre of mass is at the coordinate origin. We can now apply the regularizer $\mathbf{T}$ to the rank-deficient linear system $\mathbf{D}\mathbf{u} = \mathbf{f}_N$, yielding the well-posed problem $\mathbf{D}_R\,\mathbf{u} = \mathbf{D}^\mathsf{T}\mathbf{f}_N$, where the regularized system matrix is $\mathbf{D}_R := \mathbf{D}^\mathsf{T}\mathbf{D} + \gamma\mathbf{T}^\mathsf{T}\mathbf{T}$. We choose the parameter γ to be the squared average of the diagonal of $\mathbf{D}$: $\gamma = (\mathrm{tr}(\mathbf{D})/(3n))^2$.

Intuitively, we want the regularizer to be roughly as "important" as the system matrix itself. The exact value of γ is not crucial because the regularizer acts only on the null-space of $\mathbf{D}$ by construction. The boundary conditions, however, must be compatible with a solution free of translation and rotation, or in other words, the right hand side $\mathbf{f}_N$ must be in the range of $\mathbf{D}$ as pointed out in [78]. We present a method similar to the one in [78] to build such a surface traction field from collision impulses in the next chapter. In the presence of fractures, applying the regularization to the first row of eq. (7.1) results in:

$$\begin{pmatrix} \mathbf{D}_R & \mathbf{D}^\mathsf{T}\mathbf{X} \\ \mathbf{X}^\mathsf{T} & \mathbf{Y} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\Delta}\mathbf{u} \end{pmatrix} = \begin{pmatrix} \mathbf{D}^\mathsf{T}\mathbf{f}_N \\ \mathbf{0} \end{pmatrix}. \tag{7.4}$$

We solve eq. (7.4) following the approach of eq. (4.3). Similar to the matrix block $\mathbf{A}$ in eq. (4.2), the regularized matrix block $\mathbf{D}_R$ does not change when new crack-opening degrees of freedom are added during the fracture simulation. Consequently, we compute and store the inverse of $\mathbf{D}_R$ at the beginning of the fracture simulation. Figure 31 shows a comparison of results obtained with and without our regularizer. Note that the correct deformation is small (and magnified in the image) due to the high stiffness of the material, while the unregularized system (setting $\gamma = 0$ in $\mathbf{D}_R$) produces unphysical behaviour.

7.2 Balanced tractions from collision impulses

In this chapter we describe how to construct a traction field that is free from global translational and rotational forces and consequently admits an elastostatic equilibrium solution of eq. (7.4). We build this traction field from the collision impulses reported by the rigid-body system: we first convert collision impulses to surface tractions, using a Hertzian contact model to estimate the collision duration, similar to [19,41]. We then remove global force and torque from the traction field, similar to [78]. Finally, we map each of the object's collision points to the closest element in the BEM mesh and add a

corresponding piecewise-constant traction to this element. Recall that the total force is the integral of the traction over the element's area. Consequently, each collision point contributes a traction $J\hat{\mathbf{n}}/(t_c A_e)$ to the entry of the input traction vector $\mathbf{q}_i$ corresponding to the closest element e. Here A_e is the area of this element, J is the impulse transferred by the collision, $\hat{\mathbf{n}}$ is a unit vector pointing along the direction of approach (in the object's local coordinate system), and finally, t_c is the contact duration.

The collision impulse is independent from the rigid-body time step, therefore we need to estimate the contact duration t_c for each collision (instead of using the rigid-body step size directly). In order to do so, we briefly summarize Hertzian contact theory here. Hertz [25] describes the collision of two elastic spheres, or equivalently of one elastic sphere with a rigid plane. This sphere is described by an effective elastic modulus $E = [(1-\nu_1^2)/E_1 + (1-\nu_2^2)/E_2]^{-1}$, radius $R = (R_1^{-1} + R_2^{-1})^{-1}$, and mass $m = (m_1^{-1} + m_2^{-1})^{-1}$. Each of these three effective quantities combines the parameters of both objects involved in the collision. The collision duration is then $t_c = 2.87[m^2/(E^2 R v)]^{1/5}$, where v is the velocity (magnitude) with which the two objects locally approach one another. We refer the interested reader to [61] for a detailed derivation of the collision duration. We obtain the required parameters as follows: elasticity parameters and densities are user inputs and constant per object, the local velocities of the contact points, as well as the transferred impulse, result from the rigid-body simulation, and the effective radii are the inverse of the objects' surface mean curvature at the contact locations. We measure the volume of each object, required to compute its mass, using its high-resolution implicit surface representation.

We then find the smallest possible correction $\mathbf{q}_c$ such that the resulting traction field $\mathbf{q} := \mathbf{q}_i - \mathbf{q}_c$ has no global linear force or torque by solving a quadratic optimization problem with linear constraints. Similar to eq. (7.2) and (7.3), except that we now use piecewise constant instead of piecewise linear shape functions, we build the linear constraints $\mathbf{f}_s = \int_\Gamma \mathbf{q}(\mathbf{x})\, d\mathbf{x} = \mathbf{S}_f \mathbf{q} = 0$ and $\boldsymbol{\tau}_s = \int_\Gamma \mathbf{q}(\mathbf{x}) \times \mathbf{x}\, d\mathbf{x} = \mathbf{S}_\tau \mathbf{q} = 0$ to enforce zero global force and (linearized) torque respectively. We can now write our optimization problem as $\min \mathbf{q}_c^\mathsf{T} \mathbf{q}_c$ s.t. $\mathbf{S}_f(\mathbf{q}_i - \mathbf{q}_c) = 0$ and $\mathbf{S}_\tau(\mathbf{q}_i - \mathbf{q}_c) = 0$, or equivalently as the KKT system with Lagrange multipliers $\mathbf{v}$:

$$\begin{bmatrix} \mathbf{I} & \mathbf{S}^\mathsf{T} \\ \mathbf{S} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_c \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}, \quad \mathbf{S} := \begin{bmatrix} \mathbf{S}_f \\ \mathbf{S}_\tau \end{bmatrix}, \; \mathbf{b} := \begin{bmatrix} \mathbf{S}_f \mathbf{q}_i \\ \mathbf{S}_\tau \mathbf{q}_i \end{bmatrix}. \tag{7.5}$$

In theory, one could also use an area-weighted norm in the objective function, which could improve results in situations where mesh elements of very different areas occur. Due to the way we construct our BEM meshes (see ch. 6.1), we do not need to do so in our application. The Schur complement solution of eq. (7.5) is $\mathbf{q}_c = -\mathbf{S}^\mathsf{T}\mathbf{v}$ and $\mathbf{v} = -(\mathbf{S}\mathbf{S}^\mathsf{T})^{-1}\mathbf{b}$. Note that the matrix $\mathbf{S}\mathbf{S}^\mathsf{T}$ has dimension 6×6 regardless of the number of triangles in the mesh. This matrix can be built directly, requiring only a single pass through the element list. The projection used in [78] applies a similar idea to a tetrahedral mesh. Their method, however, requires a QR-factorization of a $3n \times 6$ system for a mesh with n nodes.

Having found the balanced piecewise-constant surface traction field $\mathbf{q}$ closest to the input tractions $\mathbf{q}_i$, we assemble the right-hand-side vector $\mathbf{f}_N$ in eq. (7.4) according to eq. (2.27). At this point, we are ready to proceed with the fracture simulation, using either the full BEM solution of ch. 4, or our fast estimators as in ch. 5. In the former case we solve eq. (7.4) instead of eq. (2.38) in every BEM time step, while in the latter case, the update to the right-hand side turns into $\mathbf{f}_N \leftarrow \mathbf{f}_N - \mathbf{X}\,\boldsymbol{\Delta}\tilde{\mathbf{u}}$, using the estimated crack-opening displacements $\boldsymbol{\Delta}\tilde{\mathbf{u}}$ according to eq. (5.6).

7.3 Generating rigid bodies for fragments

So far, we have described how we start a fracture simulation for an object that experiences a sufficiently forceful collision in the rigid-body scene. Once this fracture simulation is completed, we identify new fragments and update the rigid-body scene accordingly. As described in ch. 6, we use a sparse level-set function to represent the high-resolution surface of an object, including all fractures; we then use our modified breath-first-search algorithm to find connected components, i.e. fragments. The final step of coupling our fracture method to the rigid-body system is then to replace the broken object by its fragments. Only large enough fragments become breakable rigid bodies; this way we avoid spending time on fracture simulations for small pieces that are barely visible in the final output.

In the following discussion, we refer to the rigid body that has just fractured as the "*parent*" object. At the beginning of the rigid-body simulation we compute the total volume of every breakable object, and we copy this information from the parent to its fragments after every fracture simulation; we call this the "*original volume*". We then classify fragments into the following four groups based on their (relative) volume:

(a) Very small fragments (0.5% or less of the original volume): for efficiency reasons these fragments are not allowed to fracture any further in our implementation. In order to speed up collision detection performed by the rigid-body engine, we represent them by their convex hull in the rigid-body scene.

(b) Small fragments (0.5–2% of the original volume) cannot fracture either, but use a (possibly non-convex) mesh for collision detection.

(c) Large fragments (more than 2% of the original, but less than 95% of the parent's volume) become breakable objects and can fracture again in subsequent rigid-body time steps. These fragments have both a mesh for collision detection, as well as a (coarser) mesh for BEM computations required by later fracture simulation runs.

(d) Very large fragments: if we find a fragment that has more than 95% of its immediate parent's volume, we do not create a new BEM mesh, but instead retain the one from the parent object along with its operator matrices; we only update the collision detection mesh in the rigid-body scene. This way, we avoid the overhead incurred from constructing a new BEM mesh, as well as building, regularizing, and pre-factoring the system matrix $\mathbf{D}_R$. If there is no such fragment, we delete the parent object, along with all its associated BEM data, as soon as all its fragments have been processed.

We choose to classify fragments mostly based on the original volume, rather than the volume of their immediate parent object, because it retains a better notion of scale: a fragment of a particular size should be treated in the same way regardless of whether it has broken off of the original object right away, or after a series of intermediate fracture events. We build the meshes required for collision detection in the same way as the ones used for BEM calculations (except that we use a higher resolution than the BEM meshes, but still coarser than the output resolution). We use approximately 5 times more elements for collision meshes than for BEM meshes in all our results, except for the scene shown in fig. 42, where we keep the collision meshes very close to the resolution of the output geometry.

Except for large ones, all other fragments are easily handled by adding (or updating) their shape in the rigid-body scene and copying all material parameters from the parent. We set the initial position and velocity of all fragments such that they match the motion of the parent object *before* the collision happened. Once all fragments are in place, we repeat the rigid-body step that caused the collision (but without starting any new fracture simulations), allowing the newly created fragments to respond to the collision. This simple approach produces visually convincing collision responses in our results. A possibly more accurate approach would be to compute additional "fracture impulses" that drive fragments away from one another as elastic energy is released once they break apart, as done in [78].

The rigid-body system also handles gravity, even if objects rest on top of one another: the objects are accelerated downwards due to gravity, while the collision system restores them to a non-overlapping state. We can then use the resulting collision impulse for our fracture simulations, which allows us to ignore gravity in our BEM formulation. Figure 30 shows such a situation, where the horizontal bar breaks under its own weight. Also note that the large fragments shown in fig. 30b break again due to subsequent collisions.

Fragments containing incomplete cracks

For each large fragment, we need to build a boundary element mesh and assemble the linear system (7.4). However, the fragment may contain cracks that occurred during the previous fracture simulation, but do not cut it into separate pieces (yet). Consequently, we have to represent these cracks in the BEM mesh and allow them to grow further in subsequent fracture simulations; we refer to them as "*incomplete*" cracks, see also fig. 43.

We first build a BEM mesh of the fragment's ordinary surface (excluding incomplete cracks) as described in ch. 6.1. We then identify all incomplete fractures and copy fracture elements that are inside of the fragment from the parent's into the fragment's BEM mesh. This way, the representation of an incomplete fracture in the fragment's mesh is a subset of its representation in the parent's mesh. We take care to ensure that this subset is again a manifold with boundary, in particular the new (coarse) crack front, which is the bounding curve of this subset of triangles, has exactly two edges meeting at each node. If this criterion is not met at any node, we additionally

copy all triangles containing this node. Conversely, if a triangle consists only of crack-front nodes, it does not contribute a crack-opening degree of freedom, so we remove all such triangles. Of course, we also copy the high-resolution level-set representation of all incomplete cracks to the fragment. Additionally, wherever the new crack *front* coincides with the old one, we copy the high-resolution crack-front data as well. This treatment allows "old" fractures to propagate further if the fragment experiences a collision at a later rigid-body time step.

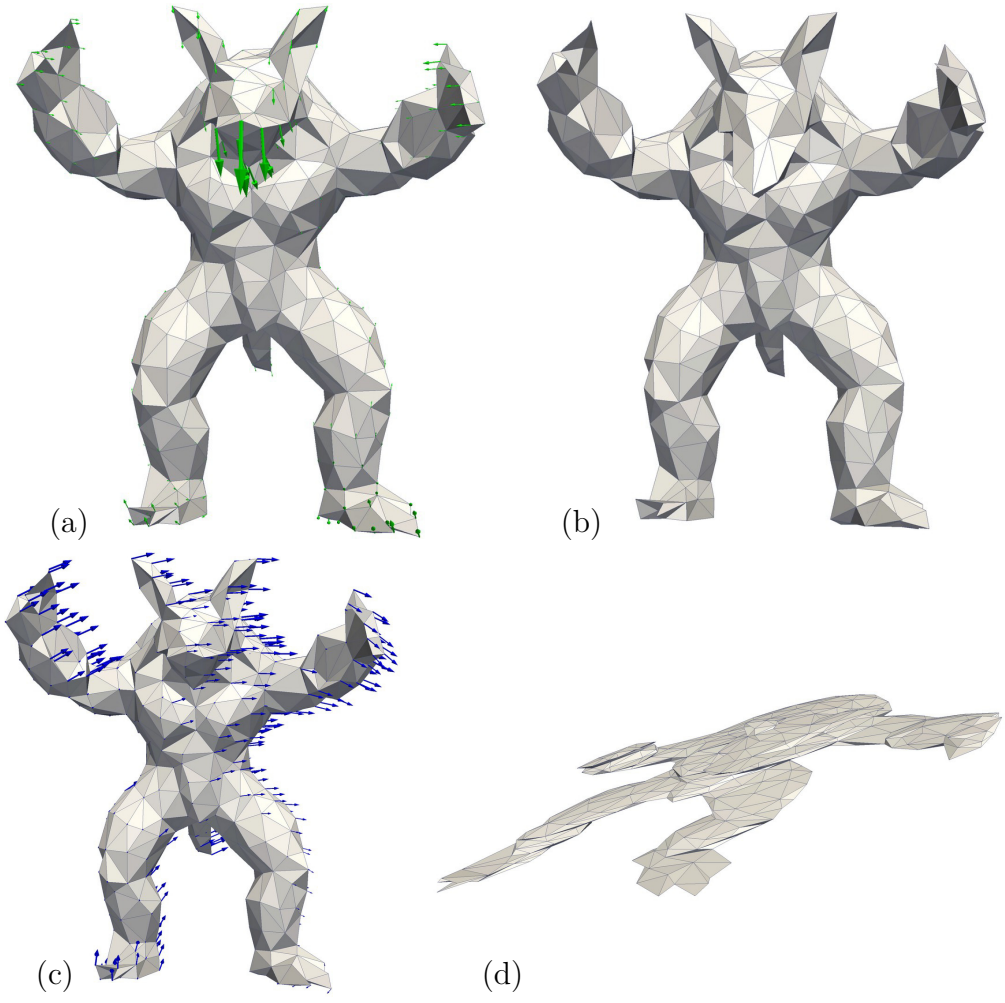


Figure 31: Hitting the Stanford armadillo on the nose: our regularizer reliably solves the pure Neumann boundary value problem (a, b); displacement vectors (green) and mesh deformation are 10x magnified. Without regularization, the displacement field (c) contains large rigid components that lead to unphysical behaviour and almost flattens the armadillo (d); displacement vectors (blue) are scaled by 0.01, mesh deformation drawn to scale.

8 Results

In the previous chapters, we have presented the components of our quasi-static fracture simulation in detail, namely a high-resolution crack propagation algorithm driven by a coarse elastostatic boundary element method, fast estimators improving the theoretical complexity, as well as speeding up the practical runtime, of our algorithm, a level-set based treatment of a fractured object's geometry and topology at high resolution, and finally, a two-way coupling between our fracture method and a standard rigid-body dynamics system. We are now ready to look at various results our method can produce, both in terms of basic verification test cases, as well as example scenes involving complex fracture behaviour of multiple objects. Tables 1–3 provide an overview of the simulation and material parameters used for the results discussed in the remainder of this chapter.

Animation output
We use either Kitware Paraview or Autodesk Maya® to render our results. For basic scenes, without the rigid-body method described in ch. 7, the output of our fracture simulation is a collection of meshes of the fracturing object or its fragments, generated either after every crack-propagation sub step (if we want to show a super-slow-motion animation of the fracturing process), after every BEM time step (for a similar slow-motion animation), or just once at the end of the fracture simulation (if only the resulting fragments are of interest). In the latter case, we can then run a rigid-body animation (within Maya) as a post process in order to produce visually interesting motion of the resulting fragments; fig. 40 shows an example of this approach.

If we want to produce a slow-motion animation, where every time step (or possibly even every sub step) becomes one animation frame, we output the complete state of the simulation after each full BEM time step. For efficiency reasons, both in terms of runtime and disk usage, we output only small incremental updates to the implicit surface after each sub step and then combine these updates with the data from the previous time step to build a high-resolution mesh for each sub step in a post-processing pass. This procedure avoids wasting time on producing high-resolution output geometry during the simulation and therefore makes it easier for the user to run a simulation, inspect the results, and decide whether to invest time in producing detailed sub-step output afterwards. Note that we can visualize the deformation of a breaking object after each time step as shown in fig. 29. In a similar way, we can also produce such results for sub steps by interpolating displacements (piecewise linearly) over time between the previous and following time step. Due to the high crack propagation speed (especially in stiff materials), our time steps are usually on the order of milliseconds to microseconds, meaning that these slow-motion animations show a much smaller time scale than a typical rigid-body animation.

Consequently, when using our two-way rigid-body coupling, as described in ch. 7, we choose not to output intermediate fracture simulation states. Instead, we set the maximal number of crack propagation time steps such that the entire facture simulation

cannot exceed the rigid-body time-step duration (nor the estimated collision duration). At the end of each fracture simulation, we then output the final simulation state, together with a surface mesh for each resulting fragment. We first output these meshes at a reduced resolution for preview purposes, and allow the user to build the high-resolution version in a post-processing step. Furthermore, when an object breaks, we need to either update its visual representation (if the fracture simulation results in a very large fragment, see ch. 7.3), or remove it from the animation (as it has been replaced by a collection of fragments). Similarly, for these rigid-body coupled scenes we also need to output the motion of all objects in the scene. To this end, we automatically generate a `MEL` script that instructs Maya to set motion key frames (position and orientation) for all active objects after each rigid-body step, as well as automatically hide all broken objects that are no longer active, and load the meshes of new fragments into the animation. The resulting scene then contains the entire animation, including the motion and fracturing of all the objects and their fragments. Apart from choosing lighting, material appearance, and camera parameters, this scene is ready for rendering.

8.1 Basic test cases

In this chapter, we first show some basic test cases, using homogeneous materials, to demonstrate that our method produces fracture surfaces in line with the theoretical predictions of LEFM. We then show examples of various material toughness fields and their influence on the resulting fracture surfaces. We compare result obtained solving the full BEM system to those of our fast approximate method. We also show that our rigid-body coupling integrates well with both fracture simulation methods.

First, we consider a cube containing a planar edge-crack under mode-I, II, and III loading respectively (see fig. 4). For these three situations, assuming a homogeneous material, fig. 7 illustrates the expected fracture behaviour described in ch. 2.4. The simulation starts with a coarsely meshed cube (similar to the one shown in fig. 34a) with a planar, horizontal initial crack in the middle of the left side face. Our results in fig. 32 use the full BEM method and reproduce the desired fracture shapes. Here, we also demonstrate the ability of our method to handle various boundary condition (BC) types: the mode-I case is defined by Dirichlet BCs (fixed displacements) at the top and bottom face of the cube (and homogeneous Neumann BCs on the sides), while the mode-II case uses a homogeneous Dirichlet BC (zero displacement) on the bottom face and an inhomogeneous Neumann BC (constant traction) on the top face. Similarly, we can define a predominantly mode-II loaded situation by applying inhomogeneous Neumann BCs (in opposite directions) on the upper and lower half of the broken side face (left/back) instead, as shown in fig. 23. In this particular test case, we also add a traction along the opening (mode-I) direction, whose magnitude is 30% of the mode-II traction magnitude, resulting in a positive K_{I} in the first time step, as shown in fig. 23d (the qualitative appearance of the fracture surface is almost unchanged by this additional mode-I loading). Note the different angles of the resulting fracture surface

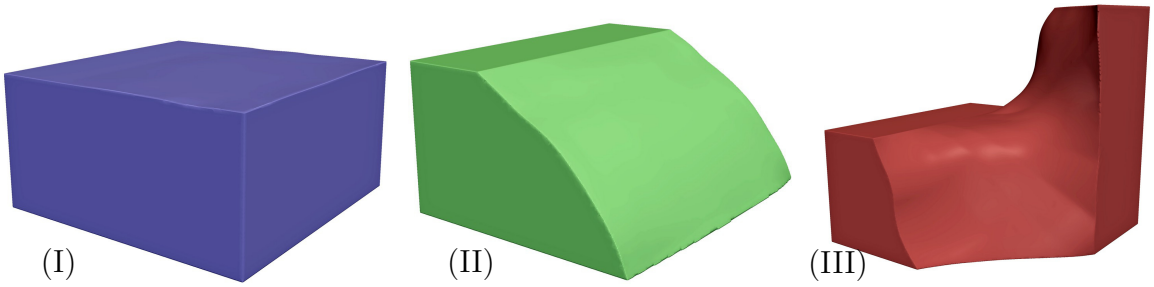


Figure 32: Edge-cracked cube under pure loading modes (I–III): results using the full BEM solution.

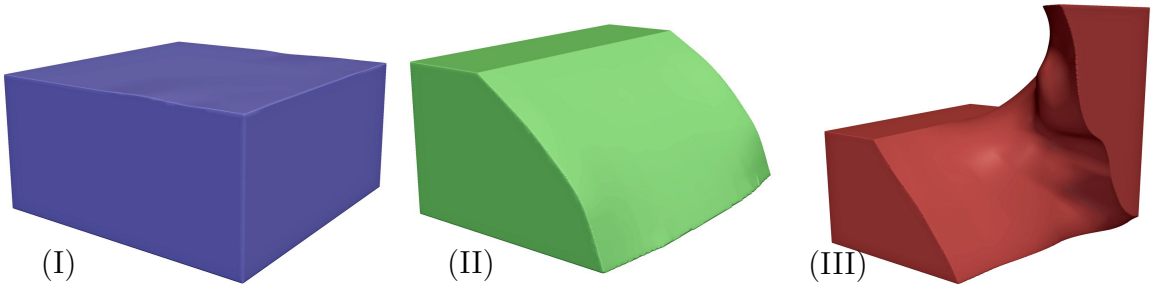


Figure 33: Edge-cracked cube under pure loading modes (I–III): results using the fast estimation method.

between fig. 32II and 23b, depending on how the boundary conditions restrict the deformation of the object. Finally, the mode-III loaded case (fig. 32III) also uses pure Neumann BCs (similar to fig. 23b) pulling the upper and lower half of the broken side face in opposite, but tangential, directions respectively. Whenever we apply Neumann boundary conditions only, we use the regularization method described in ch. 7.1. In the mode-III case we observe the characteristic "twisting" motion of the crack front. This twist is caused by variation of the local mode-II SIF along the crack front. (Note that we do not explicitly model this twisting behaviour based on the local mode-III SIF.) In the absence of toughness gradients, the piecewise-linear nature of the SIF-field driving the crack propagation simulation becomes visually apparent in fig. 32III, revealing that the BEM mesh resolution is significantly coarser than the implicit surface (and output) resolution. In all these test cases we observe that crack propagation proceeds in such a way as to maximize the local mode-I loading at the crack front, see also [58].

In chapter 5 we introduced the main components of our fast SIF estimation method, namely alignment with an eigenvector of the stress field and an update to the right-hand side of the linear elastostatic system. Figure 22 shows that a basic estimator without these modifications both suffers from oscillation artefacts on the resulting fracture surface and underestimates the stress intensities as the crack front propagates (especially when it reaches the object's ordinary surface). Figure 24 shows that with our modifications, we are able to obtain good SIF estimates for basic test cases. We also successfully avoid any oscillation artefacts, as shown in fig. 23b, as well as fig. 34c. Running the same examples as in fig. 32, using our fast approximations instead of the

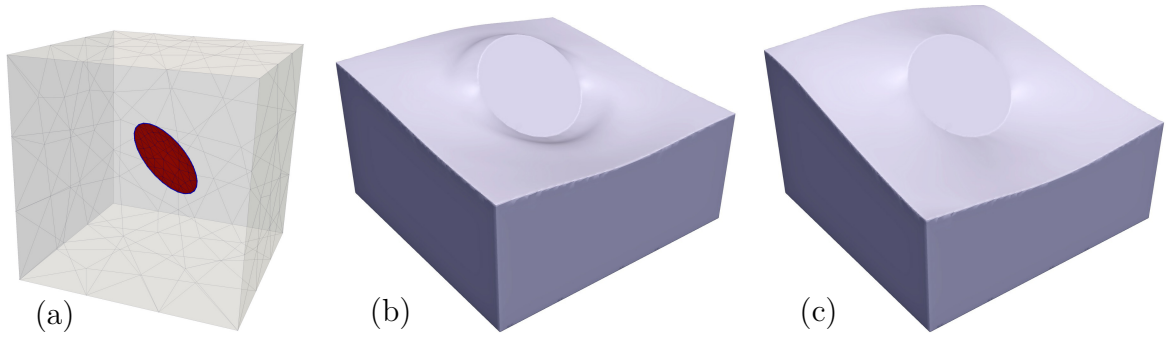


Figure 34: Cube with a 45° inclined penny-shaped crack: (a) initial BEM mesh, (b) BEM result, (c) result using fast estimators. Figure 24a shows a comparison of SIF values obtained for (b) and (c) during crack propagation.

full BEM solution, shows that our SIF and COD estimators qualitatively reproduce the fracture behaviour, as shown in fig. 33. The mode-III case is, however, not resolved as cleanly and slightly loses its symmetry due to numerical errors. In our experiments we have observed that K_{I} is still slightly underestimated in the mode-II case with mixed boundary conditions (fig. 33II), making the material appear artificially tougher. We accept this limitation, as the qualitative fracture behaviour is very close to the full BEM solution, and also because our rigid-body coupling always works with pure Neumann boundary conditions.

Next, we consider a cube under uniform tension along the vertical axis, applying a Dirichlet BC on the base and an inhomogeneous Neumann BC on the top face. This cube contains an initially penny-shaped crack inclined by 45° to the direction of tension, see fig. 34a. Assuming again a homogeneous material, the fracture surface produced by our boundary element method (fig. 34b) smoothly approaches the plane orthogonal to the axis of tension. Our (full BEM) result for this standard test case is qualitatively very close to the X-FEM result of Gravouil et al. [20]. The effect is less pronounced when using our fast approximate method (fig. 34c), but still qualitatively acceptable. Forcing the propagation path to align with a stress eigenvector avoids the oscillation issue (seen in fig. 22b), but also to some extent suppresses the characteristic kink observed at the edge of the initial crack (see also fig. 9 in [58] for a 2D example). Figure 24a shows that our estimated stress intensity values closely match the full BEM solution in this test case.

Figure 13 shows a similar situation for a uniaxially loaded cube, but this time under compression instead of tension and without any initial fractures. We apply Dirichlet BCs on the top and bottom face resulting in a compressive mode-I load. In this case, we simulate multiple fractures as created by our crack initiation method. The primary one then propagates under compressive loading, while secondary cracks appear and propagate under tension caused by Poisson's effect, roughly orthogonal to the primary fracture. The final result is that the cube breaks into multiple fragments; the four largest ones are shown in fig. 13b.

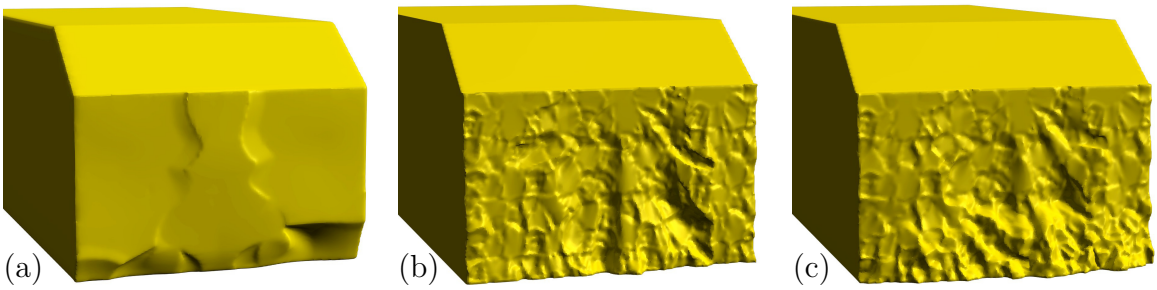


Figure 35: A notched bar in a 3-point-bending test (see also fig. 19): (a) full BEM result with a homogeneous material, (b) BEM result with a granular material, (c) result of our fast approximation method using the same granular material.

Having established that both our full BEM, as well as our fast approximate method, handle homogeneous materials successfully, we now consider inhomogeneous materials. Recall that varying *elasticity* parameters are difficult to treat in a BEM framework, and we consequently focus on (spatially) varying *fracture* parameters (strength and toughness) instead. We first demonstrate how the simulation can be controlled using carefully tailored strength and toughness fields. Consider a cube under uniform vertical tension (similar to fig. 13, but with opposite signs on the prescribed boundary displacements). We then use the strength field shown in fig. 18b to bias crack initiation towards the middle of the cube. In a homogeneous material, the crack would propagate under mode-I loading and form a planar surface, as in fig. 32I. However, we use the toughness field shown in fig. 18a to bias crack propagation towards a lower, parallel plane. As shown in fig. 18c–e, the fracture starts in the low-strength region (see also fig. 11 for a close-up view) and smoothly approaches the plane of minimal toughness. We limit crack initiation to a single fracture in order to make the propagation behaviour clearly visible.

We then reproduce the well-known 3-point-bending test of a notched bar. Figure 19a shows the initial mesh and an illustration of the applied boundary conditions: prescribed zero displacements are drawn as red bars, while arrows indicate prescribed surface tractions. In this case, there are no fractures in the initial geometry. Instead, the notch creates a stress concentration, which causes cracks to naturally initiate nearby. This test case shows that our crack initiation method (ch. 4.4) agrees with LEFM theory and experiments. Please note the small ridges in fig. 35a formed by individual cracks propagating parallel to each other and eventually intersecting. Similar patterns are often found in nature and are sometimes referred to as “river lines” in the literature [3]. This 3-point-bending test is also one of the few cases where the limitations of the COD-BEM formulation, discussed in chapter 2.5, become visually apparent as small ripples close to the bottom surface of the bar. In that region, the COD and hence SIF values are influenced by the nearby surface, causing deviations in the crack front’s path. Figure 35b shows the same setup, but now using a toughness field modelling a granular material, in which case these artefacts become barely noticeable due to the surface patterns arising from the inhomogeneous material. Again, our fast approximate method yields qualitatively equivalent results, fig. 35c, providing further evidence that

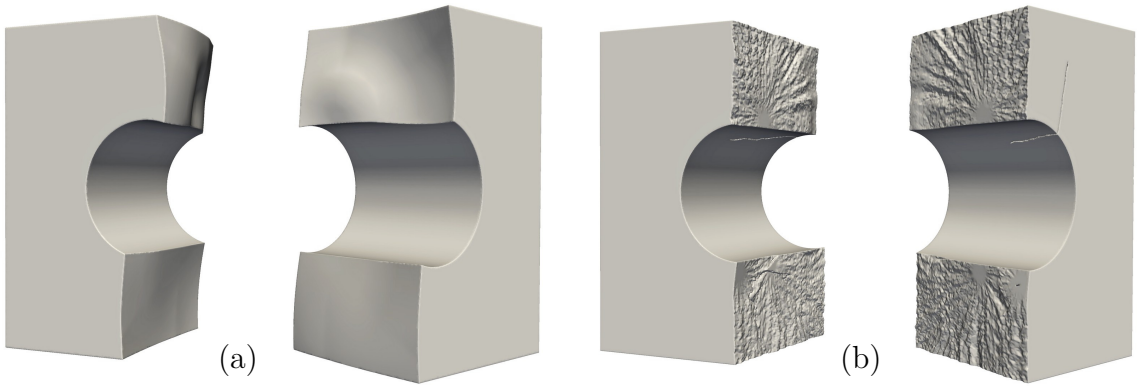


Figure 36: Breaking a "chain link": a high-strength material produces only two fractures (a), whereas a low-strength material results in additional partial cracks (b).

our SIF and COD estimates are physically reasonable. The compressive zone close to the bottom of the bar (resulting from the bending deformation), however, is not quite as well resolved by the approximate method, resulting in a fracture surface that is not as perpendicular to the length-axis of the bar compared to the full BEM solution. In general, we observe that simulating a small number of clean cuts through an object requires a more accurate deformation solution, which means that the full BEM simulation is the better choice than the fast estimators in such cases. The approximation errors quickly become visually unnoticeable as the number of fractures increases.

Figure 36 shows a "chain link" breaking under uniaxial tension. This is another example where the object's geometry naturally concentrates crack initiation around the region of minimal cross-section area. We use Dirichlet boundary conditions displacing the left and right outside faces along their normal direction (note that the images show the resulting fragments positioned differently in order to better display the fracture surfaces). In the first example (fig. 36a) we use the material properties of poly-methylmethacrylate (PMMA, also known as "acrylic glass"), which has a relatively high strength compared to its toughness. Consequently only two cracks initiate, and both propagate until the object is cut into two pieces. On the other hand, in the second example (fig. 36b) we use the material properties of concrete, which has a relatively low strength. As a result, we see multiple partial fractures (which eventually stop propagating due to insufficient stress intensity) in addition to the ones that separate the two fragments. In this example we also use a granular material model, which leads to interesting fracture surface patterns: perturbations due to varying toughness propagate with the crack front, leaving clearly visible directional patterns called "chevrons" in their wake. Consequently, the propagation direction is discernible in the final fracture surface. As multiple fractures propagate close to one another, some river lines also appear. These chevrons and river lines are a signature of brittle fractures in nature, see also [3]. Another common feature is that the surface roughness increases with distance from the initiation site, which itself is fairly smooth. To our knowledge, our approach is the first method in computer graphics to produce these physically realistic surface details.

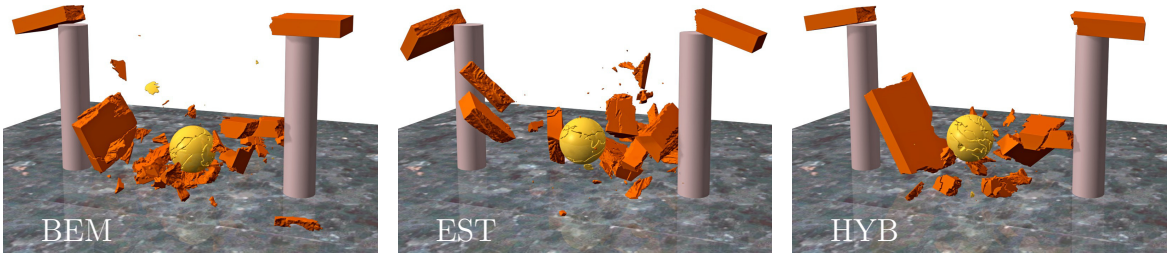


Figure 37: "Breaking II": comparison of fracture methods (full BEM, fast estimators, and hybrid method).

So far, we have discussed cases where we specify the geometry, boundary conditions, and material parameters of a single object, and then run one instance of our fracture simulation method. Tables 1 and 2 give an overview of the simulation and material parameters, as well as the total runtime required to compute these results.

Finally, we turn our attention to the rigid-body coupling described in ch. 7. Figure 30 shows a test case where a bar (without notching) rests on two pillars: in the rigid-body simulation, the bar is accelerated downwards by gravity, while the collision response system produces an impulse to counter this motion. Balancing the traction field resulting from these impulses (as described in ch. 7.2) results in a bending deformation similar to the previously discussed 3-point test. We choose material parameters for the bar such that it is so weak that the gravitational load is sufficient to break it. However, in the absence of notches, fractures occur more widely spread rather than concentrated around a small area. This result shows that we do not need to add body forces due to gravity directly to our BEM formulation (and we can safely assume $\mathbf{g} = 0$ in eq. (2.12) for all our examples). The large fragments seen in fig. 30b break again as they undergo further collisions with the ground or the pillars.

Similarly, we can of course choose a stronger material and break the bar by dropping a wrecking ball on top of it, as shown in fig. 37. We obtain qualitatively similar results from either the full BEM solution, our fast estimators, or a hybrid method formed by combining both approaches (solving the full BEM system as long as the system size is sufficiently small and switching to the fast method afterwards). For the hybrid example in fig. 37, we switch to the fast estimators when there are more than 400 elements in the BEM mesh. This threshold applies to each fracture simulation instance independently.

The granular toughness field applied to the bar in fig. 37 contains on average approximately 50×12 grains per cross section area. The interior of the grains is 40% tougher than the grain boundary surfaces. Similarly, the notched bar in fig. 35b (and c) contains roughly 21×13 grains per cross section area and the toughness increases by 10% away from grain boundaries. Please also refer to table 2 for further material parameters used in our single-fracture experiments.

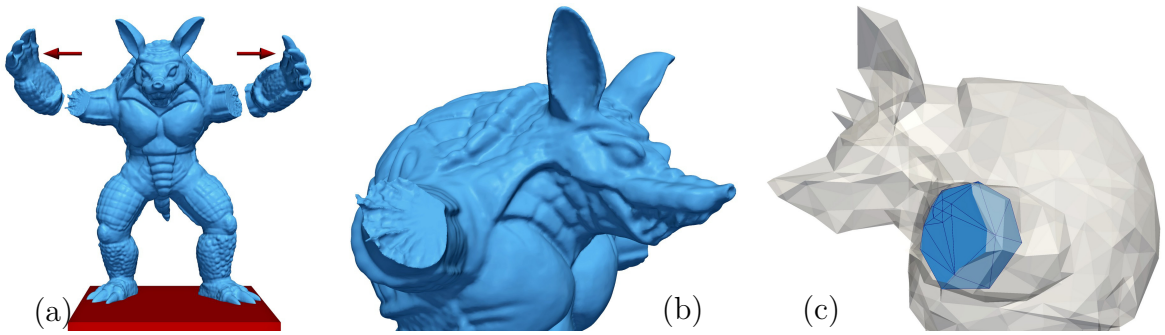


Figure 38: Tearing the armadillo; image (c) shows the BEM mesh of one of the fractures at the end of the simulation.

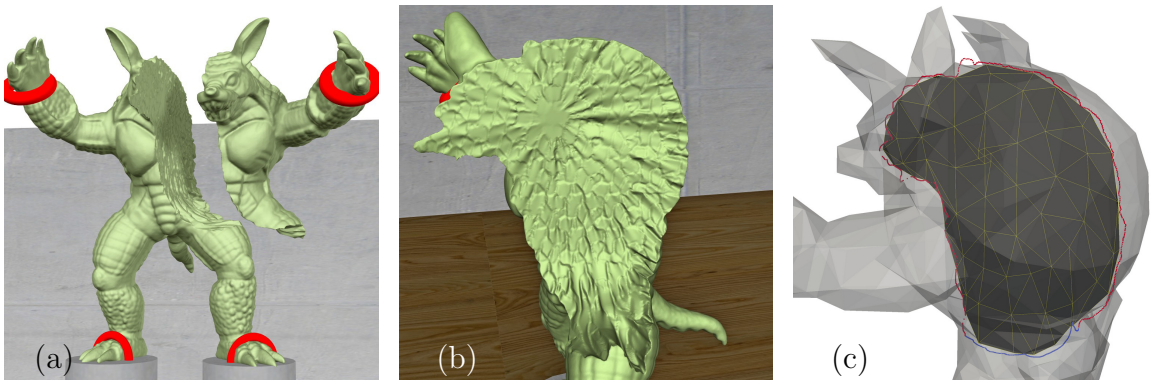


Figure 39: Splitting the armadillo with a user-specified initial fracture; image (c) shows the BEM mesh of the fracture at an intermediate step during the simulation, red points indicate crack-front markers that have reached the surface, blue points indicate markers that are still propagating.

8.2 Further examples

In this chapter we show additional examples demonstrating the versatility of our fracture simulation framework. We also compare our fast approximations to the standard BEM in more complicated scenarios, where the advantages in terms of runtime and memory consumption of our new method are much more pronounced than in the simple test cases discussed in the previous chapter.

In our first example (fig. 38), we apply a horizontal traction on the claws of the Stanford armadillo (indicated by red arrows in the image), while keeping the feet fixed. We use the material parameters of PMMA with an additional granular toughness field. The resolution of the implicit surface representation is 50x higher than the resolution of fracture elements in the BEM mesh, resulting in sharp features on the edge of the fracture surface (fig. 38b), but taking only three full BEM time steps to simulate. In this situation, the total runtime of just under one minute is dominated by the assembly of the initial BEM system matrix at the start of the simulation. Note that our surface-stress-based crack-initiation method again places cracks in regions of narrow cross-section with respect to the axis of traction.

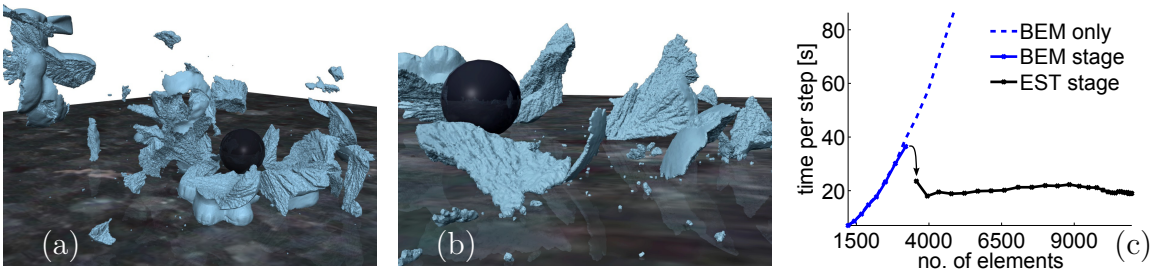


Figure 40: Bunny smash: graph shows CPU time per crack-propagation time step for our hybrid method compared to our full BEM method.

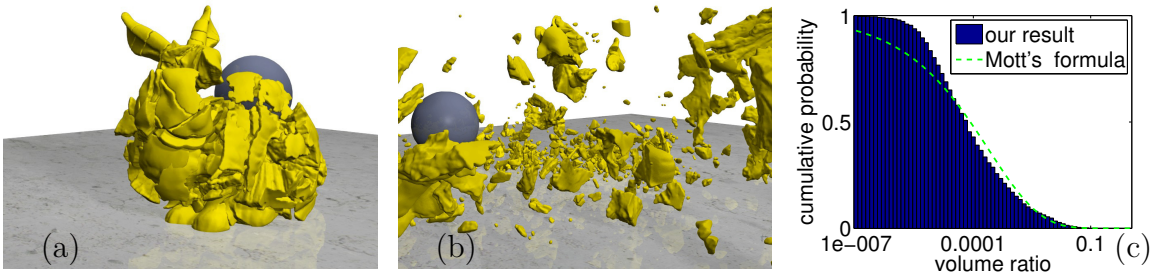


Figure 41: Bunny smash with two-way rigid-body coupling: graph shows fragment volume distribution compared to Mott's formula.

Similarly, in fig. 39 we apply the same boundary conditions as in fig. 38. We use a material of the same overall properties as well, but a coarser rolled grain structure. Instead of our stress-based crack initiation method, we manually specify one initial crack (placed vertically in the neck) and do not allow any more cracks to initiate. With this slight manipulation of the initial conditions, our fracture simulation proceeds to split the armadillo down the middle. Recall that the feet are fixed and consequently there is very little stress applied to the legs, so the fracture propagates mostly through the upper body.

Figure 40 demonstrates the efficiency of our fast SIF estimation method compared to the standard BEM approach. We smash the Stanford bunny by applying zero-displacement boundary conditions on the base (where the bunny touches the ground) and a fixed indentation on the back of the bunny. We then run a single fracture simulation and add the rigid-body dynamics of the resulting fragments in a post-processing step. During the fracture simulation, we use our hybrid method with a threshold of 3000 triangles in the mesh. We compare the CPU time required to compute each crack propagation time step to a control run solving the full BEM system in fig. 40c. While the standard BEM quickly becomes very slow, the runtime required for a time step using our estimation method is independent from the total number of elements in the mesh: it only scales with the length of the crack front, which actually reduces slightly as the crack front reaches the object's surface (in some places earlier than in others). We plot the timings for the control run until it exceeds 4GB of memory. In contrast, when switching to our fast approximate method the entire simulation runs within this memory budget.

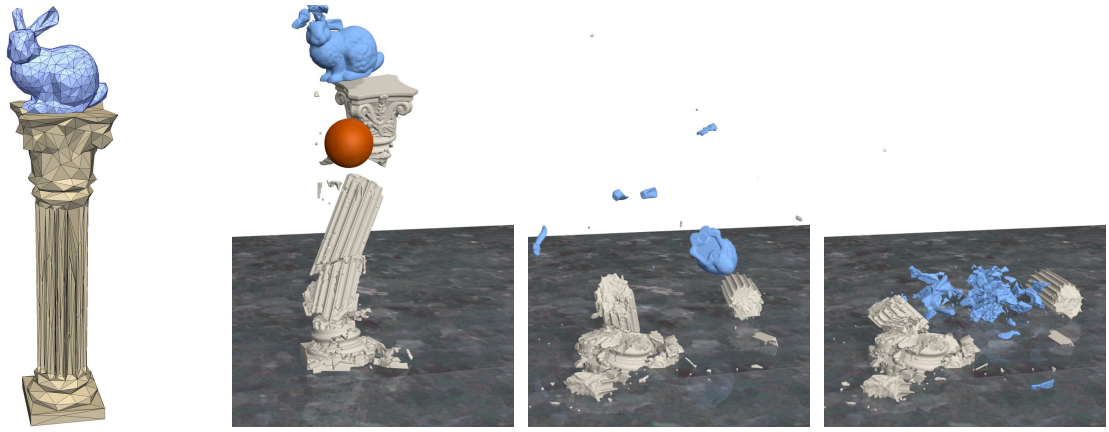


Figure 42: Bunny on a column: left image shows initial BEM meshes, note that our method robustly handles badly shaped triangles on the column.

In fig. 41 we show a similar example, but now including our two-way rigid-body coupling. In this scene we use our fast approximations right from the start of the simulation. Choosing a weaker material for the bunny, we manage to produce over 1000 fragments in four rigid-body steps. We show the distribution of fragment volumes in fig. 41c and compare it to Mott's formula, a widely accepted theoretical model for fragment size distribution, which is given by $P(V) = e^{-\sqrt[3]{\zeta V}}$, $\zeta = 6/\bar{V}$. Here $\bar{V}$ is the average fragment volume (see also eq. (20) in [14]). Figure 41c plots the probability that a randomly selected fragment has a relative volume equal or larger than a given ratio. Fragment volumes are measured relative to the original object's volume. The volume distribution of our result is in good agreement with theoretical predictions overall, except that our result contains less very small fragments than predicted. We believe this deviation to be mostly due to the resolution limit of the implicit surface on which we store the high-resolution geometry; i.e. fragments much smaller than the voxel size cannot be represented.

As described in ch. 5.3, the speed-up (i.e. the reduction of runtime spent on fracture simulation for a particular scene) depends strongly on the number of elements used to represent fractures. Our basic comparisons in figures 32–35 and 37 take about the same time for both methods as they contain very few elements. For the "bowl" example in fig. 45, a scene of medium complexity, the fast approximate fracture simulation is about 10x faster than the full BEM version. Similarly, our hybrid method used in the bunny-smash example (fig. 40) reduces the overall runtime roughly by a factor of 14, which means that the simulation completes in less than 15 minutes, instead of almost 3.5 hours required for the full BEM solution. On our largest example in terms of fracture elements, fig. 41, the fast approximate method is about 35x faster than the BEM, and the fracture simulation takes about 22 minutes instead of 13 hours. All of these timings refer only to the fracture simulation, excluding time spent on rigid-body dynamics.

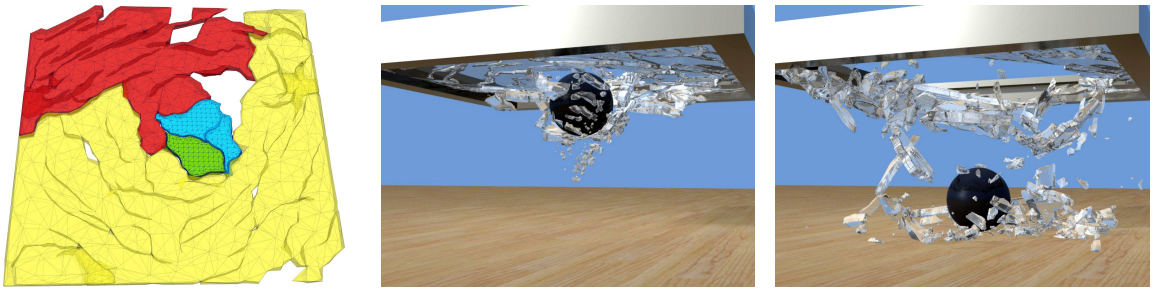


Figure 43: Breaking a window: left image shows BEM meshes of remaining large fragments after the first fracture event, containing many incomplete cracks. All four fragments break again into smaller pieces in subsequent rigid-body time steps.

More artistic examples are shown in figures 42–45, all of which use our two-way coupled rigid-body dynamics and fracture approach. Please refer to table 3 for an overview of our method's performance on these scenes. In fig. 42 the Stanford bunny rests on top of a column; we then shoot the column with a cannon ball. As the top of the column breaks, its motion also pushes the bunny upwards. Using again a weak material for the bunny, this sudden acceleration is enough to break its ears (due to their narrow cross-section). As the bunny's body falls down, it hits a sharp corner of one of the column's fragments and shatters further. For this scene we use rigid-body collision meshes that are almost the same resolution as the output geometry in order to resolve the numerous contacts between fragments. Consequently, the runtime spent on rigid-body dynamics (marked with '*' in table 3) is much higher than for most other scenes (where we use lower resolution collision shapes, see also ch. 7.3). In most of our other examples, fracture simulation accounts for about 70% of the total CPU time and rigid-body dynamics for the other 30%. Some scenes lead to fairly complex contacts between various fragments, resulting in a roughly 50-50 split of the overall runtime. (Note that we use mesh-based collision detection, but do not implement more advanced methods, such as approximate convex decomposition [43] that could improve the rigid-body runtime.)

Even though boundary element based methods are most efficient for objects with large volume to surface ratios, our method is still capable of fracturing a thin (but volumetric) window, shown in fig. 43. After the first fracture simulation, some large fragments with many incomplete cracks remain. As we copy these incomplete cracks from the parent object to the large fragment's BEM mesh, they can propagate further once the large fragment collides with another object in a subsequent rigid-body step. Eventually, many of these incomplete cracks grow further cutting out many smaller fragments in the process.

The first example of fig. 44 shows a brick-breaking setup. In this scene we also use our hybrid method, switching from the full BEM to the approximate solution at 300 elements (which is roughly half-way through a fracture simulation as we start each one with 200 elements and the largest number of elements in one run is 412). In this scene we need to accurately compute the major cut through each brick, but do not want to

invest runtime in any of the less interesting branching cracks. Our hybrid method is ideally suited for this task and we can still use a resolution ratio of $r_f/r_c = 50$ for all five bricks.

The second scene in fig. 44 starts with the orange armadillo standing in front of a wall. The blue armadillo then smashes into the first one at a high speed and both of them crash into the wall. Similarly, in the first scene of fig. 45 a bunny and an armadillo collide and then drop into a glass bowl. Finally, the second scene in fig. 45 shows various simple objects, such as boxes, cylinders, and tori, dropped from a height on top of one another. This scene produces the second largest number of fragments among our examples. Due to the high impact velocities, as well as the large number of fragments, computing rigid-body dynamics takes almost twice as long as the fracture simulation itself. All three scenes described in this paragraph use only the fast approximate fracture method.

These examples, as well as all the other results presented in this chapter, demonstrate that our fracture method is well capable of producing a wide variety of visually detailed, physically plausible fractures in both controlled test setups and complex artistic animations.

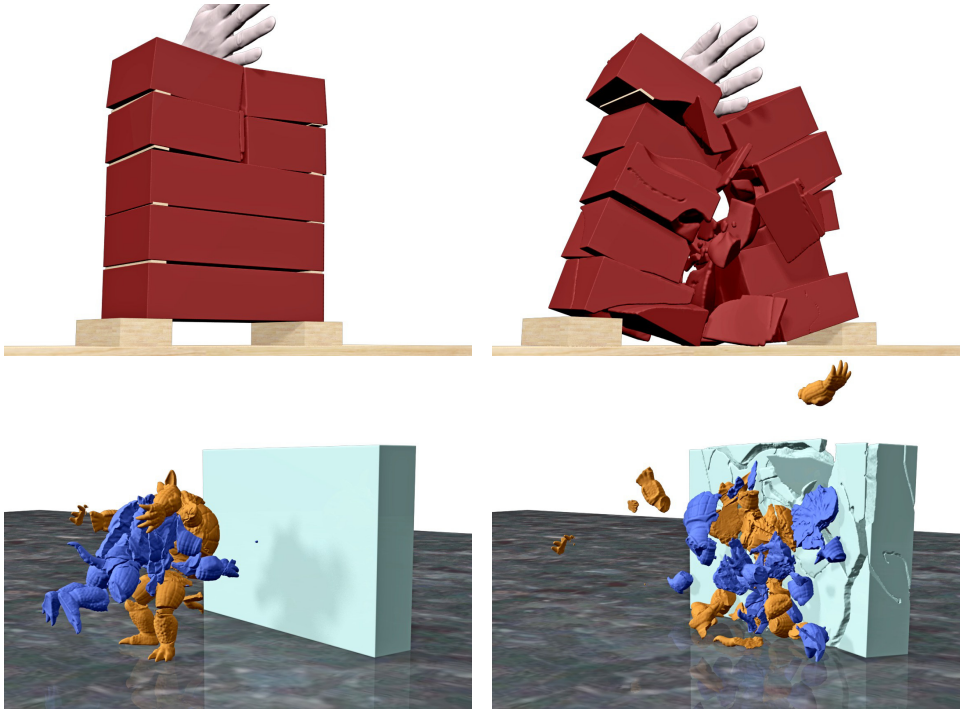


Figure 44: Further results 1: a brick-breaking scene simulated with our hybrid method, and an argument among armadillos simulated with our fast approximation method.

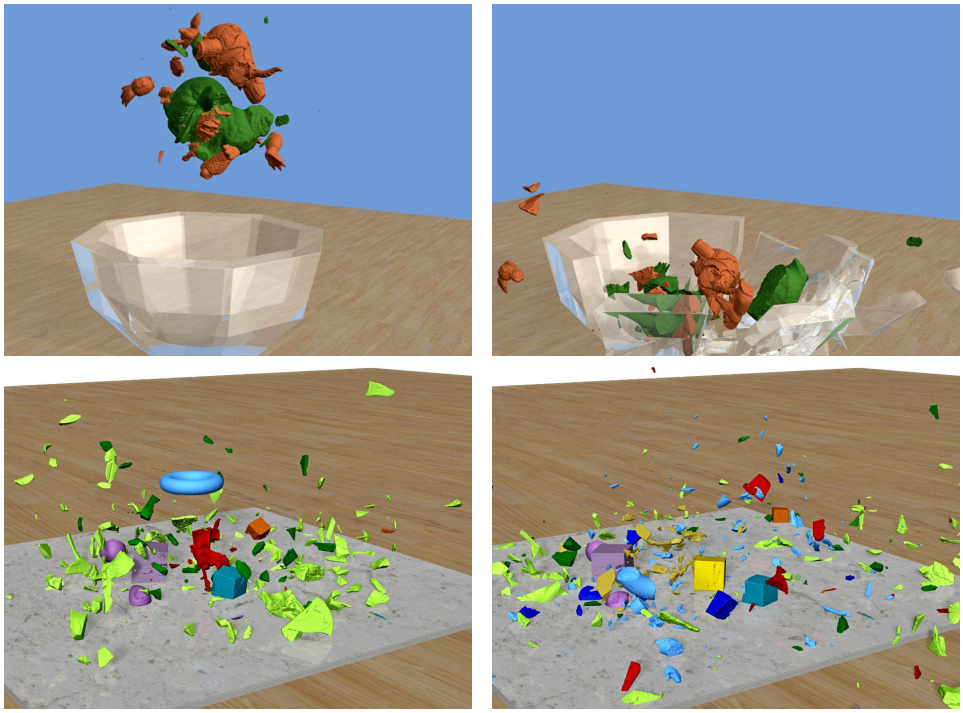


Figure 45: Further results 2: smashing a bunny and an armadillo into a glass bowl, and dropping some geometric primitives on the floor.

Example	Fig.	M	m_s	m_e	voxels	r_c/r_f	steps	t [s]	ci
Cube compressive	13	B	120	1407	339	33,3	22	91,20	*16
Cube controlled	18	B	120	389	205	20	12	7,79	*1
Notched bar (layered h)	19b	B	416	687	1005	30	7	22,11	*3
Notched bar (layered v)	19c	B	416	735	1255	37,5	9	30,22	*3
Edge-cracked cube, II n	23b	E	312	476	339	30	7	10,27	i
Edge-cracked cube, II n	23d	B	312	474	339	30	7	12,35	i
Edge-cracked cube, I	32I	B	312	562	339	26,7	10	16,43	i
Edge-cracked cube, II	32II	B	312	563	339	26,7	10	15,82	i
Edge-cracked cube, III	32III	B	312	464	339	46,7	7	14,75	i
Edge-cracked cube, I	33I	E	312	540	339	26,7	9	11,30	i
Edge-cracked cube, II	33I	E	312	565	339	26,7	11	13,46	i
Edge-cracked cube, III	33III	E	312	490	339	46,7	10	15,06	i
Centre-cracked cube	34b	B	244	1093	339	16,7	11	28,61	i
Centre-cracked cube	34c	E	244	1098	339	16,7	11	13,83	i
Notched bar	35a	B	416	822	1005	30	8	29,01	*5
Notched bar (granular)	35b	B	416	727	1005	30	8	17,35	*3
Notched bar (granular)	35c	E	416	770	1005	30	*9	17,25	*3
Chain link (PMMA)	36a	B	272	431	561	66,7	4	18,40	2
Chain link (concrete)	36b	B	310	630	673	80	8	57,44	7
Armadillo tearing	38	B	1200	1251	763	50	3	57,49	2
Armadillo splitting	39	B	1000	1169	763	50	9	51,85	i
Bunny smashing	40	H	1000	10981	525	20	38	892,25	22

Table 1: Fracture simulation examples (without rigid-body coupling): simulation parameters and runtime. Columns: Name and figure reference, used method (M): either full BEM (B), fast estimator (E), or hybrid method (H), number of elements in the BEM mesh at the start of the simulation m_s and at the end of the simulation m_e, number of voxels (grid nodes) in the implicit surface representation along the largest extent of the object's axis-aligned bounding box, ratio of implicit surface resolution to fracture BEM mesh resolution r_f/r_c (which equals the number of sub steps per BEM time step), number of BEM time steps, total simulation runtime on a 3.2 GHz quad-core desktop processor in seconds t, and number of crack initiation events (ci, 'i' indicates a pre-defined initial fracture). In columns 'steps' and 'ci' a '*' denotes that the user-specified limit of time steps or cracks has been reached.

Example	Fig.	E	ν	ρ	TS	K_c	f_c
Cube compressive	13	2,5E+10	0,2	2300	3,0E+6	5,5E+5	3
Cube controlled	18	3,1E+9	0,327	1200	*	5.0E+5 – 2.0E+6	2
Notched bar (layered h)	19b	3,1E+9	0,327	1200	7,6E+7	1.0E+3 – 5.0E+4	2
Notched bar (layered v)	19c	3,1E+9	0,327	1200	7,6E+7	1.0E+3 – 1.5E+4	2
Edge-cracked cube, II n	23b	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, II n	23d	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, I	32I	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, II	32II	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, III	32III	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, I	33I	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, II	33I	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Edge-cracked cube, III	33III	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Centre-cracked cube	34b	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Centre-cracked cube	34c	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Notched bar	35a	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Notched bar (granular)	35b	3,1E+9	0,327	1200	7,6E+7	1.0E+6 – 1.1E+6	3
Notched bar (granular)	35c	3,1E+9	0,327	1200	7,6E+7	1.0E+6 – 1.1E+6	3
Chain link (PMMA)	36a	3,1E+9	0,327	1200	7,6E+7	1,0E+6	2
Chain link (concrete)	36b	2,5E+10	0,2	2300	3,0E+6	5.5E+5 – 5.6E+5	10
Armadillo tearing	38	3,1E+9	0,327	1200	7,6E+7	1.0E+6 – 1.1E+6	2
Armadillo splitting	39	3,1E+9	0,327	1200	7,6E+7	1.0E+6 – 1.1E+6	2
Bunny smashing	40	3,1E+9	0,327	1200	1,6E+8	5,0E+5	2

Table 2: Fracture simulation examples (without rigid-body coupling): material parameters. Columns: Name and figure reference, Young's modulus E [Pa], Poisson's ratio ν, density ρ [kg/m^3], tensile strength (crack initiation threshold) TS [Pa], tensile fracture toughness (critical stress intensity) K_c [Pa m$^{1/2}$], and compressive factor f_c (defines both compressive strength and toughness relative to tensile values). (*) The tensile strength for the controlled cube example varies vertically between 7.6MPa in the centre and 76MPa on the top and bottom of the cube.

Example	Fig.	M	f	runs	m_{max}	t_f [s]	t_{rb} [s]	steps	$1/dt$
Breaking Π	30	B	105	23	626	181.03	33.22	1000	250
Breaking Π	37	B	122	17	1004	388.58	79.97	1000	250
Breaking Π	37	E	100	13	933	292.73	140.60	1000	250
Breaking Π	37	H	168	16	1225	461.36	108.76	1000	250
Bunny shooting	41	E	1099	5	15381	1328.17	565.92	500	250
Column and bunny	42	E	367	18	7980	966.81	*2628.18	1000	250
Window	43	E	455	7	6459	1256.85	1731.38	1000	250
Brick breaking	44	H	60	26	412	216.11	78.18	500	2000
Armadillos	44	E	148	14	1792	885.95	1710.42	600	250
Bowl	45	E	139	14	3250	845.99	351.32	500	250
Falling objects	45	E	654	77	704	706.43	1498.49	1000	500

Table 3: Fracture simulation examples with rigid-body coupling. Columns: Name and figure reference, used method (M): either full BEM (B), fast estimator (E), or hybrid method (H), number of fragments at the end of the scene f, number of rigid-body collisions causing new fractures (runs), maximum number of triangles contained in a single fracture simulation run m_{max}, total runtime required for all fracture simulations t_f (measured on a 3.2 GHz quad-core desktop processor), total runtime spent on rigid-body dynamics t_{rb} (the '*' indicates that we use a higher collision shape resolution for one of our scenes), number of rigid-body steps, rigid-body frame rate per seconds $1/dt$. All these examples use less than 4GB of memory.

9 Conclusion

In this thesis, we have presented a novel method for simulating brittle fracture dynamics for visual effects purposes. The basis of our approach is a symmetric Galerkin boundary element method. From the BEM solution, we compute stresses on surfaces, as well as stress intensities at crack fronts (where the stress field itself is singular). According to linear elastic fracture mechanics, we simulate crack propagation based on these stress intensities, while using surface stresses only for crack initiation. This separation effectively avoids artificial shattering often seen in purely stress-based fracture simulations. We enable artistic control over our simulations via spatially varying strength and toughness fields.

We sample the crack front at a significantly higher resolution than the BEM mesh when simulating crack propagation. The motion of the propagating crack front then defines the fracture surface. Interpolating the stress intensity factors from the coarse mesh onto the high-resolution crack front allows us to increase the resolution of the fracture surface well beyond the resolution of the BEM mesh, breaking a long-standing restriction of previous fracture simulation methods in computer graphics. This approach, combined with the ability to handle spatially varying toughness, allows us to produce highly-detailed fracture surfaces. Additionally, we have also introduced further approximations to the BEM solution, which significantly improve the efficiency and theoretical complexity of our method.

Consequently, our method is capable of producing realistic fractures at a high resolution in a reasonably short time on commodity hardware (see tables 1 and 3). While one could quickly add visual detail to coarse fracture surfaces in a post-processing step, doing so typically lacks the realism that our results exhibit, as many brittle fracture patterns found in nature arise from the motion of the propagating crack front. Furthermore, the topology of the resulting fragments would be determined by the coarse simulation; in contrast the topology is defined by the high-resolution fractures in our method (otherwise we would not be able to produce the numerous tiny fragments visible in many of our results). In theory, our method could be used to generate fracture surface templates for geometric animation methods as well (for example the methods presented in [49,70]). While we leave this avenue for future work, we instead couple our fracture simulation framework to a standard rigid-body dynamics engine. This coupling simplifies authoring animation sequences involving multiple objects and fracture events due to collisions among them. We show how to treat new fragments as additional rigid bodies in the scene and allow them to fracture further if they experience another collision.

While it is difficult to directly compare to volumetric finite element methods, our experiments showed that a single FEM-based elastostatic solution for the initial state of the edge-cracked cube example (fig. 32) takes roughly 1.2s if the crack front is resolved at a high resolution, or 0.2s for a low-resolution crack front. The tetrahedral mesh adaptively transitions to the same, coarse resolution, as our BEM surface mesh,

away from the crack front. These timings were obtained with the open source FEM implementation available at `http://elmerfem.org` (v. 8.0). Our full BEM solution in the mode-I loaded case (fig. 32I) takes roughly 270 sub steps in about 16s (on the same machine; see table 1), which amortizes the BEM cost to approximately 0.06s per sub step. This indicates that even at low-resolution, the FEM-based fracture simulation would have to work with local updates (as done in 2D in [60]) that take about 30% of a full solve in each sub step to match our runtime (ignoring time spent on meshing operations). While this comparison is by no means exhaustive, it indicates that our approach is a promising alternative to volumetric FEM-based fracture simulation, when using low-resolution surface meshes. However, a finite element method (combined with a fast *sparse* linear solver) can achieve a better runtime complexity than a standard BEM (using a *dense* linear solver) as the mesh resolution increases. Instead of trying to work with known acceleration methods to improve the BEM complexity (such as a fast multipole method [79] or adaptive cross approximation [44]), we decided to introduce even more aggressive approximations in order to address this issue. Instead of approximately solving the entire BEM system, we choose to ignore direct crack-crack interactions completely, which allows us to reduce the size of the BEM system considerably. While our approximations sacrifice accuracy of the deformation solution to a much larger extent than the previously mentioned approaches, they still produce acceptably good estimates for our fracture simulation.

Because our method is based on quasi-static *linear* elasticity, it is inherently inadequate to simulate large deformations or ductile fracture. However, it is well suited for handling brittle fracture in reasonably stiff materials. In such materials, fractures typically propagate very fast (on the order of 1km/s in PMMA or concrete) and hence a high temporal resolution is required to trace the crack front. Similarly, in these materials a fully dynamic deformation method would require very small time steps in order to resolve the rapidly changing stress field around a propagating crack. Instead, our sub-stepping scheme avoids the need to update the BEM solution in every step: we take a fixed number of sub steps (equal to the resolution ratio r_f/r_c) before this update (at the end of the full time step). During sub-stepping, we assume that the stress intensity factors remain constant. These SIFs are specified in the crack front's local coordinate frame, which automatically handles the fact that the stress singularity at the crack front moves with the front as it propagates through the material. While we emphasize that we do not need to change existing elements during our fracture simulation (resulting in less effort spent on BEM matrix assembly and meshing), it could be beneficial to the overall runtime to simplify the mesh of a fracture surface once the crack front has moved on, keeping the number of degrees of freedom as low as possible, when using the full BEM solution (instead of our fast approximations).

In our current implementation, we use a basic single-threaded rigid-body system based on *Bullet* [8] and our coupling to the fracture simulation considers collisions only. For future work, it would be interesting to also include inertial forces, such that fast spinning can cause fractures as well. We only use a standard mesh-based collision detection algorithm, which is prone to producing "popping" artefacts in the motion of

objects when trying to resolve multiple contacts within one frame. This issue could be improved, and also sped up, by convex decomposition approaches [43]. For example, the scene shown in fig. 42 contains some geometrically interlocking fragments, which require fairly high-resolution collision shapes, and consequently slow down the collision detection step of the rigid-body solver considerably. Because we did not implement known rigid-body optimizations, the time spent on rigid-body dynamics (marked with '*' in Table 3) should not be considered to be representative in this case. If rigid-body popping artefacts occur, they can fairly easily be removed by post-processing the motion. We accept these limitations, as we focus on the fracture simulation and consider improving rigid-body dynamics to be outside of the scope of this work.

In our tests (ch. 8.1), we show that our SIF estimation works well for crack propagation and yields results that qualitatively match those of the full BEM approach. However, in the current implementation, our crack initiation method handles branching only if we use the full BEM solution. In theory, we could add an additional crack branching criterion to our fast estimators. For example, during SIF estimation, we choose one eigenvalue (and -vector) of the local regular stress field to compute the stress intensities; we could similarly use (one of) the remaining two eigenvalues, which correspond to eigenvectors close to the crack surface's tangent plane, to implement a branching criterion. Furthermore, our fast estimators also ignore direct crack-crack interactions, although some analytical results exist describing these effects (sometimes referred to as "transmission factors" [22]). Even though our approximations improve the overall runtime of the simulation considerably, we still need to solve (at least) one BEM system at the start of the fracture simulation. In future work, it might be possible to further speed up this initial BEM solution by using a pre-computed proxy shape, similar to the "sound proxy" of [78].

In summary, our new approach to brittle fracture, based on boundary elements and Lagrangian crack propagation, provides the first method in computer graphics for simulating fractures, in the presence of material toughness variations, beyond the resolution of the underlying deformation method. Additionally, we have presented a way to approximate stress intensities and crack opening displacements in order to speed up crack propagation even further. Because it focuses the computational effort on the high-resolution crack-front geometry, our method is capable of efficiently generating extremely detailed and physically realistic fracture surfaces. This approach results in an efficient physics-based simulation of brittle fracture, which is both faster in practice and has theoretically superior scaling to previous finite or boundary element methods. In light of the fact that any method that generates a fracture surface mesh must iterate over each node in that mesh at least once, we consider the linear scaling of our fast approximate crack propagation algorithm to be optimal.

References

[1] Abdelaziz, Y., and Hamouine, A. A survey of the extended finite element. In *Computers & Structures* (2008), vol. 86, pp. 1141–1151.

[2] Aliabadi, M. H. Boundary element formulations in fracture mechanics. In *Applied Mechanics Reviews* (1997), vol. 50, American Society of Mechanical Engineers, pp. 83–96.

[3] Becker, W. T., and Lampman, S. Fracture appearance and mechanisms of deformation and fracture. In *Materials Park, OH: ASM International* (2002), pp. 559–586.

[4] Bender, J., Erleben, K., Trinkle, J., and Coumans, E. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. In *EUROGRAPHICS 2012 State of the Art Reports* (2012).

[5] Bernstein, D. S. *Matrix Mathematics: Theory, Facts, and Formulas with Application to Linear Systems Theory*. Princeton University Press, 2005.

[6] Bower, A. F. *ENGN2210 Lecture Notes on Continuum Mechanics*. Brown University, 2012.

[7] Chen, Z., Yao, M., Feng, R., and Wang, H. Physics-inspired Adaptive Fracture Refinement. In *ACM Trans. Graph.* (2014), vol. 33, pp. 113:1–113:7.

[8] Coumans, E. Bullet Physics Simulation. In *ACM SIGGRAPH 2015 Courses* (2015).

[9] Danson, D. Linear isotropic elasticity with body forces. In *Progress in Boundary Element Methods: Volume 2*, C. A. Brebbia, Ed. Springer, 1983, pp. 101–135.

[10] DREAM3D. A Digital Representation Environment for the Analysis of Microstructure in 3D. v. 4.2.4, `http://dream3d.bluequartz.net`.

[11] DreamWorks Animation. OpenVDB. v. 2.2.0, `http://www.openvdb.org`.

[12] DreamWorks Animation. OpenVDB Release Notes. Accessed 20.06.2017, `http://www.openvdb.org/documentation/doxygen/changes.html`.

[13] Duffy, M. G. Quadrature Over a Pyramid or Cube of Integrands with a Singularity at a Vertex. In *SIAM Journal on Numerical Analysis* (1982), vol. 19, pp. 1260–1262.

[14] Elek, P., and Jaramaz, S. Fragment size distribution in dynamic fragmentation: Geometric probability approach. In *FME Transactions* (2008), vol. 36, pp. 59–65.

[15] Feynman, R. P., Leighton, R. B., and Sands, M. *The Feynman Lectures on Physics: Volume 2.* Addison-Wesley, 1963.

[16] Frangi, A., Novati, G., Springhetti, R., and Rovizzi, M. 3D fracture analysis by the symmetric Galerkin BEM. In *Computational Mechanics* (2002), vol. 28, Springer, pp. 220–232.

[17] Freund, L. B. *Dynamic Fracture Mechanics.* Cambridge Monographs on Mechanics. Cambridge University Press, 1998.

[18] Garland, M., and Heckbert, P. S. Surface Simplification Using Quadric Error Metrics. In *SIGGRAPH 97, Annual Conference Series* (1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[19] Glondu, L., Marchal, M., and Dumont, G. Real-Time Simulation of Brittle Fracture Using Modal Analysis. In *IEEE TVCG* (2013), vol. 19, pp. 201–209.

[20] Gravouil, A., Moës, N., and Belytschko, T. Non-planar 3D crack growth by the extended finite element and level sets – part II: level set update. In *INT J NUMER METH ENG* (2002), vol. 53, John Wiley & Sons, Ltd., pp. 2569–2586.

[21] Griffith, A. A. The Phenomena of Rupture and Flow in Solids. In *Philosophical Transactions of the Royal Society of London* (1921), vol. 221, pp. 163–198.

[22] Gross, D., and Seelig, T. *Fracture Mechanics*, 2nd ed. Springer, 2011.

[23] Guennebaud, G., Jacob, B., et al. Eigen, 2010. v. 3, `http://eigen.tuxfamily.org`.

[24] Guiggiani, M. The evaluation of cauchy principal value integrals in the boundary element method – a review. In *Mathematical and Computer Modelling* (1991).

[25] Hertz, H. Über die Berührung fester elastischer Körper. In *J. reine und angewandte Math.* (1881), vol. 92, pp. 156–171.

[26] Hibbeler, R. C. *Mechanics of Materials*, 8th ed. Pearson Prentice Hall, 2010.

[27] Holzapfel, G. A. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering.* Wiley, 2000.

[28] Houston, B., Nielsen, M. B., Batty, C., Nilsson, O., and Museth, K. Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation. In *ACM Trans. Graph.* (2006), vol. 25, pp. 151–175.

[29] HyENA. Hyperbolic and Elliptic Numerical Analysis, Graz University of Technology. v. 04.11.2013, `http://www.mech.tugraz.at/HyENA`.

[30] Iben, H. N., and O'Brien, J. F. Generating surface crack patterns. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2006), pp. 177–185.

[31] Ingraffea, A., and Wawrzynek, P. Finite Element Methods for Linear Elastic Fracture Mechanics. In *Comprehensive Structural Integrity, Volume 3: Numerical and Computational Methods*, R. de Borst and H. A. Mang, Eds. Elsevier, 2003.

[32] Irving, G., Teran, J., and Fedkiw, R. Invertible Finite Elements for Robust Simulation of Large Deformation. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2004), pp. 131–140.

[33] Irwin, G. R. Analysis of Stresses and Strains Near the End of a Crack Traversing a Plate. In *Journal of Applied Mechanics* (1957).

[34] James, D. L., and Pai, D. K. ArtDefo: Accurate Real Time Deformable Objects. In *SIGGRAPH 99, Annual Conference Series* (1999), pp. 65–72.

[35] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual Contouring of Hermite Data. In *ACM Trans. Graph.* (2002), vol. 21, pp. 339–346.

[36] Kaufmann, P., Martin, S., Botsch, M., Grinspun, E., and Gross, M. Enrichment Textures for Detailed Cutting of Shells. In *ACM Trans. Graph.* (2009), vol. 28, pp. 50:1–50:10.

[37] Keeler, T., and Bridson, R. Ocean Waves Animation Using Boundary Integral Equations and Explicit Mesh Tracking. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2014), pp. 11–19.

[38] Kielhorn, L. *A time-domain symmetric Galerkin BEM for viscoelastodynamics.* Verl. der Techn. Univ. Graz, 2009.

[39] Klár, G., Gast, T., Pradhana, A., Fu, C., Schroeder, C., Jiang, C., and Teran, J. Drucker-Prager Elastoplasticity for Sand Animation. In *ACM Trans. Graph.* (2016), vol. 35, pp. 103:1–103:12.

[40] Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature Sensitive Surface Extraction from Volume Data. In *Proc. ACM SIGGRAPH 2001* (2001), pp. 57–66.

[41] Koschier, D., Lipponer, S., and Bender, J. Adaptive Tetrahedral Meshes for Brittle Fracture Simulation. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2014), pp. 57–66.

[42] Larson, M. G., and Bengzon, F. *The Finite Element Method: Theory, Implementation, and Applications.* Springer, 2013.

[43] Mamou, K., and Ghorbel, F. A simple and efficient approach for 3D mesh approximate convex decomposition. In *16th IEEE Int. Conf. Image Processing (ICIP)* (2009), pp. 3501–3504.

[44] Messner, M., and Schanz, M. An accelerated symmetric time-domain boundary element formulation for elasticity. In *Engineering Analysis with Boundary Elements* (2010), vol. 34, pp. 944–955.

[45] Moës, N., Gravouil, A., and Belytschko, T. Non-planar 3D crack growth by the extended finite element and level sets – part I: mechanical model. In *INT J NUMER METH ENG* (2002), vol. 53, John Wiley & Sons, Ltd., pp. 2549–2568.

[46] Molino, N., Bao, Z., and Fedkiw, R. A Virtual Node Algorithm for Changing Mesh Topology During Simulation. In *ACM Trans. Graph.* (2004), vol. 23, pp. 385–392.

[47] Monegato, G. Numerical evaluation of hypersingular integrals. In *Journal of Computational and Applied Mathematics* (1994).

[48] Mousavi, S. E., Grinspun, E., and Sukumar, N. Higher-order extended finite elements with harmonic enrichment functions for complex crack problems. In *INT J NUMER METH ENG* (2011), vol. 86, John Wiley & Sons, Ltd., pp. 560–574.

[49] Müller, M., Chentanez, N., and Kim, T.-Y. Real Time Dynamic Fracture with Volumetric Approximate Convex Decompositions. In *ACM Trans. Graph.* (2013), vol. 32, pp. 115:1–115:10.

[50] Müller, M., and Gross, M. Interactive Virtual Materials. In *Proc. Graphics Interface* (2004), pp. 239–246.

[51] Müller, M., McMillan, L., Dorsey, J., and Jagnow, R. Real-time Simulation of Deformation and Fracture of Stiff Materials. In *Proc. Eurographic Workshop on Computer Animation and Simulation* (2001), pp. 113–124.

[52] Museth, K. VDB: High-resolution Sparse Volumes with Dynamic Topology. In *ACM Trans. Graph.* (2013), vol. 32, pp. 27:1–27:22.

[53] Norton, A., Turk, G., Bacon, B., Gerth, J., and Sweeney, P. Animation of Fracture by Physical Modeling. In *The Visual Computer* (1991), vol. 7, Springer-Verlag New York, Inc., pp. 210–219.

[54] O'Brien, J. F., Bargteil, A. W., and Hodgins, J. K. Graphical Modeling and Animation of Ductile Fracture. In *ACM Trans. Graph.* (2002), vol. 21, pp. 291–294.

[55] O'Brien, J. F., and Hodgins, J. K. Graphical Modeling and Animation of Brittle Fracture. In *SIGGRAPH 99, Annual Conference Series* (1999), pp. 137–146.

[56] Osher, S., and Fedkiw, R. Signed Distance Functions. In *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003, ch. 2.

[57] Parker, E. G., and O'Brien, J. F. Real-time Deformation and Fracture in a Game Environment. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2009), pp. 165–175.

[58] Patricio, M., and Mattheij, R. Crack propagation analysis. In *CASA-report 0723* (2007).

[59] Pauly, M., Keiser, R., Adams, B., Dutré, P., Gross, M., and Guibas, L. J. Meshless Animation of Fracturing Solids. In *ACM Trans. Graph.* (2005), vol. 24, pp. 957–964.

[60] Pfaff, T., Narain, R., de Joya, J. M., and O'Brien, J. F. Adaptive Tearing and Cracking of Thin Sheets. In *ACM Trans. Graph.* (2014), vol. 33, pp. 110:1–110:9.

[61] Popov, V. L. Rigorous Treatment of Contact Problems – Hertzian Contact. In *Contact Mechanics and Friction*, Physical Principles and Applications. Springer, 2010, ch. 5.

[62] Portela, A., Aliabadi, M. H., and Rooke, D. P. The Dual Boundary Element Method: effective implementation for crack problems. In *INT J NUMER METH ENG* (1992), vol. 33, John Wiley & Sons, Ltd, pp. 1269–1287.

[63] Rabczuk, T. Computational Methods for Fracture in Brittle and Quasi-Brittle Solids: State-of-the-Art Review and Future Perspectives. In *ISRN Applied Mathematics* (2013), vol. 2013, p. Article ID 849231.

[64] Rice, J. R. A Path Independent Integral and the Approximate Analysis of Strain Concentration by Notches and Cracks. In *J. Appl. Mech* (1968).

[65] Sauter, S., and Schwab, C. *Boundary Element Methods*. Springer, 2011.

[66] Schvartzman, S. C., and Otaduy, M. A. Fracture Animation Based on High-dimensional Voronoi Diagrams. In *Proc. 18th ACM SIGGRAPH i3D '14* (2014), pp. 15–22.

[67] Smith, J., Witkin, A., and Baraff, D. Fast and Controllable Simulation of the Shattering of Brittle Objects. In *Computer Graphics Forum* (2001), vol. 20, Blackwell Publishers Ltd, pp. 81–91.

[68] Stomakhin, A., Schroeder, C., Chai, L., Teran, J., and Selle, A. A Material Point Method for Snow Simulation. In *ACM Trans. Graph.* (2013), vol. 32, pp. 102:1–102:10.

[69] Stomakhin, A., Schroeder, C., Jiang, C., Chai, L., Teran, J., and Selle, A. Augmented MPM for Phase-change and Varied Materials. In *ACM Trans. Graph.* (2014), vol. 33, pp. 138:1–138:11.

[70] Su, J., Schroeder, C., and Fedkiw, R. Energy Stability and Fracture for Frame Rate Rigid Body Simulations. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2009), pp. 155–164.

[71] Sutradhar, A., Paulino, G., and Gray, L. J. *Symmetric Galerkin Boundary Element Method*. Springer, 2008.

[72] Terzopoulos, D., and Fleischer, K. Modeling Inelastic Deformation: Viscolelasticity, Plasticity, Fracture. In *SIGGRAPH Comput. Graph.* (1988), vol. 22, ACM, pp. 269–278.

[73] VCGLib. The VCG Library, Visual Computing Lab, CNR-ISTI. Accessed 15.04.2014, `http://vcg.isti.cnr.it/~cignoni/newvcglib/html`.

[74] Wang, Y., Jiang, C., Schroeder, C., and Teran, J. An Adaptive Virtual Node Algorithm with Robust Mesh Cutting. In *Proc. ACM SIGGRAPH/Eurographics SCA* (2014), pp. 77–85.

[75] Westergaard, H. M. Bearing Pressures and Cracks. In *Journal of Applied Mechanics* (1939).

[76] Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., and O'Brien, J. F. Dynamic Local Remeshing for Elastoplastic Simulation. In *ACM Trans. Graph.* (2010), vol. 29, pp. 49:1–49:11.

[77] Wilde, A. J., and Aliabadi, M. H. A 3-D Dual BEM formulation for the analysis of crack growth. In *Computational Mechanics* (1999), vol. 23, Springer-Verlag, pp. 250–257.

[78] Zheng, C., and James, D. L. Rigid-body Fracture Sound with Precomputed Soundbanks. In *ACM Trans. Graph.* (2010), vol. 29, pp. 69:1–69:13.

[79] Zhu, Y., Bridson, R., and Greif, C. Simulating Rigid Body Fracture with Surface Meshes. In *ACM Trans. Graph.* (2015), vol. 34, pp. 150:1–150:11.

Appendix

A List of fracture simulation parameters

Simulation method

(1) Choose between solving the full BEM system in every time step or using our fast estimator.
(2) Hybrid method: set (1) to full BEM and define the number of elements in the BEM mesh beyond which we switch to the fast estimator.

Material

(3) E: Young's modulus
(4) ν: Poisson's ratio
(5) ρ: Density
(6) TS: Tensile Strength
(7) K_c: Toughness (critical tensile stress intensity)
(8) f_c: Ratio of compressive to tensile strength and toughness respectively
(9) Choose a spatial modifier for strength and/or toughness. (Optional)

Input

(10) Input file(s): can be either a (coarse) BEM mesh with enumerated boundary regions, or a high-resolution triangle mesh, with an additional text file specifying boundary regions (based on a convex intersection of half-spaces).
(11) Boundary conditions: for each boundary region, specify either a fixed, constant displacement (Dirichlet boundary) or a constant traction (Neumann boundary). Note that every boundary element must be contained in exactly one boundary region, and every such region must have exactly one boundary condition for the problem to be well defined.
(12) Location and orientation of small initial cracks. (Optional)
(13) The target number of triangles in the coarse BEM mesh if parameter (10) refers to a high-resolution mesh.
(14) An offset distance (in multiples of parameter (16), the level-set resolution) by which the BEM mesh vertices are moved along their outward normal after meshing. This parameter can be used to avoid the BEM mesh being inside of the high-resolution surface of the object after mesh simplification.

Resolution and output

(15) r_c: The resolution (target edge length) of the coarse BEM mesh representation of fracture surfaces.
(16) r_f: The resolution (voxel size) of all level-set grids.
(17) s: The segmentation threshold for robustly detecting fragments and writing each fragment to a separate output mesh file.
(18) Output file name (without extension); output consists of both a VTK mesh and VDB level-set data per full time step.

(19) Choose whether to output additional level-set data for each sub step as well.
(20) The maximum number of full time steps.
(21) The maximum number of cracks (prevents further crack initiation if reached).
(22) Post-processing: quality of high-resolution output meshes.
(23) Post-processing: choose whether to remove gaps introduced by the level-set representation from fractures in high-resolution output meshes.
(24) Post-processing: choose whether to add interpolated displacements to high-resolution output meshes.
(25) Post-processing: if (24) is set, choose whether to also add interpolated crack-opening displacements to high-resolution output meshes.

B Pseudo-code listings

```
GridList findSegments(Grid input, double s, bool useTiles)
  for each( value in input )
    if(useTiles || value is voxel)
      value += s
      setActiveState( value <= 0 ) // set inside values to "active"
  while( input has active values )
    segment = startNewSegment()
    queue   = emptyQueue()
    queue.push( input.firstActiveCoord() )
    while( queue not empty )
      coord = queue.pop()
      if( segment[coord] is active ) continue;
      if( input[coord] is voxel )
        segment[coord] = input[coord] // copy from input
        for( int n = 0; n < 26; n++ ) // check all 26 neighbours
          if( neighbour[n] is active in input )
            // neighbour[n] is inside
            input.deactivate( neighbour[n] )
            // if neighbour[n] is a tile, deactivate the whole tile
            if( neighbour[n] is not active in segment )
              queue.push( neighbour[n] )
          else if( neighbour[n] is not active in segment )
            // neighbour[n] is outside
            segment[ neighbour[n] ] = input[ neighbour[n] ]
            segment.activate( neighbour[n] )
      else // input[coord] is a tile
        bndBox = input.getTileBoundingBox(coord)
        n_ijks[0] = bndBox.min() + (-1, 0, 0)
        n_ijks[1] = bndBox.min() + ( 0,-1, 0)
        n_ijks[2] = bndBox.min() + ( 0, 0,-1)
        n_ijks[3] = bndBox.max() + ( 1, 0, 0)
        n_ijks[4] = bndBox.max() + ( 0, 1, 0)
        n_ijks[5] = bndBox.max() + ( 0, 0, 1)
        for( int n = 0; n < 6; n++ ) // check 6 neighbours at corners
          // process neighbour as above ...
    segment.signedFloodFill() // fill in signs for interior regions
    // activate voxels slightly outside of active region
    segment.dilateVoxels( 1+round( s/r_f ) )
    for each( active voxel in segment )
      if( voxel.value is unset ) // activated by dilateVoxels
        if( input[voxel.coord()] > s ) // value-s remains positive
          voxel.value = input[voxel.coord()] // copy from input
      voxel.value -= s
    foundSegments.push( segment )
  return foundSegments
```

Listing 1: Segmenting an OpenVDB level-set input grid into separate output grids, optionally using tile-based breath-first search and an over-segmentation threshold s of up to 1 voxel size.

```
writeVisualMesh( string file, bool addDisp, bool addCOD, bool doClose )
  fracturedObject = object.copy()
  for each( grid in fractures )
    fracturedObject = csgIntersection( fracturedObject, grid )

  segments = findSegments( fracturedObject, s, true )
  for each( segment in segments )
    mesh = VolumeToMesh( segment )

    if( doClose ) // remove "gaps" due to SDF representation of fractures
      for each( p in mesh.vertices )
        q = findClosestPointOnFracture()
        // we use OpenVDB's ClosestSurfacePoint tool
        closing[p] = q - p

    if( addDisp ){
      for each( p in mesh.vertices )
        triList = findNearbyBEMtriangles( p ) // within distance of r_c
        closestTriangle  = findClosestTriangle(    p, triList )
        u_c[p] = interpolateContinousDisplacement( p, closestTriangle )
        if( addCOD )
          cod[p] = 0
          for each( fracture in listFractures( triList ) )
            // listFractures extracts a set of unique IDs
            // of fracture surfaces occuring in triList
            closestTriangle = findClosestTriangle(
              p, matchingTris( triList, fracture )
            )
            cod[p] += interpolateCOD( p, fracture, closestFractureTri )
            // interpolateCOD chooses the sign based on
            // the high-resolution fracture surface normal and
            // data from neighbouring output mesh vertices
          p += cod[p] // add crack-opening displacement (discontinuous)
        p += u_c[p]   // add continuous displacement

    if( doClose )
      p += closing[p] // add gap-closing displacement

    writeMeshFile( file.append(segment.ID), mesh )
```

Listing 2: Building output meshes for all segments, optionally adding continuous displacements, crack-opening displacements, and removing gaps introduced by the signed distance function representation of fractures.