A Thesis for the Degree of Ph.D. in Engineering

Quasi-reversible Weathering of Rust Preventive Coating Films

July 2023

Graduate School of Science and Technology
Keio University

Akinori Ishitobi

Thesis Abstract

Quasi-reversible Weathering of Rust Preventive Coating Films

Age-related deterioration inevitably affects everything in reality. Changes in the appearance of deteriorated objects signify the passage of time to our brain, and the degree of deterioration is a meaningful factor in determining the "age" of objects at a glance. Conversely, a virtual world rendered by computer graphics (CG) differs from reality in that it is inherently unaffected by deterioration. Therefore, bridging this gap between the two worlds by assigning an "age" to CG objects necessitates the application of weathering, which expresses degradation caused by rain and wind.

Metallic corrosion is one of the commonest examples of real-life deterioration, and many weathering methods have been proposed. However, in reality, vulnerable metal is usually coated with rust preventive paint, so scenes in which metal alone is deteriorated are highly limited. In addition, most existing weathering methods focus on changing the material, while few deal with deformation. Most weathering methods use physical simulation, which computes the time evolution of the state of an object based on a mathematical model and which has the advantage of easily reproducing complex phenomena. However, when generating CG content, this one-way time progression becomes a major problem, so people generally use parametric models that allow the flexible control of time.

In this study, a novel method is proposed for the procedural simulation of cracks and bends in coated films on metallic objects. In the proposed method, a coated film is imposed on a 3D triangular mesh. Fractures are expressed by topological manipulation based on a simple computation of force, whereas the mesh is bent by position-based simulation. As further defacing elements on the coating film, chalking, runoff rust, and darkening are reproduced by changing the color and reflectance of the surface. In addition, a pseudo time reversal of the simulation is achieved by defining reverse time progressions of these processes, and directable weathering is realized by locally adjusting the control parameters of designated surface areas.

The proposed method enabled the simulation of realistic patterns of cracks and peeled areas. Thus, it was verified that the method is applicable to external polygon mesh models, and the interactive control of both the time progression and the reversal of the weathering simulation was confirmed.

Keywords

Computer graphics; visual simulation; weathering; interaction; coating; rust; deformation.

Acknowledgments

First, I would like to express my deepest gratitude to Prof. Issei Fujishiro for his continuous guidance. He has been advising me on the development of my research and the improvement of my academic skills ever since I began my research five years ago. Without his support, I would neither have skipped a grade, been accepted to a prestigious international journal, nor received numerous awards; most importantly, I would not have had so much confidence in myself. I feel that my time in his laboratory was the most fulfilling and enjoyable in my life thus far. I hope to apply his teachings to my social activities, which will allow me to demonstrate my abilities and enjoy myself.

My appreciation also goes to Mr. Masanori Nakayama of Fujishiro Laboratory for his assistance with my research as a co-author in the publications. He proposed the fundamental idea and basic policy of the study. In addition, he taught me the fun of studying CG and standard techniques, about which I knew nothing at the time.

I would like to express special gratitude to Prof. Tokiichiro Takahashi of Tokyo Denki University, Prof. Hideo Saito of Keio University, and Prof. Maki Sugimoto of Keio University, all of whom, as my committee members, offered insightful comments and helped me improve my study from various perspectives.

Finally, I would like to thank all members of Fujishiro Laboratory, especially the Reality Modeling team members, thanks to whom I was able to lead a fulfilling campus life and who often gave me sound advice.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This chapter introduces this thesis. **Section 1.1**, **1.2**, and **1.3** present the background of the thesis work. **Sec. 1.4** and **1.5** clarify the purpose and contributions of this research, respectively. Finally, **Sec. 1.6** overviews the thesis organization.

1.1 Visual Simulation

The main purpose of visual simulation is to reproduce plausibly the appearance of natural phenomena using computer graphics (CG). Thus, improvements to CG realism not only enhance the quality of works in art and entertainment but also reinforce the information transmission ability of visual media to illustrate specific information to people, such as in disaster simulation.

The most basic policy for realizing photorealistic CG is the use of physical simulation, which involves rigorous numerical computations based on the laws of nature. It ideally seems that the perfect reproduction of any natural laws in a virtual world leads to the photorealistic reproduction of the appearance of phenomena, but it is impossible in practice. The biggest reason is that the real world is too complex to construct governing equations of most phenomena to explain how they look. For example, turbulent flow, a common elemental phenomenon, is generally believed to obey Navier-Stokes equations, but this has not been rigorously analyzed yet. Even if the velocity field and density field in the turbulent flow were completely identifiable, optical analysis would be necessary to clarify how the result affects the rays of light reaching our eyes. Strictly speaking, the physiological aspect of how the stimuli from the rays would be processed cannot be ignored.

Therefore, many mathematical models have been introduced for visual simulation. Indeed, physical simulation also depends on mathematical models, but they are well-grounded in physics, and most are derived from minimum principles that must be accepted as preconditions. Conversely, the models in the visual simulation are not necessarily physical and are sometimes phenomenological and/or artificial, that is, they often explain just a phenomenon's appearance, expressed as results without consideration of the underlying mechanisms. Perlin noise [80], which generates continuous and random distributions by random sampling and vector operation, is a representative example of the artificial model. Perlin noise is used very frequently to generate photorealistic representations of natural objects, such as clouds, fire, waves, and terrain, even though it has no physical significance.

While optical analysis is typically necessary for the rigorous reproduction of appearance, three-dimensional (3D) CG usually accepts a mathematical model in which light is modeled as rays composed of three primary colors (red, green, and blue) and light behavior is approximated geometrically. The three primary colors correspond with the frequency band of electromagnetic waves, which activate three types of cone cells within the human retina, so the model is based on physiology. Although it still contains bold approximations and assumptions, the basic pipeline for drawing 3D subjects produced by visual simulation is set up, in a way. Indeed, if the result

of physical simulation is a 3D object, it can be displayed through the same pipeline. However, conversion to a 3D model and the drawing process may face a bottleneck concerning realism, and the difference in accuracy between thesimulation and drawing would be large. In this case, computational resources assigned to physical simulations are not effectively utilized. Therefore, visual simulation must adopt simplified models that properly consider the rendering capabilities of the pipeline and computational resources.

1.2 Weathering of Coating Films

Age-related deterioration inevitably affects everything in reality. Changes to the appearance of deteriorated objects signify to our brain the passage of time, and the degree of deterioration is a meaningful factor suggesting the "age" of an object at a glance. Conversely, virtual worlds rendered by CG differ from reality in that they are inherently unaffected by deterioration. Therefore, bridging this gap between the two worlds by assigning an "age" to CG objects necessitates the application of weathering, which expresses degradation caused by rain and wind.

Metallic corrosion is one of the commonest examples of real-life deterioration, and many weathering methods have been proposed to achieve this look. However, in reality, vulnerable metal is usually coated with rust-proof paint, so scenes in which metal alone deteriorates are highly limited. In addition, most existing weathering methods focus on changing the material, whereas few deal with deformation. Some methods are able to express 3D fissures in rocks or the peeling of coating films based on the result of a crack simulation in a two-dimensional (2D) texture space, but the deformation remains superficial.

Aged degradation is a comprehensive phenomenon that progresses through extremely complex interactions among various factors, and it is almost impossible to achieve by physical simulation. Particularly, extended time variations in physical properties are highly difficult to measure. Moreover, although peeled coating films are often observed in reality, they are undesirable from a safety standpoint, and engineering interest is directed to the process before deformation. Thus, there are no sufficient data to reproduce the process of coating films bending after cracking. Therefore, the visual simulation of aged degradation must involve the construction of mathematic models that describe the time variations in physical properties.

1.3 Interactive CG

The rapid development of extended reality (XR) and the metaverse presently has expanded the opportunities for people to experience virtual CG objects up close. To avoid spoiling the immersive experience, the generation and drawing of high-resolution objects that look photorealistic from any distance or angle are increasingly becoming required. In addition, because CG objects, which were mostly observed through rectangular displays, are shown as if they existed right in front of our eyes, it is expected to control the CG objects much more intuitively.

Such high-resolution CG and interactive control are important topics, not only in relation to XR and the metaverse, but also in relation to regular CG systems, and particularly, interactive control is essential for content generation tools. While content generation through visual simulation has the advantage of easily and automatically reproducing natural objects without special skills or knowledge of the user, it has the disadvantage of making it difficult to reflect user intentions. It is thus desirable to improve the directability (the property that allows users to manipulate the simulation process intuitively, including the usability of the user interface) by receiving the user's intended controls to deal with the disadvantage. However, while the next state can reflect intended control, simulation generally cannot restore an object to its previous state, because states are updated by iterating computations of parameters. A complete record of all states allows the time from the initial state to the final simulated state to be manipulated at will, but it also requires significant computational resources.

1.4 Purpose of Thesis Work

The main purpose of this thesis is to represent and simulate visually the aged degradation of coating films, which is difficult to reproduce by physical simulation using simple mathematical models. Cracks and bends of coating films are expressed by deforming a 3D polygon mesh, in consideration of mechanical effects. Moreover, stains on the surfaces of coating films are also expressed in consideration of the model shape. To improve the simulation directability, pseudo time reversal of the degradation is achieved, and simulation results for the proposed method are shown in **Fig. 1.1**.

Note that this thesis proposes a visual simulation method that aims to depict plausibly the appearance of the phenomenon, rather than to reproduce it in a physically accurate manner. This method is intended primarily for the generation of entertainment content, such as video game development, digital art production, and pre-visualization special effects for movies. In addition, unique applications of such a proposed method as a weathering technique must include a landscape assessment and the generation of analytic images that warn of the aging of social infrastructure.

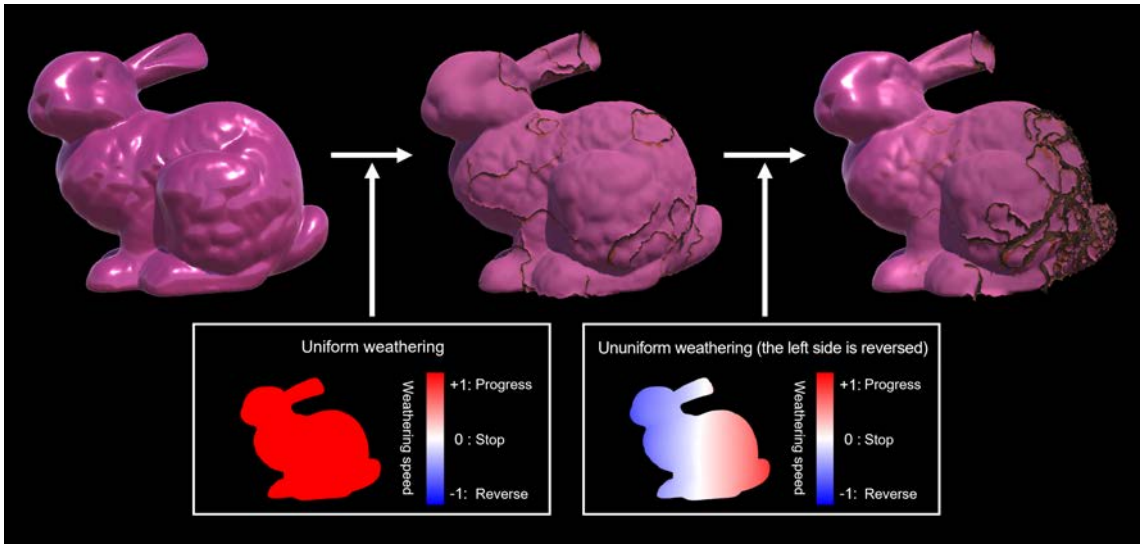


Fig. 1.1 Results of visual simulation using the proposed weathering method. By weathering the initial model (left), an aged model (middle) is obtained. In addition, partial time reversal of the aged model gives an ununiformly aged model (right)

1.5 Contributions of Thesis Work

The contributions of this thesis are five-fold, as follows:

1. Reproduction of the gradual bending of coating films by applying position-based deformation dependent on the fracture condition;
2. Statics-based fracture determination and topological manipulation of triangular polygon mesh to represent plausible cracks on coating films, as well as the generation of various peeling patterns using several control parameters;
3. Expression of stains including rust run-off from peeled areas;
4. Quasi-reversible weathering of coating films by defining a reversal process for each of the above three weathering processes; and
5. Directable weathering, where the simulation state is displayed in real time and external input during the simulation is accepted to enable interactive control.

Parts of this study were published as [MP-1], [MP-2], [RP-1], [RP-2], and [RP-3].

1.6 Organization of Thesis

The remainder of this thesis is organized as follows:

Chapter 2 Related Work

Prior works related to weathering and simulation are surveyed.

Chapter 3 Method Overview

An overview of the proposed method is presented.

Chapter 4 Data Structure

A data structure to realize the proposed method is introduced.

Chapter 5 Bend Simulation

The position-based deformation simulation to express film bending is explained.

Chapter 6 Fracture Simulation

The fracture simulation, considering force balance and topological manipulation, is presented.

Chapter 7 Stain Expression

The expression of stains caused by external effects is explained.

Chapter 8 Interactive Control

Interactive control, including the time reversal of weathering, is detailed.

Chapter 9 Results and Discussions

The results of the simulation using the proposed methods and discussions are provided.

Chapter 10 Conclusion

This chapter concludes the thesis.

Chapter 2

Related Work

This chapter gives an overview of existing simulation works related to the proposed method. **Section 2.1** introduces weathering methods to express aged degradation in CG. Then, **Secs. 2.2** and **2.3** present methods to simulate solid deformation and interactive simulation methods, respectively. Finally, the strengths of the proposed method are highlighted in **Sec. 2.4**, in comparison with related works.

2.1 Weathering Simulation

Everything in reality is affected by age-related deterioration, and to reflect this, the surveys by Merillou and Ghazanfarpour [63], Muguercia et al. [67], and Frerich et al. [29] introduced many known attempts to express changes in appearance due to aged deterioration. Weathering has been applied to various objects, including paper, corks [55], leaves [49], wood [57], stones [23][77][106], bricks [87], and concrete [99]. Crack generation methods [22][28][98] are also proposed for stiff materials. Furthermore, as examples of human aging, there exist face age manipulation by deep learning [1] and hair deterioration in consideration of biological features [2].

Metal has also become the primary target of CG weathering, and the method for generating metallic patinas proposed by Dorsey and Hanrahan [24] is one of the most recognized weathering techniques. Meanwhile, metal corrosion [45][46][62], the synthesis of rust in seawater [13], and rust generation considering the physical and chemical properties of water [44] are other examples of metal weathering. In addition, the method of generating rust texture by cell automata [94] is applied to voxels to represent cubic metal corrosion [92][93].

Conversely, few weathering methods have targeted coated objects. Paquette et al. [79] and Gobron et al. [31] demonstrate the peeling of coating films by generating cracks and bending in the surrounding films. However, cracks are simulated in texture space, while bending is simulated using a 3D polygon mesh, so combining both processes is not straightforward. As a result, only coating films near cracks bend, whereas large warps cannot be expressed.

Dust accumulation on the surface of objects [40] is another example of weathering whose simulation deals with phenomena on material surfaces rather than changes to the material. Specific objects that should be considered in the simulation of dust accumulation include fur [52], carpets [51], and cities [66]. Dust accumulation on solar panels [72] is another important topic in engineering because it affects the efficiency of energy production. In addition, a method to reproduce characteristic patterns caused by water currents conveying sediments [25] has been proposed, and there also exist an image-based method [8] and an interactive editing system [9][26] to express such phenomena. Further, weathering methods that deal with biological factors include moss propagation [76] and lichen growth [21].

As examples of versatile weathering, methods for estimating the deterioration process with an appearance manifold [101][107] and the weathering of a single image [11][27][42] have

been proposed, among which are many texture-generation methods, such as those considering the cubic shape of the object [33][60][103] and pseudo time-manipulation by processing a single texture [5][34][85]. However, these methods manipulate the material of object surfaces or objects in images, but they do not enable 3D deformation. In response, γ-ton tracing [16] is a generalized method introduced to estimate the likely weathering areas throughout the 3D space, but it must be combined with other techniques to express spatial deterioration.

2.2 Deformation Simulation

Deformation simulation is essential to 3D CG animations that must reproduce realistic object behaviors, and surveys by Nealen et al. [71] and Huang et al. [41] mention several types of simulation approaches.

The most classic model for deformation, the mass-spring system allows diverse and excellent deformation expressions, even though it is a simple model that composes objects with mass points and springs. Especially in CG, this model is often applied to hair [86] and clothes [18], which move concurrently with the acting characters. Moreover, it applies to phenomena involving topology changes, such as fractures [73] and cracks [37][38]. Acceleration of the mass-spring system for time integration schemes has been also proposed [59], and the introduction of the Maxwell-Voigt model, where sliders and dumpers in addition to springs are used as connectors among mass points, has enabled the model to express viscoelastic and plastic deformation [96].

The finite element method (FEM) [97] is a representative simulation method that is applicable to solid deformation. When applied to structural analysis in the field of physical simulation, an element equation can be formulated for each element constituting an object, the solution to each of which is displacement at the nodes located on the surface of or inside the element. Then, a hypercomplex equation whose solution is the displacement of all nodes is solved to satisfy all element equations with consideration of boundary conditions. In the field of CG, many deformation methods use FEM, among which the simulation of brittle fractures by O'Brien [75] is one of the commonest attempts. In addition, methods for contact between multiple objects [43][53] and a tearing simulation of films with adaptive mesh [81] have been proposed. Further, acceleration for elastic deformation simulation with mesh [69] is applicable to both FEM and the mass-spring system. However, existing methods using FEM in the field of CG often emphasize the fact that internal continuous stress is integrated into discrete force when formulating element equations, and they avoid solving the hypercomplex equation. For these methods, FEM is reduced to substituting elements with springs in the mass-spring model.

Alternatively, particle methods, as represented by smoothed particle hydrodynamics [30], the particle-in-cell method [36], and the moving particle semi-implicit method [56] are known as numerical analysis methods for fluids, but they are also utilized for solid simulation. Especially, the

material point method (MPM) [89] combines the particle and lattice method, which is applicable to various materials, including elastic bodies, snow, lava, and sand, as demonstrated by Jiang et al. [50]. Recently, there have been active studies on MPM, such as fractures in anisotropic materials [102], application to magnetic fluid [91], and acceleration by hierarchical processing distribution [82]. Peridynamics [88] is a molecular dynamics-based method that is similar to the particle method, which has been applied to CG to simulate fractures [15].

Deformation methods based on continuum mechanics, including FEM and particle methods, lead to exact results according to the governing equations, but applying them to complex phenomena is difficult because they require the formulation of governing equations. In addition, they require large computational complexity to obtain an exact solution. For these reasons, practical CG systems often deal with deformation in a geometrical manner. Thus, baraff et al. [3] proposed a method for avoiding penetration by geometric computation considering mechanics.

Position-based dynamics (PBD) [70], a well-known CG simulation method, replicates deformation and motion based on geometric constraints instead of natural laws. Many applications of PBD have been proposed, as presented by Bender et al. [7] and Wu et al. [104], as PBD enables the easy simulation of complex mechanical phenomena because the values of the geometric parameters produced as a result of a phenomenon can be given as constraints. However, PBD may lead to physically incorrect behavior, but Bender et al. [6] realized physically exact PBD by setting constraints based on mechanical energy.

2.3 Interactive Simulation

While physical simulation tries to reproduce phenomena accurately under specific conditions as objective facts, visual simulation focuses on the subjective plausibility of their appearances. Therefore, visual simulation often pursues directability, that is, expertise that enables the simulation to be intentionally controlled by human external inputs.

A sort of directability can be observed in a study of dents on surfaces caused by impacts [78] and interactive γ-ton tracing [35]. Further, Endo et al. proposed a design system to produce water flow stains on walls in image [26]. In addition, interactive terrain generation considering erosion caused by water currents [100] is an example of interactive weathering, and Wu et al. proposed a design system for corroded fruit [105].

There exist many methods to control fluid in virtual worlds, which is difficult to manipulate intuitively in reality, and their control targets include water [39][108], smoke [19][20], and fire [12][65]. Position-based fluid [61] is an application of PBD [70] and a particle method, and it realizes a fast, stable, and interactive fluid simulation.

Solid deformation simulations, in which the shape of the model is maintained if no external forces are exerted, achieve interactive control without special processing in most cases by using

the mass-spring system or PBD. Further, to control FEM interactively, which generally requires large computational complexity, a method using precomputed Green functions [47] and detailed deformation with deep learning [83][84] was proposed. In addition, Barbic et al. proposed a framework to generate animations interactively by FEM-based simulation [4].

PBD can also be utilized for an interactive fracture simulation, and a method related to PBD, projective dynamics [10], has been applied to the fast-tearing simulation of cloth [58]. Needle insertion for surgical simulators with FEM [17] and texture-space deformation, including penetration using a collision simulation [74], were proposed.

In the case of controlling elastic deformation or stable-state flow, it may not be necessary to restart the simulation from the beginning. For example, when an interactive simulation is used to put a wrinkled cloth model in a virtual scene, the change from one state to another is relatively easy, and it is possible to fine-tune the model's shape. As an example of fluid, a fire control system can generate various fire shapes without restarting the simulation, except for ignition, where the velocity field changes dynamically. Conversely, a fracture is an irreversible change, so an excessively broken model cannot be undone. In the simulation of irreversible phenomena, undoing the state requires restarting the simulation or saving the history.

2.4 Strengths of the Proposed Method

The proposed method focuses on the weathering of coating films covering metallic objects, which is seen regularly in reality, even though few existing methods deal with its simulation. Furthermore, as mentioned in **Sec. 2.1**, most existing methods, not just those for the weathering of coating films, manipulate the material of an object's surface, but few allow 3D deformation.

Therefore, the proposed method adopts the deformation simulation demonstrated in **Sec. 2.2**. When simulating the deformation of a thin board, such as a coating film, by FEM, the computation is more complicated, because not only the displacement but also the angles of deflection must be derived as the solution. Particle methods must position multiple particles in the thickness direction, and they require much computational complexity. In both FEM and particle methods, changes to the values of some mechanical parameters due to aged deterioration cannot be theoretically derived or actually measured. Thus, the proposed method deals with bends in coating films geometrically and phenomenologically, expresses them using PBD-based deformation, and realizes natural deformation by setting geometric constraints.

In addition, the proposed method is a kind of interactive simulation, as presented in **Sec. 2.3**, and it deals with an irreversible phenomenon, the simulation of which is usually not undone. However, this method can approximate the state of the previous step by causing virtual inverse phenomena that never occur in reality. This inverse processing makes up for the simulation's disadvantage of time unidirectionality and achieves highly directable weathering.

Chapter 3

Method Overview

This chapter gives an overview of the proposed method. **Section 3.1** explains the aged degradation of coating films in reality and **Sec. 3.2** illustrates the processing flow.

3.1 Real Coating Films

3.1.1 Bend

The main factor in the aged degradation of real coating films is the changes in the internal stress and physical properties due to drying. In reality, because it is difficult to measure stress in coating films, the stress comes from a measurable displacement by coating a small metal plate and letting the base deform together with the coating film. As shown in **Fig. 3.1**, according to Stoney's law [90], with an assumption that a coating film bends thin metal disk, the relation between the stress in the coating film σ_{F} and r can be expressed by:

$$\sigma_{\mathrm{F}} = \frac{E_{\mathrm{S}} {t_{\mathrm{S}}}^2}{6(1-\nu_{\mathrm{S}})t_{\mathrm{F}}} R^{-1}, \tag{3.1}$$

where r, t_{S}, ν_{S}, and E_{S} denote the radius, thickness, Poisson's ratio, and Young's modulus of the disk, respectively, and t_{F} denotes the thickness of the coating film and R the radius of the curvature. If the center of the disk is fixed, R is approximated as:

$$R = \frac{r^2}{2\delta},$$

where δ denotes the displacement of the circumference. Substituting this into **Eq. (3.1)**, the following equation is obtained:

$$\delta = \frac{3(1-\nu_{\mathrm{S}})t_{\mathrm{F}}}{E_{\mathrm{S}} {t_{\mathrm{S}}}^2} \sigma_{\mathrm{F}} r^2.$$

Then, multiplying σ_{F} by the volume of the coating film $\pi r^2 t_{\mathrm{F}}$ yields the contraction force of the coating film $T = \pi r^2 t_{\mathrm{F}} \sigma_{\mathrm{F}}$, and the relation between T and δ is obtained as follows:

$$\delta = \frac{3(1-\nu_{\mathrm{S}})}{\pi E_{\mathrm{S}} {t_{\mathrm{S}}}^2} T. \tag{3.2}$$

As shown in **Fig. 3.2**, F_{L} is assumed a virtual force acting on the edge of the disc to lift the coating film from the base, and it equals the force required to bend coating films with the base when the base is not fixed. F_{L} can be computed according to **Eq. (3.2)**, where if F_{L} is proportional to δ in accordance with Hooke's law and all parameters other than T are constants, F_{L} is simply given by:

$$F_{\mathrm{L}} = k_L T, \tag{3.3}$$

where k_L denotes a constant coefficient that represents the ease of separation.

If the contraction force in the tangential plane direction becomes strong enough, the coating film tears and a crack appears, as mentioned in the next subsection. In addition, the lifting force peels the film from the base.

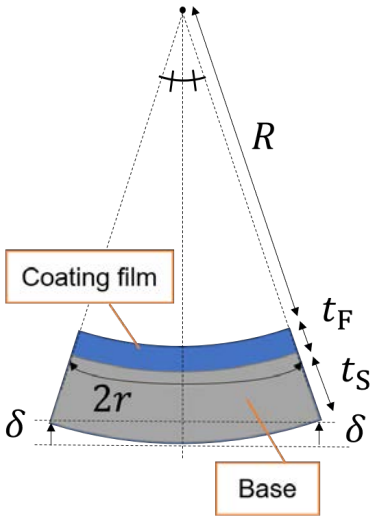


Fig. 3.1 A coating film bending together with the metallic base

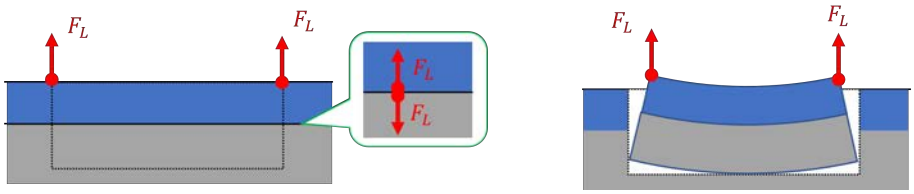


Fig. 3.2 Virtual lifting force. It acts to peel the coating film from the base (left). If the coating film were cut out in the shape of a disc with the base, it would be bent according to Stoney's law [90] (right)

3.1.2 Crack

Any material including coating film can be modeled as a continuum, whose distributions of physical properties are continuous. Continuum mechanics is a powerful theory that describes stress and strain in a wide variety of solids and fluids, and it underlies numerous physical simulations, such as FEM and particle methods. However, continuum mechanics is not applicable to phenomena strongly influenced by microscopic discontinuities, so the difference between theoretical analysis and experimental results becomes large.

A fracture is a representative phenomenon to which it is difficult to apply continuum theory, and most real materials are broken by a smaller load than the theoretical value. In contrast, fracture mechanics focuses on the stress distribution on the material surface and describes how fractures are caused by the stress concentration at the ends of cracks. According to fracture mechanics, the reason materials are easier to break than continuum theory assumes are the tiny cracks inherent in the materials as defects and the large stresses around the ends of defects. Extended defects appear as large cracks that can be seen by the naked eye. Note that except for some differences in usage,

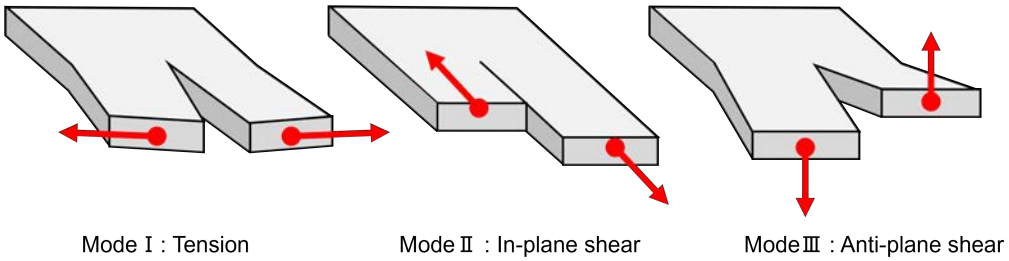


Fig. 3.3 Three modes of fracture. Mode I is dominant among thin plates

fracture mechanics assumes the same yield criterion as continuum mechanics, in which a fracture occurs if the stress is over the limit value given as a physical property.

As shown in **Fig. 3.3**, there are three modes of fracture: mode I (tension), mode Ⅱ (in-plane shear), and mode Ⅲ (anti-plane shear). Mode I is dominant among thin plates, so this thesis discusses only mode I. However, this assumption means that it is acceptable to ignore in-plane and anti-plane stresses in the crack analysis, but it does not neglect the existence of anti-plane forces.

An xy-coordinate system whose origin is the end of a crack and whose x-axis is in the direction of the crack is set parallel to the coating films, as illustrated in **Fig. 3.4**. If the stress intensity factor is denoted as K_{I} and the radius and azimuth at position $\boldsymbol{p}$ are r and θ, respectively, the stress tensor $\boldsymbol{\sigma}$ at $\boldsymbol{p}$ can be expressed by:

$$\begin{aligned}
\sigma_{xx} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\left(1 - \sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right), \\
\sigma_{yy} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\left(1 + \sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right), \\
\sigma_{xy} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} \cos\frac{\theta}{2}\sin\frac{\theta}{2}\sin\frac{3\theta}{2}, \\
\boldsymbol{\sigma} &= \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}.
\end{aligned} \tag{3.4}$$

When the factors derived from θ in σ_{xx}, σ_{xy}, and σ_{yy} are abbreviated to $k_{xx}(\theta)$, $k_{xy}(\theta)$, and $k_{yy}(\theta)$, respectively, **Eq. (3.4)** can be rewritten as:

$$\begin{aligned}
\sigma_{xx} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} k_{xx}(\theta), \\
\sigma_{yy} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} k_{yy}(\theta), \\
\sigma_{xy} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} k_{xy}(\theta), \\
\boldsymbol{\sigma} &= \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} \begin{bmatrix} k_{xx}(\theta) & k_{xy}(\theta) \\ k_{xy}(\theta) & k_{yy}(\theta) \end{bmatrix}.
\end{aligned}$$

If $\boldsymbol{n}_\theta = (-\sin\theta, \cos\theta)^{\mathrm{T}}$ denotes a unit vector in the azimuth direction at position $\boldsymbol{p}$, the opening-direction component of stress σ_θ can be obtained by:

$$\sigma_\theta = \boldsymbol{n}_\theta^{\mathrm{T}} \boldsymbol{\sigma} \boldsymbol{n}_\theta = \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} \left(k_{xx}(\theta)\sin^2\theta - 2k_{xy}(\theta)\cos\theta\sin\theta + k_{yy}(\theta)\cos^2\theta\right) = \frac{K_{\mathrm{I}}}{\sqrt{2\pi r}} k(\theta), \tag{3.5}$$

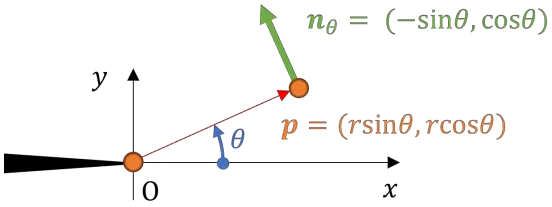


Fig. 3.4 A relative coordinate at a crack end. The black triangle on the left represents the crack. The origin is at the end, the x-axis is in the crack's direction, and the y-axis is perpendicular to the x-axis

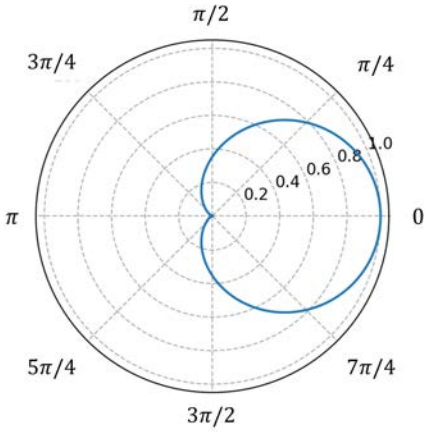


Fig. 3.5 The relation between the azimuth θ and the coefficient $k(\theta)$ on the opening-direction component of the stress tensor

where $k(\theta)$ denotes the following function of the azimuth θ, whose value range is $[0, 1]$:

$$\begin{aligned} k(\theta) = &\cos\frac{\theta}{2}\sin^2\theta\left(1-\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right) \\ &-2\cos\frac{\theta}{2}\sin\frac{\theta}{2}\cos\theta\sin\theta\sin\frac{3\theta}{2}+\cos\frac{\theta}{2}\cos^2\theta\left(1+\sin\frac{\theta}{2}\sin\frac{3\theta}{2}\right). \end{aligned} \tag{3.6}$$

Figure 3.5 illustrates the relation between θ and $k(\theta)$, which **Eq. (3.6)** represents. According to **Eq. (3.6)** and **Fig. 3.5**, the stress is strongest in the forward direction and about 35 % of the maximum value in the perpendicular direction. This stress distribution leads to the phenomenon where cracks tend to grow straight. The stress intensity factor K_I depends on stretch stress σ_{yy}^{∞} in the y-direction at the distance and length of crack a, and it is computed as:

$$K_\mathrm{I} = \sigma_{yy}^{\infty}\sqrt{\pi a}. \tag{3.7}$$

Equations (3.4), **(3.5)**, **(3.6)**, and **(3.7)** are derived from the ideal model, and there are some points to note when applying them to the simulation. First, according to **Eq. (3.5)**, the stress becomes infinitely large as r approaches 0. In this regard, plastic deformation at the crack end limits the stress to a certain value and redistributes the excess stress to the surrounding area. However, because it is extremely difficult to measure the redistributed stress, **Eq. (3.5)** must be

used as an approximate formula, while paying attention to the applicable range. Conversely, the stress is computed as 0 when r is large enough, but there exists stress at least σ_{yy}^{∞}. In addition, **Eq. (3.7)** is applicable only when the crack length a is a few dozen percent of the material thickness and the crack is a straight line. In the case of a long and complex crack, the dependence of K_{I} on a seems to weaken, but an alternative mathematical model to **Eq. (3.7)** is difficult to construct because it must consider influences on the thickness and physical properties of the material.

3.1.3 Stain

Aged degradation affects not only the shape of an object but also its superficial texture. Chalking [32] is an important chemical change in coating films, a phenomenon whereby resin in the film is decomposed by rain and light and pigment powder appears on the surface of the coating film. If a coating film is chalked, the surface diffuses light and loses its gloss. Chalking occurs earlier than deformation, but cracking and peeling usually appear right after chalking, because chalking means a lack of the rust preventive function of the coating film.

One of the characteristic effects of rust preventive coating films is rust run-off stains, which can also be seen on reinforced concrete. Rust run-off is a phenomenon where the metallic base is oxidized and outflows from peeled areas. Oxidization is promoted by moisture and rust is conveyed by water flow, so rain is the prime contributor to rust run-off stains.

Moreover, in highly populated cities where many people live, most objects are darkened from accumulating dust due to exhaust gas. Darkening appears more significantly on bumpy surfaces, which prevents rain and wind from washing the dust. In addition, dust tends to accumulate on upward-facing surfaces, which apply vertical drag force against gravity.

3.2 Simulation Flow

The processing flow is illustrated in **Fig. 3.6**, and the proposed method models a coating film with a triangle polygon mesh, as detailed in **Chap. 4**. Then, the model is deformed with a position-based bend simulation, and its topology is manipulated according to a statics-based fracture simulation to express fracturing. Either coordinate system is acceptable, but in this thesis, the left-hand coordinate system is adopted.

The position-based bend simulation, described in **Chap. 5**, parallels the fracture simulation, and it proceeds following a coherent algorithm independent of the fracture progress, but its target is updated when a fracture occurs. The effects of bend simulation cannot be seen in the initial state, but once the fracture simulation separates vertices from the base, the bend simulation moves vertices to deform the mesh.

As shown in **Fig. 3.7**, the simulation of fractures due to aged deterioration proceeds in three stages: separation, tearing, and stripping. In this thesis, the transition to the next stage is

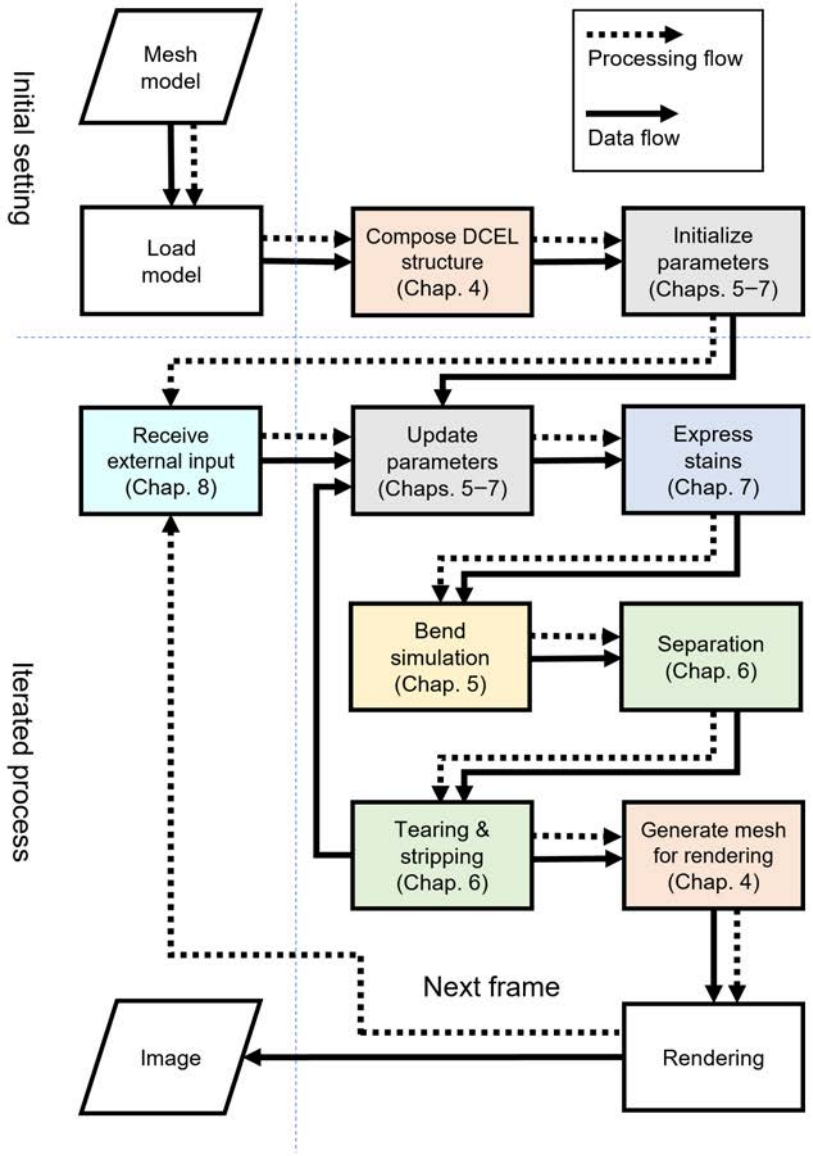


Fig. 3.6 The flow of processes in the proposed method. The input is polygon mesh data, and it is weathered as a coating film, the process of which is simulated in every frame. The deteriorating model can be displayed in real time and controlled by an external input

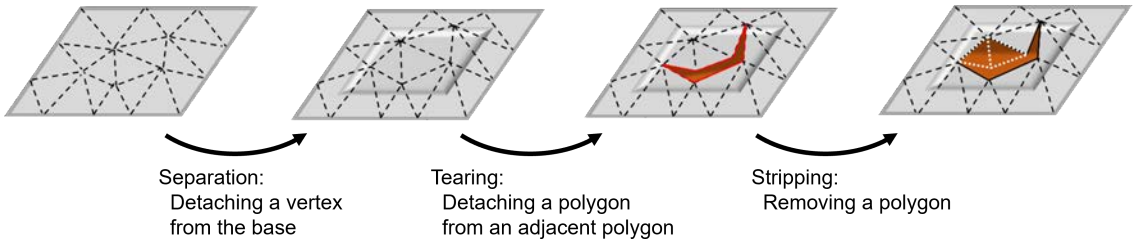


Fig. 3.7 The peeling process in the proposed method. Whether to progress to the next stage is determined by the static fracture criteria

referred to as a *fracture*. Note that these stages can coexist in the same space because a fracture is judged for each local region and is based on static equilibriums. The parameter values and topological data of the polygon mesh are updated when a part of the model fractures, the specific computation method and process of which will be detailed in **Chap. 6**.

Further, stains are expressed as non-deformation aged deterioration. As explained in **Chap. 7**, parameters concerning stains are set to each vertex, and parameter values are given by geometric computations. Then, stains are drawn by manipulating the color and reflectance at each vertex according to the degree of stains.

These processes are performed based on simplified mathematic models such that the state of the previous step can be estimated from the current and initial states, allowing for a pseudo time reversal of the simulation. In addition, the simulation process is drawn in real time by GPU parallelization. The directable simulation is achieved by manipulating parameter values according to external input, and the methods for the time reversal, parallelization, and parameter control are detailed in **Chap. 8**.

Chapter 4

Data Structure

This chapter describes the data structure and related algorithms to run the weathering simulation. First, **Sec. 4.1** introduces the doubly connected edge list (DCEL) [68], also known as the half-edge data structure, to deal effectively with 3D triangular polygon mesh. Then, **Sec. 4.2** demonstrates attribute parameters added to DCEL. Finally, **Sec. 4.3** mentions parameters to control the simulation.

4.1 Doubly Connected Edge List

The proposed method must frequently manipulate the mesh topology to express the tearing of coating films. Therefore, DCEL is adopted as a fundamental data structure to manage the mesh, which can efficiently update the reference relations among the geometric elements, that is, the topological data.

As illustrated in **Fig. 4.1**, DCEL consists of three kinds of geometric elements: vertices, faces, and half-edges, the latter of which is an edge split in two whose face has three unique half-edges. Note that in this method, an edge located on the edge of an open-curved surface model is expressed as a single half-edge that does not constitute a pair. Therefore, the total number of half-edges in a triangular polygon mesh equals three times the number of faces. Although edges do not exist as data, a pair of half-edges or a single half-edge that has no pair is simply referred to as an *edge* hereinafter. To manage all model data, the vertex list `Verts`, half-edge list `Edges`, and face list `Faces` are defined, and each geometric element is stored in the corresponding list.

Each geometric element records some of its adjacent elements as the topological data, as illustrated in **Table 4.1**. In this thesis, to simplify the notation,the vertex, half-edge, and face are implemented with the classes `Vert`, `Edge`, and `Face`, respectively, and the topological data are stored as instances of these classes that are reference-type variables. In fact, each geometric element is implemented with a structure to run in the GPU, and the topological data are stored as indexes in the corresponding list, which is a value-type variable. For example, the state "Substitute one of the faces adjacent to a face `F` to `X`" is expressed in terms of structures and classes as follows:

```
X = Faces[Edges[Edges[F.Edge].Pair].Face];// structure, X is index (value)
X = F.Edge.Pair.Face;// class, X is instance (reference)
```

The geometric data are only the 3D position vector data set to the vertex class `Vert`, as shown in **Table 4.2**.

While DCEL is suitable for phase manipulation, it includes data redundant for rendering. In contrast, an indexed face set (IFS) is the simplest data structure for shape modeling, as it expresses triangular polygon meshes with the index list `Triangles` and vertex list `Vertices`. If the index of each `Vert` in DCEL is denoted as a member variable `Index`, DCEL can be converted to IFS by **Algorithm 4.1**, and topological data referenced in the conversion are illustrated in **Fig. 4.2**.

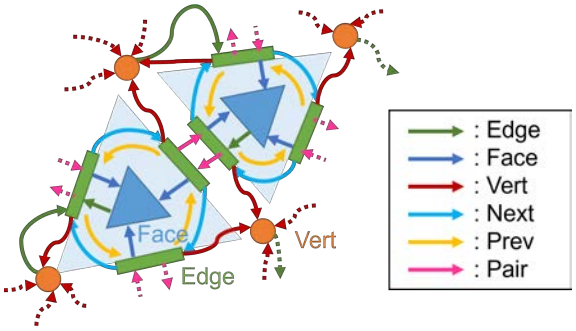


Fig. 4.1 Doubly connected edge list. Each arrow means the elements connected by it can be referenced in its specified direction. A dotted line arrow means an element reference (or is referenced by) another element that is not shown in the figure

Table 4.1 Topological data possessed by geometric elements

Element	Member variable		
(class)	Type	Name	Description
`Vert`	`Edge`	`Edge`	One of adjacent half-edges
`Edge`	`Edge`	`Pair`	Pair to the half-edge
	`Edge`	`Next`	Next half-edge clockwise around the perimeter
	`Edge`	`Prev`	Previous half-edge clockwise around the perimeter
	`Vert`	`Vert`	Vertex shared with `Prev`
	`Edge`	`Face`	Face whose perimeter consists of the edge, `Next`, `Prev`
`Face`	`Edge`	`Edge`	One of the half-edges that compose the perimeter

Table 4.2 Geometric data possessed by geometric elements

Element	Member variable		
(class)	Type	Name	Description
`Vert`	`Vector3`	`Pos`	Position vector

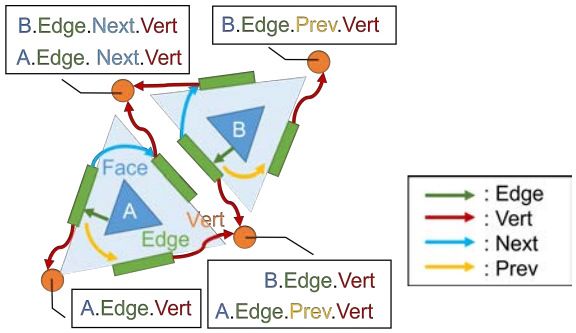


Fig. 4.2 Topological data referenced in the conversion from DCEL to IFS

Algorithm 4.1 Conversion from DCEL to IFS

Require: Vertex list `Verts`, half-edge list `Edges`, and face list `Faces` in accordance with DCEL

Ensure: Obtain Vertex list `Vertices` and index list `Triangles` in accordance with IFS

```
for all F in Faces do
    Add F.Edge.Vert.Index to Triangles;
    Add F.Edge.Next.Vert.Index to Triangles;
    Add F.Edge.Prev.Vert.Index to Triangles;
end for
for all V in Verts do
    Add V.Pos to Vertices;
end for
```

Most general file formats for 3D models, such as OBJ and PLY, are IFS-compliant, so applying the proposed method to these files requires an inverse conversion from IFS to DCEL. If data read from a model file are stored in `Triangles` and `Vertices`, and each of the `Verts`, `Edges`, and `Faces` is initialized with a list of zero elements, the procedure of the inverse conversion is as follows, also illustrated in **Fig. 4.3**:

1. Retrieve geometric data from `Vertices` and `Vert` to send to the vertex list `Verts` (**Algorithm 4.2**);
2. Generate `Face` and `Edge` according to `Triangles`, and set topological data except for `Pair` (**Algorithm 4.3**); and
3. Set the remaining topological data `Pair` (**Algorithm 4.4**).

Hereinafter, in the process where the topology data are assumed already properly configured, a half-edge `Edge` is simply depicted as an arrow that omits the topological data, with `Edge.Vert` as the start and `Edge.Next.Vert` as the end point, as shown in **Fig. 4.4**. `Edge.Vert` and `Edge.Next.Vert` are referred to as *the start* and *end point* of `Edge`, respectively. Note that `Face.Edge` and `Vert.Edge` cannot be uniquely determined, so these kinds of topological data are shown if necessary.

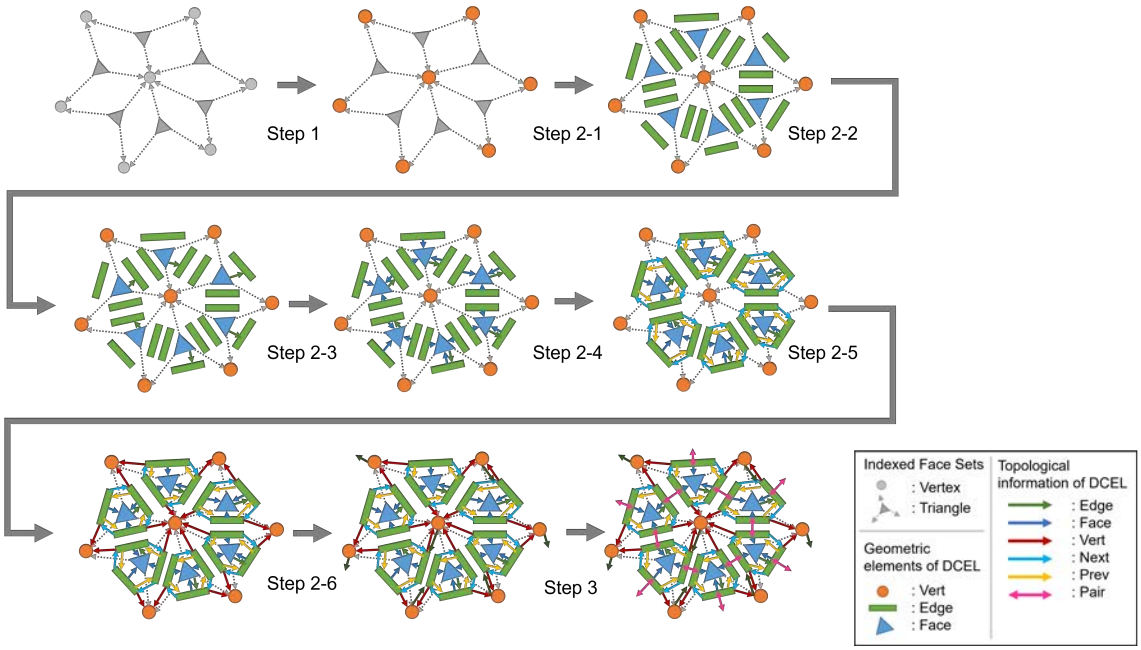


Fig. 4.3 DCEL construction procedure. As described in **Algorithm 4.3**, Step 2 is actually run concurrently for each triangle, but this figure subdivides Step 2 into Step 2-1 through Step 2-6 and describes each step as if it were performed on the entire mesh

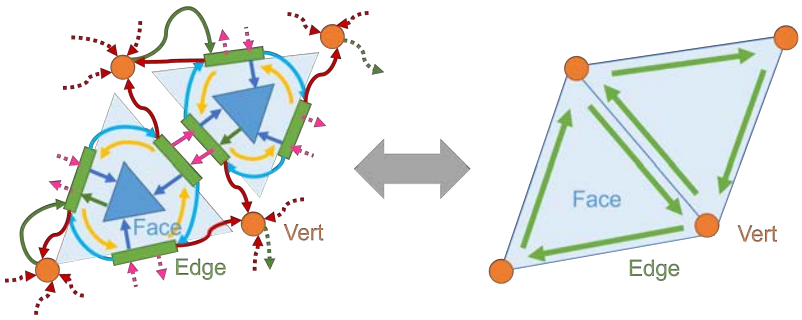


Fig. 4.4 Simplified representation of DCEL

Algorithm 4.2 The first step of DCEL construction: Initialization of `Verts`

Require: Array `Vertices` of length `n` storing the vertex positions, and vertex list `Verts` of zero elements

Ensure: Store `n` `Vert` into `Verts`

1: **for** `i` ← `0` **to** `n - 1` **do**
2: Generate a new `Vert newVert`;
3: `newVert.Pos` ← `Vertices[i]`;
4: Add `newVert` to `Verts`;
5: **end for**

Algorithm 4.3 The second step of DCEL construction: Initialization of `Faces` and `Edges`

Require: Array `Triangles` of length `m*3` storing the indexes of vertices, the configured vertex list `Verts`, half-edge list `Edges` of zero elements, and face list `Faces` of zero elements

Ensure: Store `m` `Face` into `Faces`, and `m*3` `Edges` into `Edge`

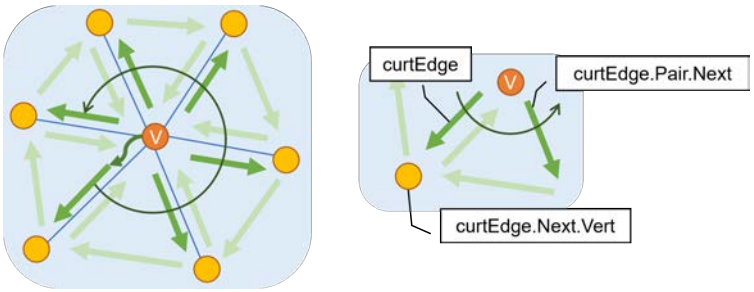
```
1: for i ← 0 to m - 1 do
2:   Generate a new Face newFace;
3:   Generate new three of Edge newEdges[j] (j = 0, 1, 2);
4:   newFace.Edge ← newEdges[0];
5:   for j ← 0 to 2 do
6:     newEdges[j].Face ← newFace;
7:     newEdges[j].Next ← newEdge[(j + 1) % 3];
8:     newEdges[j].Prev ← newEdge[(j + 2) % 3];
9:     newEdges[j].Vert ← Verts[Triangles[i * 3 + j]];
10:    newEdges[j].Vert.Edge ← newEdges[j];
11:    Add newEdges[j] to Edges;
12:  end for
13:  Add newFace to Faces;
14: end for
```

Algorithm 4.4 The third step of DCEL construction: Setting of `Pair`

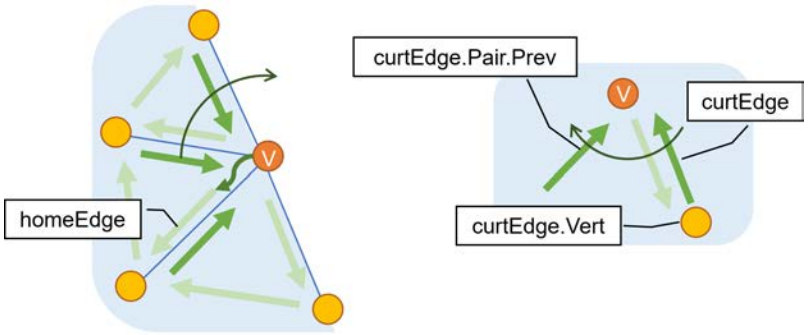
Require: The configured vertex list `Verts` and face list `Faces`, the list `Edges` whose elements' topological data are set except for `Pair`, and the number of polygons `m` constitute the model

Ensure: Set an appropriate `Edge pair` ($\in$ `Edges`) to `E.Pair`

```
1: for i ← 0 to m - 1 do
2:   for j ← 0 to i - 1 do
3:     if Edges[i].Next.Vert is Edges[j].Vert and Edges[i].Vert is Edges[j].Next.Vert
   then
4:       Edges[i].Pair ← Edges[j];
5:       Edges[j].Pair ← Edges[i];
6:     end if
7:   end for
8: end for
```



(a) Counter-clockwise search. Half-edges whose starting point is `V` are traced in a counter-clockwise order. If the current half-edge is `curtEdge`, `curtEdge.Next.Vert` is connected with `V` and the next half-edge is `curt.Pair.Next`



(b) Clockwise search. Half-edges whose end point is `V` are traced in a clockwise order. If the current half-edge is `curtEdge`, `curtEdge.Vert` is connected with `V` and the next half-edge is `curt.Pair.Prev`

Fig. 4.5 The basic policy for searching around a vertex `V` using DCEL, only if the edge of the model is reached during a counter-clockwise search (a) and if the remaining vertices are identified by the clockwise search (b)

The proposed method frequently performs search processes around a vertex `V`. As an example of search processes using DCEL, **Algorithm 4.5** represents a procedure to store vertices connected with `V` to `conVerts`. Further, as illustrated in **Fig. 4.5**(a), it is the basic policy to start with `V.Edge` as the departure half-edge (`homeEdge`) and move the current half-edge (`curtEdge`) counter-clockwise. However, if `V` is on the edge of the model and `curtEdge.Pair` does not exist, a clockwise search, as shown in **Fig. 4.5**(b), must be performed. In **Algorithm 4.5**, the body of the if statement from lines 6 to 12 represents the clockwise search and the others the counter-clockwise search.

Algorithm 4.5 Search for connected vertices

Require: Vertex list `Verts`, half-edge list `Edges` and face list `Faces` constructed according to DCEL and the target vertex `V` ($\in$ `Verts`)

Ensure: Store all `Vert` connected with `V` to list `conVerts`

```
1: homeEdge ← V.Edge;
2: curtEdge ← homeEdge;
3: repeat
4:    Add curtEdge.Next.Vert to conVerts;
5:    if curtEdge.Pair is null then
6:       curtEdge ← homeEdge.Prev;
7:       while curtEdge.Pair is not null do
8:          Add curtEdge.Vert to conVerts;
9:          curtEdge ← curtEdge.Pair.Prev;
10:      end while
11:      Add curtEdge.Vert to conVerts;
12:      break;
13:   end if
14:   curtEdge ← curtEdge.Pair.Next;
15: until curtEdge is not homeEdge
```

Table 4.3 Basic attribute parameters

Element	Member variable		
(Class)	Type	Name	Description
`Vert`	`Vector3`	`Nor`	Unit normal
	`Vector2`	`UV`	UV coordinate
	`Color`	`Col`	Vertex color
	`Vector3`	`Pos0`	Initial value of `Pos`
	`Vector3`	`Nor0`	Initial value of `Nor`
	`float`	`Cur`	Discrete mean curvature
`Edge`	`float`	`Ang`	Angle formed by adjacent polygons

4.2 Attribute Parameters

The previous section presented geometric data and topological data, but the simulation requires additional attribute parameters. As such, this section introduces attribute parameters that are set for geometric elements.

First, as listed in **Table 4.3**, the standard parameters often set in general CG systems, and the geometric properties used in several processes are set as member variables. Note that the parameters `Vert.Nor`, `Vert.UV`, and `Vert.Col` are passed to the mesh for rendering in IFS format, similar to how `Vert.Pos` was sent to `Vertices`. `Vert.Cur` is the dot product of the discrete mean curvature vector and the unit normal, which is positive and negative when convex on the front and back side, respectively.

To apply PBD [70] to the polygon mesh, the classes of the length constraint `LengthConstraint` and of the bend constraint `BendConstraint` are defined, whose member variables are listed in **Table 4.4**. These constraints and how they are applied to the mesh in the proposed method are detailed in **Chap. 5**. Note that each constraint is actually implemented as a structure but described as a class in this thesis in the same way as the geometric elements, as mentioned in the previous section. Both length constraints and bend constraints are set to half-edges, and just as half-edges are stored in `Edge`, length and bend constraints are stored to the lists `LengthConstraints` and `BendConstraints`, respectively. The indexes of the half-edge list and the constraint lists correspond to each other. In other words, to the `i`-th half-edge `Edge[i]`, the length constraint `LengthConstraints[i]` and bend constraint `BendConstraints[i]` are set. Hereinafter, `LengthConstraints[i]` and `BendConstraints[i]` are referred to as constraints that *correspond* to `Edge[i]`. In addition, three parameters shown in **Table 4.5** are set to compute the movements of vertices according to constraints.

Table 4.4 Constraint classes and their member variables

Constraint	Member variable		
(Class)	Type	Name	Description
`Length`	`Vert[]`	`Verts`	Array with 2 target vertices
`Constraint`	`float`	`Offset`	Target length
	`float`	`Offset0`	Initial value of `Offset`
	`float`	`Stiff`	Strength of constraint [0.0, 1.0]
	`float`	`Thr`	Threshold for iteration
`Bend`	`Vert[]`	`Verts`	Array with 4 target vertices
`Constraint`	`float`	`Offset`	Target angle
	`float`	`Offset0`	Initial value of `Offset`
	`float`	`Stiff`	Strength of constraint [0.0, 1.0]
	`float`	`Thr`	Threshold for iteration

Table 4.5 Geometric properties possessed by elements

Element	Member variable		
(Class)	Type	Name	Description
`Vert`	`Vector3`	`AssPos`	Assumed position
	`Vector3`	`Delta`	Assumed displacement
	`float`	`Weight`	Mobility of vertex

Next, the parameters for running the fracture simulation, as detailed in **Chap. 6**, are explained. As **Table 4.6** presents, four parameters denote the forces used to compute the criteria of a fracture, and a parameter holds the crack direction. Because the simulation requires performing different processes depending on the state of geometric elements, status parameters used for the condition determination are needed, as represented in **Table 4.7**. `Vert.Orig` basically holds the vertex itself `Vert`, but when a vertex is duplicated and all parameter values, including `Vert.Orig`, are copied, the parameter at the duplicated vertex indicates the original vertex, not itself. `Vert.OnEdge` is set to `1` if the vertex `Vert` is on the edge of the open-curved surface model and `0` otherwise. Further, each element holds a parameter `Stat` to denote the condition of the fracture, which is initially set to `0`, but at a half-edge `Edge` on the edge of the model, exceptionally, it is set to `2`. Changes to their values during the simulation are explained in **Chap. 6**.

Table 4.6 Parameters representing the state of force

Element	Member variable		
(Class)	Type	Name	Description
`Vert`	`float`	`Adhere`	Adhesion to the base
	`Vector3`	`Direction`	Direction of crack
`Edge`	`float`	`Lifting`	Lifting force to separate vertex
	`float`	`Binding`	Binding force to hold connection between polygons
`Face`	`float`	`Contraction`	Contraction forces to stretch adjacent polygons

Table 4.7 Parameters representing the state of degradation

Element	Member variable		
(Class)	Type	Name	Description
`Vert`	`Vert`	`Orig`	Original vertex
	`int`	`OnEdge`	Whether it is on the edge of the model
	`int`	`Stat`	State of degradation
`Edge`	`int`	`Stat`	State of degradation
`Face`	`int`	`Stat`	State of degradation

Table 4.8 Parameters influencing the degrees of stains

Element	Member variable		
(Class)	Type	Name	Description
`Vert`	`float`	`Chalk`	Degree of chalking
	`float`	`Rust`	Degree of rust run-off staining
	`float`	`RustBuf`	`Rust` value before update
	`bool`	`RustSource`	Presence of a source of rust
	`float`	`Dust`	Degree of dust accumulation

Table 4.8 shows the parameters needed to express stains, as detailed in **Chap. 7**. Each denotes the degree of stain at a vertex, and the initial value is `0`. `Chalk`, `Rust`, and `Dust` refer to the degree of chalking, rust run-off stains, and dust accumulation, respectively. Note that the computation of `Rust` for each frame requires a read buffer `RustBuf` holding `Rust` values before updating, because it must reference values at nearby vertices. In addition, each vertex remembers whether it is a source of rust as a boolean `RustSource`.

4.3 Control Parameters

While **Secs. 4.1** and **4.2** introduce individual parameters set to each geometric element or constraint, this section mentions global control parameters that affect the weathering progress of the entire model. **Table 4.9** lists all control parameters.

There are many parameters in the simulation, but most are fixed with empirical values. It may realize application in more varied scenes to after these values according to an external input or set non-uniform distribution. However, excessively, many parameters hinder intuitive control, so parameters that can be manipulated by an external input are limited to the five listed in the top rows of **Table 4.9**.

`dT` is the most important parameter, which denotes the virtual elapsed time per step. While weathering deals with long-term time lapse and assumes the simulation time step to be several days or months, the real time scale to display the simulation process is much shorter, at less than a second. Therefore, the weathering simulation uses the time step `dT` independent of the frame ratio. Moreover, `dT` is allowed to be a negative value, meaning the virtual time can be reversed. As for the other variable parameters, `curl` is detailed in **Sec. 6.2**, `s` and `h` in **Sec. 6.4**, and `range` in **Sec. 8.4**.

Table 4.9 Control parameters. The five parameters listed in the top rows can be manipulated by an external input

Type	Name	Description	Values
`float`	`dT`	Time elapsed per step	`[-10.0, 10.0]`
`float`	`curl`	Sensitivity for bending	`[0.0, 0.025]`
`float`	`s`	Sensitivity for stress concentration	`[0.0, 2.0]`
`float`	`h`	Persistence of crack direction	`[0.0, 1.0]`
`float`	`range`	Control range	`[0.01, 10]`
`Vector3`	`gravity`	Direction of gravity	`(0.0, -1.0, 0.0)`
`int`	`iteration`	Number of iterations in PBD	`50`
`float`	`probDefect`	Existence ratio of defects	1.0×10^{-3}
`float`	`fragDefect`	Ratio of binding force at a defect	1.0×10^{-3}
`float`	`shrink`	Sensitivity for shrinking	1.0×10^{-3}
`float`	`sppedCo`	Increasing speed of contraction force	`0.50`
`float`	`speedAd`	Decreasing speed of adhesion force	`0.25`
`float`	`kL`	Ratio of lifting force to contraction force	`0.10`
`float`	`gamma`	Sensitivity of lifting force for curvature	`2.0`
`float`	`speedBi[0]`	Decreasing speed of binding force around a face with 0 torn edges	`0.00`
`float`	`speedBi[1]`	Decreasing speed of binding force around a face with 1 torn edge	`0.20`
`float`	`speedBi[2]`	Decreasing speed of binding force around a face with 2 torn edges	`1.00`
`float`	`speedBi[3]`	Decreasing speed of binding force around a face with 3 torn edges	`0.00`
`float`	`speedCh`	Speed of chalking	5.0×10^{-3}
`float`	`sppedRu`	Speed of rust run-off	`0.10`
`float`	`rangeRu1`	Range of mainstream	`0.10`
`float`	`rangeRu2`	Range of sub-stream	`0.45`
`float`	`ratioRu`	Ratio of Rust Amount conveyed through sub-stream to one through mainstream	`0.10`
`Color`	`rustColor`	Color of rust	`(0.75, 0.37, 0.00)`
`float`	`ratioIntensity`	Strength of rust color	`0.20`
`float`	`speedDu`	Speed of dust accumulation	2.0×10^{-4}
`float`	`weightNor`	Sensitivity of darkening for normal	`4.0`
`float`	`weightCur`	Sensitivity of darkening for curvature	`1.0`
`float`	`epsilon`	Maximum length between two points considered at the same location	`0.01`
`float`	`exAdhere`	Max value of adhesion force	2.0×10^{2}
`float`	`exBinding`	Max value of binding force	2.0×10^{3}
`float`	`rateCtrl`	Sensitivity of interactive control	`0.05`

Chapter 5

Bend Simulation

This chapter introduces a simulation method for bent films. By applying a bend simulation to a coated film with cracks, the film on each side of the crack will bend and the cracks will widen. First, the underlying PBD framework [70] is introduced in **Sec. 5.1**, and the method for controlling the length of the line segment in accordance with PBD is demonstrated in **Sec. 5.2**. Then, the conventional method for maintaining the angle formed by two adjacent triangles is explained in **Sec. 5.3**, and a potential method for expressing bends of films is proposed in **Sec. 5.4**. Finally, the bend simulation algorithm is presented in **Sec. 5.5**.

5.1 Outline of Position-Based Deformation

This section introduces the outline of the PBD framework [70]. In PBD, geometric constraints are set on the vertices of a scene, and the positions of the vertices are repeatedly updated to satisfy the constraints. A constraint on N vertices is described as a function of their position vectors $C_j(\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{N-1})$, and the function is defined such that its value is 0 when the constraint is satisfied. Then, for the n-th ($n = 0, 1, ..., N-1$) vertex $\boldsymbol{p}_n$, if its inverse mass is w_n, the correction $\Delta\boldsymbol{p}_{nj}$ of the position with the constraint C_j is computed by:

$$\begin{aligned} \Delta\boldsymbol{p}_{nj} &= -s_j w_n \boldsymbol{\nabla}_{\boldsymbol{p}_n} C_j(\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{N-1}), \\ s_j &= \frac{C_j(\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{N-1})}{\sum_k w_k \|\boldsymbol{\nabla}_{\boldsymbol{p}_k} C_j(\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_{N-1})\|^2}. \end{aligned} \tag{5.1}$$

As demonstrated by **Eq. (5.1)**, the corrections are computed from the gradient of C_j with respect to the positions of the vertices. Therefore, C_j must be differentiable and have no extrema within the domain of the definition.

A vertex is generally manipulated by multiple constraints, and ts position is corrected by computing the weighted linear sum of individual corrections for each constraint. Moreover, corrections are computed each time in the iterative process. If the n-th vertex in the constraint C_j is the i-th vertex in the entire scene and the position correction $\Delta\boldsymbol{p}_{nj}$ is rewritten as $\Delta\boldsymbol{p}_{ij}$, the correction $\Delta\boldsymbol{p}_i$ for $\boldsymbol{p}_i$ considering all constraints is computed by:

$$\Delta\boldsymbol{p}_i = \sum_j \Delta\boldsymbol{p}_{ij} \left(1 - (1 - k_j)^{1/m}\right), \tag{5.2}$$

where k_j ($0 < k_j < 1$) denotes the intensity of the constraint C_j and m the current iteration count. When m reaches a predetermined number of times, the iteration process is terminated, and each position in the next step is determined.

Standard PBD methods predict rough positions in the next step by considering inertia, and then modify them using **Eqs. (5.1)** and **(5.2)**. However, because aged degradation is regarded as a quasi-static process, the proposed method directly modifies positions without considering inertia. In this thesis, such a version of PBD will be referred to as *PBD-2*. Note that the constraints explained in **Secs. 5.2**, **5.3**, and **5.4** are applicable to PBD-2, as well as to the original PBD.

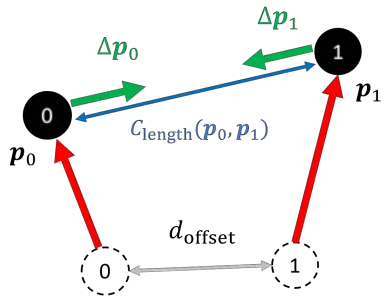


Fig. 5.1 Position corrections with a length constraint $C_{\text{length}}(\boldsymbol{p}_0, \boldsymbol{p}_1)$. Shown are two points, $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$, which are moved away from their initial positions indicated by dotted lines. In this case, the length constraint brings $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ closer together to maintain the distance d_{offset}

5.2 Length Constraint

For an edge whose ends are $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$, a constraint $C_{\text{length}}(\boldsymbol{p}_0, \boldsymbol{p}_1)$ to maintain its length at a constant value d_{offset} is defined as:

$$C_{\text{length}}(\boldsymbol{p}_0, \boldsymbol{p}_1) = \|\boldsymbol{p}_0 - \boldsymbol{p}_1\| - d_{\text{offset}}.$$

Substituting this into **Eq. (5.1)**, $\Delta\boldsymbol{p}_0$ and $\Delta\boldsymbol{p}_1$ become:

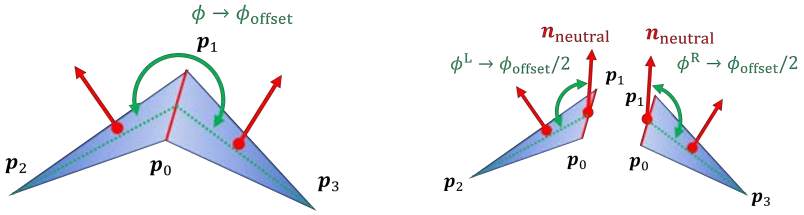
$$\Delta\boldsymbol{p}_0 = -\frac{w_0}{w_0 + w_1}(\|\boldsymbol{p}_0 - \boldsymbol{p}_1\| - d_{\text{offset}})\frac{\boldsymbol{p}_0 - \boldsymbol{p}_1}{\|\boldsymbol{p}_0 - \boldsymbol{p}_1\|},$$
$$\Delta\boldsymbol{p}_1 = +\frac{w_1}{w_0 + w_1}(\|\boldsymbol{p}_0 - \boldsymbol{p}_1\| - d_{\text{offset}})\frac{\boldsymbol{p}_0 - \boldsymbol{p}_1}{\|\boldsymbol{p}_0 - \boldsymbol{p}_1\|},$$

where both $\Delta\boldsymbol{p}_0$ and $\Delta\boldsymbol{p}_1$ are parallel to $\boldsymbol{p}_0 - \boldsymbol{p}_1$, and C_{length} acts as a spring between the two points, as illustrated in **Fig. 5.1**.

5.3 Angle Constraint

Constraints for angle control are also proposed by Müller et al. [70], but they cannot be applied to the proposed method, which must bend the mesh model. To explain why, this section introduces the angle constraint proposed by Müller et al.

As shown in **Fig. 5.2**(a), C_{angle} is a constraint to maintain the angle ϕ formed by adjacent polygons at a constant value ϕ_{offset}. Let $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ be the end positions of the edge shared by the faces, whereas $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ are the positions of the other vertices. Let us assume that $\boldsymbol{N}^{\text{L}} = (\boldsymbol{p}_0 - \boldsymbol{p}_1) \times (\boldsymbol{p}_1 - \boldsymbol{p}_2)$ and $\boldsymbol{N}^{\text{R}} = -(\boldsymbol{p}_0 - \boldsymbol{p}_1) \times (\boldsymbol{p}_1 - \boldsymbol{p}_3)$ are directed to the front side of each face. By using the unit normals of each face $\boldsymbol{n}^{\text{L}} = \boldsymbol{N}^{\text{L}}/\|\boldsymbol{N}^{\text{L}}\|, \boldsymbol{n}^{\text{R}} = \boldsymbol{N}^{\text{R}}/\|\boldsymbol{N}^{\text{R}}\|$, ϕ is given by $\phi = \text{acos}(-\boldsymbol{n}^{\text{L}} \cdot \boldsymbol{n}^{\text{R}})$. Further, the angle constraint is defined as $C_{\text{angle}}(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = \phi - \phi_{\text{offset}}$ and is substituted into **Eq. (5.1)** to enable position corrections. For more detailed derivations,



(a) Before division (b) After division

Fig. 5.2 Division of angle constraint. The neutral direction vector $\boldsymbol{n}_{\text{neutral}}$ is decided by the normals of the faces and the angles ϕ^{L} and ϕ^{R} between $\boldsymbol{n}_{\text{neutral}}$ and the faces, which are set as the targets of the constraints

refer to the paper by Müller et al. [70].

However, C_{angle} is available only if $\phi_{\text{offset}} = \pi$; otherwise, it causes a defect. For example, suppose that a pair of polygons transforms from a valley-fold shape, where $\phi = \pi - \alpha$ ($\alpha > 0$), to a mountain-fold shape, $\phi_{\text{offset}} = \pi + \beta$ ($0 < \beta < \alpha < \pi$). **Figure 5.3** shows the relation between $-\boldsymbol{n}^{\text{L}} \cdot \boldsymbol{n}^{\text{R}}$ and ϕ in this case. Because the acos function confuses the valley-fold shape $\phi = \pi - \beta$ and mountain-fold shape $\phi = \pi + \beta$, C_{angle} misunderstands the former as the target shape and stops deforming. By setting $\phi_{\text{offset}} = \pi$ as an intermediate target, this stop may be avoided. However, C_{angle} is not differentiable when $\phi = \pi$, and the corrections are computed considering a limit value of $\mathbf{0}$. Therefore, when two polygons are flat, C_{angle} cannot deform them. The atan2 function (the two-argument arctangent that divides cases according to the signs of the arguments) can be used instead of acos to distinguish between valley and mountain folds, but it is not differentiable when $\phi = \pi/2$ and $\phi = 3\pi/2$, and it cannot compute corrections. For these reasons, the proposed method cannot adopt the angle constraint, and hence, it substitutes a bend constraint, as proposed in the next section.

5.4 Bend Constraint

Physically based methods considering bending energy, such as in [14], realize a qualitative thin shell simulation, through which the bend direction can be determined, but they require more computational time than position-based methods. Thus, Jeong et al. [48] proposed a simulation method for sheet bending in a certain direction by structuring two-layer meshes, but it is only applicable to structural lattices.

In the proposed method, as shown in **Fig. 5.2**, let $\boldsymbol{n}_{\text{neutral}} = (\boldsymbol{n}^{\text{L}} + \boldsymbol{n}^{\text{R}})/\|\boldsymbol{n}^{\text{L}} + \boldsymbol{n}^{\text{R}}\|$ be a neutral direction vector and ϕ^{L} and ϕ^{R} angles between $\boldsymbol{n}_{\text{neutral}}$ and the faces. Two constraints are set such that both ϕ^{L} and ϕ^{R} approach $\phi_{\text{offset}}/2$. By setting the two angle constraints, the bend direction can be designated as a valley fold for $0 < \phi_{\text{offset}} < \pi$ and a mountain fold for $\pi < \phi_{\text{offset}} < 2\pi$. As for the example discussed in the previous section, as shown in **Fig. 5.4**, the

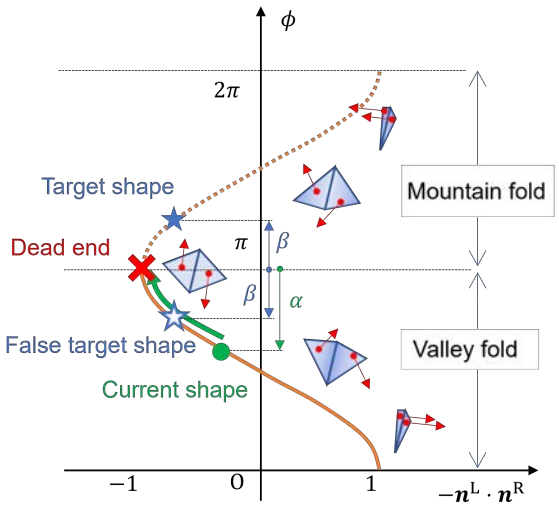


Fig. 5.3 The relation of the dot product between the normals with the angle formed by the two adjacent polygons. The acos function cannot distinguish between valley and mountain folds, and it cannot compute the gradient when the polygons are flat

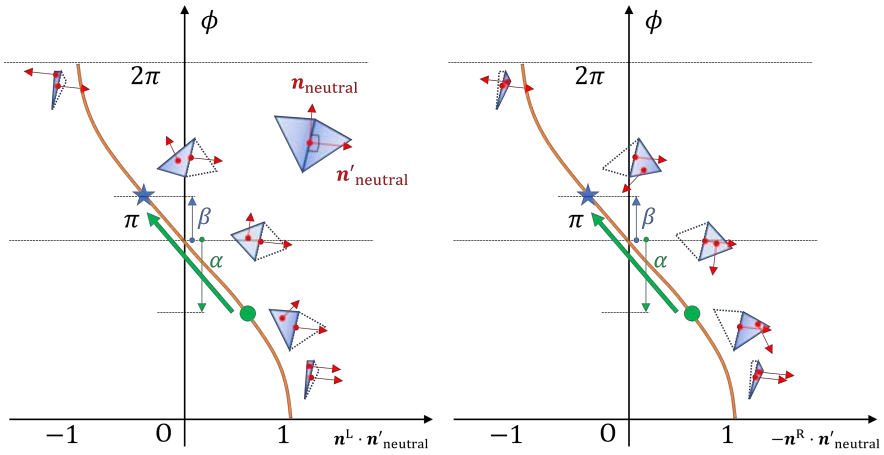
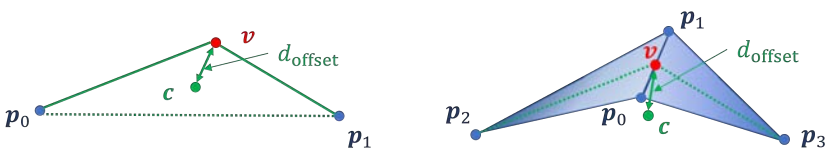


Fig. 5.4 The relation of the dot products of each normal and $\boldsymbol{n}'_{\mathrm{neutral}}$ with the angle formed by the adjacent polygons ϕ, where $\boldsymbol{n}'_{\mathrm{neutral}}$ denotes a unit vector perpendicular to $\boldsymbol{n}_{\mathrm{neutral}}$ and to the edge shared by the polygons. The constraint division reduces the behaviors of the monotonic function

polygons can reach the target shape, where $\phi = \pi + \beta$ from $\phi = \pi - \alpha$ through $\phi = \pi$.

Moreover, the application of the triangle bending constraint in [54], where an angle constraint is described as a combination of simple length constraints, provides high-quality simulations with fast convergence. As shown in **Fig. 5.5**(a), a triangle bending constraint controls the angle formed by two connected edges by setting a length constraint on the distance between the center of the triangle, which has the two edges as sides, and the junction point of the edges.

Let $\boldsymbol{v}$ be the vertex shared by the two edges and let $\boldsymbol{p}_0$ and $\boldsymbol{p}_1$ be the other vertices, so



(a) Triangle bending constraint [54] (b) Tetrahedron bending constraint

Fig. 5.5 Comparison between two bending constraints. While the target of a triangle bending constraint (a) is the angle formed by the edges, that of a tetrahedron bending constraint (b) is the angle formed by the faces

the center of three points $\boldsymbol{c}$ is obtained by:

$$\boldsymbol{c} = \frac{1}{3}(\boldsymbol{v} + \boldsymbol{p}_0 + \boldsymbol{p}_1).$$

Therefore, the triangle bending constraint C_{triangle} for $\boldsymbol{v}, \boldsymbol{p}_0$, and $\boldsymbol{p}_1$ is defined as:

$$C_{\text{triangle}}(\boldsymbol{v}, \boldsymbol{p}_0, \boldsymbol{p}_1) = \|\boldsymbol{v} - \boldsymbol{c}\| - d_{\text{offset}} = \frac{1}{3}\|2\boldsymbol{v} - \boldsymbol{p}_0 - \boldsymbol{p}_1\| - d_{\text{offset}},$$

where d_{offset} denotes the target value of the distance between $\boldsymbol{c}$ and $\boldsymbol{v}$.

Figure 5.5(b) shows a tetrahedron bending constraint, which controls the angle formed by two adjacent faces. A tetrahedron bending constraint sets a length constraint on the distance between the center of a tetrahedron, which has the two faces as surfaces, and the edge shared by the faces. Note that each polygon has a shape similar to an equilateral triangle, and the distance is equal to the distance between the center and the midpoint of the edge. If the ends of the edges are $\boldsymbol{p}_0, \boldsymbol{p}_1$ and the other vertices are $\boldsymbol{p}_2, \boldsymbol{p}_3$, the middle point of the edge $\boldsymbol{v}$ and the center of the tetrahedron $\boldsymbol{c}$ are given as:

$$\boldsymbol{v} = \frac{1}{2}(\boldsymbol{p}_0 + \boldsymbol{p}_1),$$
$$\boldsymbol{c} = \frac{1}{4}(\boldsymbol{p}_0 + \boldsymbol{p}_1 + \boldsymbol{p}_2 + \boldsymbol{p}_3).$$

Therefore, if the target value of the distance between $\boldsymbol{v}$ and $\boldsymbol{c}$ is d_{offset}, the tetrahedron bending constraint $C_{\text{tetrahedron}}$ is described as:

$$C_{\text{tetrahedron}}(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = \|\boldsymbol{v} - \boldsymbol{c}\| - d_{\text{offset}} = \frac{1}{4}\|\boldsymbol{p}_0 + \boldsymbol{p}_1 - \boldsymbol{p}_2 - \boldsymbol{p}_3\| - d_{\text{offset}}.$$

Then, the tetrahedron bending constraint is applied to the divided angle constraint, as shown in **Fig. 5.6**. Given $l^{\text{L}} = \|\boldsymbol{v} - \boldsymbol{p}_2\|, l^{\text{R}} = \|\boldsymbol{v} - \boldsymbol{p}_3\|$, the target value of the angle ϕ_{offset} $(0 < \phi_{\text{offset}} < 2\pi)$, and the positions on the neutral axis $\boldsymbol{v}^{\text{L}} = \boldsymbol{v} + l^{\text{L}}\boldsymbol{n}_{\text{neutral}}, \boldsymbol{v}^{\text{R}} = \boldsymbol{v} + l^{\text{R}}\boldsymbol{n}_{\text{neutral}}$, two tetrahedron bending constraints, $C^{\text{L}}_{\text{tetrahedron}}$ and $C^{\text{R}}_{\text{tetrahedron}}$, are defined as:

$$C^{\text{L}}_{\text{tetrahedron}}(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{v}^{\text{L}}) = \frac{1}{4}\|\boldsymbol{p}_0 + \boldsymbol{p}_1 - \boldsymbol{p}_2 - \boldsymbol{v}^{\text{L}}\| - d^{\text{L}},$$
$$C^{\text{R}}_{\text{tetrahedron}}(\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_3, \boldsymbol{v}^{\text{R}}) = \frac{1}{4}\|\boldsymbol{p}_0 + \boldsymbol{p}_1 - \boldsymbol{p}_3 - \boldsymbol{v}^{\text{R}}\| - d^{\text{R}},$$

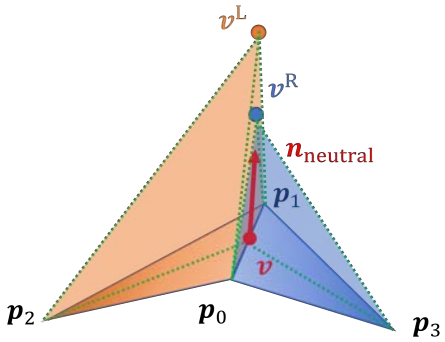


Fig. 5.6 Division of a tetrahedron bending constraint

where the constraint that offset d^{L} and d^{R} are given by:

$$d^{\mathrm{L}} = \frac{l^{\mathrm{L}}}{4}\sqrt{2\left(1+\cos\frac{\phi_{\mathrm{offset}}}{2}\right)}, \quad d^{\mathrm{R}} = \frac{l^{\mathrm{R}}}{4}\sqrt{2\left(1+\cos\frac{\phi_{\mathrm{offset}}}{2}\right)}.$$

Because $\boldsymbol{v}^{\mathrm{L}}$ and $\boldsymbol{v}^{\mathrm{R}}$ are virtual points, their inverse masses are set to 0. Let $\Delta\boldsymbol{p}_0^{\mathrm{L}}, \Delta\boldsymbol{p}_1^{\mathrm{L}}$, and $\Delta\boldsymbol{p}_2^{\mathrm{L}}$ be the corrections for $\boldsymbol{p}_0, \boldsymbol{p}_1$, and $\boldsymbol{p}_2$, respectively, according to $C^{\mathrm{L}}_{\mathrm{tetrahedron}}$, and let $\Delta\boldsymbol{p}_0^{\mathrm{R}}, \Delta\boldsymbol{p}_1^{\mathrm{R}}$, and $\Delta\boldsymbol{p}_3^{\mathrm{R}}$ be the corrections for $\boldsymbol{p}_0, \boldsymbol{p}_1$, and $\boldsymbol{p}_3$, respectively, according to $C^{\mathrm{R}}_{\mathrm{tetrahedron}}$. If the inverse mass of $\boldsymbol{p}_i$ is denoted as w_i and the sums of the inverse masses $W = w_0 + w_1 + w_2 + w_3$, $W^{\mathrm{L}} = W - w_3$, and $W^{\mathrm{R}} = W - w_2$ are defined, then **Eq. (5.1)** yields corrections for the positions, given as:

$$\begin{aligned}
\Delta\boldsymbol{p}_0 &= (\Delta\boldsymbol{p}_0^{\mathrm{L}} + \Delta\boldsymbol{p}_0^{\mathrm{R}}) \times \frac{W_0 W_1}{W(w_0+w_1)},\\
\Delta\boldsymbol{p}_1 &= (\Delta\boldsymbol{p}_1^{\mathrm{L}} + \Delta\boldsymbol{p}_1^{\mathrm{R}}) \times \frac{W_0 W_1}{W(w_0+w_1)},\\
\Delta\boldsymbol{p}_2 &= \Delta\boldsymbol{p}_2^{\mathrm{L}} \times \frac{W_0}{W},\\
\Delta\boldsymbol{p}_3 &= \Delta\boldsymbol{p}_3^{\mathrm{R}} \times \frac{W_1}{W}.
\end{aligned}$$

Hereinafter, the pair of divided angle constraints, each of which is expressed by the tetrahedron bending constraint, is referred to as a *bend constraint*.

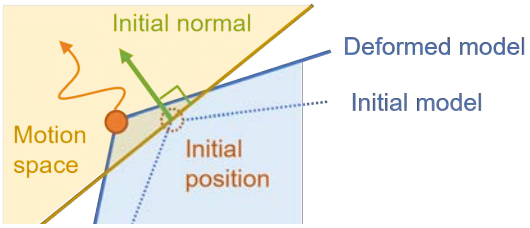


Fig. 5.7 Vertex motion space using simple methods to prevent sinking

5.5 Position Update Algorithm

This section explains the procedure to compute **Eqs. (5.1)** and **(5.2)** according to the data structure defined in **Chap. 4**. The parameters w_i, $\boldsymbol{p}_i$, and $\Delta\boldsymbol{p}_i$ correspond to member variables of `Verts[i]`: `Weight`, `AssPos`, and `Delta`, respectively, and the intensity of constraint k is saved as `LengthConstraint.Stiff` or `BendConstraint.Stiff`. Further, the procedure to update the vertex position according to the length and bend constraints is given in **Algorithm 5.1**.

Lines 10 and 18 compare the member variable of a constraint `Thr` to a variable `C` to branch the process, where the `C` holds a value computed by the definitional function of the constraint and the computation of the correction is avoided when the absolute value of `C` is less than the threshold `Thr`. In addition, because the computation is also set to inactive when the `Thr` is less than `0`, this parameter acts as a flag to alternate the constraint between active and inactive. In the proposed method, active and inactive `Thr` were set to `0.001` and `-1`, respectively; thus, substituting `0.001` or `-1` as `Thr` is referred to as *activating* or *inactivating* the constraint, respectively. Further, as can be seen from **Eq. (5.1)**, when the `Weight` of a vertex is `0`, the vertex does not move, regardless of the constraint type. Therefore, substituting `1` or `0` as the `Weight` signifies *activation* or *inactivation* of the vertex, respectively.

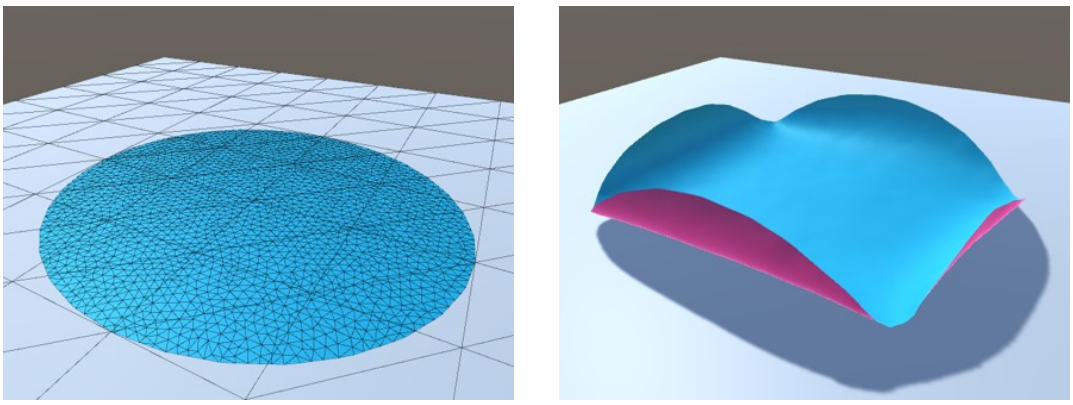
If updating positions considering only geometric constraints, coating films might sometimes sink into the surface, so judging whether the vertex would be within or outside the scope of the model to prevent sinking in line 24. General PBD detects collisions and sets constraints for resolving collisions to prevent objects from sinking, but the proposed method adopted simplified collision detection, where the motion space of a vertex is limited to the outer region of its initial position. The boundary of the motion space is the plane perpendicular to the initial normal, and it contains the initial position, as shown in **Fig. 5.7**. By setting the length and bend constraints for each edge, the polygon mesh can be bent, as shown in **Fig. 5.8**.

Algorithm 5.1 Updating vertex positions in PBD-2

Require: Length constraint list `LengthConstraints`, bend constraint list `BendConstraints`, the number of iterations `iteration`

Ensure: Update the position `V.Pos` of each element `V` in `Verts` according to the constraints

```
1: for all V in Verts do
2:   V.AssPos ← V.Pos;
3: end for
4: for m ← 1 to iteration do
5:   for all V in Verts do
6:     V.Delta ← (0, 0, 0);
7:   end for
8:   for all LC in LengthConstraints do
9:     C ← function value of LC;
10:    if 0 ≤ LC.Thr < |C| then
11:      for all V in LC.Verts do
12:        Add the weighted correction computed from LC and i to V.Delta;
13:      end for
14:    end if
15:  end for
16:  for all BC in BendConstraints do
17:    C← function value of BC;
18:    if 0 ≤ BC.Thr < |C| then
19:      for all V in BC.Verts do
20:        Add the weighted correction computed from BC and i to V.Delta;
21:      end for
22:    end if
23:  end for
24:  if V.AssPos + V.Delta is out of the initial mesh then
25:    V.AssPos ← V.AssPos + V.Delta;
26:  end if
27: end for
28: for all V in Verts do
29:   V.Pos ← V.AssPos;
30: end for
```



(a) Mesh in the initial state (b) Simulation result

Fig. 5.8 Bend simulation on a disk. The number of polygons is 4,047, and the target value of the angles formed by every two adjacent polygons is set to 0.2 rad

Chapter 6

Fracture Simulation

Chapter 5 demonstrated a method to bend an entire film, but the actual bends occur partially around cracks. Therefore, a fracture simulation should be performed in parallel with the bend simulation, and bends should be applied according to the fracture progress. Thus, this chapter presents fracture conditions based on the static balance of forces and processing when the fracture occurs, including parameter updating, topological manipulation, and constraint setting. **Section 6.1** introduces a model to consider the mechanical effects on the polygon mesh, and **Sec. 6.2** details the method to initialize and update the mechanical parameters. Finally, **Secs. 6.3**, **6.4**, and **6.5** refer to occurrence conditions and processes upon separation, tearing, and stripping, respectively.

6.1 Basic Model

To reproduce the peeling of coated films using a polygon mesh, a basic, as model shown in **Fig. 6.1**, is introduced, wherein each polygon applies lifting and contraction forces to its adjacent polygons, while each node attaches to the base in the initial state, but its adhesion force may weaken and separate from the base over time. The contraction force becomes strong during the simulation, but the connection of two adjacent polygons is torn when the contraction force becomes stronger than the binding force of the polygons.

6.2 Initialization and Updating of Parameters

This section explains the initialization and updating of parameters expressing the magnitude of forces. Note that lifting force does not require initialization because it is computed from contraction force in every frame, as detailed in the next section.

The initial distributions of the adhesion force and binding force are generated by Perlin noise [80] to reproduce a realistic output that is non-uniform and continuous. The correlation between the distributions is debatable, but in this thesis, they are assumed independent of each other, and the adhesion force and binding force were initialized by Perlin noise, whose mean values

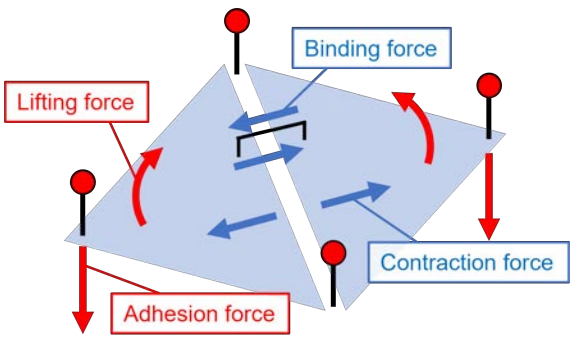


Fig. 6.1 Basic model. Red and blue arrows indicate forces that are compared to judge separation and tearing, respectively

for each are 100 and 750, respectively. The binding force is set to be extremely low on certain edges to reproduce defects, as mentioned in **Sec. 3.1**. The existence probability of the weak edges was empirically set to 1 % and the binding force of the weak edges was also empirically set to 0.1 % of the other edges. Further, the contraction force of each face was set to 100, assuming contraction forces are balanced.

When the elapsed time per frame is `dT`, update formulas of the adhesion force `Vert.Adhere`, the contraction force `Face.Contraction`, and the binding force `Edge.Binding` vary linearly with time as follows:

$$\begin{aligned} &\texttt{Vert.Adhere} \leftarrow \texttt{Vert.Adhere - dT * speedAd};\\ &\texttt{Face.Contraction} \leftarrow \texttt{Face.Contraction + dT * speedCo};\\ &\texttt{Edge.Binding} \leftarrow \texttt{Edge.Binding - dT * speedBi[n]}; \end{aligned} \tag{6.1}$$

where `speedAd`, `speedCo`, and `speedBi[n]` are coefficients representing the speed change of each force, and `speedAd` and `speedC` were set to `0.25` and `0.50`, respectively. The parameter `n (n = 0, 1, 2, 3)` denotes the number of torn edges on the perimeter of `Edge.Face`, while the values for `speedBi[0]`, `speedBi[1]`, `speedBi[2]`, and `speedBi[3]` were empirically set to `0.0`, `2.0`, `10.0`, and `0.0`, respectively. Note that if a force parameter is a negative value, it is continuously decreased in this updating process, though it is regarded as `0` in the other processes. This is to prevent the parameter distributions from flattening, and the negative parameter value acts as a delay until it begins to increase when reversing the simulation.

Next, the initialization of constraints in PBD-2 is explained, such that `LengthConstraint.Stiff` and `BendConstraint.Stiff` were set to `1.00` and `0.95`, respectively, and these values are fixed during the simulation. Both the `Offset` and `Offset0` values of each constraint are initialized so that the constraint is completely satisfied. At either of the two half-edges that make up an edge, both length and bend constraints are activated, while on the other half-edge, both are inactivated. The effects of PBD-2 do not appear at all in the initial state because all constraints and vertices are inactivated. The `Offset` values of the `LengthConstraint` and `BendConstraint` corresponding to a half-edge `E` are updated by:

$$\begin{aligned} &\texttt{LengthConstraint.Offset} \leftarrow \texttt{LengthConstraint.Offset0}\\ &\qquad\qquad\qquad\qquad \texttt{* (1 - shrink * E.Face.Contraction)};\\ &\texttt{BendConstraint.Offset} \leftarrow \texttt{BendConstraint.Offset0 + curl * E.Lifting}; \end{aligned} \tag{6.2}$$

where `shrink` and `curl` denote sensitivity parameters of shrink and bend, respectively. The `curl` value was able to vary in the range `[0, 0.05]` by an external control, while the `shrink` value was fixed to `0.001`. These update formulas cause the coating films to shrink and bend as the contraction and lifting force increase.

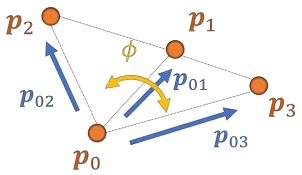


Fig. 6.2 Angle formed by two polygons that sandwich an edge

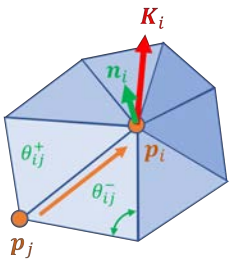


Fig. 6.3 Discrete mean curvature at a vertex

The angle formed by adjacent polygons `Edge.Ang`, and the curvature `Vert.Cur` is a geometric property computed from vertex positions for each frame, and it does not require initialization. The procedure of computing `Edge.Ang` is given in **Algorithm 6.1**, while the parameters used for this computation are illustrated in **Fig. 6.2**. This algorithm is almost the same as the computing procedure of ϕ in **Sec. 5.3**, but it differs in that the triple product of vectors is found to distinguish mountain and valley folds on line 8. Conversely, `Vert.Cur` is obtained using the discrete mean curvature [64]. As illustrated in **Fig. 6.2**, the discrete mean curvature value at a vertex $\boldsymbol{p}_i$ is denoted by $\boldsymbol{K}_i$, the set of vertices connected to the vertex is denoted by $N(i)$, and the respective angles of the corners opposite the line segment $\boldsymbol{p}_i\boldsymbol{p}_j$ in each of the two polygons that have $\boldsymbol{p}_i\boldsymbol{p}_j$ as an edge are denoted by θ_{ij}^+ and θ_{ij}^-. If the sum of areas of polygons that have $\boldsymbol{p}_i$ as a vertex is A_{all}, $\boldsymbol{K}_i$ is computed by:

$$\boldsymbol{K}_i = \frac{1}{2A_{\text{all}}} \sum_{\boldsymbol{p}_j \in N(i)} \left(\cot\theta_{ij}^+ + \cot\theta_{ij}^-\right)(\boldsymbol{p}_i - \boldsymbol{p}_j).$$

The discrete mean curvature is a vector in the direction of convexity whose length represents the magnitude of the curvature. Therefore, by computing the dot product with the normal, it is converted to a scalar that is positive when convex on the front side. The procedure to obtain `Vert.Cur` based on the above consideration is given in **Algorithm 6.2**.

Algorithm 6.1 Computing the angle formed by two polygons that sandwich an edge

Require: A half-edge `E` that is not torn and has the pair

Ensure: Store angle formed by polygons connected to `E` to `E.Ang`

```
p0 ← E.Vert.Pos;
p01 ← E.Pair.Vert.Pos − p0;
p02 ← E.Pair.Prev.Vert.Pos − p0;
p03 ← E.Pair.Prev.Pos − p0;
n012 ← normalize(p01 × p02);
n013 ← normalize(p01 × p03);
phi ← acos(−n012 · n013) − π;
E.Ang ← phi × sign((p01 × p02) · p03);
```

Algorithm 6.2 Computing the curvature at a vertex

Require: `V` such that `E.Stat = 0` for each `E` connected to `V`

Ensure: Store the dot product of the discrete mean curvature and `V.Nor` to `V.Cur`

```
sumA ← 0;
count ← 0;
k ← (0, 0, 0);
homeEdge ← V.Edge;
curtEdge ← homeEdge;
repeat
  pi ← curtEdge.Vert.Pos;
  pj ← curtEdge.Next.Vert.Pos;
  pk ← curtEdge.Prev.Vert.Pos;
  pl ← curtEdge.Pair.Prev.Vert.Pos;
  cosA ← normalize(pi - pk) · normalize(pj - pk);
  cosB ← normalize(pi - pl) · normalize(pj - pl);
  cosA2 ← cosA * cosA;
  cosB2 ← cosB * cosB;
  cotA ← sqrt(cosA2 / (1 - cosA2)) * sign(cosA);
  cotB ← sqrt(cosB2 / (1 - cosB2)) * sign(cosB);
  k ← k + (cotA * cotB) * (pi - pj);
  sumA ← sumA + length((v1 - v0) × (v2 - v0));
  curtEdge ← curtEdge.Pair.Next;
until curtEdge is not homeEdge
V.Cur ← k · V.Nor / sumA;
```

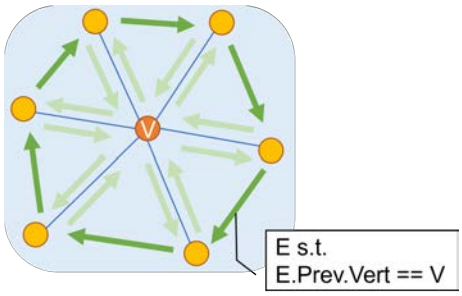


Fig. 6.4 Half-edges around a vertex, which provide lifting forces to the vertex

6.3 Separation

A vertex `V` is determined to separate from the base if `V.Stat == 0`, and the sum of lifting forces from half-edges around `V` is greater than `V.Adhere`, where half-edges around `V` refer to half-edges `E` such that `E.Prev.Vert == V`, as illustrated in **Fig. 6.4**. In this case, `V.Stat` is set to `1`, and `V` is activated in PBD-2.

A half-edge `E` gives Vertex `E.Prev.Vert` lifting force `E.Lifing`. Thus, if `E` is on a flat surface, according to **Eq. (3.3)**, the value of the lifting force is computed as:

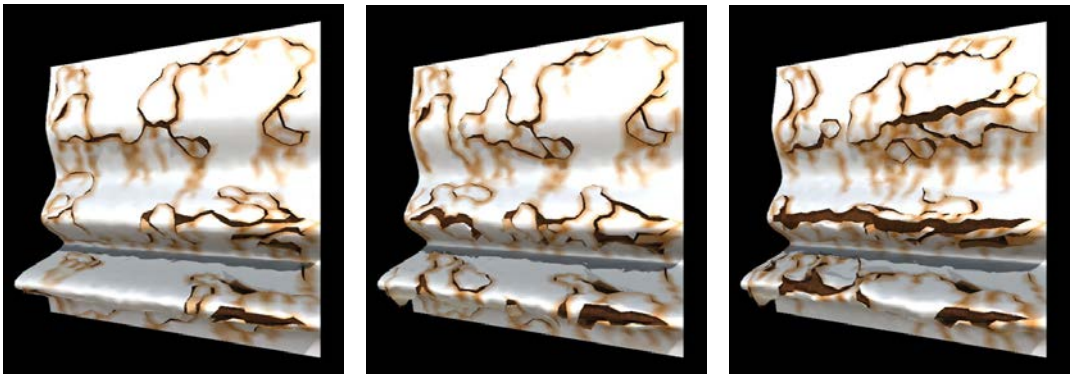
```
E.Lifing ← kL * E.Face.Contraction;
```

where `kL` denotes a proportion coefficient that expresses the ease of separation, and its value was empirically set to `kL = 0.1`.

The series of computations described in **Sec. 3.1** assumes the model is initially flat. However, when observing actual coated films on curved objects, convex parts appear to peel easily. To reproduce this kind of curvature effect, the lifting force is multiplied by the value of the sigmoid function `sigmoid`, whose input is the angle `E.Ang` formed by faces connected to `E`:

```
sigmoid ← 1 / (1 + exp(-gamma * E.Ang));
E.Lifting ← kL * E.Face.Contraction * sigmoid;
```

where `gamma` is a positive constant. The larger its value, the greater the effect of the curvature on the ease of peeling. As shown in **Fig. 6.5**(a), if `gamma = 1`, the effect of the curvature is so small that aged deterioration uniformly progresses, even along the steep curve at the bottom of the model. On another front, as shown in **Fig. 6.5**(c), if `gamma = 4`, the effect of the curvature is so large that the difference in the degree of aged deterioration appears even on the gentle curve at the top of the model. Thus, a moderate condition of `gamma = 2` was adopted, where the difference in the degree of aged deterioration appears only on somewhat steep curves, as shown in **Fig. 6.5**(b).



(a) `gamma = 1.0` (b) `gamma = 2.0` (c) `gamma = 4.0`

Fig. 6.5 Sensitivity analysis of the geometric property `gamma`

6.4 Tearing

For a half-edge `E` to tear, all the following preconditions must be satisfied:

- `E` has not yet torn: `E.Stat == 0`;
- The starting point of `E` is separated: `E.Vert.Stat > 0`;
- The end point of `E` is separated: `E.Vert.Next.Stat > 0`;

`E` is determined to tear when it meets a standard or special condition, as described below, in addition to the above three conditions. By letting `E.Pair` be `Ep`, the standard condition of tearing is:

```
E.Face.Contraction + Ep.Face.Contraction > E.Binding + Ep.Binding;
```

This means the sum of the contraction forces from the adjacent polygons is greater than the binding force.

In contrast, the special condition is applied when `E.Vert.Stat` or `Ep.Vert.Stat` is `2`, and it is satisfied by contraction forces less than the standard condition due to the stress concentration. Because the computation when `E.Vert.Stat == 2` is the same as that for `Ep.Vert.Stat`, only the former is explained below. When `E.Vert.Stat` is `2` and the starting point of `E` is one end of a crack, stress concentrates around the crack end, and the effect of concentrated stress is computed according to **Eq. (3.5)** in **Sec. 3.1**. As for **Eq. (3.5)**, because the existing crack is much longer than the thickness of the coating film, it is assumed that the crack length will not affect the stress concentration and that the stress intensity factor is proportional to the contraction force. Moreover, if opening-direction stress at the end point of `E` is supposed to be proportional to an additional contraction force, the special condition is formulated as:

```
E.Vert.Stat == 2 and
(1 + s * kT / sqrt(E.Length)) * (E.Face.Contraction + Ep.Face.Contraction)
> E.Binding + Ep.Binding;
```

where `s` denotes a proportion coefficient that expresses the intensity of the stress concentration, `kT` the value of the function introduced in **Eq. (3.6)**, and `sqrt(E.Length)` the square root of the length of `E`. Note that the azimuth used for the computation of `kT` is determined by using the crack direction parameter `E.Vert.Direction` as the polar axis.

If `E` tears, `E.Stat` and `Ep.Stat` are set to `1`, and `E.Vert.Stat` and `Ep.Vert.Stat` are incremented by `1`. As for PBD-2, bend constraints corresponding to `E` and `Ep` are inactivated, whereas length constraints are activated, and moreover, the vertex parameters are updated and the topology is manipulated. These processes are also applied to `E.Vert` and `Ep.Vert` similarly, so only the case of `E.Vert` is explained below. The tearing process for the vertex `E.Vert` branches into three patterns according to the values of `E.Vert.Stat` and `E.Vert.OnEdge`. Note that `E.Vert.Stat` is more than `1` at this time because the condition `E.Vert.Stat > 0` is required for tearing, and it is incremented by `1` just after tearing. The three branched processes are as follows:

Pattern A: `E.Vert.Stat == 2` and `E.Vert.OnEdge == 0` (**Fig. 6.6**(a))

This process occurs when a crack is generated or extended inside the model and `E.Vert` becomes the end of the crack. In this case, the crack direction vector is stored in `E.Vert.Direction`.

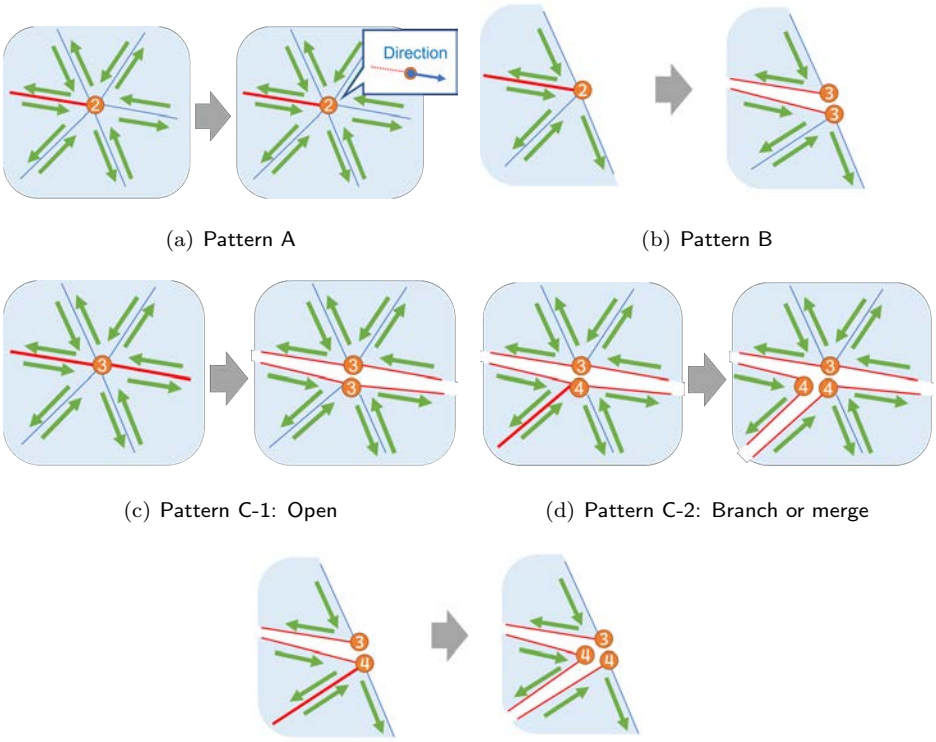
Pattern B: `E.Vert.Stat == 2` and `E.Vert.OnEdge == 1` (**Fig. 6.6**(b))

This process occurs when a crack is generated at the edge of the model or has reached the edge. In this case, `E.Vert.Stat` is changed to `3` and `E.Vert` is duplicated.

Pattern C: `E.Vert.Stat > 2` (**Fig. 6.6**(c)–(e))

This process occurs when a crack is extended through `E.Vert` or `E.Vert` is a junction of cracks. In this case, `E.Vert` is duplicated.

Figure 6.7 illustrates the method to compute the crack direction in Pattern A. The crack direction is basically a unit vector in the direction from the end point to the starting point of `E`. If the extension of a crack in Pattern A has been already applied to `Ep.Vert`, the end of a crack is moved from `Ep.Vert` to `E.Vert`, and the crack direction is changed from `Ep.Vert.Direction` to `E.Vert.Direction`. However, the crack direction is updated each time the crack extends, so the complexity of the crack shape strongly depends on the resolution of the mesh. Therefore, crack shape manipulation is realized by reflecting the previous crack direction in `E.Vert.Direction`. If `oldDir` denotes a vector obtained by projecting `Ep.Vert.Direction` onto the plane perpendicular to `E.Vert.Nor` and `newDir`, a unit vector in the direction from the end point to the starting point



(a) Pattern A (b) Pattern B

(c) Pattern C-1: Open (d) Pattern C-2: Branch or merge

(e) Pattern C-3: Merge on the model edge

Fig. 6.6 Patterns of the tearing process. The number in vertices shows the `Stat` value.

of `E`, `E.Vert.Direction` is found to be the weighted linear sum of these vectors:

```
E.Vert.Direction ← oldDir * h + newDir * (1 - h);
```

where `h` is a parameter that varies in the range `[0, 1]`, which expresses the reflection ratio of the previous direction, that is, the persistence of the crack direction. **Figure 6.8** shows a sensitivity analysis of `h`, where because `h` is large, the crack direction is maintained, and the crack shape becomes linear. The reason why `Ep.Vert.Direction` is projected and normalized is that the crack must proceed along the surface. **Figure 6.9** compares a case where `Ep.Vert.Direction` is directly set as `oldDir` and a case with projection and normalization. Without projection or normalization, the crack direction vector gradually leaves the surface and points in the air or inside the model, so the cracks stop in an unnatural pattern.

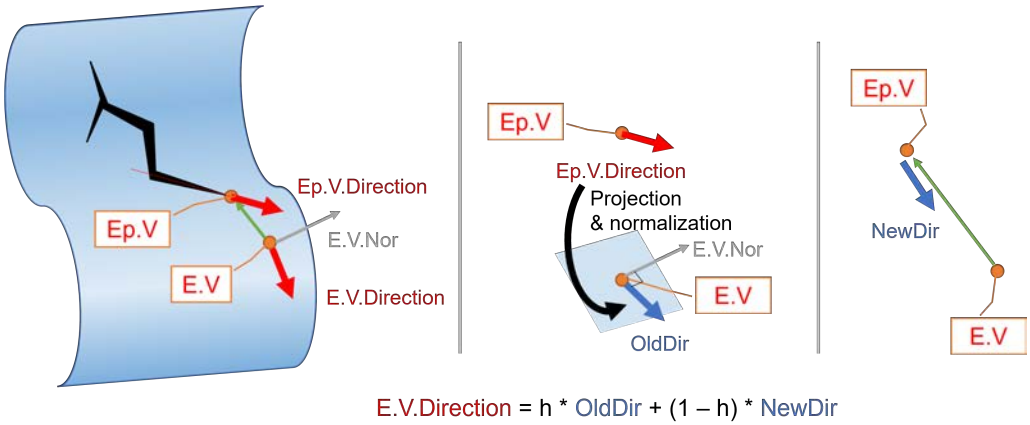
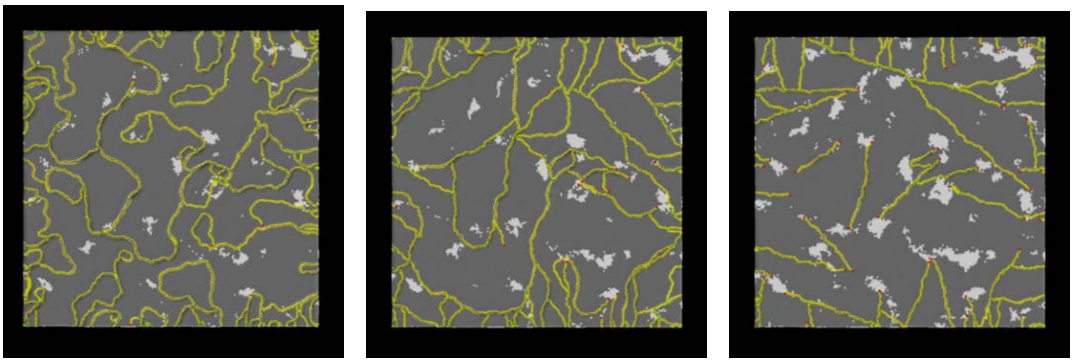


Fig. 6.7 Update of the crack direction reflecting the previous direction. The direction is the weighted sum of the direction of the newly extended crack `newDir` and the projected and normalized previous direction `oldDir`

The duplication of `E.Vert` in Patterns B and C requires updating of the topological data and target vertices of the constraints. In addition, to achieve the binding of cracks, as detailed in **Sec. 8.1**, `E.Stat` and `Ep.Stat` are set to 3 if either of the following conditions is satisfied:

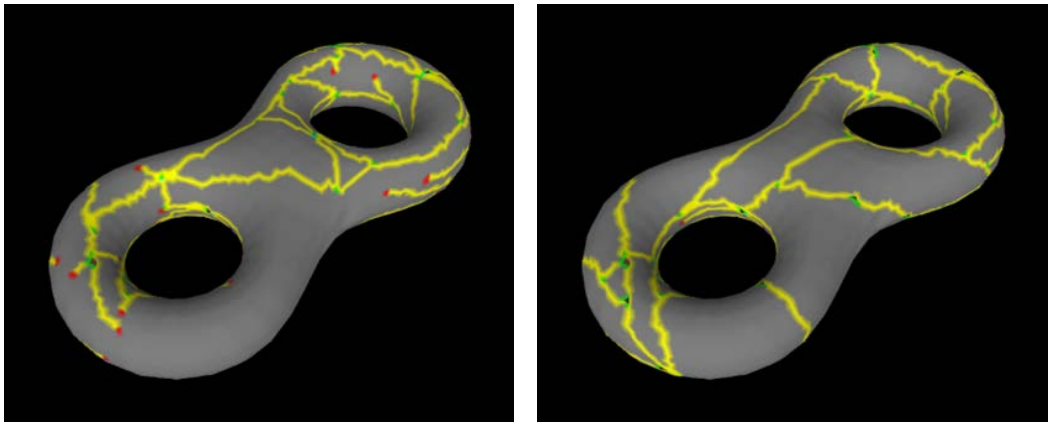
- `E.Vert` was processed following Pattern B (**Fig. 6.10**(a));
- `E.Vert` was processed following Pattern C, and all of the following conditions are satisfied (**Fig. 6.10**(b)):
  - `E.Vert.Stat > 2`;
  - `Ep.Vert.Stat > 2`;
  - `E.Vert.Stat == 3` **and** Ep.Vert.Stat $==$ 3;

In these cases, the end of a crack disappears due to the tearing of `E`. Strictly speaking, when `Ep.Vert.Stat` is 2 in Pattern B, the end is only on the model edge right after tearing, but `E.Vert` is split soon after to remove the end.



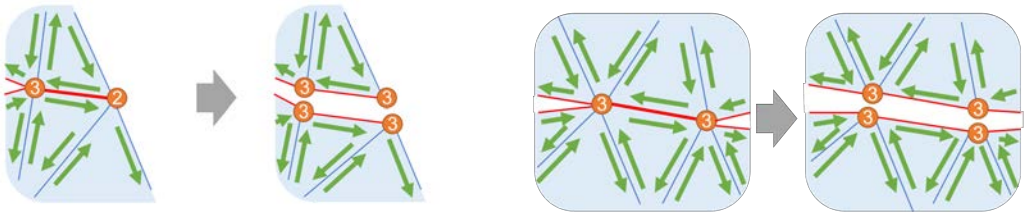
(a) `h = 0.0` (b) `h = 0.5` (c) `h = 1.0`

Fig. 6.8 Sensitivity analysis of the persistence of the crack direction `h`. The color indicates the vertex status `Stat`. White denotes `0` (bonding), gray `1` (separated), red `2` (end of a crack), yellow `3` (path of a crack), and green `4` (junction of cracks). Cracks are likely to proceed straight, as `h` is large



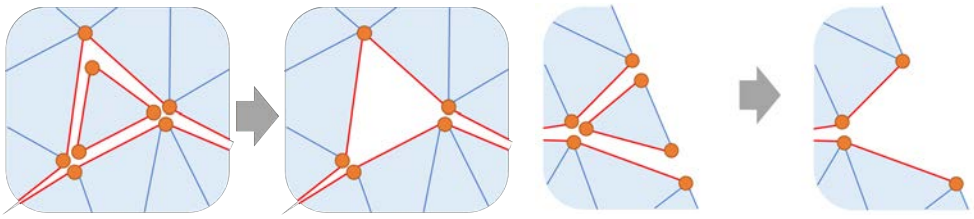
(a) Without projecting the crack direction (b) With projecting the crack direction

Fig. 6.9 The effect of projecting the crack direction onto the model surface. In both (a) and (b), the persistence of cracks was set to `h = 1.0`, so the cracks run straight. While cracks stop on the curved surface because the directions point in the air or inside the model in (a), cracks can extend, as in the case of a flat plane, because the directions are updated to run along the model surface



(a) Arrival at the model edge (b) Connection between cracks

Fig. 6.10 Cases in which a crack end disappears



(a) Case when all three edges on the perimeter are torn (b) Case when two of edges on the perimeter are torn and the other is on the model edge

Fig. 6.11 Stripping process

6.5 Stripping

A face is determined to be stripped if each of the three edges on the perimeter of the face is torn or on the model edge, as shown in **Fig. 6.11**. It is sufficient to determine the stripping only when a half-edge `E` tears. If `E.Next.Stat > 0` and `E.Prev.Stat > 0`, `E.Face` is stripped.

If `E.Face` is stripped, the following processes must be performed:

1. Set `E.Face.Stat` to `1`;
2. Inactivate each length constraint corresponding to `E`, `E.Next`, and `E.Prev`;
3. Inactivate `E`, `E.Next`, and `E.Prev` in PBD-2.

In the algorithm, to generate mesh for rendering, as shown in **Algorithm 4.1**, a face `F` such that `F.Stat == 1` is not generated in the mesh for rendering. Rather, it hides stripped polygons while preserving their DCEL data.

Chapter 7

Stain Expression

Aged deterioration of coating films causes not only deformation but also color and material changes. However, these changes to the texture of coating films due to aged degradation are difficult to construct using rigorous mathematical models, which were not considered in the existing weathering methods for coating films [79][31]. This research focuses on chalking caused by light exposure, rust run-off stains caused by rains, and darkening caused by dust accumulation, and the methods for depicting them are described in this chapter. The process for computing the degree of chalking, rust run-off stain, and dust accumulation is presented in **Secs. 7.1**, **7.2**, and **7.3**, respectively. In addition, **Sec. 7.4** represents how these stains are reflected in the model's appearance.

7.1 Chalking

In reality, chalking is a chemical change caused by light, heat, and ambient air, and it progresses almost uniformly on an object, though there may be some bias depending on the shape. The degree of chalking `V.Chalk` at each vertex `V` is initialized to `0` and linearly updated as follows, independent of the geometric data:

$$\texttt{V.Chalk} \leftarrow \texttt{V.Chalk + dT * speedCh};$$

where `speedCh` denotes the speed of chalking, whose value was empirically set to `speedCh` $= 5.0 \times 10^{-3}$.

7.2 Rust Run-off

Rust run-off stains are expressed by controlling the amount of rust `V.Rust` at each vertex `V`, which varies in the range `[0, 1]`. In the initial state, each `Rust` value is set to `0`. At rust sources set around cracks, `Rust` values are set to `1` and are propagated downward along the surface.

Whether a vertex `V` is a rust source is stored in the boolean `V.RustSource`, which is initially set to `false`. If a half-edge `E` whose starting point is `A` and end point is `B` tears, `RustSource` is determined according to the following four criteria:

- If `A` is the end of a crack and below `B`, `A` is designated a source (**Fig. 7.1** (a)) :
  if `A.Stat` **is** 2 **and** `(A.Pos - B.Pos)` $\cdot$ `gravity > 0`, **then** `A.RustSource` $\leftarrow$ `true`;
- If `A` is the end of a crack and above `B`, `A` is designated a non-source (**Fig. 7.1** (b)) :
  if `A.Stat` **is** 2 **and** `(A.Pos - B.Pos)` $\cdot$ `gravity < 0`, **then** `A.RustSource` $\leftarrow$ `false`;
- If `A` is not the end of a crack and below `B`, `A` is unchanged (**Fig. 7.1** (c)) :
  if `A.Stat` **is not** 2 **and** `(A.Pos - B.Pos)` $\cdot$ `gravity < 0`, **then** do nothing;
- If `A` is not the end of a crack and above `B`, `A` is designated a source (**Fig. 7.1** (d)) :
  if `A.Stat` **is not** 2 **and** `(A.Pos - B.Pos)` $\cdot$ `gravity < 0`, **then** `A.RustSource` $\leftarrow$ `false`;

Note that if `A` is duplicated, the duplicate `A'` is located above `A`, so `A'` is designated a non-source.

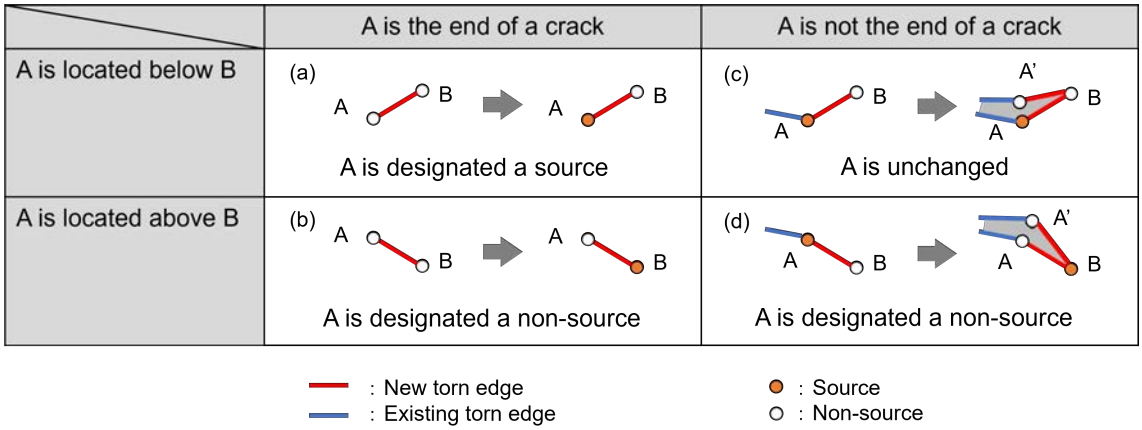


Fig. 7.1 Setting of rust sources at tearing

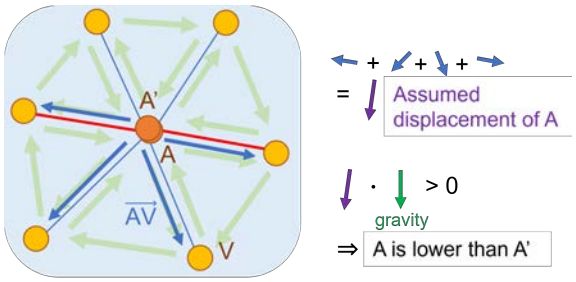


Fig. 7.2 Determination of vertical position relation between vertex `A` and its duplicate `A'`. Because `A` and `A'` are at the same position immediately following duplication, the relation is determined by predicting the direction in which `A` moves

However, `A.Pos` and `A'.Pos` are the same values, and the heights of these two vertices cannot be compared right after tearing. Therefore, the displacement of `A` by PBD-2 is approximately predicted by summing vectors $\overrightarrow{AV}$, such that `V` is connected to `A`, as shown in **Fig. 7.2**. Then, if the dot product of the predicted displacement and the gravity direction vector `gravity` are positive, `A` is determined to be below `A'`; otherwise, `A` is above `A'` and swapped for `A'`.

Because external factors, such as wind, often alter flow paths over the long term, rust diffuses and propagates downward, and **Algorithm 7.1** gives the procedure for propagating the value of `V.Rust` downward. The propagation process must reference the rust amounts at nearby vertices. However, if `Rust` is directly referenced, the result depends on the order of the vertices to be updated, because `Rust` contains a mixture of values before and after the update. Therefore, the current value is saved to `V.RustBuf` at first. In subsequent processes, by using `Rust` for write and `RustBuf` for read, each vertex can reference the pre-updated values of `Rust`.

Next, the rust inflow and outflow amounts at each vertex `V` are computed. However, the vertex `V` designated as a rust source is assumed supplied with rust from the base to maintain the rust amount, so `V.Rust` at `V` is not updated. Meanwhile, the direction of rust flow `flowDir` is

determined by projecting the gravity direction `gravity` onto the plane perpendicular to `V.Nor` and by normalizing it, as illustrated in **Fig. 7.3**.

By allowing `S` to be one of the vertices connected to the vertex `V`, the rust transfer between `V` and `S` can be computed. If the normalized vector of $\overrightarrow{SV}$ is `pathDir`, the rust transfer is classified into the following five patterns, as shown in **Figure 7.5**, and the amount of rust transferred `flowRust` from `S` to `V` is obtained:

Large inflow: when `flowDir · pathDir` $\geq$ `dirThr1` and `S.RustBuf > amoThr`:

`flowRust` $\leftarrow$ `(flowDir · pathDir - dirThr1) * (S.RustBuf - amoThr);`

Small inflow: when `dirThr1 < flowDir · pathDir < dirThr2` and `S.RustBuf > amoThr`:

`flowRust` $\leftarrow$ `(flowDir · pathDir - dirThr2) * (S.RustBuf - amoThr) * rustRatio;`

Large outflow: when `- flowDir · pathDir` $\geq$ `dirThr1` and `V.RustBuf > amoThr`:

`flowRust` $\leftarrow$ `(- flowDir · pathDir + dirThr1) * (V.RustBuf - amoThr);`

Small outflow: when `dirThr1 < - flowDir · pathDir < dirThr2` and `V.RustBuf > amoThr`:

`flowRust` $\leftarrow$ `(- flowDir · pathDir + dirThr2) * (V.RustBuf - amoThr) * rustRatio;`

No inflow or outflow: when other than the above four cases:

`flowRust` $\leftarrow$ `0;`

where `dirThr1` and `dirThr2` are the parameters that control the range of areas with much and little rust flow, respectively, satisfying `0 < dirThr1 < dirThr2 < 1`. Further, `rustRatio` denotes the ratio of a small amount to a significant amount of rust. In addition, rust cannot completely flow from any area, so it can be assumed that a vertex that has received rust inflow always holds at least a certain amount of rust `amoThr`, and only excess rust propagates downward. The values of these control parameters of rust run-off were empirically set to `dirThr1 = 0.10`, `dirThr2 = 0.45`, `rustRatio = 0.20`, and `amoThr = 0.20`.

The computed `flowRust` is multiplied by the time step `dT` and the speed of rust run-off `speedRu` and then added to `V.Rust`, where the value of `speedRu` was set to `0.10`. `V.Rust` is updated completely by performing this computation for each nearby vertex `S`, as given in line 23.

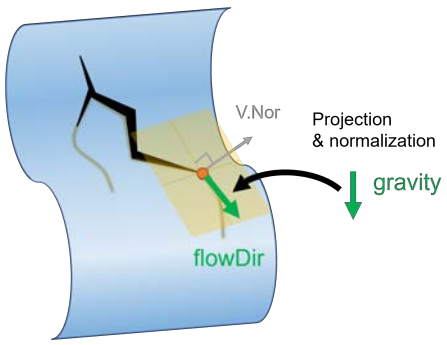


Fig. 7.3 The computation of the rust flow direction `flowDir` at a vertex `V`. The direction is found by projecting the gravity direction vector `gravity` onto the plane perpendicular to `V.Nor` and by normalizing it

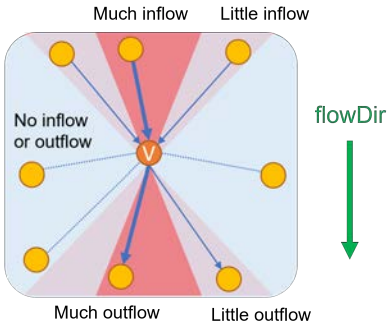


Fig. 7.4 Transfer of rust from a vertex `V` to its nearby vertices

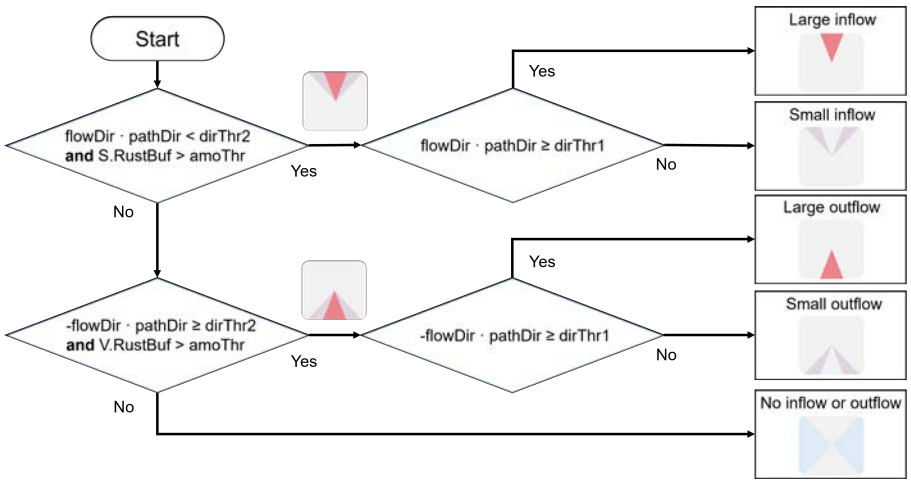


Fig. 7.5 Flowchart for determining the transfer amount

Algorithm 7.1 Propagating rust

Require: Vertex list `Verts`, gravity direction vector `gravity`, the time step `dT`, and the control parameters for rust run-off `dirThr1`, `dirThr2`, `amoThr`, `rustRatio`, and `speedRu`

Ensure: Update `V.Rust` for each `V` that is an element of `Verts`

```
for all V in Verts do
  V.RustBuf ← V.Rust;
end for
for all V in Verts do
  if V.Source is true then
    continue;
  end if
  flowDir ← a vector normalized after projecting gravity onto the plane perpendicular to V.Nor;
  for all S in set of vertices connected to V do
    pathDir ← a normalized vector of V.Pos - S.Pos;
    flowRust ← 0;
    if flowDir · pathDir < dirThr2 and S.RustBuf > amoThr then
      flowRust ← (flowDir · pathDir - dirThr1) * (S.RustBuf - amoThr);
      if flowDir · pathDir ≥ dirThr1 then
        flowRust ← flowRust * rustRatio;
      end if
    else if -flowDir · pathDir ≥ dirThr2 and V.RustBuf > amoThr then
      flowRust ← (- flowDir · pathDir + dirThr1) * (V.RustBuf - amoThr);
      if -flowDir · pathDir ≥ dirThr1 then
        flowRust ← flowRust * rustRatio;
      end if
    end if
    V.Rust ← V.Rust + flowRust * dT * speedRu;
  end for
end for
```

7.3 Dust Accumulation

As shown in **Fig. 7.3**, the amount of accumulated dust `V.Dust` at a vertex `V` is increased according to two geometric properties: the normal `V.Nor` and the curvature `V.Cur`. If `speedDu` denotes the accumulation speed of the dust, the update formula of `V.Dust` is:

$$\texttt{V.Dust} \leftarrow \texttt{V.Dust + dT * speedDu * (1 + rN + rC);}$$

where `rN` and `rC` are the influence ratio of the normal and curvature to the factor independent of geometry, respectively, the computation of which is explained below. In addition, the accumulation speed was set to `speedDu = 2.0` $\times 10^{-4}$.

The closer the direction of the normal `V.Nor` is to the opposite direction opposite the gravity vector `gravity`, the faster dust accumulates there, because the surface is upward. Therefore, the influence ratio of normal `rN` is computed as:

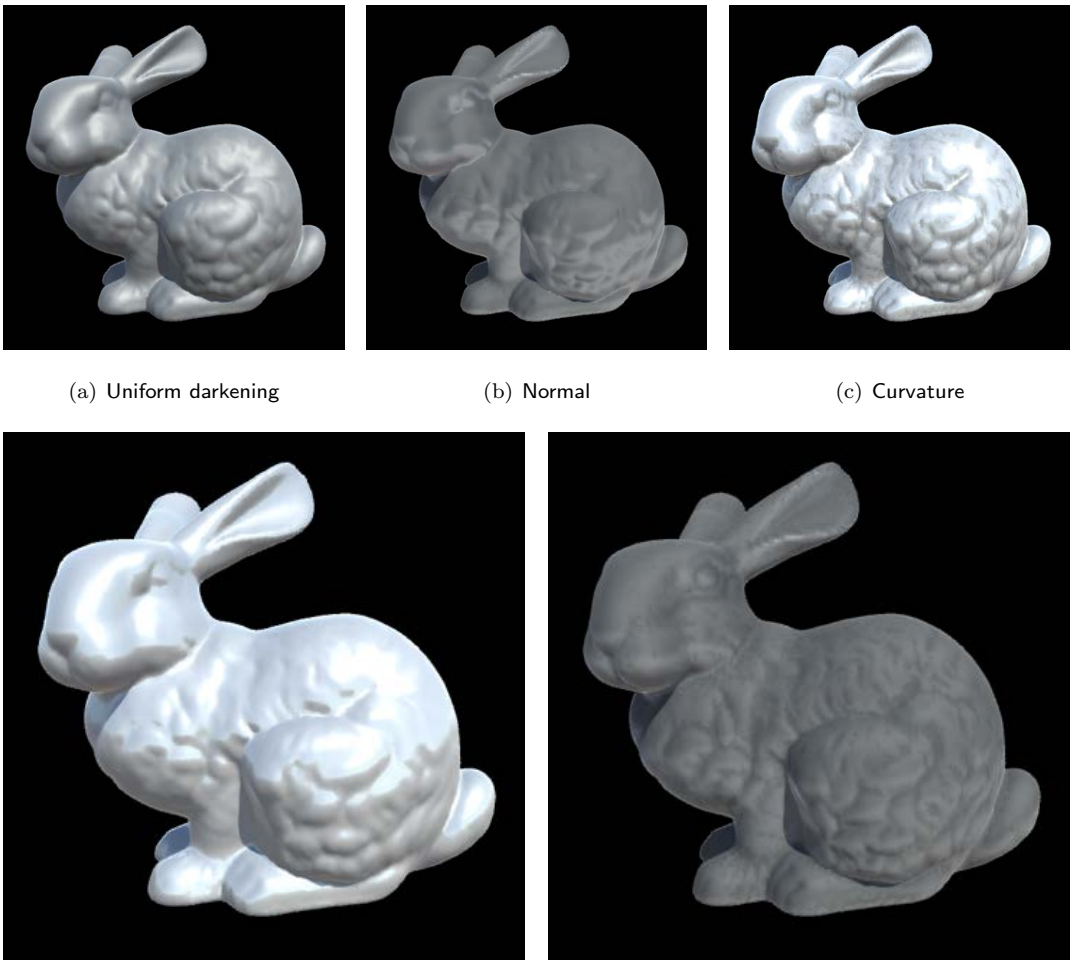
$$\texttt{rN} \leftarrow \texttt{weightNor / (1 + exp(-gainN * asin(-v.Nor} \cdot \texttt{gravity)));}$$

where `weightNor` is a coefficient to manipulate the influence ratio of the normal `rN`, whose value was empirically set to `weightNor = 4.0`. Conversely, `gainN` is the sensitivity of the accumulation speed to the normal, where the larger its value, the greater the difference of darkening between the upward and downward surfaces. The value of `gainN` was empirically set to `gainN = 10.0`. As the direction is changed, the value of `rN` varies smoothly in the range of `[0, weightNor]`.

In contrast, `rC` is computed so the smaller the curvature `V.Cur`, the larger its value, as follows:

$$\texttt{rC} \leftarrow \texttt{weightCur / (1 + exp(gainC * V.Cur));}$$

where `weightCur` denotes a coefficient for manipulating the influence ratio of the curvature `gainC` and the sensitivity of the accumulation speed to the normal, and these values were empirically set as `weightCur = 1.0` and `gainC = 1.0`, respectively. The value of `rC` varies in the range of `[0, weightCur]`.



(a) Uniform darkening (b) Normal (c) Curvature

(d) Initial state (e) All factors

Fig. 7.6 Geometric factors and appearance of dust accumulation

7.4 Rendering Stains

The time variation of the texture at a vertex `V` is expressed by manipulating the vertex color `col` and reflection `ref` according to parameters that represent the degree of stains `V.Chalk`, `V.Chalk`, and `V.Chalk`. First, the following process normalizes the stain parameters `V.Chalk`, `V.Chalk`, and `V.Chalk` to the range[0, 1] to compute the variants `chalk`, `rust`, and `dust`, respectively:

```
chalk ← 1 - exp(- V.Chalk);
rust ← clump01(V.Rust);
dust ← 1 - exp(- V.Dust);
```

where `clamp01(V.Rust)` is a function that returns `max(0, min(V.Rust, 1))`. Let `whiteColor` be the white color, `blackColor` the black color, and `rustColor` the color of rust. The `rustColor` was set to dark brown, which is represented as `(0.75, 0.34, 0.00)` in RGB. If the base color of

the model is `c`, the stained color `col` is given as:

```
col ← c * lerp(rustColor, whiteColor, rust);
col ← lerp(rustColor, col, rust * rustIntensity);
col ← lerp(blackColor, col, dust);
```

where `lerp` is a function for linear interpolation, and `lerp(A, B, x)` returns `A * x + B * (1 - x)`. Note that the product between two colors is obtained by multiplying the two colors in a component-wise manner. Thus, the effect of rust on color is computed in two steps. The first step finds the reddish color according to `rust`. However, this step computes the product with the base color `c`, so the rust color is ignored when `c` is a dark color, each of whose components is close to `0`. Therefore, the second step computes the weighted linear sum with `rustColor` so that the rust color appears in any base color, where `rustIntensity` is a weight of `rustColor` and controls the color intensity of rust, as shown in **Fig. 7.7**. The value of `rustIntensity` was set to `0.2`. In addition, a darkened color can be obtained by computing the linear interpolation with `blackColor` according to the value of `dust`. Although chalking causes a color change in reality, the method of this change varies with the materials used, and it is difficult to describe using a general model, so the proposed method does not deal with the color change from chalking.

If the reflection ratio of the initial state `r` of the stained surface `ref` is computed as:

```
ref ← r * (1 - chalk);
ref ← ref * (1 - rust * rustIntensity);
ref ← ref * (1 - dust);
```
(7.1)

When each value of `chalk`, `rust`, and `dust` is increased, the reflection decreases. Note that the reflection ratio means the material parameters `Metallic` and `Smoothness` in Surface Shader of Unity 2019.4.14f1, and both were computed according to **Eq. (7.1)** with the initial values of `0.2` and `0.8`, respectively. For details on `Metallic` and `Smoothness`, please refer to the official Unity document [95].

(a) rustIntensity = 0.0 (b) rustIntensity = 0.2 (c) rustIntensity = 0.5 (d) rustIntensity = 1.0

Fig. 7.7 Sensitivity analysis of the coloring intensity of rust `rustIntensity`. If `rustIntensity` is high, the base color is not reflected well, and the rust color is deeply drawn. Note that when `rustIntensity = 0`, the mixed color is computed only by multiplication, so rust does not appear at all in the black base color. Using this method, `rustIntensity = 0.2` was adopted

Chapter 8

Interactive Control

This chapter presents methods for achieving interactive control to improve the directability of the weathering simulation. **Section 8.1** introduces the time reversal of the simulation, whereas **Sec. 8.2** describes GPU parallelization to allow for real time rendering and **Sec. 8.3** describes how to display the progress of weathering. Finally, **Sec. 8.4** explains the method to control the degree of weathering.

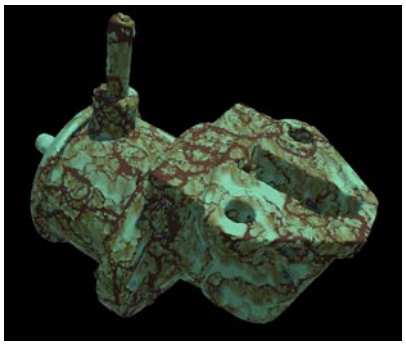
8.1 Pseudo Time Reversal

This section describes methods for the pseudo time reversal of the weathering processes mentioned thus far. In general, simulations include irreversible manipulations, and they cannot restore states completely unless properties are saved for each frame. While this method is no exception, as the basic parameters vary linearly with time, as shown in **Eq. (6.1)**, it is easy to approximate the state in the previous step. In the time reversal of weathering simulation, the time variations of mechanical parameters according to **Eq. (6.1)** can be reversed by assigning a negative value to the time step `dT`.

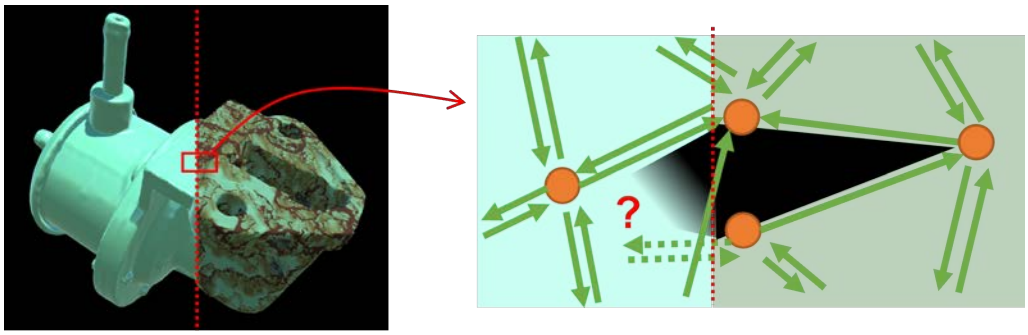
Time reversal has two advantages over complete history preservation, one of which is to save the spatial complexity. For example, if a model with 83,139 vertices and 166,408 polygons, as shown in **Fig. 8.1**, is weathered using the proposed method and the parameter values required to restart the simulation are saved for each frame, the spatial computation required per frame is 36 MB, and it takes 61 GB of memory to store 1 minute of states at a frame ratio of 30 frames per second (fps). In contrast, time reversal requires only 36 MB of the initial state. To rewind to a certain state in the past, it is necessary to estimate sequentially the state of the previous step, which requires more spatial computation than directly restoring the data to a certain state. However, this is not a major problem because it is possible to return to a previous state at a high speed by increasing the absolute value of the time step, and the images presented at each step can be used as a reference for the user to consider how far to reserve. The other advantage is partial recovery of the model. Because saved data of a certain state reference the entire model, the partial recovery must combine two states at different time points, as shown in **Fig. 8.1** (c). However, topological discrepancies may arise on the border of the two states. Conversely, time reversal allows partial recovery, because it repairs the model with an algorithm that preserves phase correctness.

However, because the time reversal of weathering requires the initial state, that is, the model before weathering, the proposed method does not allow the presumption of the initial state from a weathered model. In addition, note that especially, many assumptions and mathematic models must be introduced for the time reversal, which never occurs in reality.

(a) Before weathering



(b) After weathering



(c) Weathered model partially restored

Fig. 8.1 Partial recovery by history preservation. The numbers of vertices and polygons in the model are 83,139 and 166,408, respectively. Topological inconsistencies may occur at the boundary

8.1.1 Stretch of Bent Films

To reverse manipulate the bend simulation mentioned in **Chap. 5**, the process where bent films stretch and return to their initial states is presented. This stretch simulation is also thanks to PBD-2, which, because it ignores inertia, namely, the model shape depends only on the geometric constraints, it does not require any special time manipulation, and pseudo time reversal can be realized by updating the constraints properly.

As can be seen from the update formula for `Offset` in **Eq. (6.2)**, as lifting and contraction forces approach `0`, `Offset` also approaches its initial value `Offset0`. In the time reversal, lifting and contraction forces are decreased, and bent films return to their initial states without any special measures.

8.1.2 Fracture Repair

This subsection details the *repair* manipulation, which means the process of reversing the fracture simulation presented in **Chap. 6**. The repair manipulations corresponding to the three stages of fractures mentioned in **Secs. 6.3**, **6.4**, and **6.5**, are referred to as *bonding*, *binding*, and *attaching*, respectively. In addition, repairing requires a prerequisite: `dT < 0`.

If a vertex `V` satisfies the following three conditions, `V` is bonded to the base:

- It is separated but not torn: `V.Stat == 1`;
- The adhesion force is more than the sum of the lifting force from the surrounding half-edges: `V.Bond` $> \sum_{\texttt{E}\in\texttt{Es}}$ `E.Lifing` (`Es` is a set of `E` such that `E.Prev.Vert == V`);
- The current position is close enough: $\|$`V.Pos - V.Pos0`$\|$ `< epsilon`;

where `epsilon` is the maximum distance of two vertices considered at the same location, and its value was set to `0.01`. The second condition is derived by reverting the inequality sign for the separation condition on the forces. Note that the adhesion force at the separated vertex `V` should be `0` by rights, so `V.Bond` is interpreted as the predicted value of the adhesion force exerted when the vertex bonds. Therefore, the second condition ensures the vertex does not separate as soon as it bonds, and whether it actually bonds is determined by the third condition. Thus, if the vertex `V` bonds, the following process should be performed to reset the parameters, except for the degree of stains, to their initial values:

- Set `V.Stat` to `0`;
- Inactivate `V` in PBD-2;
- Substitute `V.Pos0` for `V.Pos`;

There are standard and special conditions for binding, and they are corresponded to conditions for tearing. As requirements common to both, the state `E.Stat` of a half-edge `E` must be `1` or `3`. If `E.Pair` is abbreviated to `Ep`, the standard condition for binding `E` is defined as follows:

[ST-1] Each end of `E` is the end of a crack:
`E.Vert.Stat == 2` **and** `Ep.Vert.Stat == 2`;

[ST-2] Binding forces are stronger than contraction forces:
`E.Face.Contraction + Ep.Face.Contraction < E.Binding + Ep.Binding`;

The standard condition of tearing is for generating a new crack, so its reversal is for removing a crack. As such, [ST-2] denotes the condition for maintaining binding, so it does not consider the effects of stress concentration.

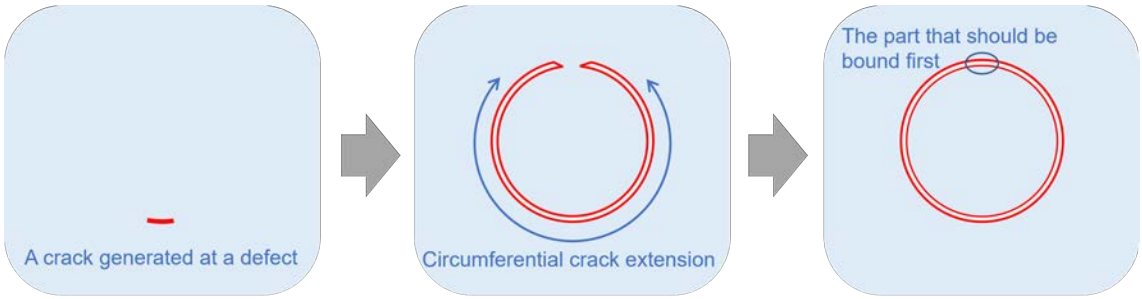


Fig. 8.2 An example of the difficulty in determining the area that should be bound due to the disappearance of crack ends

In contrast, the special condition for binding is described as follows:

[SP-1] The starting point of `E` is a crack end or `E` is an edge where a crack end has disappeared:
`E.Vert.Stat == 2` **or** `E.Stat == 3`;

[SP-2] Binding forces are stronger than contraction forces, including effects of stress concentration:
`(1 + s * kT / sqrt(E.Length)) * (E.Face.Contraction + Ep.Face.Contraction)`
`< E.Binding + Ep.Binding`;

[SP-3] Either pair of ends of `E` or `Ep` are close to each other:
`‖E.Vert.Pos - Ep.Next.Vert.Pos‖ < epsilon`
or `‖E.Next.Vert.Pos - Ep.Vert.Pos‖ < epsilon`;

As in the case of bonding, [SP-2] indicates the condition, ensuring the edge does not tear right after binding, and `s`, `kT`, and `sqrt(E.Length)` denote the same parameters as in the special condition for tearing. However, `kT` is computed assuming the crack end is `Ep.Vert` and the crack direction is `E.Vert.Direction`. In addition, `epsilon` in [SP-3], which indicates the geometric condition, denotes the same parameter as in the case of bonding. Thus, if the starting point of the half-edge is a crack end in [SP-1], that is, the pair of half-edges shares the vertex, [SP-3] is necessarily met.

The reason [SP-1] includes `E.Stat == 3` is provided below. The stress concentration occurs at a crack end, so [SP-2] must consider its effect. If `E.Vert` is a crack end, the crack end after binding can be predicted as `Ep.Vert`. However, if crack ends disappear, as mentioned in **Sec. 6.4**, it is difficult to determine edges to be bound without any exception processing. **Figure 8.2** shows a simple example where a crack extends circumferentially from a defect and its ends are connected. In this example, the edge that connected the crack ends should be bound first, but this edge does not satisfy the condition that either end is a crack end. If this condition is removed, the edge is certainly bound, but each edge on the circumference is bound at once, which differs significantly from the time reversal of the crack-extending process. Therefore, exception processing for setting `E.Stat` to a special value of `3` is required so the edge in the area where a crack has disappeared can be preferentially bound.

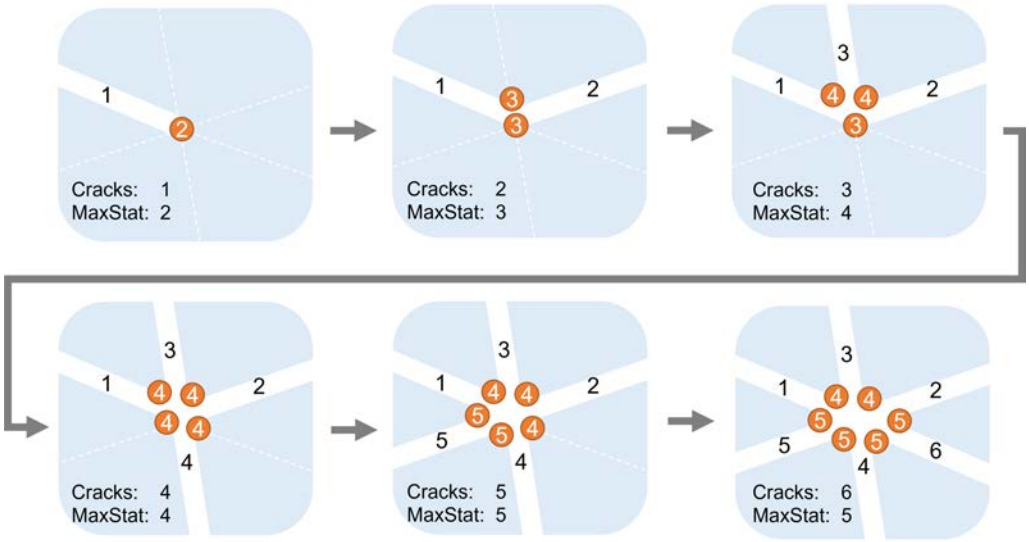


Fig. 8.3 An example in which the number of cracks near the junction of cracks does not match the maximum `Stat` value. If edges tear in the order of the black numbers, the `Stat` values are incremented, as indicated in white numbers. Finally, their maximum value is 5, whereas the number of cracks is 6

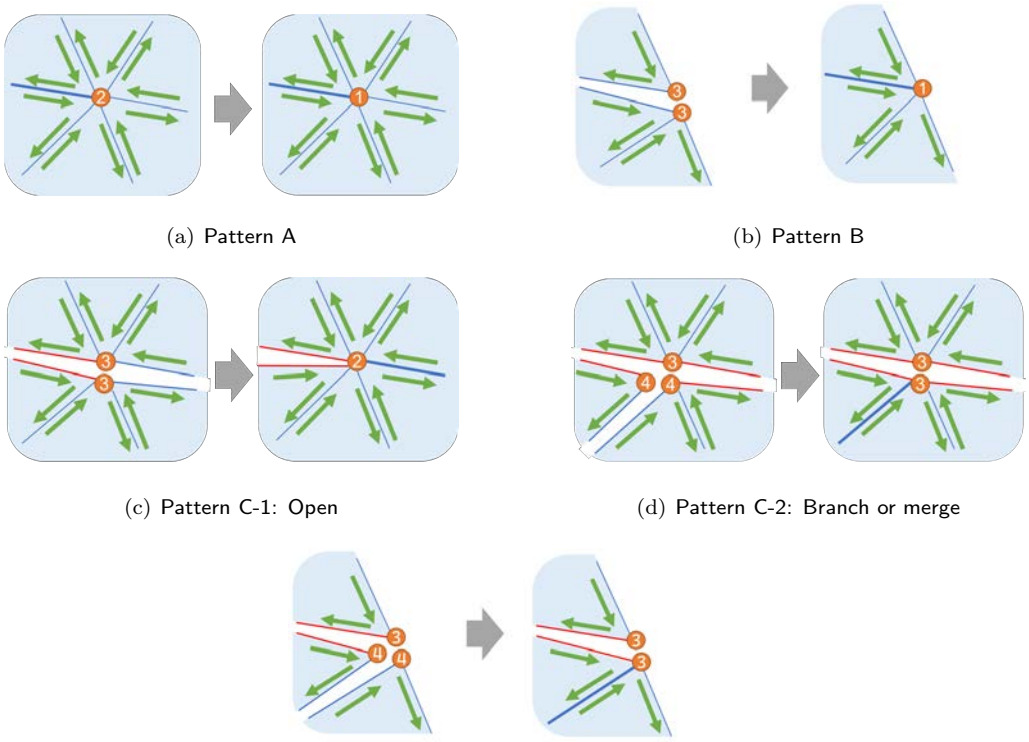
Algorithm 8.1 gives the process of binding the half-edge `E`, and **Fig. 8.4** shows cases of binding that correspond to the tearing patterns shown in **Fig. 6.6**. The pair `Ep` is also bound at the same time, but the bend and length constraints are activated only on either `E` or `Ep`. The statuses `E.Stat` and `Ep.Stat` are changed, and then `E.Vert` and several parameters are set to elements around `E.Vert`. In principle, `E.Vert.Stat` is set to the number of vertices, including `E.Vert` that, which has the same value of `Orig`, with `E.Vert`. Except at the model edges, this number is equal to the number of nearby vertices, including `E.Vert`, or of cracks around `E.Vert`. Although the number matches the maximum value of `Stat` at nearby vertices in all examples shown above, it is not generally equal to the maximum `Stat` value when many cracks are generated, as shown in **Fig. 8.3**, and it is necessary to count the number by searching around the vertex. However, because `Stat` at the vertex on the edge of the model cannot be 2, in Pattern B (**Fig. 8.4** (b)), the `Stat` is set to `1` after binding, though the number of vertices before binding is 2. The vertex `V1` in **Algorithm 8.1** indicates a vertex that is duplicated during tearing but no longer requires the binding process. By overwriting all topological data pointed to `V1` with `V0`, `V1` appears to be merged into `V0`. `V1` is not referenced but remains as data, and the memory is reused by overwriting these data when tearing occurs, even if in a different place. Therefore, it is ensured that the number of vertices remaining as data will not exceed that of half-edges.

Algorithm 8.1 Binding a half-edge `E`

Require: A half-edge `E` satisfies the condition for binding, geometric data, and constraints data

Ensure: Bind half-edges `E` and `E.Pair`

```
1: Ep ← E.Pair;
2: E.Stat←0;
3: Ep.Stat←0;
4: Activate length and bend constraint on either E or Ep
5: if E.Vert.Stat > 2 then
6:   Vs ← array with all vertices V such that V.Orig == E.Vert.Orig as elements;
7:   newStat ← the number of elements in Vs;
8:   if newStat is 2 and E.Vert.OnEdge is 1 then
9:     newStat ← 1;
10:  end if
11:  for all V in Vs do
12:    V.Stat ← newStat;
13:  end for
14:  if E.Vert.Orig in E.Vert then
15:    V0 ← E.Vert;
16:    V1 ← Ep.Next.Vert;
17:  else
18:    V0 ← Ep.Next.Vert;
19:    V1 ← E.Vert;
20:  end if
21:  Overwrite all data references to V1 with V0;
22:  if newStat is 2 then
23:    Update V0.Direction;
24:  end if
25: else
26:  E.Vert.Stat ← 1;
27: end if
28: Swap E with Ep and then perform the if-else state again;
```



(a) Pattern A

(b) Pattern B

(c) Pattern C-1: Open

(d) Pattern C-2: Branch or merge

(e) Pattern C-3: Merge on the model edge

Fig. 8.4 Examples of binding corresponding to tearing patterns. The `Stat` value is basically set to the number of surrounding vertices before binding but forced to be `1` only in Pattern B

Attached is a manipulation to revive a stripped face, which always occurs immediately after binding, so by merging these two processes together, they can be considered for extending coating films. When a half-edge `E` has been bound and the face `E.Pair.Face` is abbreviated to `F`, `F` is attached if `F.Stat` is `1`.

The stripped face `F` remains as data but is hidden, and a piece of the coating film appears to be added by setting `F.Stat` to `0` for redisplay. Moreover, `E.Pair.Prev.Vert` is activated in PBD-2, and length constraints corresponding to `E.Pair.Next` and `E.Pair.Prev` are activated.

8.1.3 Stain Removal

This subsection explains the reversal of reversing chalking, rust run-off, and dust accumulation. Note that properties that must be reconsidered regarding the computation method are member variants `V.Chalk`, `V.Rust`, and `V.Dust` of each vertex `V`, whereas the rendering method presented in **Sec. 7.4** is used as is. Moreover, the increasing speeds of `V.Chalk` and `V.Dust` are almost constant, regardless of the simulation progress, that is, `V.Chalk` and `V.Dust` vary linearly with time, so the time reversal of chalking and dust accumulation are automatically achieved only by setting the time step `dT` to a negative value.

In contrast, rust run-off requires some exceptional manipulations. As described in **Sec. 7.2**, when weathering progresses, the more rust located above a vertex, the more rust will flow into the vertex, and the more rust a vertex has, the more rust will flow downward from the vertex. However, if `dT` is a negative value, the more rust located above a vertex, the less rust will flow into the vertex, and the more rust the vertex has, the less rust will flow downward from the vertex. As a result, the two kinds of points appear alternately, and horizontal stripes linger, as shown in **Fig. 8.5**: the point where the rust flow is stopped and the point where rust is completely absorbed by the former points. Therefore, the time reversal of rust run-off requires a different algorithm.

The rust propagation algorithm for the time reversal is shown in **Algorithm 8.2**, which includes three changes from the original. The first is to solve the above problem by swapping `V` and `S` on lines 13 and 18, respectively, which is almost the same as reversing the gravity direction. The second is to cancel the threshold of the rust amount `amoThr`. To return to the initial state where each `Rust` value is `0`, the setting that does not allow `Rust` values to fall below the threshold must be removed. The conditions regarding `amoThr` on lines 2 and 17 and such factors as `amoThr` on lines 13 and 18 have been removed. However, the flow speed is increased by removing the factors, so `(1 - amoThr)` has been multiplied to compensate for the increase on line 26. The third is to add a process on lines 23–25 so `flowRust` becomes `0` when `V.RustBuf` is below `amoThr` and `flowRust` is below `0`. This process prevents the rust amount from increasing at vertices having no rust before the time reversal.

8.2 GPU Parallelization

This section describes the parallelization of the simulation processes through GPU implementation to achieve a fast execution speed that allows for interactive control. Note that this thesis focuses only on the theoretical process, while actual GPU parallelization requires technical attention, such as using structure instead of class, and `uint` type instead of `bool` type.

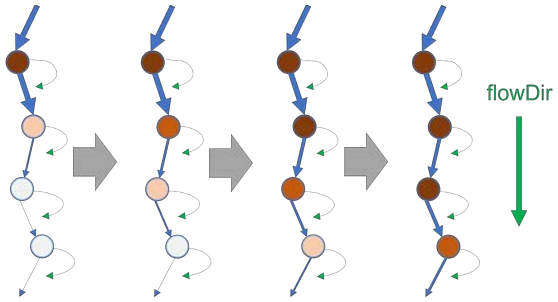
Most processes in the proposed method are performed for each element of the vertex list `Verts`, half-edge list `Edges`, face list `Faces`, length constraint list `LengthConstraints`, or bend constraint list `BendConstraints`. If a process is performed for each element `E` of a list `Elements` and updates only parameters of `E`, the process can be parallelized by assigning a thread to each element. **Figure 8.6** shows an outline of parallelization per vertex `V`. For example, **Algorithm 7.1** is the process of updating the amount of rust, performed through two parallelized processes. The first, which substitutes `V.Rust` into `V.RustBuf`, is self-contained only with vertex data, so it can be parallelized. Conversely, the second process computes the change in rust amount at each vertex `V` and it references other vertices. However, this process does not change the parameters of other vertices, so it can also be parallelized.

Algorithm 8.2 Condensation of rust

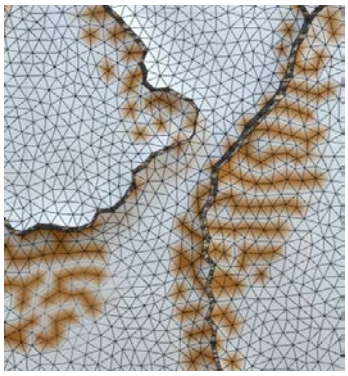
Require: Vertex list `Verts`, gravity direction vector `gravity`, the time step `dT`, and the control parameters for rust run-off `dirThr1`, `dirThr2`, `amoThr`, `rustRatio`, and `speedRu`

Ensure: Update `V.Rust` of each element `V` in `Verts`

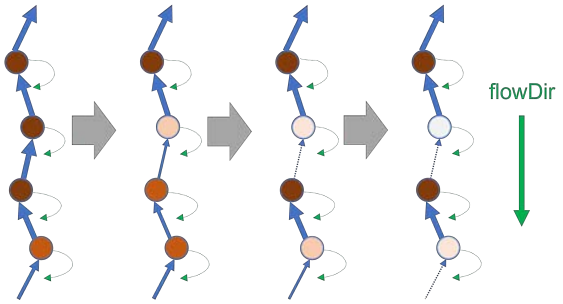
```
for all V in Verts do
  V.RustBuf ← V.Rust;
end for
for all V in Verts do
  if V.Source is true then
    continue;
  end if
  flowDir ← gravity projected onto the plane perpendicular to V.Nor and normalized;
  for all S in set of vertices connected to V do
    pathDir ← normalized vector of V.Pos - S.Pos;
    flowRust ← 0;
    if flowDir · pathDir < dirThr2 then
      flowRust ← (flowDir · pathDir - dirThr1) * V.RustBuf;
      if flowDir · pathDir ≥ dirThr1 then
        flowRust ← flowRust * rustRatio;
      end if
    else if -flowDir · pathDir ≥ dirThr2 then
      flowRust ← (- flowDir · pathDir + dirThr1) * S.RustBuf;
      if -flowDir · pathDir ≥ dirThr1 then
        flowRust ← flowRust * rustRatio;
      end if
    end if
    if  thenV.RustBuf < amoThr and flowRust < 0
      flowRust ← 0;
    end if
    V.Rust←V.Rust + flowRust * dT * speedRu * (1 - amoThr);
  end for
end for
```



(a) Outline of rust run-off



(b) Failure of the time reversal



(c) Outline of the time reversal of rust run-off

Fig. 8.5 Outline of normal rust run-off (a), time reversal using the same algorithm as with normal rust run-off (b), and its outline (c). In the time reversal scenario, the difference among rust amounts at vertices grows, and points at which rust stops and at which there is no rust appear alternately

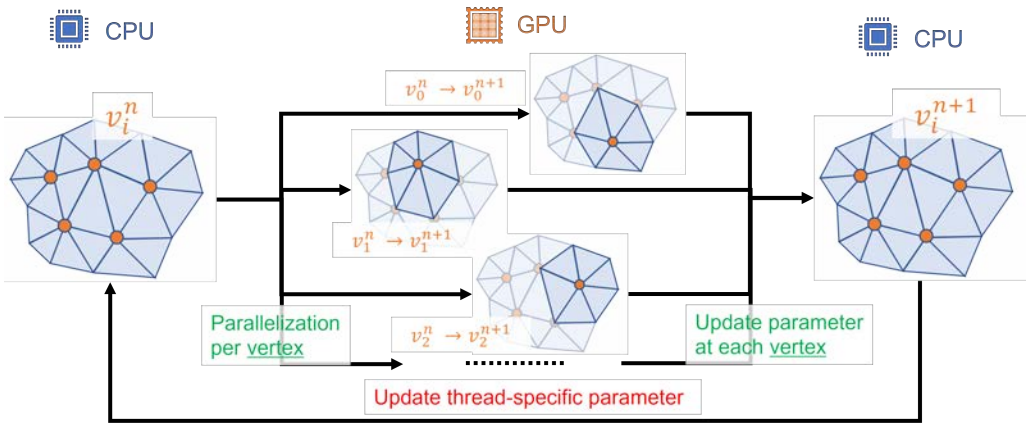


Fig. 8.6 Parallelization per vertex, where the process updates only a one vertex parameter. The parameter of the i-th vertex in the n-th step is denoted as v_i^n, and it is updated to v_i^{n+1} by the parallelization process

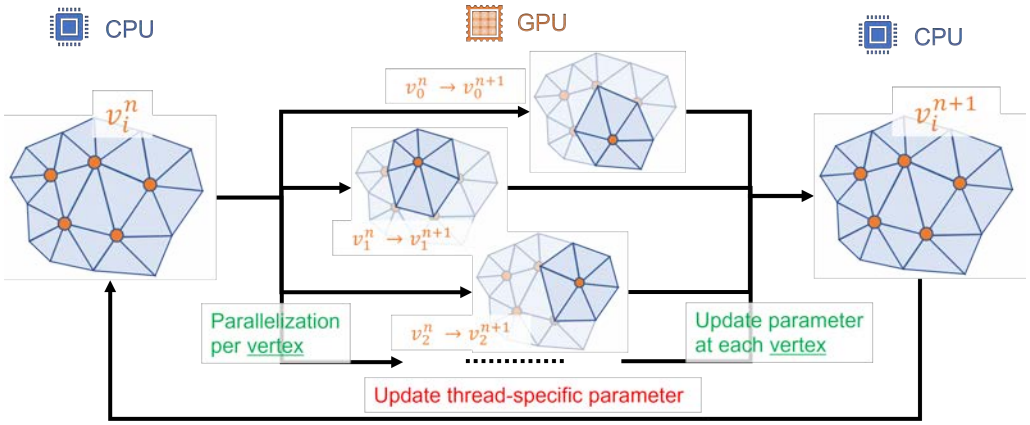
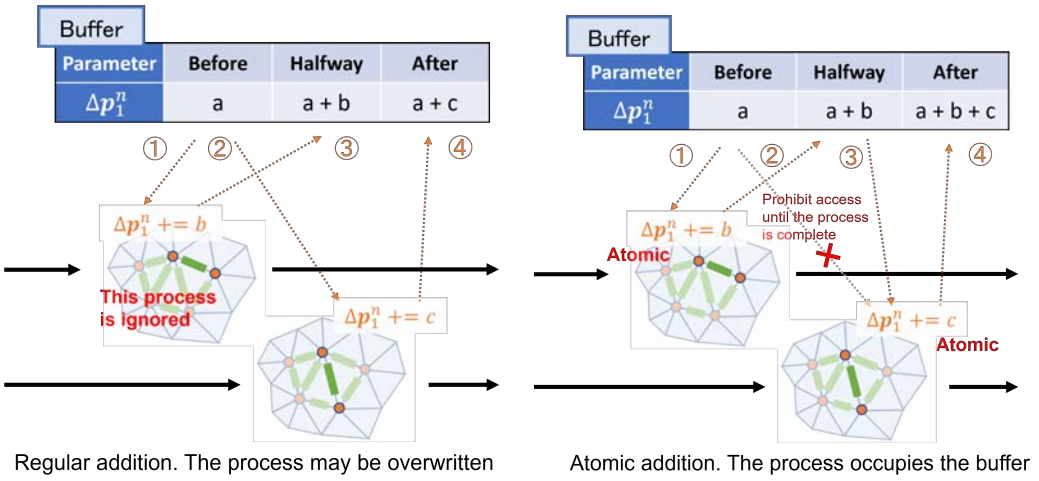


Fig. 8.7 Parallelization per length constraint, where the process updates the assumed displacements of the target vertices. Letting the value of `Verts[i].Delta` in n-th step be $\Delta \boldsymbol{p}_i^n$, and it is updated to $\Delta \boldsymbol{p}_i^{n+1}$ by the parallelization process

Although most of the processes in the proposed method can be parallelized in the same way, two processes need measures. The first is the process to update the assumed displacement `Delta` according to constraints in PBD-2, the parallelization of which is illustrated in **Fig. 8.7**. This process is parallelized for each constraint, but the parameters of the vertices must be updated, so multiple threads may write the same parameter simultaneously, that is, conflicts could arise. **Figure 8.8**(a) shows an example of conflict. Each thread accesses the buffer that stores parameter values to retrieve a value, updates the value, and accesses the buffer to write the updated value. Suppose that while thread A accesses the buffer twice, thread B retrieves the same value and B writes the updated value after A, so B does not reflect the update by A, which is discarded. To avoid the conflict, `Delta` is updated using an atomic operation, as shown in **Fig. 8.8**(b). If a thread performs an atomic operation, it is ensured that other threads will not access the same value until the thread finishes writing the updated value to the buffer, so any update is not discarded.

The second process requiring measure for parallelization is the topological manipulation of tearing and binding, which is a process in which each half-edge updates the parameters at its ends and duplicates or merges vertices depending on the situation. This process is too complex to perform by atomic operation, which supports only a simple calculation, such as addition. Therefore, the number of topological manipulations a vertex can undertake in a single step are limited to one, and subsequent manipulations are postponed to the next or a latter step. If the topological information at `V1` is changed, the topological manipulation of `V2` such that `V2.Orig` matches `V1.Orig` is also prohibited. To realize the management of topological manipulations, an array `locked` of `uint` type whose number of elements is equal to that of half-edges is set, and whether each vertex is topologically manipulated is saved in `locked`. Each element in `locked` is initialized to `0` at every step, and the condition that `locked[E.Vert.Orig] == 0` and `locked[E.Vert.Orig] == 0` is added to both the tearing and binding conditions corresponding to half-edge `E`. If `E` is torn



(a) When a conflict arises (b) When a conflict is avoided by atomic operations

Fig. 8.8 Conflicts due to parallelization and atomic operations

or bound, `locked[E.Vert.Orig]` and `locked[E.Pair.Vert.Orig]` are increased by `1`. Note that these comparisons to `0` and increments by `1` must be performed together using atomic operations to prevent other threads from interrupting the process, from comparison to increment.

8.3 Visualization of Degradation Level

The distribution of parameter values is pseudo color-coded to check the progress of weathering using information other than the realistic appearance. Three display modes are set to focus on separation, tearing, and stains, and the models visualized using these modes are shown in **Fig. 8.9**.

In the separation mode, to visualize the areas likely to be separated, the color at each vertex is varied according to the difference between its adhesion force and the sum of the lifting force the vertex receives. As shown in the legend of **Fig. 8.9**(b), the separated areas are displayed in red, and the areas that take time to separate are set to be blue. Note that if the difference between the adhesion and lifting force is over `100`, the areas are shown in pure blue.

Next, the tearing mode visualizes areas likely to be torn by coloring the model, according to the difference between the binding force of an edge and the sum of contraction forces the edge receives. Thus, the color is computed for each vertex, and the mean of the difference in the values of the edges connected to the vertex is linked to the vertex color. The method to link mechanical value and color is almost the same as that in separation mode, but in tearing mode, the areas where the difference is over `250` are shown in pure blue.

Finally, the degree of stains is displayed by setting `Chalk`, `Rust`, and `Dust` to `G`, `R`, and `B` components of the vertex color, respectively. All values are initially `0`, so the model is displayed in black. As the simulation progresses, the model gradually turns blue-green, displaying rust in red.

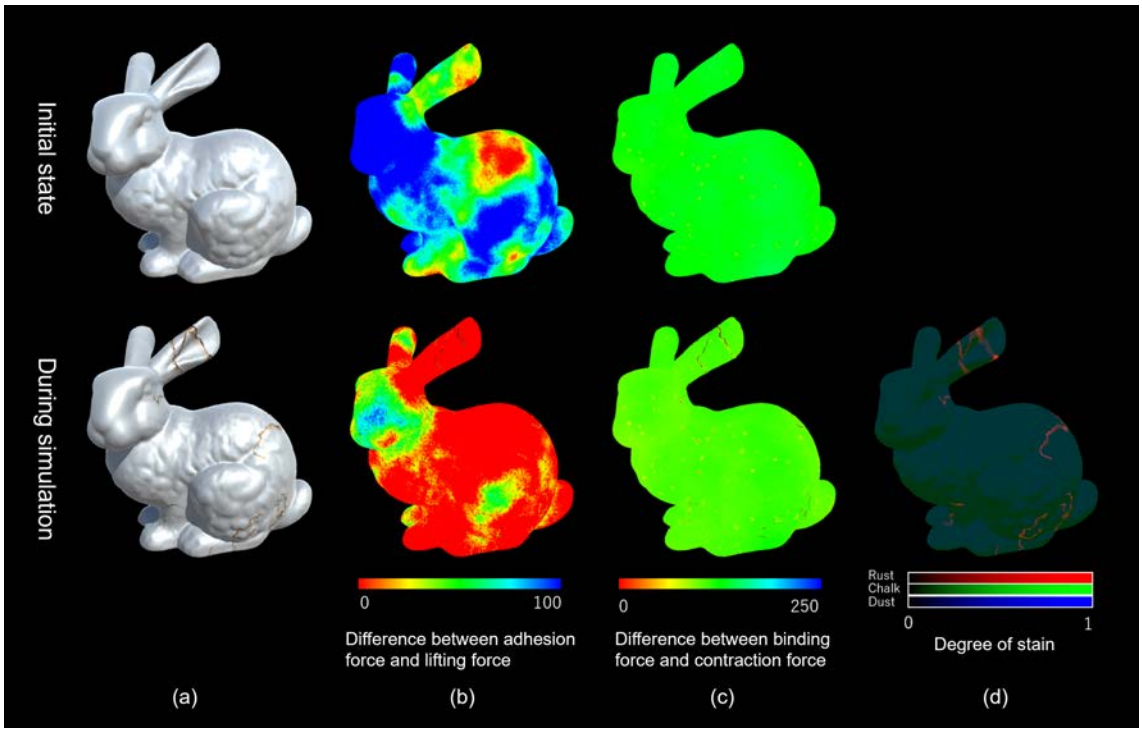


Fig. 8.9 Pseudo color-coding visualization of parameter values. The upper and lower row display the initial state and the state of weathering to some extent, respectively. Each column shows, from left to right, the normal appearance, ease of separation, ease of tearing, and degree of stains. Note that the model is initially shown in black to visualize stains

8.4 Simulation Control

To determine the target of interactive control, the intersections of the polygon mesh and a ray from the camera position to the pointer on the screen are found; thus, let `P` be the closest to the camera among the intersections. If the distance between `P` and a vertex `V` is less than `range`, `V` is determined as the target. In addition, if the distance between a half-edge `E`' midpoint `(E.Vert.Pos + E.Vert.Next) / 2` and `P` is less than `range`, `E` is the target. Note that `range` can be varied by external control in the range of `[0.01, 10]`. Four types of control modes are set: weathering, maintaining, washing, and repairing.

8.4.1 Weathering Mode

In weathering mode, `V.Bond` and `E.Binding` are set to approach `0`, as follows:

```
V.Bond ← V.Bond - V.Bond * rateCtrl;
E.Binding ← E.Binding - E.Binding * rateCtrl;
```

where `rateCtrl` denotes the sensitivity of the interactive control, and the value was set to `rateCtrl = 0.05`. In this mode, the designated areas become easy to fracture.

8.4.2 Maintaining Mode

In maintaining mode, `V.Bond` and `E.Binding` are increased using the following updating expressions:

```
V.Bond ← V.Bond + (V.Lifting + exBond - V.Bond) * rateCtrl;
E.Binding ← E.Binding + (exBinding - E.Binding) * rateCtrl;
```

where `exBond` and `exBinding` denote the maximum value of the adhesion force and binding force, and the values were set to `exBond = 200` and `exBinding = 2,000`, respectively. In addition, `V.Lifting` denotes the sum of the lifting forces the vertex `V` receives. The areas designated in maintaining mode are less likely to fracture, but if they have already done so, the fracture state is maintained.

8.4.3 Washing Mode

In washing mode, `V.Chalk`, `V.Rust`, and `V.Dust` are set to approach `0` as follows:

```
V.Chalk ← V.Chalk - V.Chalk * rateCtrl;
V.Rust ← V.Rust - V.Rust * rateCtrl;
V.Dust ← V.Dust - V.Dust * rateCtrl;
```

This manipulation removes the stains that are set to the vertices.

8.4.4 Repairing Mode

Repairing mode performs the repairs mentioned in **Sec. 8.1**, in addition to manipulation, in the maintaining and washing modes. Although the prerequisites of repair include `dT < 0`, the repairing operation exceptionally needs not satisfy this condition.

Chapter 9

Results and Discussions

This chapter provides the simulation results and discussions using the method proposed in this thesis, which was implemented on a gaming laptop with an Intel i9-10980 2.40 GHz CPU, 32.0 GB RAM, and an NVIDIA GeForce RTX 2080 GPU. Unity 2019.4.14f1 was used as an integrated development environment, and CPU and GPU implementation parts are coded in C# and HLSL, respectively. **Section 9.1** shows the results of an automatic weathering simulation and **Sec. 9.2** the effects of the external input. Meanwhile, **Sec. 9.3** compares the simulation results to peeled coating films in reality, **Sec. 9.4** mentions the execution time of the weathering simulation, and **Sec. 9.5** shows a specific scene in which an object is weathered using the proposed method. Finally, **Sec. 9.6** discusses the applicability and limitations of the proposed method.

9.1 Progress and Reversal of Weathering

Figure 9.1 shows the result of weathering a coating film on a curved surface using the proposed method. As the simulation time progresses, cracks are generated, from which the coating film warps. In addition, rust flows from pealed areas along the surface, causing the entire model to darken and lose its gloss.

Another result of applying the proposed method to an oil pump model is shown in **Fig. 9.2**, where pseudo time reversal was performed after weathering. By reversing the direction of the time progress, the weathered model can be returned to its initial state. Note that because this simulation does not record the history of previous states, the weathering process cannot be completely reversed. Especially, the binding condition is comparatively difficult to satisfy, so cracks often remain when stain removal is finished.

9.2 Weathering Control by External Input

Figure 9.3 shows a weathering process with an external input. By setting the control mode to weathering or maintaining and designating some areas on the model, the timing when fractures

Fig. 9.1 A result of weathering a coated curved plate. As time progresses, cracks and bends are generated

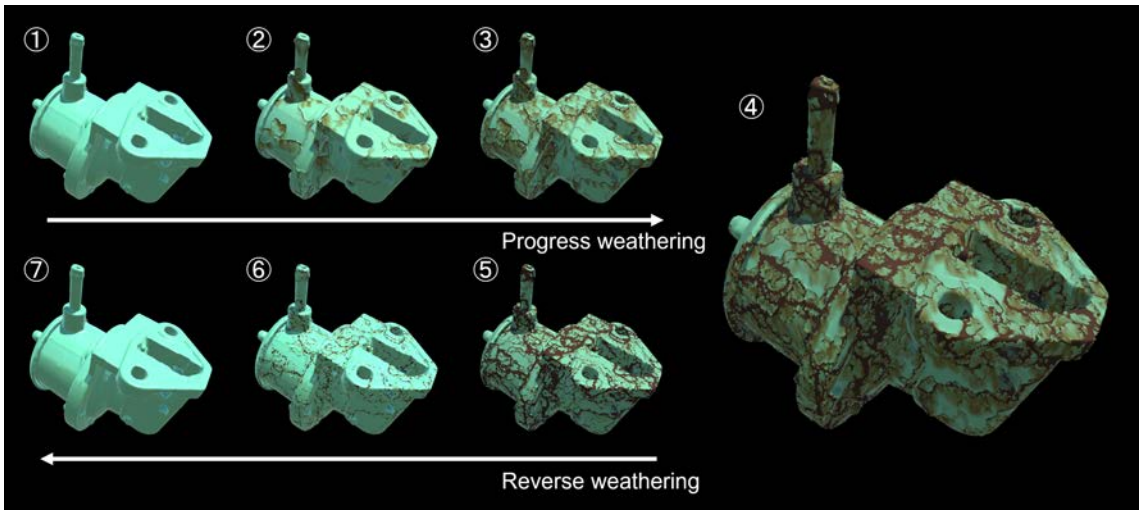


Fig. 9.2 Progress and reversal of the simulation. Weathering is progressed from ① to ④ and then reversed from ④ to ⑦. The pseudo time reversal cannot completely reproduce the reverse process of weathering, but it can gradually return the model to its initial state

occur can be controlled on a part-by-part basis within the model. Moreover, the separation and tearing modes visualize the areas likely to fracture.

Next, a machine part model that was weathered and partially repaired is displayed in **Fig. 9.4**, for which manipulation with the repairing mode repaired fractures and removed stains. Even around the boundary between weathered and repaired areas, no discrepancy arose among the topological data, and proper cracks were generated.

9.3 Reproduction of Real Peeled Films

The variations in peeling patterns caused by changing control parameters are shown in comparison to snapshots of peeled coating films. Note that simulations in this section were automatically performed without designating peeled areas to avoid draw realistic patterns intentionally.

Figure 9.5 compares the simulation result to a snapshot of a coated vertical pole. On the coating films shown in **Fig. 9.5**(b), cracks are not noticeable, peeled areas appear in small clusters, and there are few bends. Thus, to reproduce this texture, parameters were set to `s = 0.10`, `h = 0.10`, and `curl = 0.00`, and the result in **Fig. 9.5**(a) was obtained.

Conversely, **Fig. 9.6** compares the simulation result to a snapshot of a coated car stop fence. In the snapshot shown in **Fig. 9.6**(c), peering areas are scattered with few cracks, and coating films around peeled areas are slightly curved. Considering these factors, parameters were set to `s = 0.00`, `h = 0.00`, and `curl = 0.01`. Because the snapshot in **Fig. 9.6**(c) was taken under a roof, the fence is less dirty than in the simulation result shown in **Fig. 9.6**(a). **Figure 9.6**(b) shows the intentionally washed version of the automatically weathered model in **Fig. 9.6**(a).

Fig. 9.3 Designation of areas for weathering and maintaining by interactive control. The lower row visualizes the progress of weathering using the tearing mode, whereas red and blue areas indicate the promotion and prevention of weathering, respectively

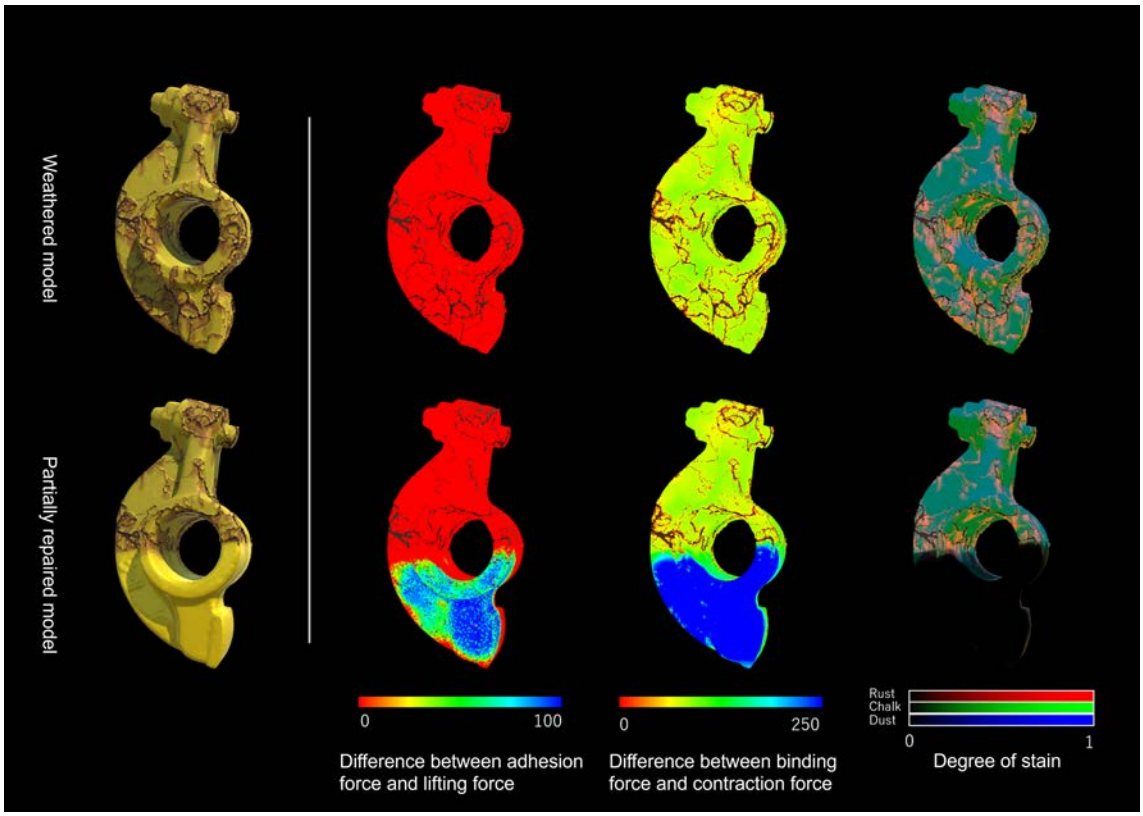
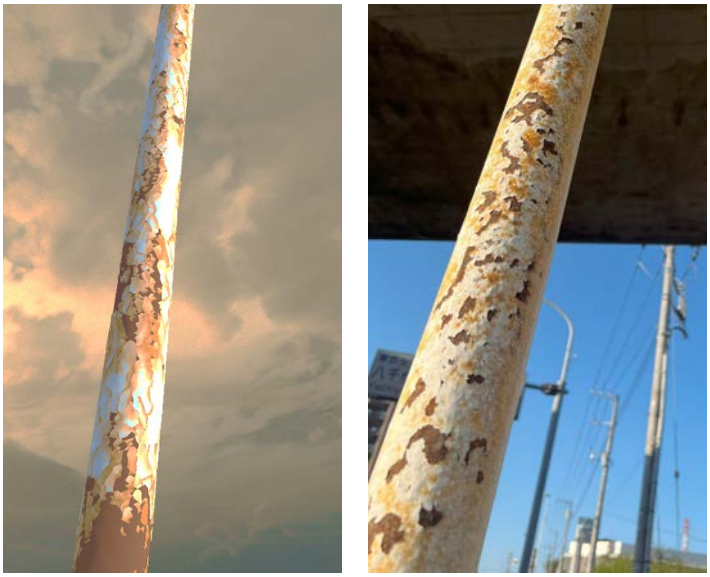
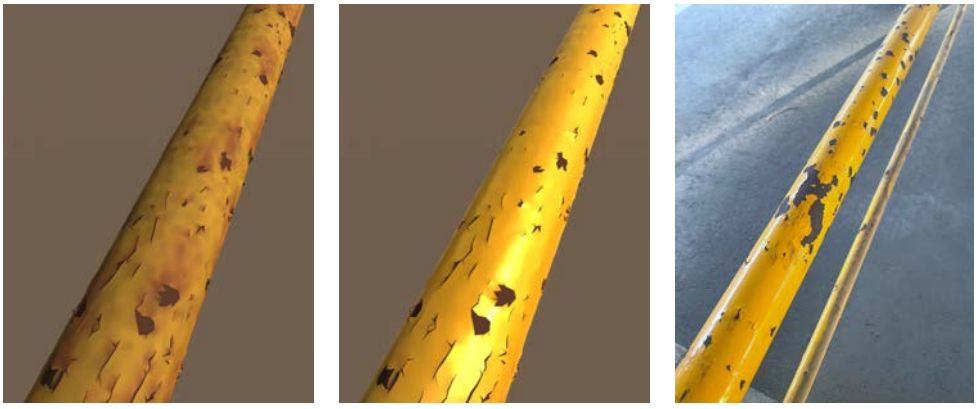


Fig. 9.4 Repair of a weathered model by interactive control. The controlled model (bottom) is the version whose lower part is the repaired weathered model (top)



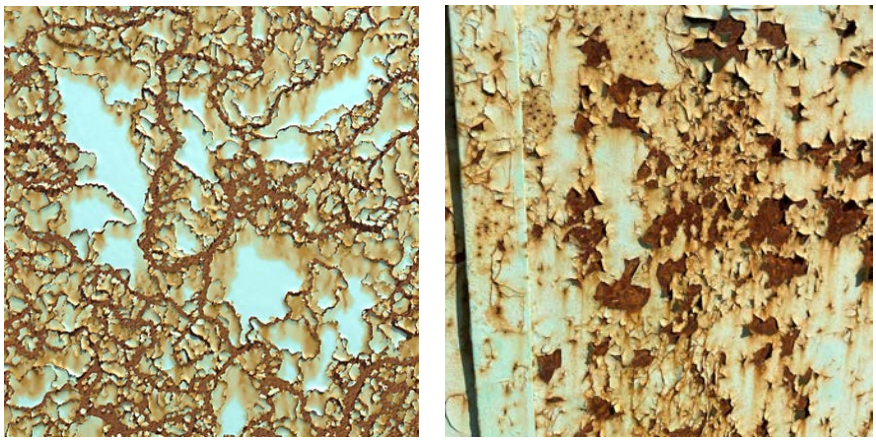
(a) Weathered pole model (b) Coated pole in reality

Fig. 9.5 Comparison between the simulation result (a) and a snapshot (b) of a coated iron pole in reality. The simulation was performed with the settings `s = 0.10`, `h = 0.10`, and `curl = 0.00` and without interactive control



(a) Weathered fence model (b) Model with (a) washed (c) Coated fence in reality

Fig. 9.6 Comparison between the simulation result (a) and (b) and a snapshot (c) of a coated car stop fence in reality. The simulation was performed with the settings `s = 0.00`, `h = 0.00`, and `curl = 0.01` and without interactive control. (a) is the full automatic simulation result, while (b) was washed by an interactive control to resemble the texture of (c)



(a) Weathered wall model (b) Coated iron wall in reality

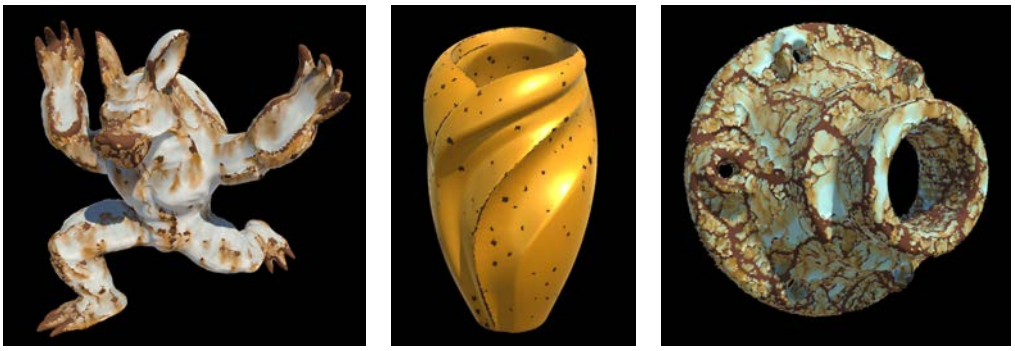
Fig. 9.7 Comparison between the simulation result (a) and a snapshot (b) of a coated iron wall in reality. The simulation was performed with the settings `s = 1.00`, `h = 0.50`, and `curl = 0.02` and without interactive control

Figure 9.7 shows an additional example of a coated iron wall. In the real coated film shown in **Fig. 9.7**(b), many fine cracks are generated, and pieces of the film are lifting around the cracks. The result in **Fig. 9.7**(a) was obtained by a simulation with large parameter values: `s = 1.00`, `h = 0.50`, and `curl = 0.02`, and it reproduces the variation in crack density.

Furthermore, it is possible to apply weathering to another model with a similar texture to the original by a simulation with the same parameter values. **Figure 9.8** shows the simulation results of weathering other models with the same parameter values of the above three cases. In each case, the texture of the original is reproduced in the different model. Note that texture mapping is not used, but the simulation is re-performed on another model, so it is possible to apply weathering considering the geometry of the model without distortion caused by UV mapping.

9.4 Temporal Complexity Analysis

Concerning the three types of models weathered automatically, as shown in **Fig. 9.9**, simulation statistics are summarized in **Table 9.1**. Step numbers depend on the number of polygons, but the execution time of a single step amounts to less than 30 milliseconds, even if the model has 10^5 polygons. This implies the possibility of real time animation and interactive control. Furthermore, **Table 9.2** shows the computation time required for each part of the simulation. For every model, the fracture simulation takes up most of the processing time, where the time required for mesh generation and other categories is less than 20 % of the full amount. This is because local processing only requires reference to data around cracked edges and peeled triangular faces. In addition, proportion of items in the mesh generation and other categories is high, presumably because it takes much time to transfer data between the GPU and CPU.



(a) Armadillo model (b) Vase model (c) Machine part model

Fig. 9.8 Results of reproducing the textures of objects on other models. (a), (b), and (c) reproduce the textures of objects shown in **Fig. 9.5**(b), **Fig. 9.6**(c), and **Fig. 9.7**(b). Because these simulations do not use texture mapping, no distortion caused by UV mapping arises, and the texture can be transferred considering the geometry of the target model

Table 9.1 Simulation statistics of the three models in **Fig. 9.9**

	Flat plane	Double torus	Stanford Bunny
Number of steps	1,000	1,000	1,000
Number of polygons	68,159	67,482	84,758
Pre-computation time	0.603 s	0.545 s	0.703 s
Simulation time	24.7 s	25.03 s	28.4 s
Computation time per frame	24.7 ms	25.03 ms	28.4 ms

9.5 Application to Specific Scene

Figure 9.10 shows a weathered object embedded within a specific scene provided by the model Destroyed City FREE for Unity by Profi Developers. In many realistic scenes, in which most other objects are more or less aged, weathering harmonizes an object with its environment, and it reinforces the reality of said object.

As an example that needs designation of weathered areas, **Fig. 9.11** shows a case of weathering playground equipment that biases people's tactile saliency. The handle and seating area, which people frequently touch, were designated to be degraded easily and designated areas cracked and peeled, while the entire model was darkened by weathering.

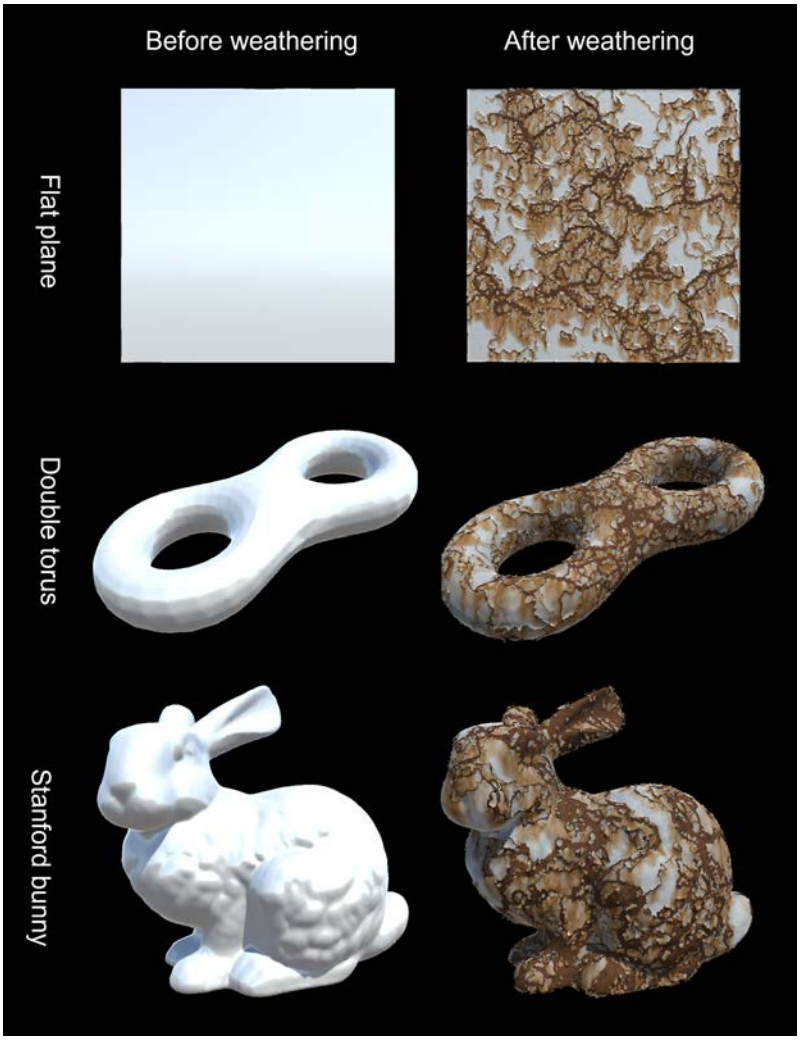
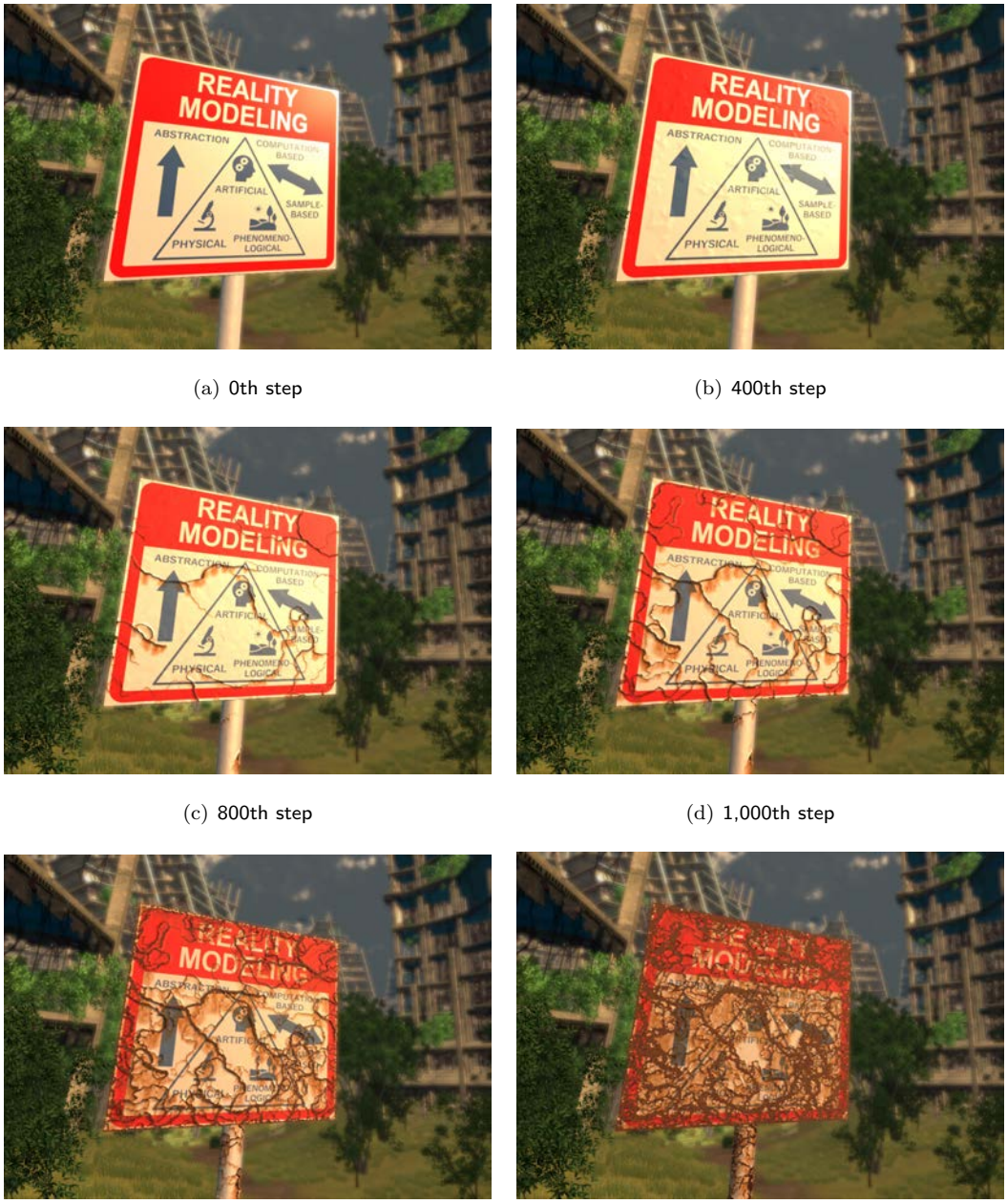


Fig. 9.9 Results of automatic weathering simulations whose net execution time was measured. See the obtained statistics in **Table 9.1** and **Table 9.2**

Table 9.2 Ratio of time required for each simulation part

	Flat plane	Double torus	Stanford bunny
Parameter updating	0.39 %	0.45 %	1.77 %
Bend simulation	0.23 %	0.23 %	0.20 %
Fracture simulation	60.92 %	60.84 %	52.79 %
Stain expression	0.02 %	0.02 %	0.02 %
Mesh generation	25.99 %	25.83 %	23.36 %
Others	12.35 %	12.63 %	21.86 %



(a) 0th step (b) 400th step

(c) 800th step (d) 1,000th step

(e) 1,200th step (f) 1,600th step

Fig. 9.10 Weathering of a signboard model in a devastated city scene. Weathering integrates the object into the scene

(a) Visualization of degradation (b) Before weathering (c) After weathering

Fig. 9.11 Weathering of playground equipment. The areas people frequently touch are designated by an external input for easy degradation

(a) Object uniformly weathered (b) Placing a parasol to block sunlight (c) Repairing the blocked areas

Fig. 9.12 Example of partial recovery. If an object is uniformly weathered (a) and a parasol is placed to block some of the sunlight hitting the object (b), repairing only the blocked area (c) renders the parasol's appearance to reflect as if it were placed before (a)

Moreover, **Fig. 9.12** shows an example of intended manipulation of time series for partial recovery. **Figure 9.12**(a) shows an object that is uniformly weathered, while in **Fig. 9.12**(b), a parasol was placed to block some of the sunlight hitting it. The parasol was certainly placed after (a), but restoring only the blocked areas made the image appear as if the degradation in the blocked areas was suppressed by the parasol, as shown in **Fig. 9.12** (c). In this situation, the parasol is interpreted as if it were placed before (a), thus reversing the time sequence.

9.6 Discussions

This section refers to future research to be conducted in order of feasibility, as indicated in **Fig. 9.13**, with some experimental examples. Finally, the limitations of the proposed method are described.

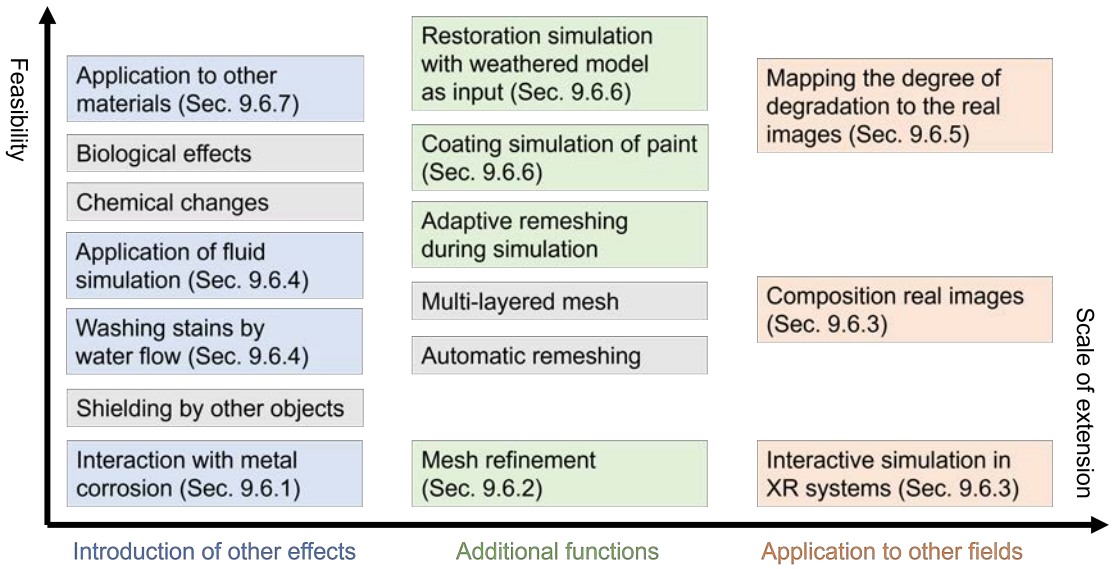


Fig. 9.13 Technology map of the weathering method. Colored items are discussed in **Sec. 9.6**

9.6.1 Corrosion of Metallic Base

Although there are various factors of aged degradation in reality and their complex interactions, the proposed method assumed the results could be summarized as linear variations in several fundamental parameter values, as shown in **Fig. 9.14**. Therefore, many factors could not be dealt with adequately in this study. In particular, base corrosion is one of the most important factors, as it interacts with the degradation of coating films. Weathering simulation that considers both coating films and metal bases may be achieved by combining with related works to deal with metal [45][46][62].

Figure 9.15 shows a result of an experimental simulation in which base corrosion is simply modeled. In the corroded areas, the base is set to expand as it lifts the coating films, and its color is set to change from silver to dark brown. In addition, the adhesion force of coating films on the corroded base is set to weaken rapidly. The introduction of base corrosion reproduced the variation in base degradation, as observed through the cracks, the bumpiness of the paint film due to rust formation, and the pattern of peeling, which appears in clusters.

In addition, yet another important topic related to metallic corrosion is galvanic corrosion, which is a phenomenon where the deterioration of one side is accelerated at contact points of dissimilar metals. Galvanic corrosion is not an uncommon phenomenon, because metal objects generally consist of multiple parts, and the screws and bolts that connect them are often made of different materials than the parts themselves, as shown in **Fig. 9.16**.

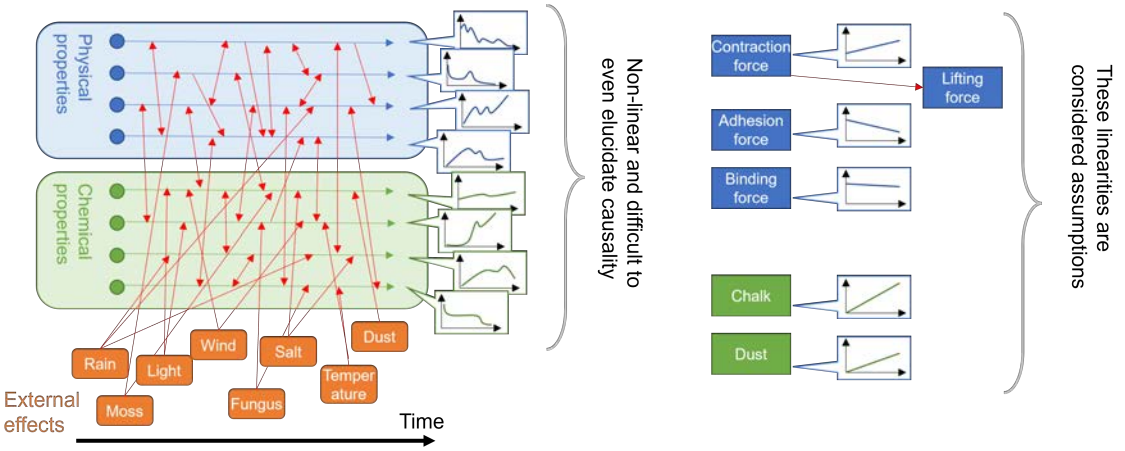


Fig. 9.14 Modeling concept of the proposed method. Aged degradation is caused by complex interactions among various factors in reality (left), whereas the proposed method assumes the interactions can be summarized as variations in several parameters (right)

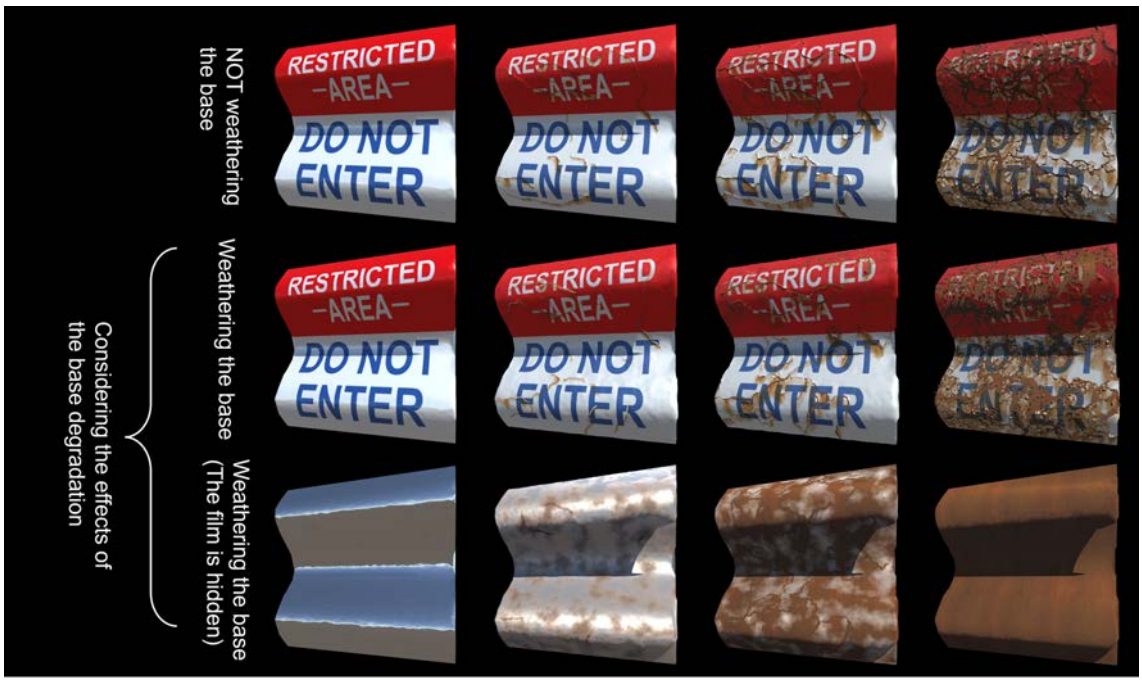


Fig. 9.15 Weathering of coating films considering base corrosion. The base is unchanged in the upper row, while a prepared texture is gradually applied to exposed base areas in the middle and bottom rows, and the coating film is hidden in the bottom row

Fig. 9.16 Snapshot of actual galvanic corrosion. The screw has a lower ionization tendency than the body, so it retains its metallic luster, but the body corrosion is accelerated near the screw

9.6.2 Dependence of Crack Generation on Mesh Shape

The problem with crack generation using mesh is that the shape of a crack changes depending on how the mesh is structured, even if the model has the same form; thus, consistent results cannot be obtained. The underlying cause is the difficulty in analyzing real crack generation, and the realism of the crack shape is greatly compromised at the analytical modeling stage by continuum theory, as depicted in **Fig. 9.17**. **Section 3.1** stated that the effect of the stress concentration was maximal in the direction of crack growth, but according to the results of this analysis, the crack should travel in a straight line without any bending.

However, real cracks have zigzag shapes, due to the effects of microstructural disorder in the order of tens of micrometers. In other words, while continuum theory is insufficient to explain detailed crack shapes, it is also difficult to construct mathematical models that cover the microstructure of materials or to perform simulations at the microstructure level.

In visual simulations, macroscopic mathematical models have been used to represent plausible cracks by actively incorporating discretization errors and randomness. For example, Hirota et al. [37] used a regular hexagonal lattice to prevent cracks from running straight, and Paquette et al. [79] used a square lattice as a base, but randomized the distance and direction of crack propagation to reproduce zigzag crack shapes. The need for discretization errors and randomness in crack generation also applies to the case of using re-meshing. Pfaff et al. [81] determined a method of re-meshing by considering the stress distribution, but in this case, a straight crack propagation is avoided because the stresses are discretely assigned to each polygon, which should be continuously distributed in reality.

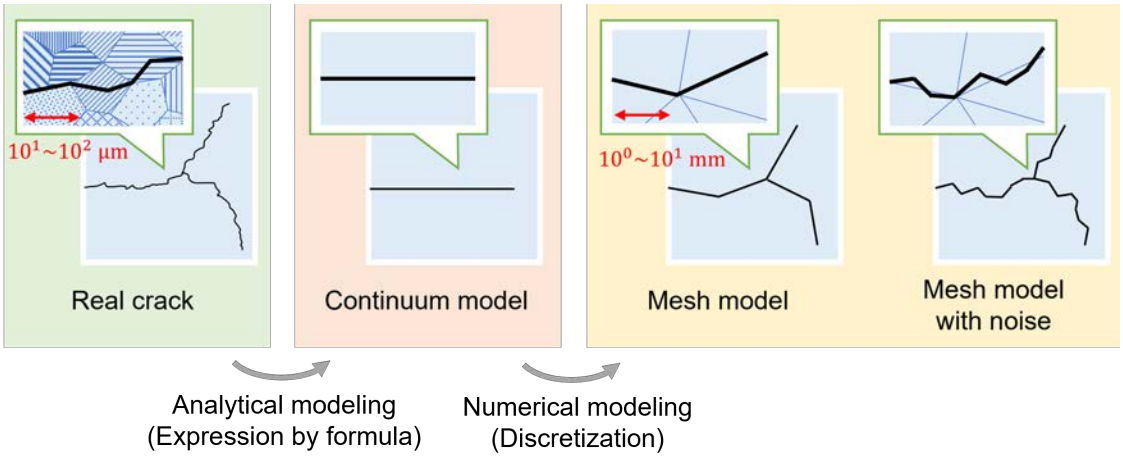


Fig. 9.17 Mathematical modeling and discretization flow in crack generation using mesh. Restricting crack extension to polygon boundaries results in a plausible appearance, whereas in the continuum model, the crack is considered to go straight through

The problem with the proposed method is that the shape of the crack changes when the mesh resolution is increased to maintain the plausible appearance of the crack, even from a close viewpoint. Two ways to deal with this problem can be proposed: use of the discretization error and the introduction of randomness. First, in the method using a discretization error, a mesh for simulation and a finer mesh for rendering are prepared, and the geometric elements of the two meshes are mapped to each other in advance. As mentioned above, simulation on a mesh introduces discretization errors, and fine errors are added when mapping the simulation result to the mesh for rendering. This method provides consistent results because the simulation itself is performed on the dedicated mesh, and the resolution of the fine mesh can be set independently of the simulation. **Figure 9.18** shows the results of the mesh mapping, where it can be seen that the fine contours of the cracks are represented while maintaining the general shape. This method requires pre-computation for the preparation and mapping of the fine mesh, and the execution time per frame was about 4 times longer in this case.

Alternatively, in the method introducing randomness, each triangular polygon is divided radially from the center, and fine contours are expressed by randomly moving vertices along the torn edges. The result of refinement by quartering each edge by length is shown in **Fig. 9.19**. In addition to reproducing a plausible crack shape, this method does not require another mesh input or pre-computation, and the running time doubles. This method is expected to be more efficient, as it dynamically changes the number of edge segments depending on the viewpoint and to generates a mesh that does not lose detail, regardless of how close one gets.

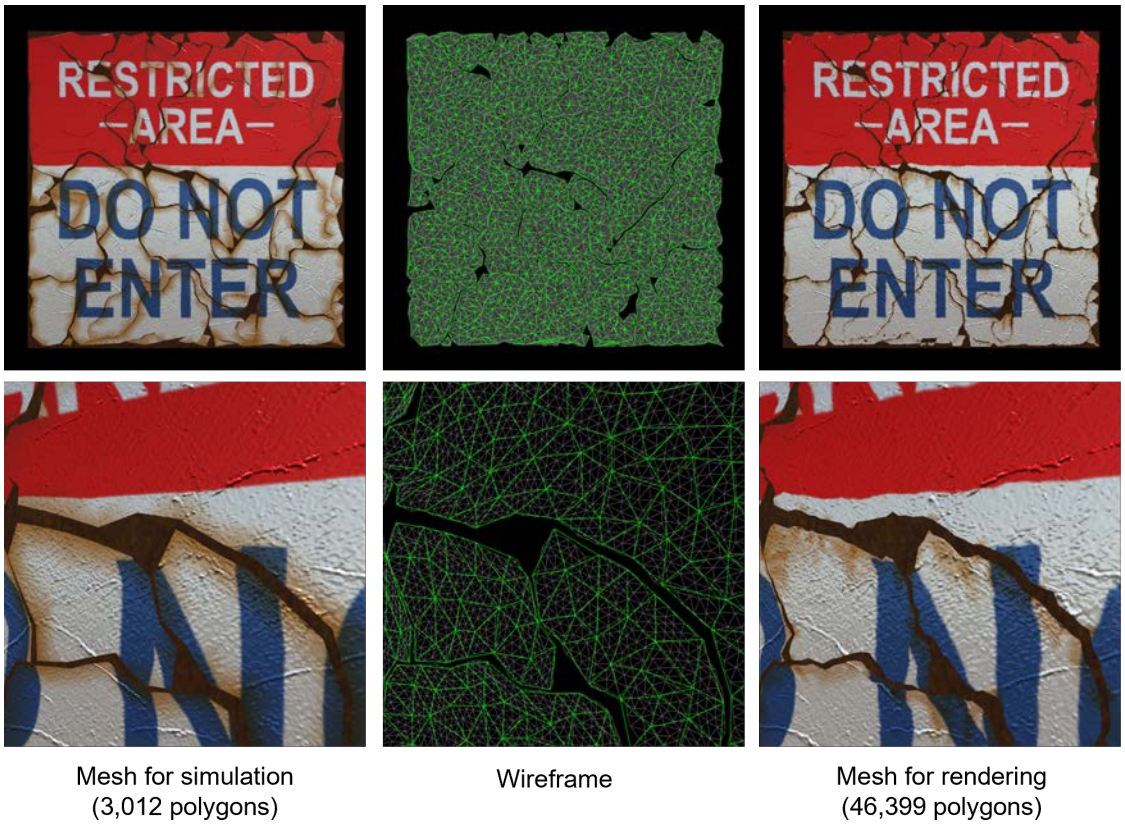


Fig. 9.18 High-resolution rendering by mesh mapping. The simulation mesh is drawn as is in the left column, while the mesh is projected to the fine mesh before rendering in the right column so that fine contours can be expressed while maintaining the general crack shape. The middle column superimposes the wireframes of the simulation mesh and rendering mesh in green and gray, respectively

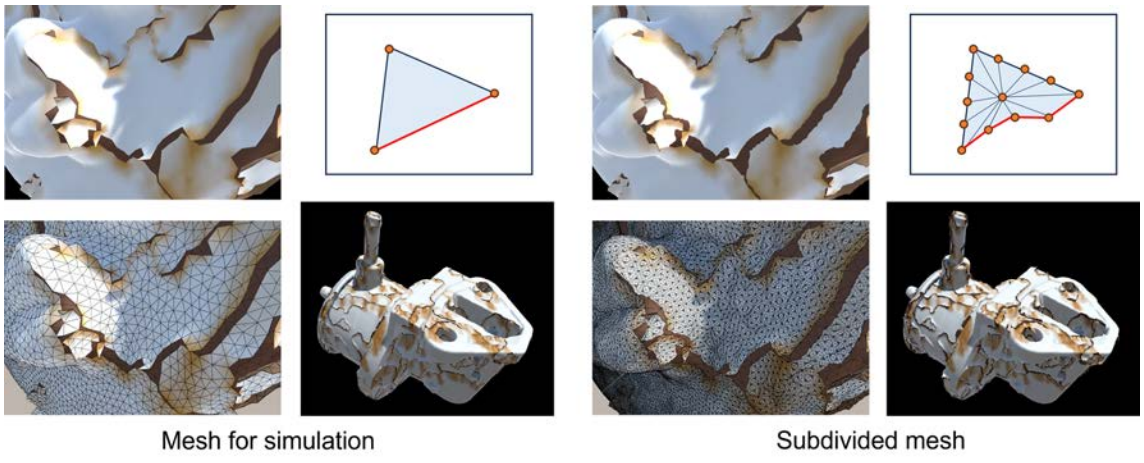


Fig. 9.19 High-resolution rendering by mesh subdivision. The torn edges are randomly distorted to express fine contours

9.6.3 Application to XR Systems

Because the proposed method provides an interactive simulation of mesh deformation, it is suitable for application to XR systems, which display CG as if they were really in front of users. In addition, one of the main purposes of weathering is to improve the realism of virtual objects, as it seems especially compatible with augmented reality (AR), which displays CG objects on real images rather than being completed in the virtual world. In other words, weathering is expected to have the effect of harmonizing CG objects with real scenes in AR systems.

Figures 9.20 and **9.21** show composite photographs with CG objects, whereas in **Fig. 9.20**, it is reasonable that the weathered object (b) looks more natural than the non-weathered one (c), because CG is overlaid on the originally weathered pulley. However, in the case where the original pole is not very weathered, as shown in **Fig. 9.21** (a), the weathered object (c) does not differ clearly from the non-weathered one (b). Therefore, the moderate weathering of CG objects improves affinity to real images, as well as improves realism, and it is expected to achieve a natural composition of CG and reality.

9.6.4 Acceleration of Deterioration and Transport of Stains by Water

Besides base corrosion, another factor not adequately addressed in this research is the effect of water, where even though the proposed method focused on its mechanical effects, such as water flow that conveys dust, the chemical effects might have a greater influence in some cases. The moisture accelerates not only the chemical deterioration of coating films but also base corrosion, which is mainly caused by electrochemical effects. Furthermore, if such biological factors such as moss growth are also considered, moisture can be assumed to have a significant impact on them.

In addition, mechanical effects should also be considered, such as outflow to another model, dropping, and puddling in pits. The water flow was assumed to transport only rust, but in fact, it also washes away pigments caused by chalking and accumulated dust. Such complex situations are expected to be reproduced by the application of a fluid simulation, but special efforts are required to maintain quasi-reversibility.

9.6.5 Mapping to Physical Time

Aged deterioration in the real world is a phenomenon that takes years to progress, and it is difficult to measure the changes in physical properties that occur during aging. In other words, it is impossible to construct a mathematical model that maps changes in parameter values associated with aging to physical time. Therefore, in this research, time in the simulation is treated as logical time, which differs intrinsically different from physical time. Further, the proposed method cannot be used to predict the future, including how many cracks will appear in a specific number of years.

(a) Real photograph (b) Weathered CG on the photograph (c) Non-weathered CG on the photograph

Fig. 9.20 A scene of a weathered pulley. (a) is a real photograph, (b) is overlaid with a weathered CG, and (c) is overlaid with a non-weathered CG. The virtual objects were manually overlaid with fine-tuning

(a) Real photograph (b) Weathered CG on the photograph (c) Non-weathered CG on the photograph

Fig. 9.21 A scene of a non-weathered pole. (a) is a real photograph, (b) is overlaid with a weathered CG, and (c) is overlaid with a non-weathered CG. The virtual objects were manually overlaid with fine-tuning

In contrast, the results obtained from the simulation based on logical time are interpreted by the viewer as imitations of real deterioration and are unconsciously mapped to physical time, making applications to the real world possible. For example, if an interpretation by a skilled coating inspector was treated as the correct answer, it could be used as a pseudo case study for training purposes to determine whether a new inspector could provide the same interpretation or to augment machine learning data to estimate the degree of deterioration.

9.6.6 Application to Restoration Simulation

Although this research realized the pseudo time reversal of weathering, the initial model is required as the input, that is, it is impossible to presume the original state by assigning a weathered model as input. However, such virtual repairing of damaged objects is an extremely significant component of computational archaeology. If machine learning was applied to this task, the proposed method might be used to augment training data.

In contrast, not presuming the original state but rather simulating restoration work has industrial value. In the current situation of many deteriorated coated objects, industrial interest is in predictions of the future, such as how the deterioration will be repaired and how it will proceed after the restoration, rather than the past history of how the deterioration has been proceeded. In fact, as shown in **Figure 9.22**, because real weathered coated objects cannot be returned to their initial state through restoration work, the progress of deterioration after restoration depends on the state before restoration. Therefore, the simulation of restoration by coating new paint on a weathered object is a possible application of this research.

9.6.7 Application to Other Materials

The proposed method focused on coating films on metallic objects using Stoney's law [90], but it may be possible to apply similar theories to bases made of other materials. **Figure 9.23** shows a simulation result in which the model is regarded as an oil painting, another example of objects with noticeable peeling effects. Because the patterns of cracks in rust preventive coating films and oil painting are highly similar, the proposed method expresses plausible deterioration of oil painting only by removing rust run-off stains. It is expected that the proposed method can be applied to coating films on the bases of such materials as wood and concrete by considering the specific degradation factors of each.

(a) Weathered coated plate (October 30, 2018)

(b) Coated plate just after restoration (November 17, 2018)

(c) Coated plate weathered again (May 8, 2023)

Fig. 9.22 A coated plate that is restored once but then weathered again. The subject is the backside of the signboard of Hiyoshi Shrine, which is adjacent to Keio University Yagami Campus. In the restoration work, warped coating films were removed and then paint was coated, but over time, the restored area deteriorated again

Fig. 9.23 Weathering simulation of oil painting. The effects of rust run-off stains are turned off and a model that imitates a canvas is placed behind the painting model

9.6.8 Limitations of the Proposed Method

This subsection summarizes the limitations of the proposed method, the first of which is its assumption that some typical parameter values change linearly with time as a result of various degradation factors. Therefore, the bias in the degree of degradation caused by each degradation factor, such as base corrosion, weather, light, heat, and people contact, is not automatically reproduced and must be specified by an external input. In addition, the factors that change the texture of the coating are limited to three: chalking, rust, and dust. As such, this method must be combined with other methods to represent changes in appearance caused by surface wear, light-initiated fading, moss growth, and other factors.

This method is a visual simulation that aims to reproduce plausible appearances, and it is performed according to a logical time. Therefore, although the generated images can impress the viewer with degradation, they cannot be used to predict the future by presenting physically precise information.

Finally, the input simulation mesh must be re-meshed in advance, and consistent results cannot be obtained if the resolution of the simulation mesh is changed. To reproduce the detailed shapes of cracks, it is necessary to prepare a high-resolution mesh for rendering that differs from a mesh used for simulation or to subdivide the mesh for simulation and to move vertices randomly.

Chapter 10

Conclusion

In this thesis research, an interactive weathering method for rust preventive coating films was proposed by expressing the complex phenomenon of aged degradation with simple mathematic models. By moving the vertices of a polygon mesh representing coating films according to position-based dynamics, this method efficiently expressed a thin-film bend, which requires complex computations to be reproduced by physical simulation. Fracturing was represented by manipulating the topology of the mesh while considering force balance, and various crack patterns were generated by setting simple properties related to the stress concentration as control parameters. The stains caused by external effects on the coating film were also expressed by controlling the vertex color in consideration of the geometry of the mesh model. Moreover, by defining reverse manipulations for these deformation and staining processes, the pseudo time reversal of weathering was achieved, which is difficult for conventional visual simulation methods. These processes were accelerated by parallel computation, and the parameter values could be varied in response to external inputs to achieve an interactive weathering simulation.

The proposed method enabled the simulation of realistic patterns of cracks and peeled areas. In addition, the time reversal of weathering improved the directability of the simulation to achieve both the advantages of automatically generating complex patterns in simulation and of intuitive and free manipulation in manual modeling.

In conclusion, this research has provided compelling evidence that weathering plays a crucial role in enhancing the realism of virtual objects. Moreover, it has brought to light the significance of incorporating comprehensive modeling techniques that account for both surface texture alterations and geometric deformations when weathering coating films. By recognizing these critical factors, even greater levels of authenticity and immersion can be expected to be achieved in virtual environments.

Publications

Main Publications

[MP-1] Akinori Ishitobi, Masanori Nakayama, and Issei Fujishiro: "Visual simulation of crack and bend generation in deteriorated films coated on metal objects—combination of static fracture and position-based deformation," to appear in *The Visual Computer,* vol. 39, Special Issue of CG International 2023, Springer-Verlag.

[MP-2] Akinori Ishitobi, Masanori Nakayama, and Issei Fujishiro: "Visual simulation of weathering coated metallic objects," *The Visual Computer,* Springer-Verlag, vol. 36, no. 10–12 (Special Issue of CG International 2020), pp. 2383–2393, September 30, 2020 (online first on August 8, 2020) [DOI: 10.1007/s00371-020-01947-w].

Reference Publications

[RP-1] Akinori Ishitobi, Masanori Nakayama, and Issei Fujishiro: "Crack generation and bend of coated films due to aged deterioration—Visual simulation of a quasi-static process by combination of fracture judgement and bend simulation—," *Visual Computing 2022,* October 2022, VC Paper Award (2nd prize), CGVI Best Presentation Award, CGVI Student Presentation Award, IPSJ Yamashita SIG Research Award. [refereed, in Japanese]

[RP-2] Akinori Ishitobi, Masanori Nakayama, and Issei Fujishiro: "Visual simulation of weathering coated metallic objects," in *Visual Computing 2020,* December 2020. [invited talk, in Japanese]

[RP-3] Akinori Ishitobi, Masanori Nakayama, and Issei Fujishiro: "A deformation method for simulating coating degradation while taking mechanical behavior into account," *2019 International Conference on Cyberworlds (CW),* Kyoto, Japan, October 2019 [DOI: 10.1109/CW.2019.00066].

References

[1] Yuval Alaluf, Or Patashnik, and Daniel Cohen-Or, "Only a matter of style: Age transformation using a style-based regression model," *ACM Transactions on Graphics*, vol. 40, no. 4, pp. 45:1–45:12, July 2021. DOI: 10.1145/3450626.3459805

[2] Arthur E. Balbão and Marcelo Walter, "A biologically inspired hair aging model," *ACM Transactions on Graphics*, vol. 41, no. 6, pp. 223:1–223:9, November 2022. DOI: 10.1145/3550454.3555444

[3] David Baraff and Andrew Witkin, "Dynamic simulation of non-penetrating flexible bodies," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp. 303–308, July 1992. DOI: 10.1145/142920.134084

[4] Jernej Barbič, Funshing Sin, and Eitan Grinspun, "Interactive editing of deformable simulations," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 70:1–70:8, July 2012. DOI: 10.1145/2185520.2185566

[5] Rachele Bellini, Yanir Kleiman, and Daniel Cohen-Or, "Time-varying weathering in texture space," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 141:1–141:11, July 2016. DOI: 10.1145/2897824.2925891

[6] Jan Bender, Dan Koschier, Patrick Charrier, and Daniel Weber, "Position-based simulation of continuous materials," *Computers & Graphics*, vol. 44, pp. 1–10, November 2014. DOI: 10.1016/j.cag.2014.07.004

[7] Jan Bender, Matthias Müller, and Miles Macklin, "A survey on position based dynamics, 2017," in *Proceedings of the European Association for Computer Graphics: Tutorials (EG '17)*, pp. 6:1–6:31, April 2017. DOI: 10.2312/egt.20171034

[8] Carles Bosch, Pierre-Yves Laffont, Holly Rushmeier, Julie Dorsey, and George Drettakis, "Image-guided weathering: A new approach applied to flow phenomena," *ACM Transactions on Graphics*, vol. 30, no. 3, pp. 20:1–20:13, May 2011. DOI: 10.1145/1966394.1966399

[9] Carles Bosch and Gustavo Patow, "Controllable image-based transfer of flow phenomena," *Computer Graphics Forum*, vol. 38, no. 1, pp. 274–285, July 2018. DOI: 10.1111/cgf.13530

[10] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly, "Projective dynamics: Fusing constraint projections for fast simulation," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 154:1–154:11, July 2014. DOI: 10.1145/2601097.2601116

[11] Ivaylo Boyadzhiev, Kavita Bala, Sylvain Paris, and Edward Adelson, "Band-sifting decom-

position for image-based material editing," *ACM Transactions on Graphics*, vol. 34, no. 5, pp. 163:1–163:16, November 2015. DOI: 10.1145/2809796

[12] Richard Bukowski and Carlo Séquin, "Interactive simulation of fire in virtual building environments," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*, pp. 35–44, August 1997. DOI: 10.1145/258734.258757

[13] Yao-Xun Chang and Zen-Chung Shih, "The synthesis of rust in seawater," *The Visual Computer*, vol. 19, no. 1, pp. 50–66, March 2003. DOI: 10.1007/s00371-002-0172-0

[14] Hsiao-Yu Chen, Arnav Sastry, Wim M. van Rees, and Etienne Vouga, "Physical simulation of environmentally induced thin shell deformation," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 146:1–146:13, July 2018. DOI: 10.1145/3197517.3201395

[15] Wei Chen, Fei Zhu, Jing Zhao, Sheng Li, and Guoping Wang, "Peridynamics-based fracture animation for elastoplastic solids," *Computer Graphics Forum*, vol. 37, no. 1, pp. 112–124, June 2018. DOI: 10.1111/cgf.13236

[16] Yanyun Chen, Lin Xia, Tien-Tsin Wong, Xin Tong, Hujun Bao, Baining Guo, and Heung-Yeung Shum, "Visual simulation of weathering by γ-ton tracing," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1127–1133, July 2005. DOI: 10.1145/1073204.1073321

[17] Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K. Hauser, Ken Goldberg, Jonathan R. Shewchuk, and James F. O'Brien, "Interactive simulation of surgical needle insertion and steering," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 88:1–88:10, July 2009. DOI: 10.1145/1531326.1531394

[18] Kwang-Jin Choi and Hyeong-Seok Ko, "Stable but responsive cloth," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 604–611, July 2002. DOI: 10.1145/566654.566624

[19] Jonathan M. Cohen, Sarah Tariq, and Simon Green, "Interactive fluid-particle simulation using translating eulerian grids," in *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '10)*, pp. 15–22, February 2010. DOI: 10.1145/1730804.1730807

[20] Qing Dai and Xubo Yang, "Interactive smoke simulation and rendering on the GPU," in *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry (VRCAI '13)*, pp. 177–182, November 2013. DOI: 10.1145/2534329.2534358

[21] Brett Desbenoit, Eric Galin, and Samir Akkouche, "Simulating and modeling lichen growth," *Computer Graphics Forum*, vol. 23, no. 3, pp. 341–350, August 2004. DOI: 10.1111/j.1467-8659.2004.00765.x

[22] ——, "Modeling cracks and fractures," *The Visual Computer*, vol. 21, no. 8, pp. 717–726, August 2005. DOI: 10.1007/s00371-005-0317-z

[23] Julie Dorsey, Alan Edelman, Henrik Wann Jensen, Justin Legakis, and Hans Køhling Pedersen, "Modeling and rendering of weathered stone," in *Proceedings of the 26th Annual Con-*

ference on Computer Graphics and Interactive Techniques (SIGGRAPH '99), pp. 225–234, July 1999. DOI: 10.1145/311535.311560

[24] Julie Dorsey and Pat Hanrahan, "Modeling and rendering of metallic patinas," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, pp. 387–396, August 1996. DOI: 10.1145/237170.237278

[25] Julie Dorsey, Hans Køhling Pedersen, and Pat Hanrahan, "Flow and changes in appearance," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*, pp. 411–420, August 1996. DOI: 10.1145/237170.237280

[26] Yuki Endo, Yoshihiro Kanamori, Jun Mitani, and Yukio Fukui, "An interactive design system for water flow stains on outdoor images," in *Proceedings of Smart Graphics: 10th International Symposium on Smart Graphics*, pp. 160–171, June 2010. DOI: 10.1007/978-3-642-13544-6_16

[27] ——, "Image editing for weathering effects with geometric details," in *Proceedings of Computer Graphics International 2011*, pp. S24:1–S24:4, July 2011.

[28] Pavol Federl and Przemyslaw Prusinkiewicz, "Finite element model of fracture formation on growing surfaces," in *Computational Science—ICCS 2004*, pp. 138–145, June 2004. DOI: 10.1007/978-3-540-24687-9_18

[29] Dhana Frerichs, Andrew Vidler, and Christos Gatzidis, "A survey on object deformation and decomposition in computer graphics," *Computers & Graphics*, vol. 52, pp. 18–32, November 2015. DOI: 10.1016/j.cag.2015.06.004

[30] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly Notices of the Royal Astronomical Society*, vol. 181, no. 3, pp. 375–389, December 1977. DOI: 10.1093/mnras/181.3.375

[31] Stéphane. Gobron and Norishige Chiba, "Simulation of peeling using 3D-surface cellular automata," in *Proceedings of Ninth Pacific Conference on Computer Graphics and Applications*, pp. 338–347, October 2001. DOI: 10.1109/PCCGA.2001.962890

[32] MPI Group, "Fitz's atlas of coating defects 2," *Anti-Corrosion Methods and Materials*, vol. 59, no. 3, May 2012. DOI: 10.1108/acmm.2012.12859cab.016

[33] Jinwei Gu, Chien-I Tu, Ravi Ramamoorthi, Peter Belhumeur, Wojciech Matusik, and Shree Nayar, "Time-varying surface appearance: Acquisition, modeling and rendering," in *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*, pp. 762–771, July 2006. DOI: 10.1145/1179352.1141952

[34] Geoffrey Guingo, Frédéric Larue, Basile Sauvage, Nicolas Lutz, Jean-Michel Dischler, and Marie-Paule Cani, "Content-aware texture deformation with dynamic control," *Computers & Graphics*, vol. 91, pp. 95–107, October 2020. DOI: 10.1016/j.cag.2020.07.006

[35] Tobias Günther, Kai Rohmer, and Thorsten Grosch, "GPU-accelerated interactive material aging," in *Proceedings of Vision, Modeling and Visualization*, pp. 63–70, November 2012.

DOI: 10.2312/PE/VMV/VMV12/063-070

[36] Francis H Harlow, "The particle-in-cell method for numerical solution of problems in fluid dynamics," March 1962. URL: https://www.osti.gov/biblio/4769185. DOI: 10.2172/4769185

[37] Koichi Hirota, Yasuyuki Tanoue, and Toyohisa Kaneko, "Generation of crack patterns with a physical model," *The Visual Computer*, vol. 14, no. 3, pp. 126–137, July 1998. DOI: 10.1007/s003710050128

[38] ——, "Simulation of three-dimensional cracks," *The Visual Computer*, vol. 16, pp. 371–378, November 2000. DOI: 10.1007/s003710000069

[39] Rama Hoetzlein and Tobias Höllerer, "Interactive water streams with sphere scan conversion," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*, pp. 107–114, February 2009. DOI: 10.1145/1507149.1507166

[40] Siu-Chi Hsu and Tien-Tsin Wong, "Simulating dust accumulation," *IEEE Computer Graphics and Applications*, vol. 15, no. 1, pp. 18–22, January 1995. DOI: 10.1109/38.364957

[41] Jin Huang, Jiong Chen, Weiwei Xu, and Hujun Bao, "A survey on fast simulation of elastic objects," *Frontiers of Computer Science*, vol. 13, no. 3, pp. 443–459, June 2019. DOI: 10.1007/s11704-018-8081-1

[42] Satoshi Iizuka, Yuki Endo, Yoshihiro Kanamori, and Jun Mitani, "Single image weathering via exemplar propagation," *Computer Graphics Forum*, vol. 35, no. 2, pp. 501–509, May 2016. DOI: 10.1111/cgf.12850

[43] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw, "Volume conserving finite element simulations of deformable models," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 13:1–13:6, July 2007. DOI: 10.1145/1276377.1276394

[44] Tomokazu Ishikawa, Yusuke Kameda, Masanori Kakimoto, Ichiro Matsuda, and Susumu Itoh, "Rust simulation based on chemical reaction processes," *IIEEJ Transactions on Image Electronics and Visual Computing*, vol. 6, no. 2, pp. 82–88, December 2018. DOI: 10.11371/tievciieej.6.2_82

[45] Nisha Jain, Prem Kalra, and Subodh Kumar, "Simulation and rendering of pitting corrosion," in *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing (ICVGIP '14)*, pp. 38:1–38:8, December 2014. DOI: 10.1145/2683483.2683521

[46] Nisha Jain, Prem Kalra, Rohit Ranjan, and Subodh Kumar, "User guided generation of corroded objects," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP '16)*, pp. 89:1–89:8, December 2016. DOI: 10.1145/3009977.3010031

[47] Doug L. James and Dinesh K. Pai, "Multiresolution Green's function methods for interactive simulation of large-scale elastostatic objects," *ACM Transactions on Graphics*, vol. 22, no. 1, pp. 47–82, January 2003. DOI: 10.1145/588272.588278

[48] SoHyeon Jeong, Tae-Hyeong Kim, and Chang-Hun Kim, "Shrinkage, wrinkling and ablation of burning cloth and paper," *The Visual Computer*, vol. 27, no. 6, pp. 417–427, June 2011. DOI: 10.1007/s00371-011-0575-x

[49] SoHyeon Jeong, Si-Hyung Park, and Chang-Hun Kim, "Simulation of morphology changes in drying leaves," *Computer Graphics Forum*, vol. 32, no. 1, pp. 204–215, January 2013. DOI: 10.1111/cgf.12009

[50] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle, "The material point method for simulating continuum materials," in *ACM SIGGRAPH 2016 Courses (SIGGRAPH '16)*, pp. 24:1–24:52, July 2016. DOI: 10.1145/2897826.2927348

[51] Shaohui Jiao, Youquan Liu, and Enhua Wu, "Time-varying simulation for image-based carpets," in *2009 Fifth International Conference on Image and Graphics*, pp. 571–576, September 2009. DOI: 10.1109/ICIG.2009.30

[52] Shaohui Jiao and Enhua Wu, "Simulation of weathering fur," in *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry (VRCAI '09)*, pp. 35–40, December 2009. DOI: 10.1145/1670252.1670262

[53] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai, "Staggered projections for frictional contact in multibody systems," *ACM Transactions on Graphics*, vol. 27, no. 5, pp. 164:1–164:11, December 2008. DOI: 10.1145/1409060.1409117

[54] Micky Kelager, Sarah Niebe, and Kenny Erleben, "A triangle bending constraint model for position-based dynamics," in *Proceedings of the Seventh Workshop in Virtual Reality Interactions and Physical Simulation*, pp. 31–37, November 2010. DOI: 10.2312/PE/vriphys/vriphys10/031-037

[55] Bradley W. Kimmel, Gladimir V. G. Baranoski, T. F. Chen, Daniel Yim, and Erik Miranda, "Spectral appearance changes induced by light exposure," *ACM Transactions on Graphics*, vol. 32, no. 1, pp. 10:1–10:13, February 2013. DOI: 10.1145/2421636.2421646

[56] Seiichi Koshizuka and Yoshiaki Oka, "Moving-particle semi-implicit method for fragmentation of incompressible fluid," *Nuclear Science and Engineering*, vol. 123, no. 3, pp. 421–434, July 1996. DOI: 10.13182/NSE96-A24205

[57] Julian Kratt, Marc Spicker, Alejandro Guayaquil, Marek Fiser, Sören Pirk, Oliver Deussen, John C. Hart, and Bedrich Benes, "Woodification: User-controlled cambial growth modeling," *Computer Graphics Forum*, vol. 34, no. 2, pp. 361–372, June 2015. DOI: 10.1111/cgf.12566

[58] Jing Li, Tiantian Liu, Ladislav Kavan, and Baoquan Chen, "Interactive cutting and tearing in projective dynamics with progressive cholesky updates," *ACM Transactions on Graphics*, vol. 40, no. 6, pp. 254:1–254:12, December 2021. DOI: 10.1145/3478513.3480505

[59] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan, "Fast simulation of mass-spring systems," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 214:1–214:7,

November 2013. DOI: 10.1145/2508363.2508406

[60] Jianye Lu, Athinodoros S. Georghiades, Andreas Glaser, Hongzhi Wu, Li-Yi Wei, Baining Guo, Julie Dorsey, and Holly Rushmeier, "Context-aware textures," *ACM Transactions on Graphics*, vol. 26, no. 1, pp. 3:1–3:22, January 2007. DOI: 10.1145/1189762.1189765

[61] Miles Macklin and Matthias Müller, "Position based fluids," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 104:1–104:12, July 2013. DOI: 10.1145/2461912.2461984

[62] Stéphane Mérillou, Jean-Michel Dischler, and Djamchid Ghazanfarpour, "Corrosion: Simulating and rendering," in *Proceedings of the Graphics Interface 2001 Conference*, pp. 167–174, June 2001. DOI: 10.20380/GI2001.20

[63] Stéphane Mérillou and Djamchid Ghazanfarpour, "A survey of aging and weathering phenomena in computer graphics," *Computers & Graphics*, vol. 32, no. 2, pp. 159–174, January 2008. DOI: 10.1016/j.cag.2008.01.003

[64] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Visualization and Mathematics III*, pp. 35–57, June 2003. DOI: 10.1007/978-3-662-05105-4_2

[65] Keisuke Mizutani, Yoshinori Dobashi, and Tsuyoshi Yamamoto, "Interactive control of fire simulation based on computational fluid dynamics," in *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications—GRAPP, (VISIGRAPP 2016)*, pp. 242–247, January 2016. DOI: 10.5220/0005746902400245

[66] Imanol Muñoz Pandiella, Carles Bosch, Nicolas Mérillou, Gustavo Patow, Stéphane Mérillou, and Xavier Pueyo, "Urban weathering: Interactive rendering of polluted cities," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 12, pp. 3239–3252, December 2018. DOI: 10.1109/TVCG.2018.2794526

[67] Lien Muguercia, Carles Bosch, and Gustavo Patow, "Fracture modeling in computer graphics," *Computers & Graphics*, vol. 45, pp. 86–100, December 2014. DOI: 10.1016/j.cag.2014.08.006

[68] David E. Muller and Franco P. Preparata, "Finding the intersection of two convex polyhedra," *Theoretical Computer Science*, vol. 7, no. 2, pp. 217–236, March 1978. DOI: 10.1016/0304-3975(78)90051-8

[69] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler, "Stable real-time deformations," in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*, pp. 49–54, July 2002. DOI: 10.1145/545261.545269

[70] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, April 2007. DOI: 10.1016/j.jvcir.2007.01.005

[71] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, no. 4, pp. 809–836, December 2006. DOI: 10.1111/j.1467-8659.2006.01000.x

[72] Jian Ni, Mei Yang, and Yingtao Jiang, "Virtual reality simulation of dust accumulation on the surface of solar panel," in *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*, pp. 425–430, December 2017. DOI: 10.1109/ICCSEC.2017.8446946

[73] Alan Norton, Greg Turk, Bob Bacon, John Gerth, and Paula Sweeney, "Animation of fracture by physical modeling," *The Visual Computer*, vol. 7, no. 4, pp. 210–219, July 1991. DOI: 10.1007/BF01900837

[74] Scott Nykl, Chad Mourning, and David M. Chelberg, "Interactive mesostructures with volumetric collisions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 7, pp. 970–982, July 2014. DOI: 10.1109/TVCG.2014.2317700

[75] James F. O'Brien and Jessica K. Hodgins, "Graphical modeling and animation of brittle fracture," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*, pp. 137–146, August 1999. DOI: 10.1145/311535.311550

[76] Yuuji Ogasawara, Kazunobu Muraoka, and Norishige Chiba, "Visual simulation of moss taking into account local environment on temperature and humidity," *The Journal of The Society for Art and Science*, vol. 2, pp. 31–39, January 2003. DOI: 10.3756/artsci.2.31

[77] Nao Ozawa and Issei Fujishiro, "A morphological approach to volume synthesis of weathered stones," *Volume Graphics*, pp. 367–378, March 2000. DOI: 10.1007/978-1-4471-0737-8_24

[78] Eric Paquette, Pierre Poulin, and George Drettakis, "Surface aging by impacts," in *Proceedings of the Graphics Interface 2001 Conference*, pp. 175–182, June 2001. DOI: 10.20380/GI2001.21

[79] ——, "The simulation of paint cracking and peeling," in *Proceedings of the Graphics Interface 2002 Conference*, pp. 59–68, May 2002. DOI: 10.20380/GI2002.08

[80] Ken Perlin, "Improving noise," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 681–682, July 2002. DOI: 10.1145/566654.566636

[81] Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien, "Adaptive tearing and cracking of thin sheets," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 110:1–110:9, July 2014. DOI: 10.1145/2601097.2601132

[82] Yuxing Qiu, Samuel Temple Reeve, Minchen Li, Yin Yang, Stuart Ryan Slattery, and Chenfanfu Jiang, "A sparse distributed gigascale resolution material point method," *ACM Transactions on Graphics*, vol. 42, no. 2, pp. 22:1–22:21, January 2023. DOI: 10.1145/3570160

[83] Cristian Romero, Dan Casas, Maurizio M. Chiaramonte, and Miguel A. Otaduy, "Contact-centric deformation learning," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 70:1–70:11, July 2022. DOI: 10.1145/3528223.3530182

[84] Cristian Romero, Dan Casas, Jesús Pérez, and Miguel Otaduy, "Learning contact corrections

for handle-based subspace dynamics," *ACM Transactions on Graphics*, vol. 40, no. 4, pp. 131:1–131:12, July 2021. DOI: 10.1145/3450626.3459875

[85] Amir Rosenberger, Daniel Cohen-Or, and Dani Lischinski, "Layered shape synthesis: Automatic generation of control maps for non-stationary textures," *ACM Transactions on Graphics*, vol. 28, no. 5, pp. 1–9, December 2009. DOI: 10.1145/1618452.1618453

[86] Andrew Selle, Michael Lentine, and Ronald Fedkiw, "A mass spring model for hair simulation," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 1–11, August 2008. DOI: 10.1145/1360612.1360663

[87] Salman Shahidi, "Salt weathering of brick walls," in *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications (VISIGRAPP 2012)—GRAPP*, pp. 7–15, February 2012. DOI: 10.5220/0003807600070015

[88] Stewart A. Silling, "Reformulation of elasticity theory for discontinuities and long-range forces," *Journal of the Mechanics and Physics of Solids*, vol. 48, no. 1, pp. 175–209, January 2000. DOI: 10.1016/S0022-5096(99)00029-0

[89] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle, "A material point method for snow simulation," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 102:1–102:10, July 2013. DOI: 10.1145/2461912.2461948

[90] George Gerald Stoney, "The tension of metallic films deposited by electrolysis," in *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 82, no. 553, pp. 172–175, May 1909. DOI: 10.1098/rspa.1909.0021

[91] Yuchen Sun, Xingyu Ni, Bo Zhu, Bin Wang, and Baoquan Chen, "A material point method for nonlinearly magnetized materials," *ACM Transactions on Graphics*, vol. 40, no. 6, pp. 205:1–205:13, December 2021. DOI: 10.1145/3478513.3480541

[92] Yuta Suzuki, Tomoaki Moriya, and Tokiichiro Takahashi, "A fast method of iron rust by corrosion based on high resolution voxel models," in *Proceedings of International Workshop on Advanced Image Technology 2019*, pp. 117:1–117:6, March 2019. DOI: 10.1117/12.2521563

[93] Yuta Suzuki, Yuta Yamabe, Tomoaki Moriya, and Tokiichiro Takahashi, "A corrosion and deformation simulation method of 3d iron objects based on voxel automaton," in *Proceedings of International Workshop on Advanced Image Technology 2018*, pp. 92:1–92:4, January 2018. DOI: 10.1109/IWAIT.2018.8369703

[94] Ryoma Tanabe, Tomoaki Moriya, Yuki Morimoto, and Tokiichiro Takahashi, "A generation method of rust aging texture considering rust spreading," vol. 38, no. 16, pp. 95–98, January 2014. DOI: 10.11371/wiieej.13.05.0_94

[95] Unity Technologies, "Unity documents: Material parameters," 2019, https://docs.unity3d.com/2019.4/Documentation/Manual/StandardShaderMaterialParameters.html

[96] Demetri Terzopoulos and Kurt Fleischer, "Modeling inelastic deformation: Viscolelasticity,

plasticity, fracture," *SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 269–278, June 1988. DOI: 10.1145/378456.378522

[97] M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp, "Stiffness and deflection analysis of complex structures," *Journal of the Aeronautical Sciences*, vol. 23, no. 9, pp. 805–823, September 1956. DOI: 10.2514/8.3664

[98] Gilles Valette, Stéphanie Prévost, and Laurent Lucas, "A generalized cracks simulation on 3D-meshes," in *Eurographics Workshop on Natural Phenomena*, pp. 7–14, September 2006. DOI: 10.2312/NPH/NPH06/007-014

[99] Adrien Verhulst, Jean-Marie Normand, Guillaume Moreau, and Gustavo Patow, "Deep weathering effects," *Computers & Graphics*, vol. 112, pp. 40–49, May 2023. DOI: 10.1016/j.cag.2023.03.006

[100] Ondřej Šťava, Bedřich Beneš, Matthew Brisbin, and Jaroslav Křivánek, "Interactive terrain modeling using hydraulic erosion," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08)*, pp. 201–210, July 2008. DOI: 10.5555/1632592.1632622

[101] Jiaping Wang, Xin Tong, Stephen Lin, Minghao Pan, Chao Wang, Hujun Bao, Baining Guo, and Heung-Yeung Shum, "Appearance manifolds for modeling time-variant appearance of materials," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 754–761, July 2006. DOI: 10.1145/1141911.1141951

[102] Joshuah Wolper, Yunuo Chen, Minchen Li, Yu Fang, Ziyin Qu, Jiecong Lu, Meggie Cheng, and Chenfanfu Jiang, "Anisompm: Animating anisotropic damage mechanics," *ACM Transactions on Graphics*, vol. 39, no. 4, pp. 37:1–37:16, August 2020. DOI: 10.1145/3386569.3392428

[103] Tien-Tsin Wong, Wai-Yin Ng, and Pheng-Ann Heng, "A geometry dependent texture generation framework for simulating surface imperfections," in *Proceedings of the Eurographics Workshop on Rendering Techniques '97*, pp. 139–150, June 1997. DOI: 10.1007/978-3-7091-6858-5_13

[104] Jun Wu, Rüdiger Westermann, and Christian Dick, "A survey of physically based simulation of cuts in deformable bodies," *Computer Graphics Forum*, vol. 34, no. 6, pp. 161–187, March 2015. DOI: 10.1111/cgf.12528

[105] Sheng Wu, Teng Miao, Boxiang Xiao, and Xinyu Guo, "A realistic modeling and real time rendering method of fruit decay based on interactive design," in *2017 International Conference on Virtual Reality and Visualization (ICVRV)*, pp. 125–128, October 2017. DOI: 10.1109/ICVRV.2017.00033

[106] Su Xue, Julie Dorsey, and Holly Rushmeier, "Stone weathering in a photograph," vol. 30, no. 4, pp. 1189–1196, July 2011. DOI: 10.1111/j.1467-8659.2011.01977.x

[107] Su Xue, Jiaping Wang, Xin Tong, Qionghai Dai, and Baining Guo, "Image-based material

weathering," *Computer Graphics Forum*, vol. 27, pp. 617–626, April 2008. DOI: 10.1111/j.1467-8659.2008.01159.x

[108] Xiao Yan, Yun-Tao Jiang, Chen-Feng Li, Ralph R. Martin, and Shi-Min Hu, "Multiphase sph simulation for interactive fluids and solids," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 79:1–79:11, July 2016. DOI: 10.1145/2897824.2925897