FEDERAL UNIVERSITY OF CEARÁ
DEPARTMENT OF COMPUTATION
MDCC - MASTER AND DOCTORATE IN COMPUTER SCIENCE
CRAb - COMPUTER GRAPHICS, VIRTUAL REALITY AND ANIMATION

ARTUR PEREIRA SAMPAIO

TEMPORALLY COHERENT SCULPTURE OF COMPOSITE OBJECTS

FORTALEZA
2017

ARTUR PEREIRA SAMPAIO

TEMPORALLY COHERENT SCULPTURE OF COMPOSITE OBJECTS

Thesis presented to the graduate program in Computer Science of the Federal University of Ceará as part of the prerequisites for obtaining the title of Doctor in Computer Science.

Advisors: Prof. Dr. Raphaëlle Chaine
Prof. Dr. Creto Augusto Vidal
Coadvisor: Prof. Dr. Joaquim Bento Cavalcante Neto

FORTALEZA
2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S181t Sampaio, Artur Pereira.
Temporally Coherent Sculpture of Composite Objects / Artur Pereira Sampaio. – 2017.
67 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2017.
Orientação: Prof. Dr. Creto Augusto Vidal.
Coorientação: Prof. Dr. Joaquim Bento Cavalcante Neto.

1. Sculpting. 2. Interactive Techniques. 3. Agglomerates. I. Título.

CDD 005

ARTUR PEREIRA SAMPAIO

TEMPORALLY COHERENT SCULPTURE OF COMPOSITE OBJECTS

Thesis presented to the graduate program in Computer Science of the Federal University of Ceará as part of the prerequisites for obtaining the title of Doctor in Computer Science.

Approved in: 29 / 08 / 2017.

THESIS JURY

Prof. Dr. Creto Augusto Vidal (Orientador)
Universidade Federal do Ceará (UFC)

Profª. Dra. Raphaëlle Chaine (Orientadora)
Université Lyon 1

Prof. Dr. Joaquim Bento Cavalcante Neto (Coorientador)
Universidade Federal do Ceará (UFC)

Profª. Dra. Emanuele Marques dos Santos
Universidade Federal do Ceará (UFC)

Prof. Dr. Esdras Soares de Medeiros Filho
Universidade Federal do Ceará (UFC)

Prof. Dr. Anselmo Cardoso de Paiva
Universidade Federal do Maranhão (UFMA)

To my wife, for the support.
To my brothers, for the inspiration.
To my friends, for the reprieve.
To my parents, for everything.

ACKNOWLEDGEMENTS

To CAPES, FUNCAP and PALSE, for financing this endeavor.

To prof. Raphaëlle, for the leap of faith taken when she took me in, and for all the insight and hard work she put on our research.

To profs. Creto and Bento, for everything they taught me inside and outside of class, and for balancing my countless shortcomings with their experience and knowledge.

To the jury professors Emanuele, Esdras and Anselmo, for taking time out of their busy schedules to watch my presentation and for the invaluable contributions and suggestions.

To my Ph.D. colleagues Lenz, Lilian, Laise and Arnaldo, for walking this difficult path beside me.

"Every flight begins with a fall."
(George R.R. Martin, A Game of Thrones)

ABSTRACT

We address the problem of sculpting and deforming shapes composed of small, randomly placed objects. Objets may be tightly packed - such as pebbles, pills, seeds and grains, or may be sparsely distributed with an overarching shape - such as flocks of birds or schools of fishes. Virtual sculpture has rapidly become a standard in the entertainment industry, as evidenced by the extensive use of software such as ZBrush Pixologic (2017b). Composites, though, are still usually created in a static way by individually placing each object or by sculpting a support surface and procedurally populating the final shape, which raises problems for the generalization of evolving shapes with visual continuity of the components. Large amounts of geometrical data are generated that must be maintained and processed, both by the CPU and by the GPU. Whenever the shape is stretched, pressed or deformed, one has to define how these compositing objects should turn, displace or disappear inside the volume, as well as how new instances should become visible to the outside. It is difficult to rely on a physical system to perform that task in real time. The system we suggest can be constructed upon any uniform mesh-based representation that can be deformed and whose connectivity can be updated by operations such as edge splits, collapses, and flips. We introduce the notion of CompEls as composing elements used to populate the mesh, with aperiodic distribution. These can be automatically updated under deformation. The idea is to sculpt the shape as if it were filled with little objects, without handling the complexity of manipulating volumetric objects. For this purpose, we suggest exploiting the properties of the uniform sampling of the surface with distances between vertices greatly exceeding the CompEls distances. Both the surface and the CompEls are immersed into deformation fields, such that the update of the uniform sampling can be used to track the movement of the CompEls, to identify those which should disappear inside the shape, and empty areas where further CompEls should be generated. This system uses GPU optimizations to efficiently perform the rendering of individual components. To our knowledge, no previous sculpting system allows the user to simultaneously see and sculpt agglomerates in such a fast and reliable fashion.

Keywords: Sculpting, Interactive techniques, Agglomerates.

LIST OF FIGURES

SUMMARY

1 INTRODUCTION

Sculpting is the process of creating an object by modifying, carving or adding material. In computer graphics, this means deforming a digital object using virtual tools. Unlike sculpting real materials, computer graphics sculpting systems are limited in how much geometry they are able to track and update in each step of the sculpting process. For that reason, virtual sculpting requires the development of a host of different strategies to optimize performance and to maintain the illusion of working with a real object. The problem is even more complex when sculpting agglomerate objects, collections of otherwise independent objects assembled in a larger scale form. In fact, the challenge is so great, that little research has been done in the direction of sculpting such materials. Literature on the subject is usually limited to extracting a composite from a support shape, which can be produced with traditional sculpting techniques and applications.

In our research, we developed efficient ways of digitally sculpting and rendering agglomerates. The remainder of this chapter outlines the context in which this research is inserted (Section 1.1), the objectives we pursue (Section 1.2), how we propose to achieve those objectives (Section 1.3) and how the rest of this work is organized (Section 1.4).

1.1 Contextualization

Agglomerate materials are common in nature: rock piles, armies of ants, castles of sand, beautiful works of tiled art - basically any group of similar, randomly oriented elements that, when viewed together, form a larger scale shape. Mankind quickly learned how to create such materials to produce compelling art. Figures 1 and 2 illustrate such composites in nature and in real-world sculptures.

With the development of computer graphics, humans discovered a new canvas onto which they could create and display their art. Modeling virtual objects quickly became essential to the entertainment industry.

Today, artists expect to use abilities developed from real-life sculpting in digital environments. Many systems now allow users to sculpt objects as they would do with virtual clay.

In that context, the industry naturally began to use agglomerates in digital settings. Since they are free from real material constraints, which are usually static once sculpted, such agglomerates can display dynamic shapes, as can be seen in Figure 3.

Unlike regular, non-composite sculpted objects, deforming an agglomerate shape has an effect on the distribution of its composing elements. In the real world, an artist would create an agglomerate by either:

- placing each composing element in its final position, one by one, or;
- designing an intermediate support shape and populating it afterwards.

Figure 1: An ant bridge reveals nature sculpting through dynamic agglomerates.

Source: (Land, 2016).

Though both approaches have traditionally been used in computer graphics settings, the sheer object count and the possibility of employing automation to determine the position of each individual element has usually favored the latter strategy. This means that artists were prevented from experiencing the agglomerate material until completion of the support shape. This limitation is even more serious when the object is meant for animation.

Despite its importance and obvious limitations, though, little research has been done in the direction of sculpting and animating such agglomerates in computer environments.

Representing and updating those kinds of assemblies efficiently remains an open problem. The large quantity of elements on a surface and the polygons required to represent them can cripple even powerful GPUs.

1.2 Objectives

This work aims to develop a technique that allows for the efficient representation, sculpture and rendering of agglomerate objects. With the deformation of the agglomerate, composing elements should change position and appear or disappear automatically amidst the group of elements. Artists expect to be free to deform such shapes or to create animations without worrying about placing each individual composing element. The sculpting process should be experienced in real time and with temporal continuity,

Figure 2: Gaudi's Salamander in Barcelona, Spain shows how agglomerates are used in the depiction of real objects.

Source: (Boyton, 2017).

Figure 3: Deus Ex Machina confronts Neo. The swarm of bugs forming the machine-god's face illustrates an industry inspiration for our work.

Source: (Robertson, 2003).

avoiding sudden appearance and disappearance of composing elements between frames.

The specific objectives of this work are:

- To maintain a user-defined composing element density throughout the sculpted object and sculpting session;
- To allow objects to move according to the deformation field, regardless of the connectivity of the support shape;
- To allow close objects to move apart if a deformation field so requires;
- To have objects appear from and disappear to the interior of the support shape;
- To have objects move smoothly, to avoid popping effects;
- To prevent object intersection whenever possible;
- To allow object variability in color, shape, orientation and pose (for animated objects).

1.3 Proposal

At the core of our proposed method, we use a state of the art sculpting framework (Stanculescu, Chaine, and Cani, 2011) to maintain a quasi-uniform triangular mesh as a proxy for the underlying shape. Composing elements can then be sampled at the desired rate from the surface's faces. Under deformation, we track face area updates to trigger insertion or removal of elements to maintain the required density. We move composing elements about faces participating in each connectivity update, triggered by the sculpting framework, in order to achieve minimal element displacement, insertion and removal.

We apply instance-based deformations in the rendering pipeline to minimize visual intersection between elements.

We use instance-based rendering to minimize bandwidth between the CPU and the GPU. We add extra per-instance ids that can be sampled at the rendering pipeline to perform texture, shape, orientation and animation variation.

1.4 Organization

The remainder of this thesis is organized as follows.

In Chapter 2, we present and discuss the most relevant work related to our research. We proceed to review both the surface sculpting and agglomerate material generation fields. We also discuss the applicability of state of the art techniques to our problem, to better explain the importance of our research.

In Chapter 3 we detail our method, algorithms and techniques. We begin by describing our composite shape representation system and how it relates to the underlying sculpting framework. This explanation is followed by sections that explain in-depth how such system is initialized and maintained throughout the sculpting section. Finally, we present additional rendering optimization and variability strategies.

Chapter 4 reveals the results of our algorithms. We propose different evaluation criteria for our intended applications, and evaluate how our technique performs in each one.

Lastly, we present a summary of our work in Chapter 5. We also discuss its limitations and possibilities for future research.

2 RELATED WORK

This chapter is divided into two parts. Section 2.1 reviews the most relevant literature pertaining sculpting in general. Since our work is a sculpting application in essence, the two major competing sculpting strategies are described and compared against each other, in their usefulness for the purposes of our research. Section 2.2 then details literature pertaining the sculpture and representation of composite objets specifically. In it, we discuss the use, the applicability and the limitations of different methods of producing agglomerates in a real-time sculpting application.

2.1 Sculpting

Virtual sculpting is a set of modeling techniques designed to replicate, in a digital environment, the experience of a sculptor in their work with clay or other materials in a realistic and intuitive way. Information and control over the exact position of each item that constitutes the model is withheld from the user in favor of a simpler approach to the modeling process. The technique seeks to reconcile ease of description of the object intended by the user with flexibility, speed and good topological and geometrical quality.

By simulating the work of a sculptor, the technique allows the generation of organic models, such as characters and environments for games and animations, with great ease. Its level of abstraction and relative difficulty in describing perfectly flat surfaces or edges with high curvature however, limit, but not impair, its use in engineering or architecture.

Moreover, because it is easy to learn and use, the technique has been gaining in popularity amongst artists and studios. This trend has been strengthened by the the development of hardware capable of high model editing speeds, and of haptic input devices.

In general, sculpting applications simulate forces acting on the surface of an initial object, generated using blocking techniques, which deforms in response to the stimulus. The sequence, position, duration and intensity of the force stimuli are controlled by the user by means of a circular brush tool positioned in the 3D space. The tool is a three-dimensional analog to the cursor widely used in 2D image editing software. It establishes the center and the radius of the region to be sculpted, labeled Region of Interest (RoI). Additional input, such as the tool's normal, can also be easily conveyed.

The generation of algorithms and the choice of the best representation for sculpting systems, therefore, are the biggest challenges in the development of sculpting software. Volume representations, for instance, are able to perform boolean operations easily, enabling the creation and removal of holes and new features in the solid trivially. Their usual caveat is the presence of high memory footprints, no explicit connectivity

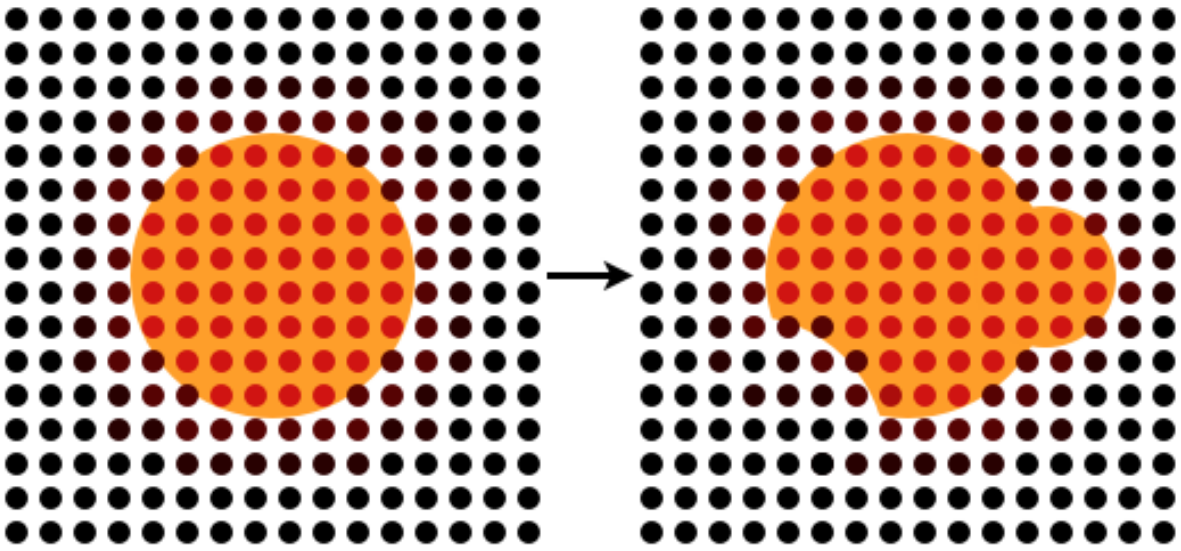
information, and aliasing effects and sampling issues in surface regions not aligned with the coordinate axes. In their turn, boundary representations (B-reps) present faster processing and rendering, more compact storage and explicit surface and connectivity information. As a downside they tend to display a more rigid structure, with greater difficulty in representing boolean operations and addition or removal of material. Both sculpting strategies and pertaining research are discussed in depth in the following sections.

2.1.1 Volume sculpting

As discussed in (Aghdaii, Younesy, and Zhang, 2012), a number of sculpting frameworks use volumetric models with adaptive topology. Those systems operate by locally carving and merging materials under the action of the user. Also known as grid-based methods, algorithms based on such strategies represent solids as uniform or adaptive scalar fields. The object is represented as a discrete density function sampled in each node, called a "voxel", where a density value is stored. The object's surface is implicitly defined as an isosurface, and extracted using the marching-cubes algorithm (Lorensen and Cline, 1987).

(Galyean and Hughes, 1991) were the first to introduce a sculpting system based on volumetric models. Such systems allow the addition and removal of material interactively by changing the density values stored in the voxels. A 2D illustration of their operation can be seen in Figure 4.

Figure 4: 2D grid-based modeling example. Red and black points indicate, respectively, voxels with high and low material density. The boundary of the object is calculated by interpolating these densities. Left: pre-deformation space. Right: geometry changes as a result of the brush.



Source: the authors.

Such systems have been extended with the introduction of adaptive resolution (Ferley, Cani, and Gascuel, 2001) and fixed volume editing (Dewaele and Cani, 2004).

In this kind of representation, it is quite straightforward to implement volume operations, such as digging or adding clay to the solid. A tool moving through the 3D space, capable of inducing local changes in the voxel values, controls the insertion of details. Topological operators, such as opening and closing cuts and holes, can also be implemented in a trivial manner and without the need for any special mechanism. The technique further ensures the formation of closed, manifold, self-intersection free solids. The success of commercial software that use volumetric representations (Pilgway, 2017) attests to the efficiency of this type of representation.

As hinted in the previous section, though, the strategy is not without caveats. Though they are straightforward to implement, volumetric sculpting systems need to store large amounts of data, of the order of the cube of the available space for modeling. The explicit surface also needs to be extracted for processing and rendering. Current algorithms available for the task are liable to sampling, aliasing and continuity artifacts as sections of the surface need to be constantly re-extracted. The continuity issues, most of all, is of special concern to our research's goal. While updates on an undifferentiated surface material can be overlooked by the user, resampling entire regions of composing elements can cause unbearable discontinuities.

2.1.2 Surface sculpting

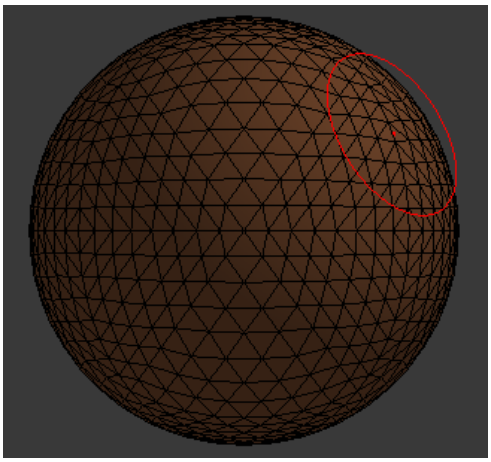
Boundary representations and sculpting systems, in turn, appear as the most widespread in computer graphics. This preference is mainly due to the increased rendering and geometric processing speeds promoted by the existence of boundary and connectivity information (Eyiyurekli and Breen, 2010).

While a number of works deal with the editing of point-based surfaces (Zwicker *et al.*, 2002), the majority focuses on sculpting triangular meshes to represent objects. This is the case for the majority of commercial sculpting applications. This choice favors higher performances by promoting a reduction of traffic between the application and the graphics card (Hernandez, 2011), but usually comes at the cost of not supporting changes in mesh topology. In early systems, changes in the model were limited to vertex displacements. As these displacement algorithms always operated on the same set of vertices, successive geometric operations tended to reduce vertex density. This could hinder the addition of details and features to the model after a certain limit. Figure 5 illustrates the issue.

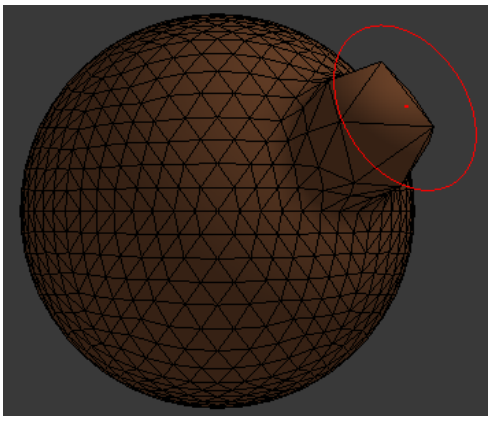
Seeking to minimize this limitation, most applications in the industry, such as Blender (Blender Foundation, 2017) and ZBrush (Pixologic, 2017b), allow the user to perform uniform subdivision on the model. This global boost in vertex density may produce overly dense regions outside the RoI. It also increases the size of the final model

Figure 5: Feature insertion in fixed topology systems.

(a) Initial model.



(b) Model after inflate operation. Affected region presents low vertex density, hindering further detail insertion.



Source: the authors.

while requiring the user to know, even if only a little, about the details of the underlying representation.

A different approach to sculpting, in turn, was the development of Direct Manipulation Of Free-Form Deformation (DMFFD) by Hsu, Hughes, and Kaufman (1992). The technique is an extension of FFD (Free-Form Deformation), described by Sederberg and Parry (1986), that allows it to be considered a sculpting technique. It consists of involving the model space in a three-dimensional parallelepipedic grid (hereafter called FFD grid). By changing the position of the control points of the grid, the contained object can be deformed.

The objects' constituent elements, such as vertices, are parameterized with respect to the FFD grid. The initial position of each element is set as a weighted average of the positions of the control points. The editing of an object is thus made indirectly through manipulation of the FFD grid. The parameterization only needs to be done once, when creating the grid. Figure 6 illustrates a 2D example of the technique.

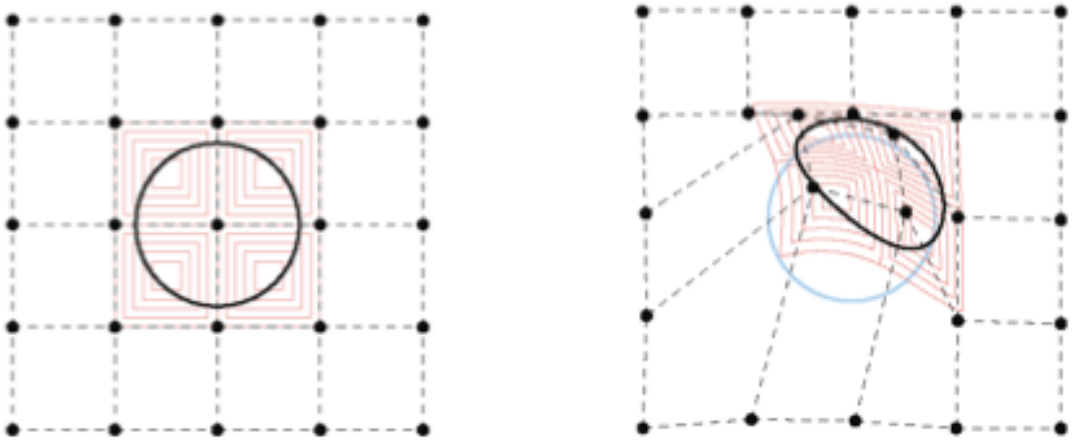
The extension, DMFFD, consists in allowing the user to edit the object directly. To Gain and Dodgson (2001), to deform an object with the manipulation of FFD grid points can be complicated and not very intuitive. Points of the grid can even be hidden inside the model, making them difficult to access, especially in 3D applications.

The technique consists of mapping, to the FFD grid, changes requested by the user. The user indicates changes by moving mesh points directly to a target position. Editing this way generates displacement vectors called *restrictions*. The idea is to identify changes in the FFD grid that would allow the selected point to move according to the restriction. By displacing the grid, the region around the point used to define the restriction can be moved by reapplying the original parametrization.

Figure 6: 2D FFD example.

(a) 2D FFD grid. Control points evenly distributed.

(b) Grid after deformation by the user. The contained object is distorted to conform to the original parametrization.



Source: Gain and Dodgson (2001).

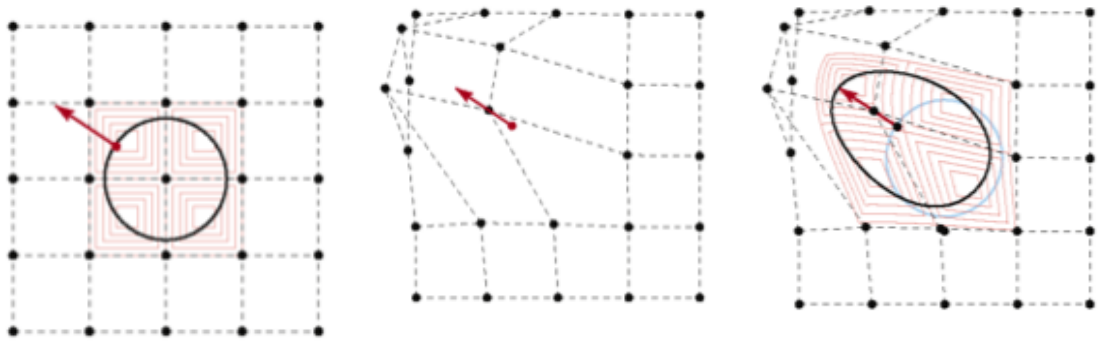
The technique employed to determine these changes in the FFD grid is the least squares. It aims at moving each control point as little as possible. The evaluation of a displacement is made by weighing the proximity of a point to the one used in determining the restriction. Figure 7 illustrates a DMFFD example in 2D.

Figure 7: 2D DMFFD example.

(a) A single point of an object is selected and a restriction (red arrow) is defined, indicating the expected displacement.

(b) Control points displaced in order to conform to the restriction.

(c) Modified FFD grid is applied to the original object.



Source: Gain and Dodgson (2001).

Although the system uses adaptive resolution in the object's representation, the quality of features inserted by DMFFD are closely linked to the density of the FFD grid points. When that density becomes sufficiently small, the representation of new features can become problematic.

Sculpting systems based on triangular meshes such as those proposed by (Gain

and Marais, 2005) and (Stanculescu, Chaine, and Cani, 2011), in turn, used adaptive subdivision and decimation for the addition and removal of features in an intuitive and efficient manner. Those systems are also able to maintain topological and geometrical quality despite the deformation requested.

Such systems also have the merit of incorporating to B-reps capabilities almost exclusively related to volumetric modelers. Examples of such features are fixed-volume editing and self-intersection prevention, possible in both works, and topological modifiers (Stanculescu, Chaine, and Cani, 2011), such as creating and closing holes in the mesh.

The work of Gain and Marais (2005) uses spatial warping techniques to determine displacement vectors at the vertices, along with the adaptive refinement algorithm proposed by Gain and Dodgson (2001). The refinement algorithm takes into account edge lengths and angle between the normals at its incident vertices, performing local subdivision and collapses after certain thresholds. Despite its advantages, sculpting with this method still depends on the use of haptic interfaces in order to become intuitive. To the authors, without the diffusion of this type of interface, manipulating tools in 3D space with just a mouse or touch screen can be somewhat tedious and laborious.

Since then, several applications managed to deform and refine a mesh in real time, by embedding it into a deformation field (Stanculescu *et al.*, 2013). Deformation fields are vector fields or more general position functions that return displacement vectors used to produce local perturbations on surfaces (Gain and Marais, 2005). It assigns a displacement to each point of the evolving surface. Those fields may be defined in the ambient space, while still holding properties such as volume preservation (Angelidis *et al.*, 2006), (von Funck, Theisel, and Seidel, 2006) or detail preservation (Sumner, Schmid, and Pauly, 2007) of the shape being deformed. They can also depend on differential properties of the surface or even on features to be preserved (Stanculescu, Chaine, and Cani, 2011). Laplacian editing (Sorkine *et al.*, 2004) can be used to extend the deformation field between a fixed part of the shape being deformed and a handle. An extension of Laplacian editing can be used to preserve details when stretching adaptive meshes (Dekkers and Kobbelt, 2014). Although such systems can convey a wider range of operations, the performance cost related to updating and solving sparse linear systems usually restricts their use, in the context of solid sculpting, to mesh posing and detail-preserving deformations. (de Goes and James, 2017) exploit closed-form analytical expressions to produce a wide family of deformation tools with physical realism and real-time feedback.

Hernandez (2011) extended Blender (Blender Foundation, 2017) by incorporating adaptive resolution in its sculpting pipeline. The paper attempts to establish a theoretical basis for adaptive resolution systems. It also attempts to describe the inner workings of commercial systems that already offer adaptive subdivision tools to the market, but without technical documents that reveal their internal mechanics, such as Sculptris (Pixologic, 2017a).

The resulting system allows the user to add mesh features in any level of detail (LoD), keeping an adequate density throughout the model. The author defines the operators of subdivision, collapse and relaxation, to be used along with traditional sculpting operators, namely domain selection and sculpting (vertex displacement).

The system described in (Stanculescu, Chaine, and Cani, 2011) seeks to maintain an uniform triangle density by changing their connectivity (addition/removal of triangles and edge rotation operations). The work is based on the maintenance of a good quality, quasi-uniform mesh, in which the edges' lengths of all triangles are within a predetermined range. The lengths of the edges within the RoI are monitored at each brush stroke. Edges whose lengths exceeds the threshold value are subdivided (edge split), while edges smaller than the minimum are collapsed (edge collapse).

Limiting the minimum size of edges not only cleans excessively dense regions, but also gives support to genus changes. The mesh, after being made compliant with the maximum and minimum sizes, has the movement of its vertices limited to a fraction of the minimum edge size. The approximation of two vertices can thus be detected before an intersection occurs. In cases where the distance between two non-adjacent vertices is less than a limit value, the neighborhoods of the two vertices are merged by connecting their 1-rings. The strategy allows authors to replace expensive polygon intersection tests by simpler sphere collision ones. Though fast intersection detection and prevention between multiple rigid objects has been vastly studied, the proposed solutions so far tend to be too slow for real-time (Tang, Manocha, and Tong, 2010).

For the purposes of this research, it is interesting to maintain a quasi-uniform distribution of vertices and good quality triangles on the mesh. A local sampling performed on a good quality triangle grid will present distribution patterns similar to a jittered grid sampling. It is also important to track and correctly manage connection updates, in order to ensure that existing composing elements are not suddenly inserted or removed. The framework described by (Stanculescu, Chaine, and Cani, 2011), thus, strikes as an ideal support onto which our research can be built.

2.2 Sculpting composites

A composite is an assemble of parts that form a larger-scale figure or shape. An ideal composite sculpting framework, though, does more than just create a proxy through concatenation. Besides closeness to the intended shape, a user creating such composites can also be concerned with several other aspects of the agglomerate object and of the sculpting experience, such as: the number of frames per second (fps); temporal continuity; the position, quantity, density and distribution of each composing element; their individual aspect, uniqueness, how they interact with and if they intersect each other; and how those elements are preserved or how smoothly they evolve through the

sculpting session. The temporal continuity that we aim to obtain is similar to the one defined by Medeiros *et al.* (2014). Objects should be inserted, displaced and removed without popping effects or loss of visual continuity between frames.

As stated in (Ma *et al.*, 2013), the biggest challenge in creating an agglomerate is controlling the position of individual objects. Similarly to what happens in real composite sculpture, users had to place each composing element in its final position. This could be done either by simply adding elements floating in an empty 2D or 3D space or by first creating an intermediate representation and to use automated methods to populate it with smaller elements afterwards. The remainder of this section classifies the related work into techniques that automatize either approach.

Research concerning the automatic positioning of surface elements or details under surface deformation or sketching will be called *agglomerate sketching*. Section 2.2.1 provides further discussion on the subject.

On the other hand, approaches that generate an agglomerate from an intermediate shape will hereafter be dubbed *composite extraction* algorithms. An in-depth literature review of such methods is presented in Section 2.2.2.

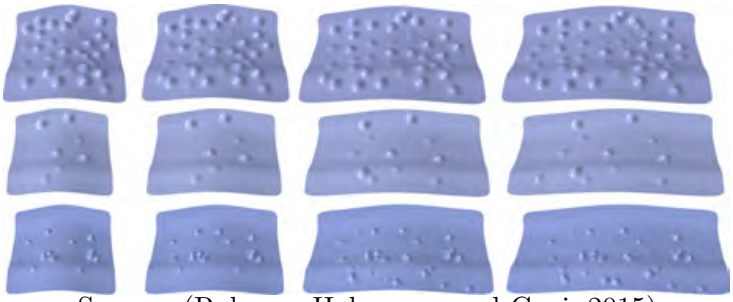
2.2.1 Agglomerate sketching

What we call agglomerate sketching or agglomerate surface editing frameworks are those that employ common sculpting techniques to insert and manage details on a changing shape. Users usually expect to have new composing elements added along a brush curve or to have previously defined patterns automatically managed through surface deformation.

Abdrashitov *et al.* (2014) employ a sketch-based interface to interactively create 2D mosaics. The user is able to add new details to an object along curves and contours. The framework is also capable of automatically converting images to mosaics in a way that more closely resembles a Texture bombing technique (Section 2.2.2.3).

More recently, research has been done on detail-preserving deformation (Rohmer, Hahmann, and Cani, 2015). The possible edits are those that do not drastically affect the overall aspect of a shape, like the stretching of one part. The method is capable of maintaining temporally coherent details on a meshed surface stretch with local updates performed on the geometry and on the connectivity. For instance, the number of details increases on an enlarging area. Details are represented as high frequency changes applied on top of the objects' surfaces. Furthermore, new details are added at the location of preexisting ones and moved, scaled and rotated to their final position. Figure 8 illustrates the technique. Though capable of decoupling mesh and detail resolutions, their method is not intended for agglomerate material representation, for large-scale deformations or for real-time applications.

Figure 8: Three different scenarios with varying detail quantity and size.



Source: (Rohmer, Hahmann, and Cani, 2015).

2.2.2 Composite extraction

As a whole, composite extraction algorithms and frameworks are capable of producing very high quality element distributions and images. Extensive research has been done on how to generate and render such composites.

Users can usually expect their composite material to present desirable element distributions, such as regular, blue noised or even sample-based ones (Emilien *et al.*, 2015). This a posteriori element sampling can either be limited to the object's surface or encompass the entire volume (Peytavie *et al.*, 2009). Usually, the representation of the support shape does not need to be exact or to have high quality. It can result from the rough meshing of an implicit surface or from any other meshing design. Constraints to the quality of the facets are desirable but, ultimately, not required, since the composite material's extraction usually involves a parameterization of the shape.

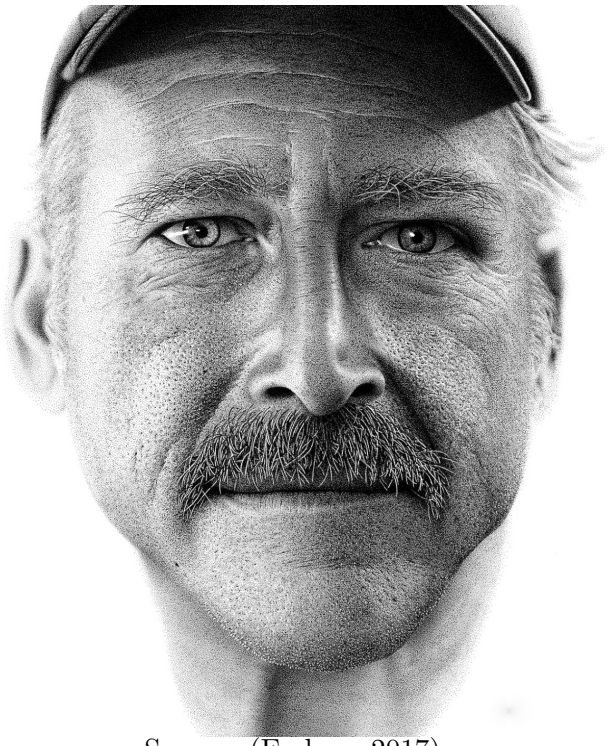
As a downside, the optimization techniques involved in those methods are generally not intended for real-time applications or for dynamic surfaces. The corollary is that the user cannot see the desired agglomerate pattern before the end of the design process of the intermediate shape. This can be a problem for the design of object clouds, where it becomes very difficult to have an idea of the final shape's outer silhouette without the presence of the actual objects. The problem is even more severe in the representation of animations, where temporal continuity or coherence is also of concern. Merely resampling elements on updated regions of an animated surface can cause significant visual artifacts, due to sudden changes associated with rapid object insertion and removal, as described by Schwarz and Stamminger (2009). A common approach to minimize the cost of positioning such elements over the surface is to delay their involvement until the rendering pipeline. At rendering stage, element positioning can be determined with the aid of adaptive sampling functions and projections, capable of reacting to changes in shape density. However, they still rely on previous parameterization and, thus, are not directly usable for our purposes.

In this section, we classify the different types of composite extraction algorithms based on how each composing element is represented or rendered: stippling, tiling, and texture bombing.

2.2.2.1 *Stippling*

An intuitive way to use individual elements to represent a composite shape is to use dots (called stipples). That process of rendering images is called stippling. For the purpose of our research, a stipple can later be transformed into the anchor of an orientable texture unit. But dots can be used not only to represent images and contours, but also to darken or highlight sections of an image. The invention of this type of art is attributed to Giulio Campagnola around 1510 (Simpson and Wood, 1987). A hand-made example of the technique can be seen in Figure 9.

Figure 9: "Benjaman Kyle", by Miguel Endara. The artist used around 2.1 million ink dots to convey shape, shading and depth.



Source: (Endara, 2017).

When producing stippling procedurally, one usually aims at obtaining blue noise samplings (Vanderhaeghe *et al.*, 2007). Hierarchical tiles with different levels of resolution (Ostromoukhov, Donohue, and Jodoin, 2004), (Wachtel *et al.*, 2014) can be used to generate adaptive samplings in images with fast timings (except for preprocessing costs).

With respect to meshes, Pastor, Freudenberg, and Strothotte (2003) use a dynamic stippling generation system that operates directly on adaptive subdivision meshes. Their hierarchical framework and mesh surface can be edited for nearly isometric deformations. The direct use of their framework is limited for the scope of this research, however, since they rely on fixed topology, shape and overall connectivity. An additional limitation is the tight coupling between vertices and stipples. Medeiros *et al.* (2014) alleviate the

shape restriction and the tight coupling by precomputing a hierarchical Poisson disk sampling on the surface. It can be later evaluated and rendered according to an importance function. Their system, however, uses fixed connectivity meshes.

2.2.2.2 *Tiling*

Textures are also a fast and reliable way of adding information to meshed surfaces without requiring any additional geometry description. For that reason, they are natural candidates for representing composites. Tiled composite textures can be mapped to large surfaces by reusing texture cells. The use of textures in such a way is called tiling.

From the point of view of our research, a composite could be created by aperiodic tile packing and good surface parameterization, such as the one by (Ying, Li, and He, 2014). This makes it unecessary to generate individual compositing elements on the surface. The resulting textured surface can also be made non-periodic while retaining most of the attributes of the tile packing. Most of the related research outlined in this section follows this strategy (Wei, 2004), (Li *et al.*, 2010), (Vanhoey *et al.*, 2013).

The tile packing is usually created from methods such as Wang Tiles (Grünbaum and Shephard, 1986) and Corner tiles (Lagae and Dutré, 2006), which have an expensive preprocessing step: the construction of a good distribution, such as Poisson Disk Distribution, of the initial tile packing, and surface parameterization.

The first step, as described by Lagae *et al.* (2008), involves producing a finite set of tiles used to pave the surface being textured. Each tile is square-shaped, has fixed orientation, and displays tags or colors in its edges (Wang Tiles) or corners (Corner Tiles). The different possible color combinations will produce the different stencil combinations that will be stamped on the surface. Edges or corners with the same color are painted in a way that there is no discontinuity. This strategy will produce the atlas illustrated in Figure 10.

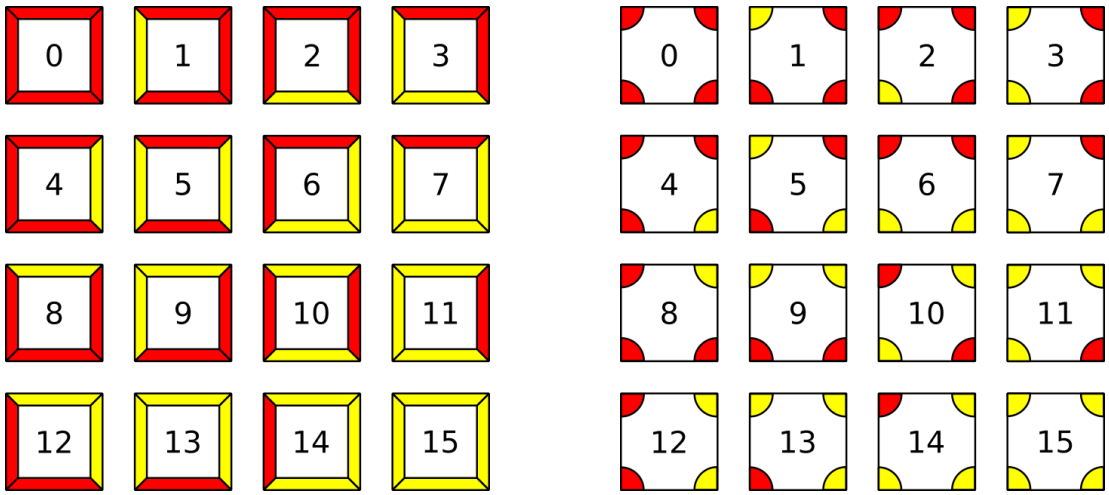
In both strategies, the atlas is reorganized over a large square texture in such a way that adjoining tiles touch at an edge or a corner of the same color. Tiles along the edge of the texture must also match colors with their respective mirror tiles in each direction to prevent discontinuities.

After surface parameterization, the resulting tiled texture can be mapped on the fly by Direct Stochastic algorithms implemented by hashing functions, as discussed in (Lagae *et al.*, 2008). In order to generate the most photorealistic agglomerate materials, recent research has expanded the idea to 3D texture mapping. This allows for the representation of complex details on parameterized shapes (Koniaris *et al.*, 2014), (Meng *et al.*, 2015).

The main restriction to this kind of strategy to our intended application on dynamic meshes is that re-parameterization after deformation may be time consuming

Figure 10: Wang Tile and Corner Tile atlases created from all possible tiles for 2 colors.

(a) Wang Tiles. (b) Corner Tiles.



Source: the authors.

or may be temporally non-preserving. Surface tracking methods can ensure consistent parameterization (Bojsen-Hansen, Li, and Wojtan, 2012) and insertion of details (Mercier *et al.*, 2015) through animations, even on large-scale surface deformations, but, so far, have not been able to do it in real-time.

2.2.2.3 *Texture bombing*

Texture bombing is a more direct approach to using textures for agglomerate extraction. It involves sampling texture or mesh elements directly on the target surface (Dischler *et al.*, 2002). Unlike regular textures, no combined or tiled texture atlases are ever produced through the rendering process. Instead, elements are placed on the surface directly through parameterization, and rendered individually. The combined rendering of each element produces the composite image. This allows for greater user control since, unlike tiled textures, adding new elements to the shape has no impact on other surface regions. The technique can also result in reduced memory usage, as each different type of composing element need only be stored once.

Sampling variation and rendering efficiency can be improved by adding noise and by precomputing levels-of-detail (Wang *et al.*, 2013). Figure 11 shows agglomerates produced by their technique.

A common limitation is the requirement of static connectivity and geometry. Changing the underlying mesh connectivity would require recalculating all texture and element particle maps. de Groot *et al.* (2014) reduce this limitation somewhat by performing only local parameterization. This allows for small deformations on the surface with consistent texturing and real-time rendering. 2D texture decals can then be placed on the surface at user-specified positions. It is not, however, targeted at accumulating or

Figure 11: Texture bombing used to create different shapes with the appearance of agglomerates and corresponding composing elements.

Source: (Wang *et al.*, 2013).

large scale deformations. Nonetheless, for the purposes of our research, texture bombing remains the closest technique to what we try to accomplish. In that regard, our algorithms can be thought of as texture bombing on dynamic meshes.

2.3 Final considerations

In this chapter, we overviewed the different approaches to both general sculpting and composite material generation. The unique characteristics as well as similarities of the different strategies in each domain were analyzed and explained.

In the domain of sculpting, we opted for the use of a surface method over a volume one, due to the better tracking of individual samples the former provides. We found the work of Stanculescu, Chaine, and Cani (2011) to be optimal as the basis for our technique, as it provides good triangle quality and sampling distribution whilst actively tracking connectivity updates.

Regarding composites, the strategies we described can be visually compared in Table 1.

We found that the bulk of the literature to which these techniques pertain revolves around their *a posteriori* extraction. To the best of our knowledge, no agglomerate sculpting framework exists previous to the one we describe in this work. We also found that neither of the existing composite extraction techniques is capable of simultaneously providing:

- high quality element distribution;
- real-time rendering and;
- dynamic shapes with visual continuity between frames.

This absence reinforces our belief that our approximative solution can be of use to the industry and academia. In the next chapter, we present the proposed technique

Table 1: Comparison of the different agglomerate object sculpting techniques.

	High Quality	**Real Time**	**Dynamic Shape With Visual Continuity**
One by one placement	Any / Ideal	No	Yes
Physical Systems	Blue noise	No	Yes
Populate Support Shape (Meng *et al.*, 2015), (Peytavie *et al.*, 2009)	Blue noise	No	No
Stippling (Dots) (Medeiros *et al.*, 2014)	Blue noise	Yes	No
Texture (Wei, 2004)	Blue noise	Yes	No
Texture Bombing (Wang *et al.*, 2013)	Blue noise	Yes	No

Source: the authors.

in detail.

3 TEMPORALLY COHERENT SCULPTURE OF COMPOSITE OBJECTS

In this thesis, we address the problem of sculpting shapes composed of small 3D elements. For now, we exclude the use of shapes considered "too elongated", such as hair fibers, since they can be seen as lower dimensional objects. Furthermore, we are interested in shapes whose component elements can be distant or loosely glued to each other. Under the action of a deformation field, two touching elements should be able to separate, if so required.

In Section 3.1, we describe the inner workings of the underlying sculpting system we use. This system favor a uniform sampling of the evolving shape, which we use to describe the relative positions of 3D elements to one another.

In Section 3.2, we suggest how composite shapes can be empirically described by a system of elements referred to as CompEls. We do not implement interaction forces for this, but only identify local area variations on the shape's envelope.

Section 3.3 describes how to initialize the CompEl coverage of the shape and how to update it under the action of a deformation field. Such fields control the positions of the surface's vertices and of the CompEls, even when they do not correspond to isometric deformations. Intuitively, CompEls undergo the same deformation of the faces of the support mesh, whose vertices present uniform sampling, except when a triangle disappears or is split. We suggest fast redistribution schemes aiming at little to no CompEls displacement in Section 3.3.1. While splitting a triangle is quite natural, the transfer of CompEls from a destroyed triangle imposes a stretch in sub-regions of neighboring triangles. This is necessary to allow space for the inherited elements. The proposed heuristics only uses simple barycentric coordinates tests and computation of pairs of closest points between two edges.

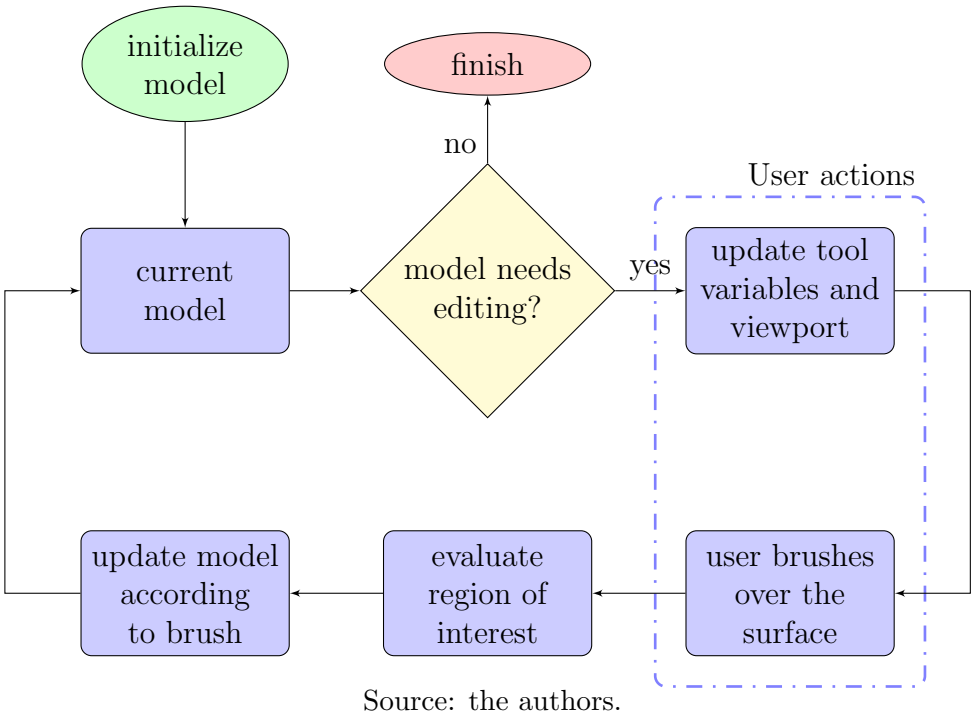
Finally, in Section 3.4 we discuss how to use the graphic card to perform CompEl variation, visual intersection prevention and to speed up the display of the CompEl-made shapes.

3.1 Maintaining a uniform set of anchors on the shape during sculpting

A sculpting system is an application responsible for enabling, tracking and conveying deformations requested by the user. The sculpting process can be seen as a sequence of edits performed on a surface by brushes, as can be seen in Figure 12.

Since we are interested in empirically simplifying the deformation of a compound shape, it may be worthwhile to get rid of internal components, and to focus on visible elements only. These are the elements of the outer layers of the evolving shape. We also need descriptors in order to locally identify the effects of deformation on the surface

Figure 12: Sculpting flowchart.



Source: the authors.

of the shape. Indeed, we aim to know at all times where the surface is contracting and where it is stretching. This will be useful to avoid computing exact collisions between components. To this end, we propose to maintain a uniform sampling of a rough envelope of the shape with a set of anchors. Wherever the shape is deformed isometrically, the sampling should remain stable. In contrast, the emergence of new samples is an indicator of the areas of expansion, where new components should become visible. Finally, the disappearance of samples will identify areas where the shape contracts locally.

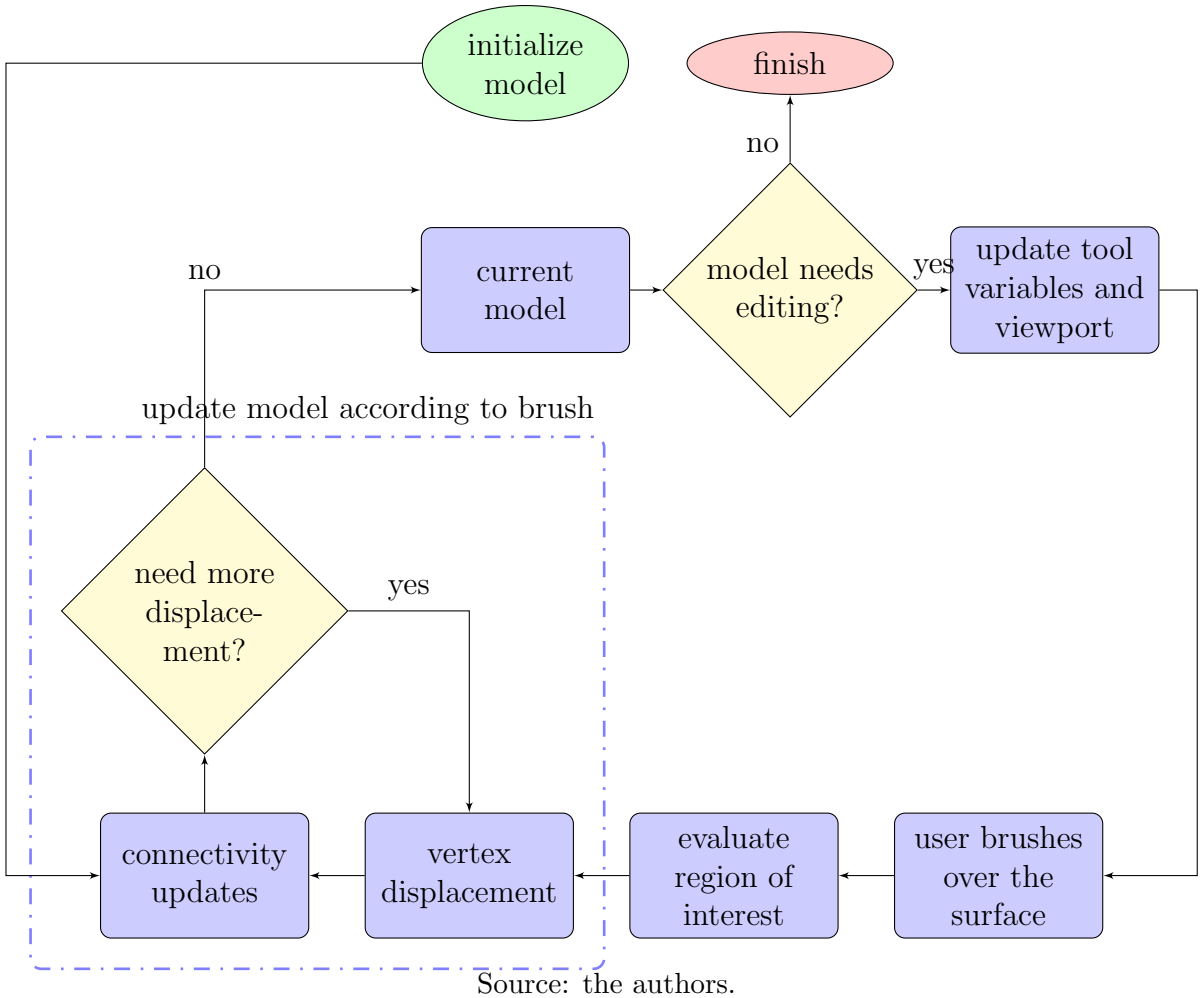
It is important, therefore, to locate the components that we track with respect to the uniform sampling. We do not aim to couple each component with a sample, since we do not want the elements to be distributed regularly over the surface. Furthermore, we want the number of samples to be significantly smaller than the number of components. We suggest connecting the samples into a triangulated mesh to anchor the components therein. For that aim, we focused on the quasi-uniform meshes proposed by Stanculescu, Chaine, and Cani (2011) to maintain a nearly uniform sampling of a shape that is sculpted under the action of deformation fields.

Within the general sculpting flow (Figure 12), that algorithm expands the "update model" task into two consecutive subroutines, namely "vertex displacement" and "connectivity updates". Those subroutines can be iterated over several times before returning to the "current model" node, under conditions explained in Section 3.1.1.

Vertex displacements are, in our case, simple local evaluations of a displacement field centered at the user's tool. The "connectivity updates" subroutine is better explained in Section 3.1.1. This subroutine is also added to the end of the "initializa-

tion" step, and ensures that the current model is always quasi-uniform compliant at the beginning of a sculpting cycle. Figure 13 shows how this model deviates from the general flow.

Figure 13: Modified sculpting flow of the Freestyle framework (Stanculescu, Chaine, and Cani, 2011).



Source: the authors.

3.1.1 Model updates under freestyle sculpting.

Quasi-uniform meshes are characterized by a level of detail d, the maximum distance between two adjacent samples. Unlike similar systems (Lachaud and Montanvert, 1999) previously proposed in image processing, quasi-uniform meshes do not guarantee a minimum length for the edges but still promote the elimination of edges that are too short. To that effect, the user also establishes a minimum edge length d_{min}. The relation between the two variables is

$$d_{min} \leq d/2. \tag{1}$$

Whenever the surface is deformed, edges that have become too long are broken through a sequence of flips or splits, which means that samples moving apart are no longer

connected, and that new samples are inserted in between whenever a split is performed. The sequence that subdivides long edges is followed by a sequence that merges short edge extremities. When this step is over, all the edges that once again grew larger than d are flipped or split. This algorithm optimizes the uniformity of the edges, and does not require any additional step to achieve it. It also does not require a relaxation loop. The resulting mesh is only consistent with the constraint regarding the level of detail d. The uniformity obtained in practice turns out to be sufficient for our purpose.

Finally, the "connectivity update" subroutine is also characterized by a minimum thickness t, below which the shape breaks locally. In order to ensure an accurate tracking of the deformation (at the precision d) and of the changes in topological genus, deformations have to be subdivided into smaller steps of maximum length d_{move}. The relation between those variables and d is

$$4d_{move}^2 \leq t^2 - d^2/3. \tag{2}$$

Algorithm 1 shows the final pseudocode for the "update model" routine, while Figure 14 illustrates the four basic connectivity operations invoked by the subroutine. For an in-depth explanation, the reader is referred to Stanculescu, Chaine, and Cani (2011).

Algorithm 1 The update model routine

```
1: procedure UpdateModel(dField, mesh, d, d_min, t, d_move)
2:     if d_move = 0 then return
3:     n ← ⌈dField.max/d_move⌉
4:     dField.max ← dField.max/n
5:     for i ← 1, n do
6:         mesh.vertices ← Displace(dField, mesh.vertices)
7:         mesh ← SplitEdgesGreaterThanD(d, mesh.edges)
8:         mesh ← CollapseEdgesSmallerThanDmin(d_min, mesh.edges)
9:         mesh ← FlipOrSplitEdgesGreaterThanD(d, mesh.edges)
10:        mesh ← JoinRingOfVerticesCloserThanT(t, mesh.vertices)
```

3.2 Description of outer layers of a shape by a system of CompEls

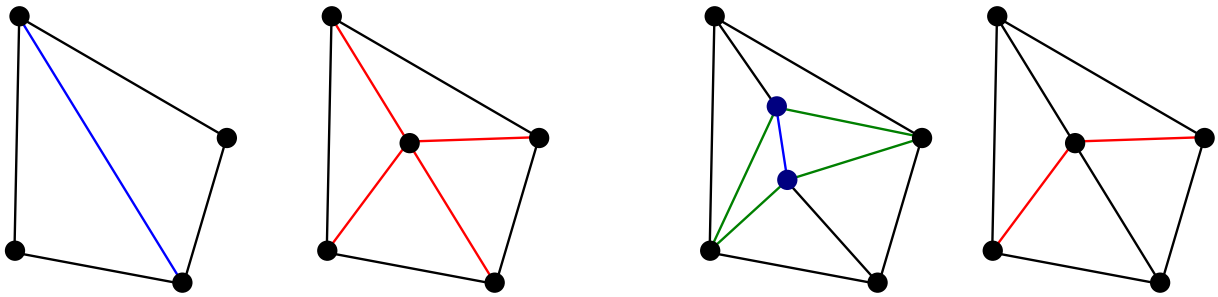
We suggest to use quasi-uniform meshes as an anchor for the component elements of the shape. Given a quasi-uniform mesh of the input shape, we propose to populate it with a set of elements that we shall refer to individually as CompEl (Composing Element) hereafter.

Formally, a CompEl is a 3D object anchored in a triangle. They are the individual elements rendered when representing an agglomerate object. They are positioned within a triangle according to each of their barycentric coordinates with respect to that triangle's vertices. They are also characterized by a depth level, with respect to the en-

Figure 14: Freestyle (Stanculescu, Chaine, and Cani, 2011) basic operations.

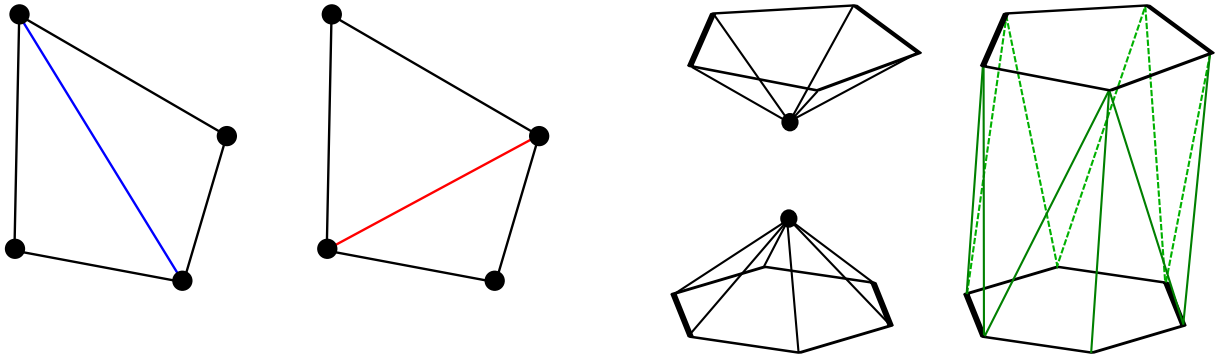
(a) Edge split. As the blue edge becomes larger than d, it is split into two equal segments.

(b) Edge collapse. This operation is performed if the size of the blue edge is smaller than d_{min}.



(c) Edge flip. The blue edge larger than d is flipped. This may be interesting when the flipped edge's size is smaller than d and when a split would create edges smaller than d_{min}.

(d) Ring join. When two non-adjacent vertices get within t of each other, the surface changes topology by removing both vertices and stitching their 1-rings together.



Source: the authors.

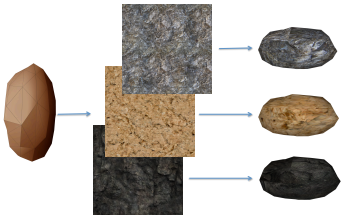
velope of the object, and by an orientation (rotation) relative to that of their support triangle. Figure 15 visually reveals how a CompEl is created. A thorough explanation of its defining arguments can be found in Section 3.3.

Whenever a composite shape is deformed, its elements are displaced under the action of the deformation. If the displacement is not constant throughout the shape, it is possible to have elements come in contact with or be separated from one another. The nature of the interactions between components that collide can be very complex to model, leading to numerical systems that are difficult to solve in real time. Although not physically accurate, we decided to make a compromise by basing our approach on empirical descriptions instead. We stress that our goal is to offer the user a magnified experience of sculpture, still in real time, and we do not necessarily adhere faithfully to the physics of the real world. The observations we have decided to focus on are the following:

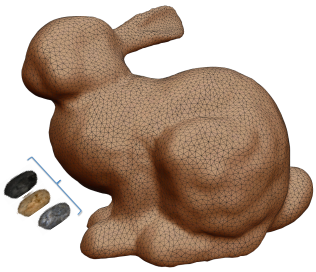
- Elements that move toward and touch each other are clamped in their movements, they cannot get closer. This implies a change in their position at this time step, compared to the expected position that would have been reached in absence of collision (position induced by the local deformation of the shape exclusively). Among the elements visible from the outside, some only undergo tangential additional movement

Figure 15: CompEl definition and placement.

(a) A CompEl produced from a rock mesh and three different textures.

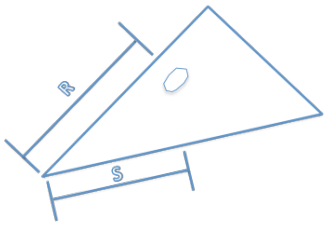


(b) A quasi-uniform Stanford Bunny before CompEl insertion.

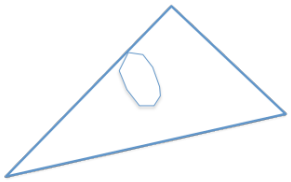


(c) The Stanford Bunny rendered as an agglomerate object.

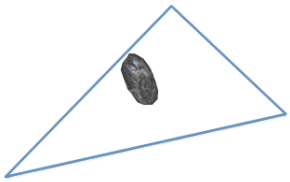
(d) A CompEl position within a triangle defined by randomly picking a position within a triangle.



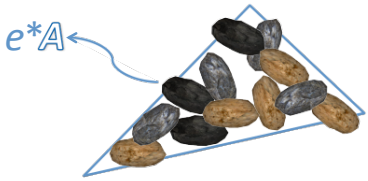
(e) Its orientation and size are also specified.



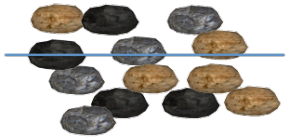
(f) Final CompEl placement relative to the triangles's vertices.



(g) The actual number of CompEls inserted on a triangle, based on a user-defined density e and the triangle area A.



(h) A side view of the triangle in Figure 15g reveals how CompEls are distributed over different layers.



Source: the authors.

on the surface, while others are pushed inside the shape and disappear behind other elements.

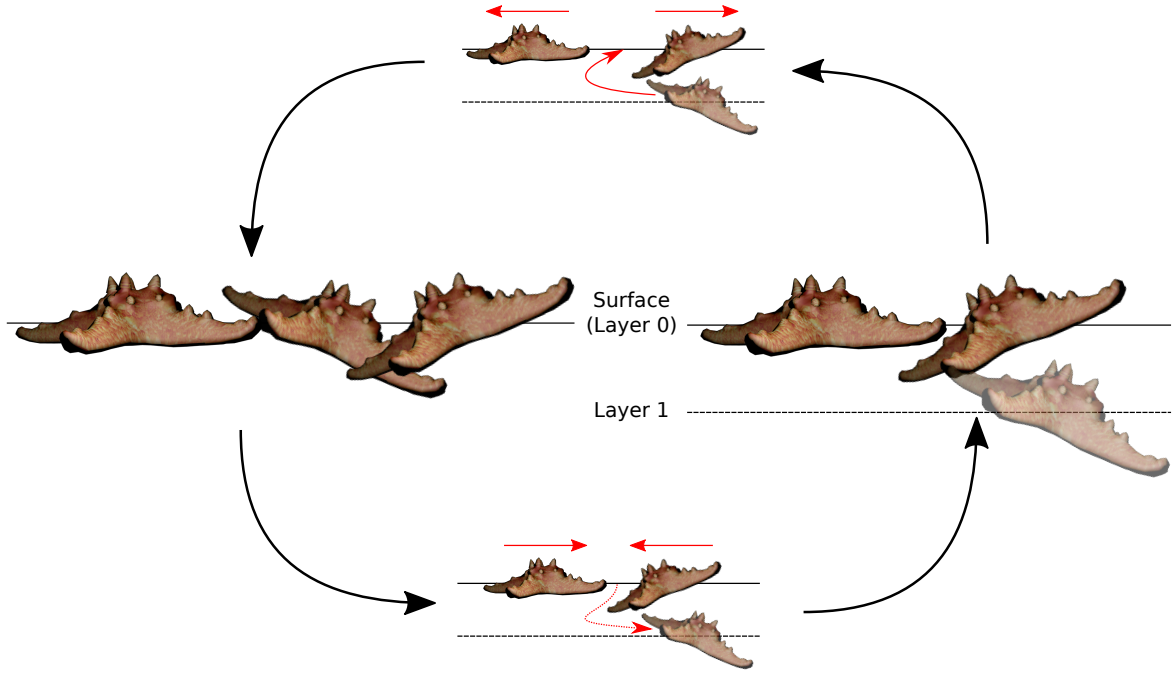
- Conversely, when the deformation field separates elements that were previously in contact, new elements in the gap left between them, emerging from the interior of the shape.

These simple rules result in the expected behavior shown in Figure 16.

3.3 Generation and update of CompEls under deformation

The initial distribution of CompEls is obtained using an accelerated dart throwing technique on an input quasi-uniform mesh (which can be obtained from a regular

Figure 16: Promotion and demotion of elements in a system of two layers. Left → bottom → right: three elements, under deformation, move towards each other. One of the set is moved to a new layer, away from the surface, where it can be removed if necessary. Right → top → left: two elements, under deformation, are set to be separated, creating a void zone. An element is inserted in a new layer, if one does not already exist, and moves to the surface to fill the gap.



Source: the authors.

manifold mesh by means of the UpdateModel procedure described in Algorithm 1). The different CompEl types used can be added to the application as a series of models. Each model contains not only geometry and connectivity, but also color, texture and animation information, when applicable.

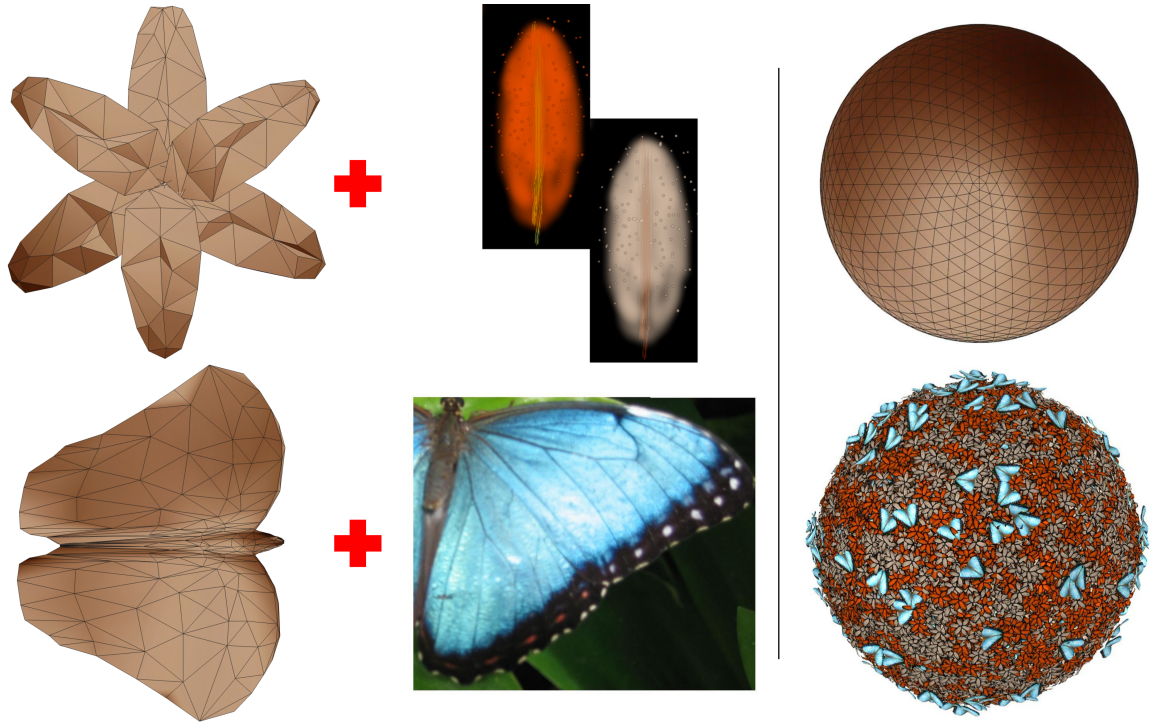
In addition to the variables d, d_{min}, d_{move} and t, required by the Freestyle framework and described in Section 3.1.1, the user should also provide the target density e_i per CompEl type i, which corresponds to the number of CompEls of that type visible or partially visible per unit area, and the number of layers n over which the CompEls are distributed.

CompEls are then inserted in the faces by randomly picking their barycentric coordinates, their orientation and their depth level. This depth level is just an artificial information to be used by the graphics card during the rendering step, as described in section 3.4.

The method is fully parallelizable, as it only requires information from the given face. Figure 17 illustrates this initialization.

Considering that the faces may not exactly have the same shape or size, each

Figure 17: Initializing a system of CompEls. Left: input CompEls and accompanying textures. right: initial sphere before and after CompEl insertion over its surface.



Source: the authors.

face receives a number of CompEls proportional to its area. In a face f of area $a(f)$, with CompEl density e_i for CompEls of type i, the number $c_i(f)$ of CompEls inserted can be defined as

$$c_i(f) = a(f) * e_i. \tag{3}$$

If one wishes to sculpt a shape with the appearance of being full of CompEls of a given type i, it is important to choose e_i sufficiently large so that:

e_i*(area of the smallest square obtainable from all projections of a CompEl of type i) ≥ 1. (4)

However, since this calculation is local, it is still possible to have empty regions if individual faces have areas too small. To mitigate the problem, we propose adding to $c_(f)i$ a random number on the interval [-1, 1] with uniform distribution. The final definition for the number $c_i(f)$ of CompEls then becomes:

$$c_i(f) = a(f) * e_i + \text{random}[-1, 1]. \tag{5}$$

This naïve initialization is simple and powerful enough to create a good jittered grid-like distribution, provided that the faces are approximately regular, with a controlled area, which is favored by the underlying sculpting framework.

3.3.1 Temporal coherence and continuity under freestyle sculpting

Whenever the shape is changing under the action of a deformation field, the vertices of the quasi-uniform mesh move accordingly and the triangles are deformed. As described in Section 3.1.1, outgrown triangles are automatically divided into smaller ones, while triangles too small are collapsed. It is important to set rules for internal evolution and transfer of CompEls between those triangles.

Wherever the shape undergoes a rigid transformation locally, faces keep their initial dimensions. In those situations, no CompEl migration is performed between faces and CompEls do not change position relative to their current support faces. Anywhere else, we suggest the evolution of CompEls to be directly coupled with the elementary operations performed on the triangles: flip, split and collapse of edges and the merge of the 1-ring of two vertices.

The following rules should be interpreted as CompEls migration rules, from the triangles being destroyed towards the triangles being created and towards neighboring triangles whose dimensions are impacted by the operation. Those rules remain local and simple, while still ensuring temporal continuity of CompEls. Thus, the evolution of the resulting CompEl distribution does not relate to some force equalizing algorithm, as it is the case with a physical system. The update can be performed in real time, possibly involving parallelism.

3.3.1.1 Edge split

Splitting an edge that has grown beyond d after applying a deformation results into replacing its two incident triangles with four others. We first address the case where the edge is cut in the middle. We suggest a very simple CompEls coordinate update within these triangles that ensures that CompEls remain where they were positioned before the division step. Figure 18 illustrates the technique on a face f affected by an edge split.

Split update: Suppose that B(C) = (α, β, γ) is the barycentric coordinates of a CompEl C with respect to the vertices V_1, V_2 and V_3 of a face f to be split on V_1V_2 with:

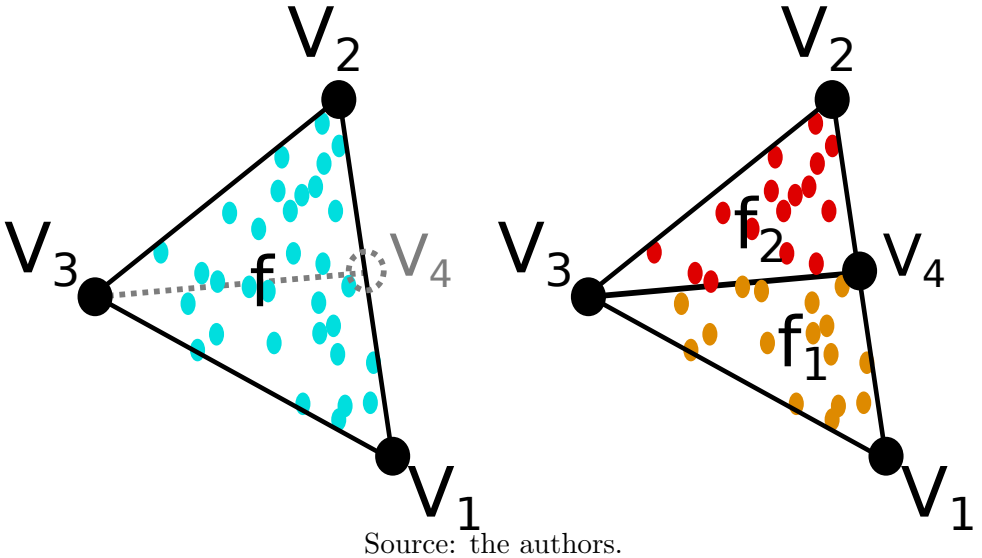
$$C = \alpha V_1 + \beta V_2 + \gamma V_3 \tag{6}$$

a straightforward implementation to the CompEls distribution between each of the two resulting faces f_1 and f_2 is simply to transmit to f_1 the CompEls whose barycentric coordinate with respect to V_1 is bigger than that to V_2, and the remaining to f_2. The barycentric coordinates B'(C) of the CompEls attached to f_1 ($\alpha > \beta$) thus become:

$$B'(C) = (\alpha - \beta, 2\beta, \gamma), \tag{7}$$

with respect to V_1, V_4 and V_3.

Figure 18: Splitting an edge at its center. Left: CompEls in f before split. Right: Distribution of CompEls between f_1 and f_2 after V_4 splits the edge V_1V_2 at its center.



Source: the authors.

This easily demonstrated to be true. As we know that f is split in half, with the insertion of a vertex V_4 in the edge V_1V_2, we can calculate the coordinates of V_4 with respect to V_1 and V_2 as

$$V_1 + V_2 = 2V_4, \tag{8}$$

we can then update the barycentric coordinates of the CompEl in order to maintain it immovable as

$$C = \alpha V_1 + \beta(2V_4 - V_1) + \gamma V_3 = (\alpha - \beta)V_1 + 2\beta V_4 + \gamma V_3. \tag{9}$$

The barycentric coordinates of the CompEls attached to f_1 ($\alpha > \beta$) thus evolve into:

$$B'(C) = (\alpha', \beta', \gamma') = (\alpha - \beta, 2\beta, \gamma) \tag{10}$$

as stated above, with $(\alpha', \beta', \gamma')$ expressed with respect to V_1, V_4 and V_3, respectively.

In the case where a deformation is decomposed into steps that are sufficiently small with respect to d and the minimum thickness t, the split of an edge using its middle is sufficient to track the deformation with precision d (Stanculescu, Chaine, and Cani, 2011). However, if we want to improve the quality of the mesh produced by the edge splits, we propose choosing another position for the added vertex that better corresponds to the sampling strategy proposed by von Funck, Theisel, and Seidel (2006), revealed in Algorithm 2.

In this case, whenever an edge is split, the new vertex is positioned at the location corresponding to the deformation field applied at the middle of the edge in the space before deformation (line 3). This position may not be equidistant from the two extremities of the deformed edge. We adapt the update of the CompEls so as to take

Algorithm 2 The edge split algorithm.

```
 1: procedure EdgeSplit(mesh, edge)
 2:     faces ← edge.faces
 3:     V4 ← VonFunckTheiselSeidelSplit(edge)
 4:     for each f ∈ faces do
 5:         [V1, V2, V3] ← f.vertices                    // V1 and V2 are vertices of edge.
 6:         f1 ← (V1, V4, V3)
 7:         f2 ← (V4, V2, V3)
 8:         mesh.faces ← (mesh.faces ∪ {f1, f2}) − {f}
 9:         r ← ||V1V4||/||V2V4||
10:         for each compEl ∈ f.compEls do
11:             [α, β, γ] ← compEl.barycentric           // C = αV1 + βV2 + γV3
12:             if α ∗ r > β then                                                  // f1
13:                 compEl.barycentric ← [α − (β/r), β + (β/r), γ]
14:                 f1.compEls ← f1.compEls ∪ {compEl}
15:             else                                                               // f2
16:                 compEl.barycentric ← [α + (α ∗ r), β − (α ∗ r), γ]
17:                 f2.compEls ← f2.compEls ∪ {compEl}
```

these relative distances into account, while still providing a fast scheme. We calculate the ratio between the distances

$$r = ||V_1V_4||/||V_2V_4|| \tag{11}$$

as the new threshold for CompEls placement in f_1 or f_2 (line 12). We, therefore, avoid overly sparse regions in f_1 and overly crowded regions in f_2 by updating CompEl coordinates according to lines 13 and 16. An illustration of the procedure is shown in Figure 19.

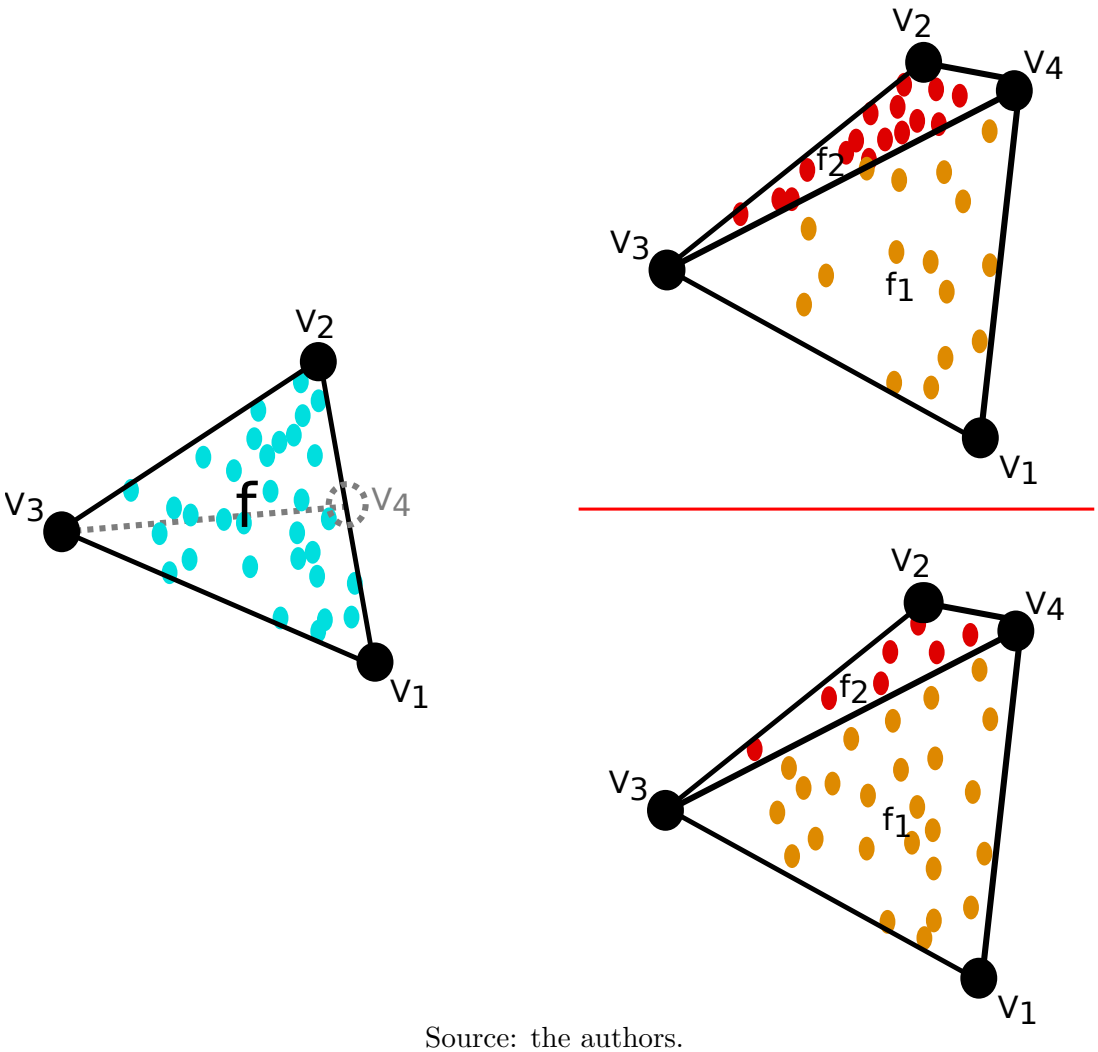
3.3.1.2 Edge collapse

The collapse of an edge affects all the triangles that are incident to either one of its vertices: the two that are incident to the edge disappear and the remaining are distorted.

In the case where the surface is nearly planar, the migration of CompEls could be obtained by superposing the configurations of the triangles before and after the edge collapse, with the initial CompEls being projected on the remaining triangles. This strategy has already been proposed by Hale (1998) for traditional texture update on edge collapse algorithms. It amounts to not moving CompEls during redistribution (see Figure 20a).

The overlay of the original and modified projected connectivities results in a set of convex polygonal cells, whose determination imposes the calculation of a number of points that, in the worst case, is proportional to the square of the number of neighboring edges (Cohen, Manocha, and Olano, 1997). Time requirement to correctly place CompEls during the edge collapses at a given frame, therefore, sharply increases with the number

Figure 19: Modified edge split algorithm. Left: face before split of V_1V_2. Top-right: naïve approach produces overcrowded and overly sparse regions. Bottom-right: modified approach.



Source: the authors.

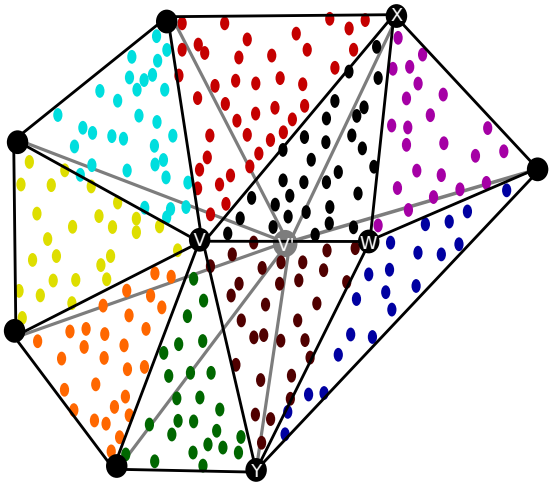
of adjoining triangles and of anchored CompEls. Furthermore, if the surface is not flat, the extension of this approach to the general case requires the determination of a projection plane that is suitable for all the participant triangles, before and after deformation. Especially, it is necessary to avoid the folding of the projected surface. The choice of such a plane is not always obvious or legitimate.

In the following, we propose a simple heuristic that is an order of complexity faster and that does not require explicit projection calculations. Among the faces that are affected by the collapse of an edge VW, the two faces VWX and WVY are destroyed. The other affected faces are maintained, but their vertex incident to the collapsed edge (V or W) is moved to the midpoint V' of VW. We suggest only redistributing CompEls on destroyed faces to the remaining ones. In our heuristic, the surviving faces also keep their own original CompEls, but those are redistributed over a sub-region (of triangular or quadrangular shape), in order to make room for the CompEls of the triangles being

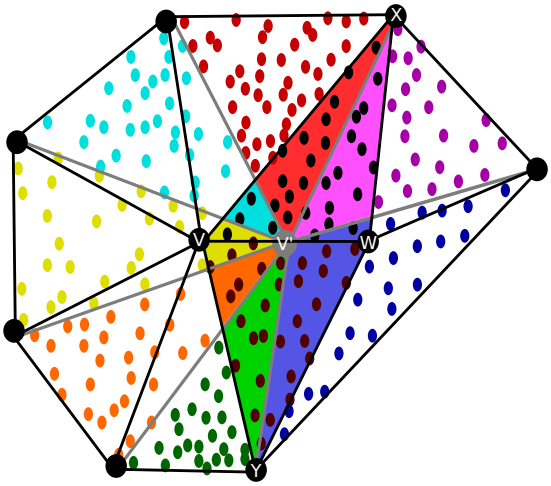
destroyed. Figure 20 reveals the aforementioned operations and visually compares both approaches.

Figure 20: CompEls redistribution on edge collapse. The original and modified connectivities are shown overlaid in black and gray, respectively. CompEls are colored according to their originating faces. Their original positions are the ones in Figure 20a.

(a) The slow but ideal distribution. CompEls' final positions shown match their initial.



(b) Compel distribution by using the proposed heuristic. CompEls are shown in their final position. Colored areas identify the face that will receive the CompEls in that region.

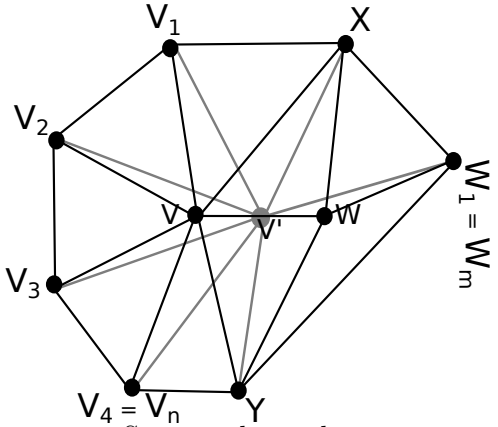


Source: the authors.

3.3.1.2.1 Collapse update

Let $V_1, .., V_n$ denote the vertices incident to the new vertex V' that are located counterclockwise between X and Y, and $W_1, .., W_m$ the vertices that are located counterclockwise between Y and X. We execute our CompEl distribution by visiting the faces around V' in a specified order according to Algorithm 3. Figure 21 illustrates this naming convention.

Figure 21: Vertex naming convention example for the proposed heuristic.



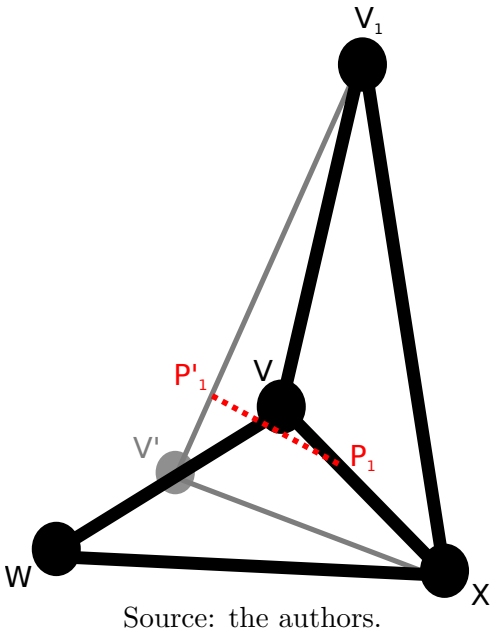
Source: the authors.

Algorithm 3 The edge collapse algorithm.

```
1:  procedure EdgeCollapse(mesh, V, W, f1, f2)
2:      X ← f1.vertices − {V, W}
3:      Y ← f2.vertices − {V, W}
4:      V′ ← MergeAtCenter(V, W)              // 1-ring of V′ is sorted, starting with X.
5:      faces[] ← V′.star                     // faces is sorted according to 1-ring of V′.
6:      P_0 ← X
7:      P′_0 ← X
8:      for i ← 0, faces.count − 1 do
9:          V_i ← V′.1ring[i]
10:         V_{i+1} ← V′.1ring[i + 1]
11:         P_{i+1}, P′_{i+1} ← ClosestCorrespondence(XV, V′V_{i+1})
12:         if (P_{i+1} ∈ XV)and(P′_{i+1} ∈ V′V_{I+1}) then
13:             LiftRegion(faces[i], P_{i+1}, P′_{i+1})
14:             faces[i].compEls ← faces[i].compEls ∪ Inherit(f1, P_i, P′_i, P_{i+1}, P′_{i+1})
15:         else                                  // Stop condition. Begin next loop.
16:             P_{i+0.5} ← V
17:             P′_{i+0.5} ← Project(V, V′VIVI + 1)
18:             V_{i+0.5} ← Intersection(V_iV_{i+1}, V′P′_{i+0.5})
19:             LiftRegion(faces[i], P_{i+0.5}, P′_{i+0.5})
20:             faces[i].compEls ← faces[i].compEls ∪ Inherit(f1, P_i, P′_i, P_{i+0.5}, P′_{i+0.5})
21:             V ← V_{i+0.5}
22:             P_i ← P_{i+0.5}
23:             P′_i ← P′_{i+0.5}
24:             f1 ↔ f2          // Swap f1 and f2. Inherit from the other collapsing face.
25:             X ↔ Y            // Swap X and Y. Calculate P_{i+1}, P′_{i+1} using the other apex.
26:             i ← i − 1                         // Do not increment i
27:         if V_{i+1} = X then                   // Reached apex. Begin next loop.
28:             V ↔ W            // Swap collapsing vertex used to Calculate P_{i+1}, P′_{i+1}.
29:     V ← V′
30:     W ← V′
31:     mesh.faces ← mesh.faces − {f1, f2}
```

In order to determine the CompEls that migrate from VWX to $V'XV_1$, our approach computes the pair of closest points P_1 and P'_1 that are located on the straight lines (VX) and $(V'V_1)$ respectively (line 11 of Algorithm 3, as illustrated in Figure 22).

Figure 22: Example of closest points determination. The dotted line reveals the shortest segment between edges VX and and $V'V_1$. CompEls in V_1VX are lifted to the subregion $V_1P'_1X$ of triangle $V_1V'X$. CompEls in the subregion $P_1V'X$ of the triangle VWX are moved to the subregion $P'_1V'X$ of triangle $V_1V'X$.



Source: the authors.

Closest correspondence: The closest correspondence points P and Q between two parametric lines

$$L_1 = P_0 + u\mathbf{s} \text{ and } L_2 = Q_0 + v\mathbf{t}, \tag{12}$$

with $\mathbf{s}$ and $\mathbf{t}$ their respective parameters, can be easily calculated as

$$P = P_0 + u\frac{(u \bullet v)(v \bullet (P_0 - Q_0)) - (v \bullet v)(u \bullet (P_0 - Q_0))}{(u \bullet u)(v \bullet v) - (u \bullet v)^2} \tag{13}$$

and

$$Q = Q_0 + v\frac{(u \bullet a)(v \bullet (P_0 - Q_0)) - (u \bullet v)(u \bullet (P_0 - Q_0))}{(u \bullet u)(v \bullet v) - (u \bullet v)^2}, \tag{14}$$

so long as L_1 and L_2 are not parallel (Teller, 2006).

If P_1 and P'_1 are located within the interior of the segments VX and $V'V_1$ respectively, these two points can be used to define a kind of local projection of the CompEls. This migration algorithm can be then described as follows for the first face:

1. (line 13) CompEls initially located in VXV_1 are pushed above segment XP'_1 on the resulting face $V'XV_1$. Everything happens as if the point V were pushed towards P'_1, which means that the barycentric coordinates (α, β, γ) of a CompEl C expressed in P'_1XV_1 are the same as its original coordinates in VXV_1. We say that the triangle

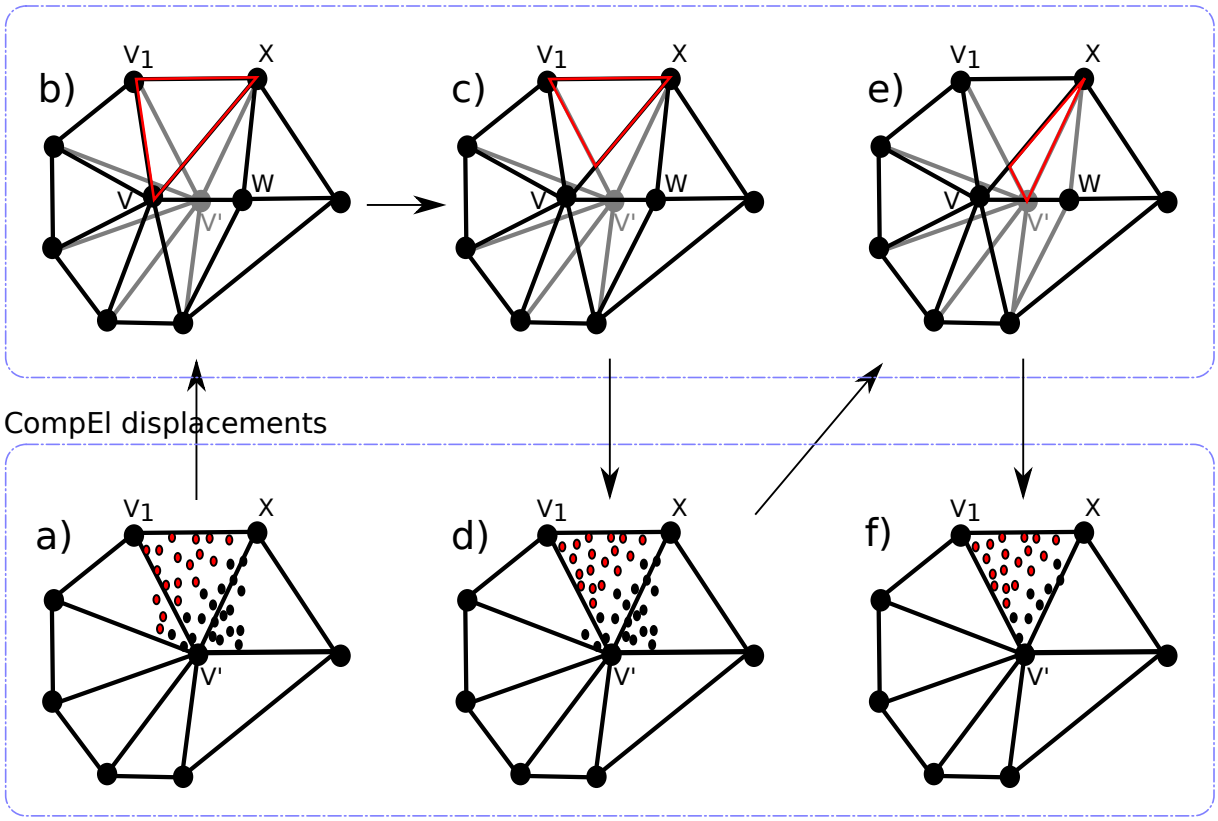
VXV_1 is lifted to P'_1XV_1. Since we know the barycentric coordinates (μ'_1, ν'_1) of P'_1 with respect to V_1 and V', the new coordinates of C in the triangle $V'XV_1$ are:

$$B'(C) = (\alpha * \nu'_1, \beta, \alpha * \mu'_1 + \gamma) \tag{15}$$

2. (line 14) CompEls of VWX that are initially located within $V'XP_1$ are lifted to the triangle $V'XP'_1$ by keeping their barycentric coordinates unchanged within those two triangles. Therefore, their new coordinates within $V'XV_1$ can be updated using the coordinates (μ'_1, ν'_1) of P'_1 and the coordinates (μ_1, ν_1) of P_1 within VX using Equation 15.

Figure 23 visually illustrates both steps.

Figure 23: Region lifting and CompEl inheritance for the triangle V_1VX. a) Original CompEl positions after collapse. b-c) Region correspondence calculation between V_1VX and $V'XV_1$. d) Lifted CompEls from triangle V_1VX. e) CompEl inheritance. CompEls in the highlighted region of VWX will be moved to $V'XV_1$. f) Final CompEl distribution for triangle $V'XV_1$.



Source: the authors.

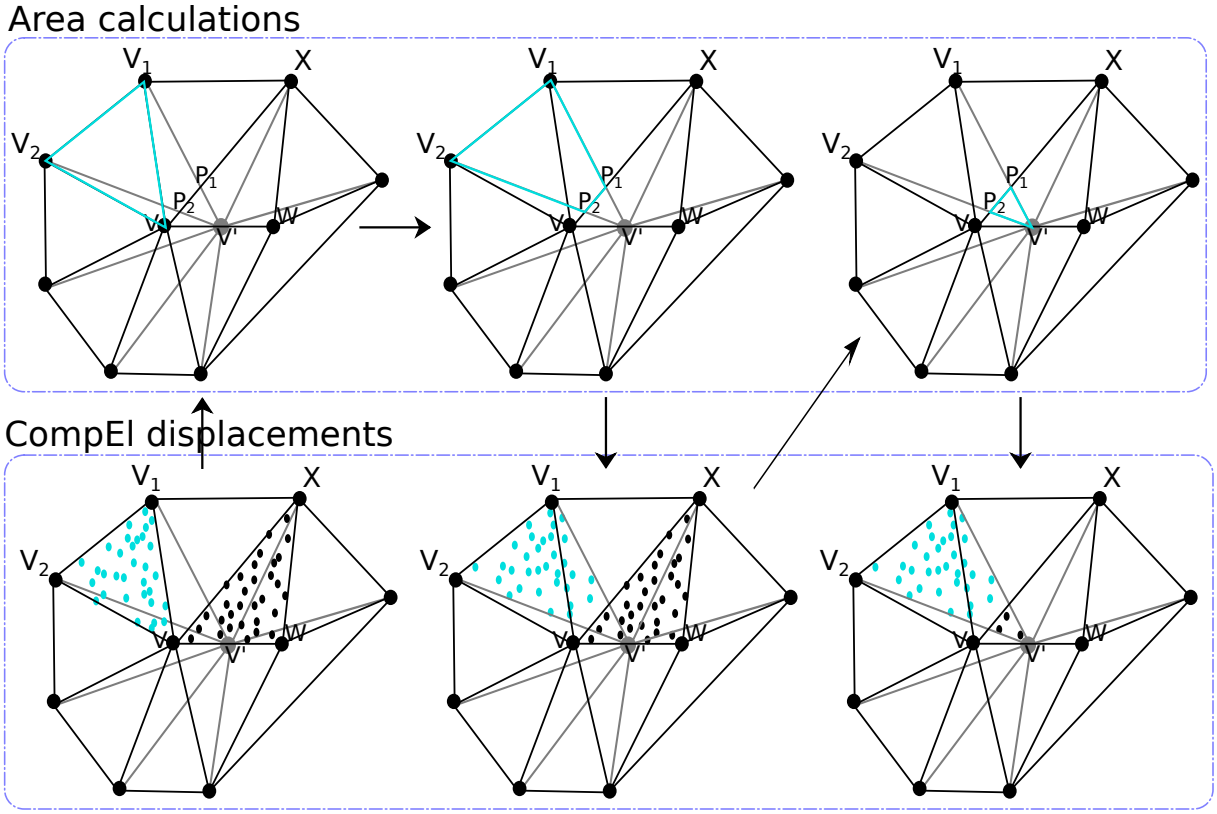
We iteratively carry a similar treatment with CompEls contained in the triangles VV_iV_{i+1} for increasing values of i ranging from 1 to a stop condition (line 15) defined below. In order to determine which of the triangle VWX's CompEls are reassigned to the triangle $V'V_iV_{i+1}$, we compute the pair of closest points P_{i+1} and P'_{i+1} that are located on

the straight lines (VX) and $(V'V_{i+1})$ respectively (line 11). If P_{i+1} and P'_{i+1} are located in the interior of the segments VX and $V'V_{i+1}$ (line 13), the following migrations are performed:

1. (line 13) CompEls initially located in VV_iV_{i+1} are lifted above the segment $P'_iP'_{i+1}$ on the resulting face $V'V_iV_{i+1}$. This amounts to lifting the initial triangle VV_iV_{i+1} into the quadrangle $P'_iV_iV_{i+1}P'_{i+1}$. Let C be a CompEl (α, β, γ) in VV_iV_{i+1} and let C' be the position corresponding to the same coordinates (α, β, γ) within $V'V_iV_{i+1}$. The target position C" of the CompEl C will be placed along the line $(V'C')$: one determines the intersection $P'_{(i,C)}$ (resp. $V'_{(i,C)}$) of this line with $P'_iP'_{i+1}$ (resp. with V_iV_{i+1}) and we consider the linear mapping of $V'P'_{(i,C)}$ into $V'V_{(i,C)}$. C" is obtained as the image of C' by this linear mapping. This defines the lifting of C into C".
2. (line 14) CompEls of VWX that are initially located within $V'P_iP_{i+1}$ are lifted to the triangle $V'P'_iP'_{i+1}$ by keeping their barycentric coordinates unchanged within these two triangles using Equation 15.

Those rules are essentially a generalization of the region lifting shown in Figure 23, which becomes a particular case for when a P' coincides with an apex. Figure 24 shows an example for this general case.

Figure 24: Quad region lifting and CompEl inheritance for the triangle V_2VV_1.



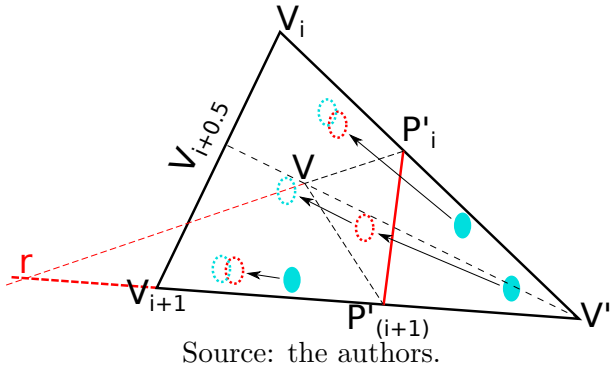
Source: the authors.

Stop condition: The index i is incremented until we reach the value I such that P_{i+1} and P'_{i+1} are not located in the interior of the segments VX and $V'V_{I+1}$ (stop condition).

In that case, V is projected onto $P'_{I+0.5}$ within the triangle $V'V_IV_{I+1}$. The CompEls of VWX initially included in $V'P_IV$ are lifted to the triangle $V'P'_IP'_{I+0.5}$. It is also necessary to compute a virtual vertex $V_{I+0.5}$ as the intersection of $V'P'_{I+0.5}$ with V_IV_{I+1}. CompEls located inside the triangle $VV_IV_{I+0.5}$, are lifted within the quadrangle $P'_{I+0.5}P'_IV_IV_{I+0.5}$. When the stop condition is met, we do not move to the next triangle and the index i is not incremented.

In the same way, if P_1 and P'_1 do not belong to the segments VX and $V'V_1$ respectively, we project V on the triangle $V'XV_1$ at point P' (and compute $V_{0.5}$ as described above). The sub-triangle $V'XV$ (within VWX) is then lifted to $V'XP'$ (within $V'XV_1$). CompEls located inside the triangle $VXV_{0.5}$ are lifted to the triangle $P'XV_{0.5}$, and no iteration is performed on i. This final case can be better seen in Figure 25.

Figure 25: Region lifting under stop condition. It is treated as two quad region liftings using $V_{i+0.5}$ as the divisor. The dashed lines in red show the stop condition, where the calculated P'_{i+1} would fall in point r, outside the edge $V'V_{i+1}$. The dotted CompEls in red show the CompEl placement if the lifting took place using the quadrangle $P'_iV_iV_{i+1}P'_{i+1}$. Dotted CompEls in cyan show their final position by using the virtual vertex $V_{i+0.5}$.



Source: the authors.

The same process is performed starting from the edge VY in the direction of V_n, from WX in the direction of W_m and from the edge WY in the direction of W_1. If any triangle incident to vertex V' is not reached by these four iterations, it keeps its CompEls unchanged.

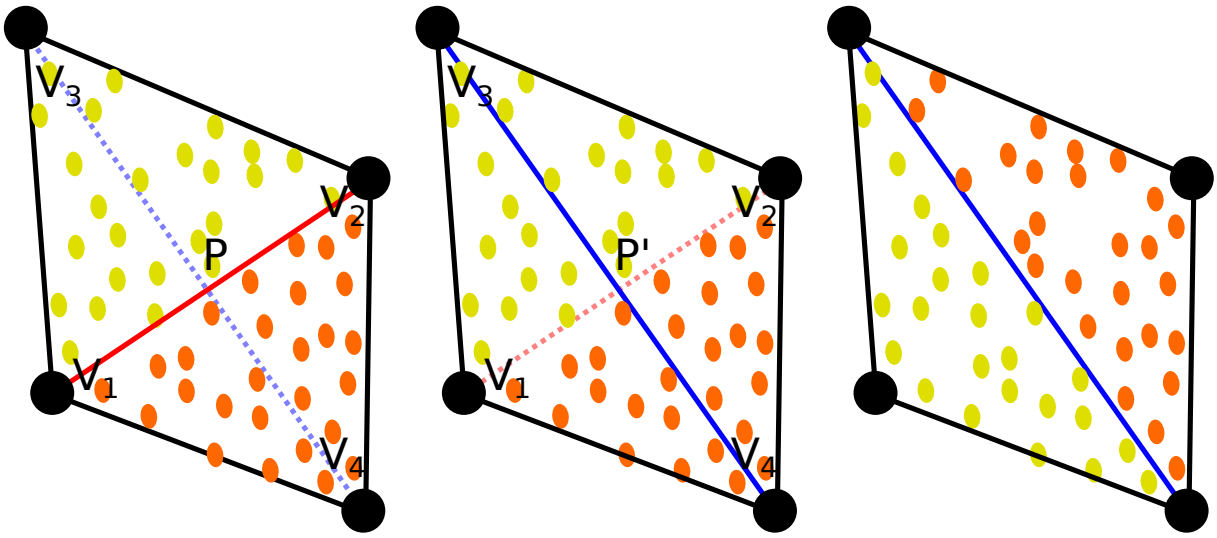
3.3.1.3 Edge flip

In nearly planar areas, edge flip is a preferred alternative to edge split and edge collapse, whenever it is sufficient to restore the level of detail d or the quality of a triangle. This favors a better stability of the quasi-uniform sampling since it prevents creation and destruction of vertices.

Using the notation from Figure 26, whenever an edge V_1V_2 is replaced by its flipped counterpart V_3V_4, one first calculates the pair of points P and P' on the edges V_1V_2 and V_3V_4 that minimize the distance between the lines supported by these edges. Compel barycentric coordinates are first updated as if V_1V_2 were to be split in P. CompEls of the triangle PV_3V_1 (resp. PV_1V_4) are then lifted to $P'V_3V_1$ (resp. $P'V_1V_4$) using Equation 15.

Since P' is on V_3V_4, we perform the inverse of a split barycentric coordinate update to merge $P'V_3V_1$ and $P'V_1V_4$ into $V_4V_3V_1$. The same procedure is performed on the other side of V_4V_3.

Figure 26: Edge flip. Left: edge V_1V_2 shows P. Center: flipped edge revealing P'. Right: final result with CompEls recolored according to their new faces.



Source: the authors.

3.3.1.4 Change in topological genus

Whenever the distance between two vertices located in different sides of the quasi-uniform mesh becomes smaller than the minimum thickness t, the shape locally flattens and breaks. This amounts to anticipating intersections and changes in topological genus. This change is performed through vertex ring join of the two vertices (Stanculescu, Chaine, and Cani, 2011). It can cause major topological and visual changes in the mesh. In order to maintain a minimum temporal continuity, we propose to reuse the faces of the star of the joining vertices. The faces and their corresponding CompEls are maintained by merely replacing the joining vertex with one in the other's 1-ring. This effectively stitches both 1-rings together, as depicted in Figure 27. No barycentric coordinate update is performed.

3.3.1.5 Finishing touches

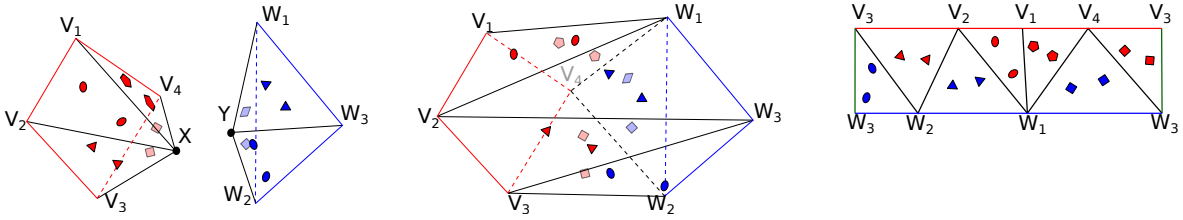
Following the guidelines set in Section 3.2, the proportion of CompEls per unit of area is maintained after each operation. We create new elements in the deeper layers of any triangle with a shortage in the number of CompEls. We also promote CompEls from those layers to upper ones, if necessary. Similarly, CompEls can be pushed to deeper layers or be destroyed whenever a triangle is too crowded. Both creation and destruction are performed behind the elements on the surface, while the promotion and demotion are

Figure 27: CompEl redistribution on ring join operations.

(a) The original neighborhoods of two vertices before ring join.

(b) The two vertices are removed and the two neighborhoods are stitched together.

(c) Resulting connectivity unwind at the edge V_3W_3 (shown in green, duplicated)



Source: the authors.

carried over a period of time. With this maintenance strategy, visual artifacts related to the sudden insertion or removal of CompEls can be minimized or overcome entirely.

3.4 Rendering

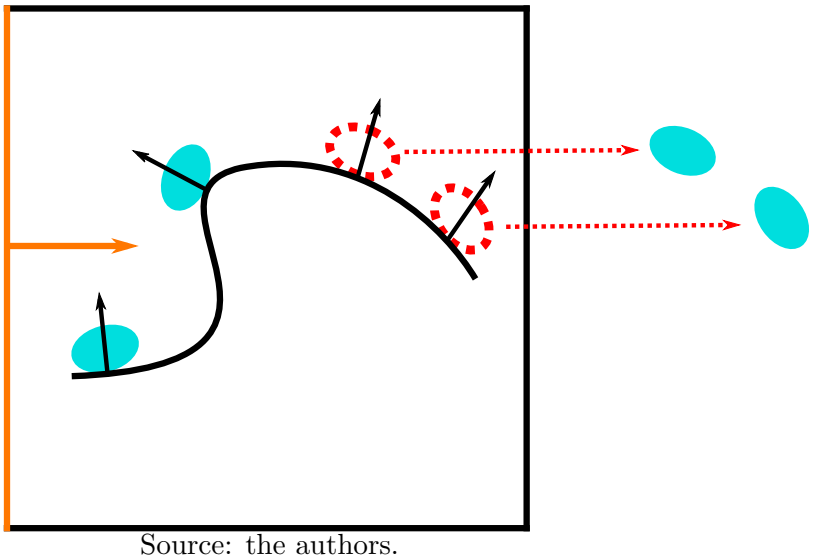
Storing and rendering, in real time, a large number of CompEls can be a considerable technical challenge. A single model can have tens of thousands of CompEl instances, which can hinder performances due to a sheer triangle count. Taking advantage of the capabilities of the graphics card can be useful for that challenge. We also use the programable rendering pipeline to lower visual problems related to CompEls intersecting one another, which is a problem not commonly present in traditional 2D texture bombing. Finally, we will explain how our system is compatible with the introduction of variability in the models of CompEls being used without loosing the benefits of instancing. In this section, we outline the algorithms employed to improve rendering performance, reduce inter-object intersection and promote variability.

3.4.1 "Back CompEl" Culling

Backface culling and clipping algorithms are common place in modern graphics cards. CompEl meshes, after vertex processing in the rendering pipeline, may have already been automatically clipped out. However, if they have not been clipped, usually, there will be triangles on their surface facing the camera (always, if they are closed manifold inside the viewing frustum). In crowded composite objects, this may result in several hidden CompEls being passed on to primitive assembly and fragment processing.

We can alleviate the problem by discarding all the CompEls that face away from the camera. At the level of each vertex (in Normalized Device Coordinates (NDC)), we compare the quasi-uniform mesh normal closest to the CompEl's barycenter and the eye vector. If the CompEl is deemed backfacing, we displace the vertex outside the viewing frustum, resulting in the entirety of the CompEl being clipped. Figure 28 illustrates this technique.

Figure 28: Back CompEl culling. CompEls are shown distributed along a surface in NDC. The screen and the viewing vector are shown on the left. CompEls placed where the surface normal forms an angle smaller than 90° with the viewing vector, shown dotted in red, are sent outside the NDC to be clipped.



Source: the authors.

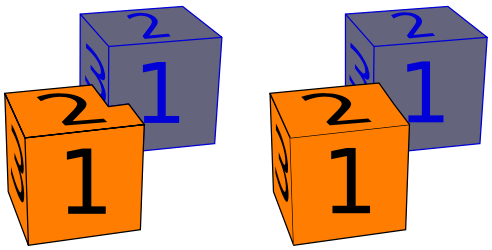
3.4.2 Visual intersection prevention

Intersections in a scene are a constant concern in computer graphics (Gain and Dodgson, 2001). Applications that work with 3D objects clumped together usually solve this problem by making use of physical simulations, polygonal collision tests or other indirect methods, such as bounding boxes, to prevent pairs of rigid objects from intersecting each other. Collisions may not only be costly to identify, but also troublesome to solve when more than two objects are involved. Our framework aims at being fully local in order to better support interactive sculpting times, so we cannot resort to costly collision detection calculations.

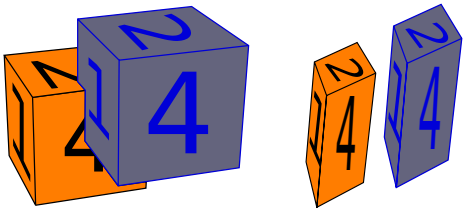
To tackle the issue, we propose a CompEl squish in the rendering pipeline. After the transformation to Normalized Device Coordinates, we add a displacement to CompEl vertices perpendicular to the canvas. We move the vertex so that its depth approaches the same depth as its CompEl's projected center of mass. As per Gain and Dodgson (2001), this deformation can be represented as a simple linear scale in a cubic set of hyperpatches around the CompEl. This squish operator cannot cause any new self-intersections in a mesh, as long as the scale factor s is less than 100%. Also, since the displacement is perpendicular to the canvas, only the intersection of objects with different centers of mass is altered. The rendering of individual CompEls is not affected. In the limit case, when $s = 100\%^-$ (as close as possible to, but less then 100%), all CompEls become equivalent to parallel 2D images, which cannot intersect (Figure 29). Though arguably ideal for static viewports, applying the squish operator with $s = 100\%^-$ may generate animation artifacts.

Figure 29: Particle Squish.

(a) View from the camera before (left) and after (right) squish.



(b) A hypothetical side view before (left) and after (right) squish reveals that the two intersecting CompEls had their vertices moved towards the depth of their respective centers of mass in NDC.



Source: the authors.

3.4.3 Variability

CompEl models are displayed using traditional instance-based rendering. This means that we use a single CompEl model, and that the graphics card performs the numerous mesh instantiations of this model. This alleviates memory requirements in the graphics card and allows us to render a large number of objects in real time. Variability can be introduced directly during instancing. Given a very small number of input models, we are able to synthesize a possibly large number of different CompEls. We control variations between CompEls originated from the same model by means of a randomization seed, assigned to each element, that is used by shader operations.

We increase variability by adding low frequency noise to the positions of CompEl vertices. A low frequency noise can be efficiently generated in the graphics card by a number of algorithms. We opted for a Perlin Noise-like function, as defined by Gustavson (2005). Noise intensity is evaluated using vertex normals offset by the randomization seed. The resulting noise strength is applied as a vertex displacement along its normal direction.

Noise can further be used to randomize the orientation of each CompEl and the initial frame of its animation, if applicable.

Results of the use of this technique can be seen in Section 4.5.

3.5 Final considerations

In this chapter, we presented our CompEl-based framework. We began by outlining the requirements and special challenges we took in consideration when modeling our system. Next, we explained how we used the work of Stanculescu, Chaine, and Cani (2011) as a support to our own. Then, we explained the nature of our CompEls, how they are distributed and how they relate to the support shape. We then explained how we are able to minimize CompEl destruction and creation under the various connectivity

updates. Finally, we discussed additional strategies we employ to minimize rendering overhead and to promote variability.

The next chapter is devoted to the analysis of these strategies and to their comparison with alternative methods.

4 RESULTS

In this chapter, we present and discuss the results obtained with the proposed model. Over the following sections, we evaluate our technique based on the criteria listed below:

- **Completeness and distribution quality.** Our local sampling strategies must maintain, throughout the mesh, the required CompEl density requested by the user. If this density, coupled with CompEl size, is sufficient to completely cover the surface, we must show, in addition, that our algorithm can cover the entire surface, without empty spaces. The quality of the CompEl distribution is also important.
- **Performance and real-time compliance.** Our application needs to allow users to sculpt with several thousand CompEls in real time in order for our method to be useful in real world applications.
- **Temporal continuity.** We must show that we are able to minimize the removal and insertion of CompEls in the face of connectivity updates. This is necessary in order for our method to be superior to the simple repopulation of removed and inserted faces. Additionally, CompEl insertion and removal must be made as smooth as possible to avoid popping effects when they come in and out of existence.
- **Intersection prevention.** Ideally, CompEls should not intersect each other upon rendering.
- **Variability.** Our CompEl framework should maximize the number of CompEl variations it can create from a single mesh.

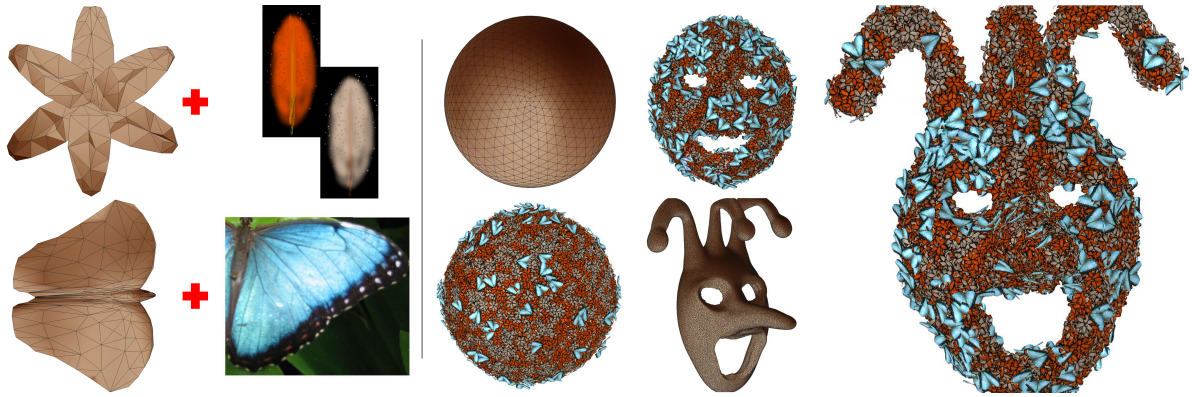
Following the analysis of our system in these dimension, we proceed to compare it with several competing methods with our final remarks in Section 4.6.

4.1 Completeness and distribution quality

Completeness can be seen as the absence of empty regions on the surface of the mesh. This is important when the user wants to convey the idea of an object full of composing elements, not only on the surface. This should usually be true in most regular uses of our software, as having many faces with no CompEl would mean the user can displace elements in empty regions of the model. It is not, however, mandatory. Low e_is can, in fact, generate CompEls disconnected from each other, which can produce interesting effects. Figure 30 reveals an example produced by our technique, with and without completeness.

Another major concern is with the distribution quality. What configures a good distribution, however, can vary for each user. A common expectation involves randomness and uniformity of the distributed elements, without noticeable patterns. With that in mind, a good target distribution is the Poission-disk sampling (Ying, Li, and He, 2014).

Figure 30: Carnival mask sculpted using our framework. Left: input CompEls and accompanying textures. Center: initial sphere mesh and intermediate steps of the sculpting process, with and without CompEls on the outer surface. Right: final rendering. Sculpt time: 4 minutes.



Source: the authors.

In our system, the user implicitly signals his wish for completeness by establishing a big CompEl frequency through the e_i variable. e_i must be big enough so that a good distribution of CompEls covers the entirety of the surface (Equation 4). Since our sculpting framework greatly favors good quality triangles, our local distribution algorithm tends to display at least jittered sampling quality levels, which will also result in completeness at no additional cost. This ensures a nice distribution with the desired CompEl density throughout the sculpting process.

4.2 Performance

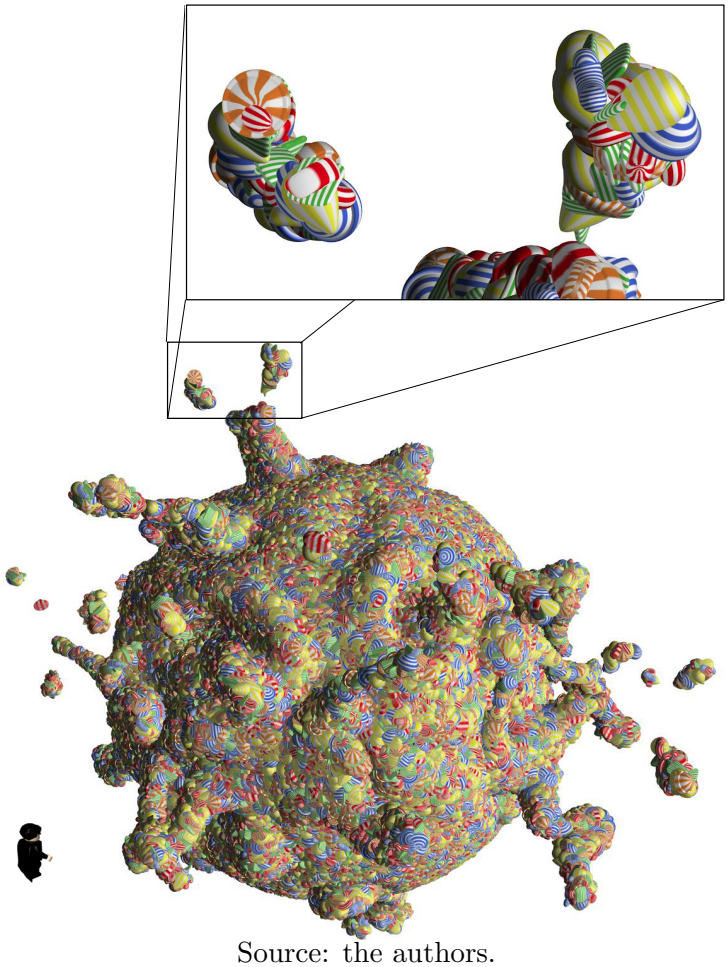
Simulations were performed on a MacBook with 2.2 GHz Intel Core i7 processor, 8.0 GB of RAM, AMD Radeon HD 6750M 1024 MB graphics card. We used C/C++ and OpenGL 3.3 for graphics.

Unlike procedurally generated models, where users can be tolerant to offline processing, real time compliance can be a red flag for sculpting applications. Artists expect to see and evaluate changes to their models as they sculpt before deciding the next batch. When sculpting agglomerate materials with our framework, a displacement induced on the surface produces a corresponding movement of anchored CompEls. Since each CompEl is a mesh object in its own right, this means the user will effectively manipulate massive amounts of geometric information. That information must be readily available to the graphics card in quantities it is able to process in interactive rates. Moreover, polygon intersections can increase the number of overdraws, which negatively impact performance and can sharply decrease fps. Such intersections are common in applications with several objects displayed in a tight space, such as ours.

In spite of said limitations, our tests indicate that our frames per second are on

par with the rendering of an intersection-free mesh with an equivalent number of polygons, as can be seen in Figure 32 with respect to Figure 31.

Figure 31: “My World”. Example sculpture by a 6-year-old from candy meshes, populated as to show the number of CompEls that can be drawn at interactive rates. A zoomed view shows individual CompEls. The artist is shown on the left, looking at his creation. Sculpt time: 5 minutes. Total number of CompEls: 91.456. Face count per CompEl: 150.

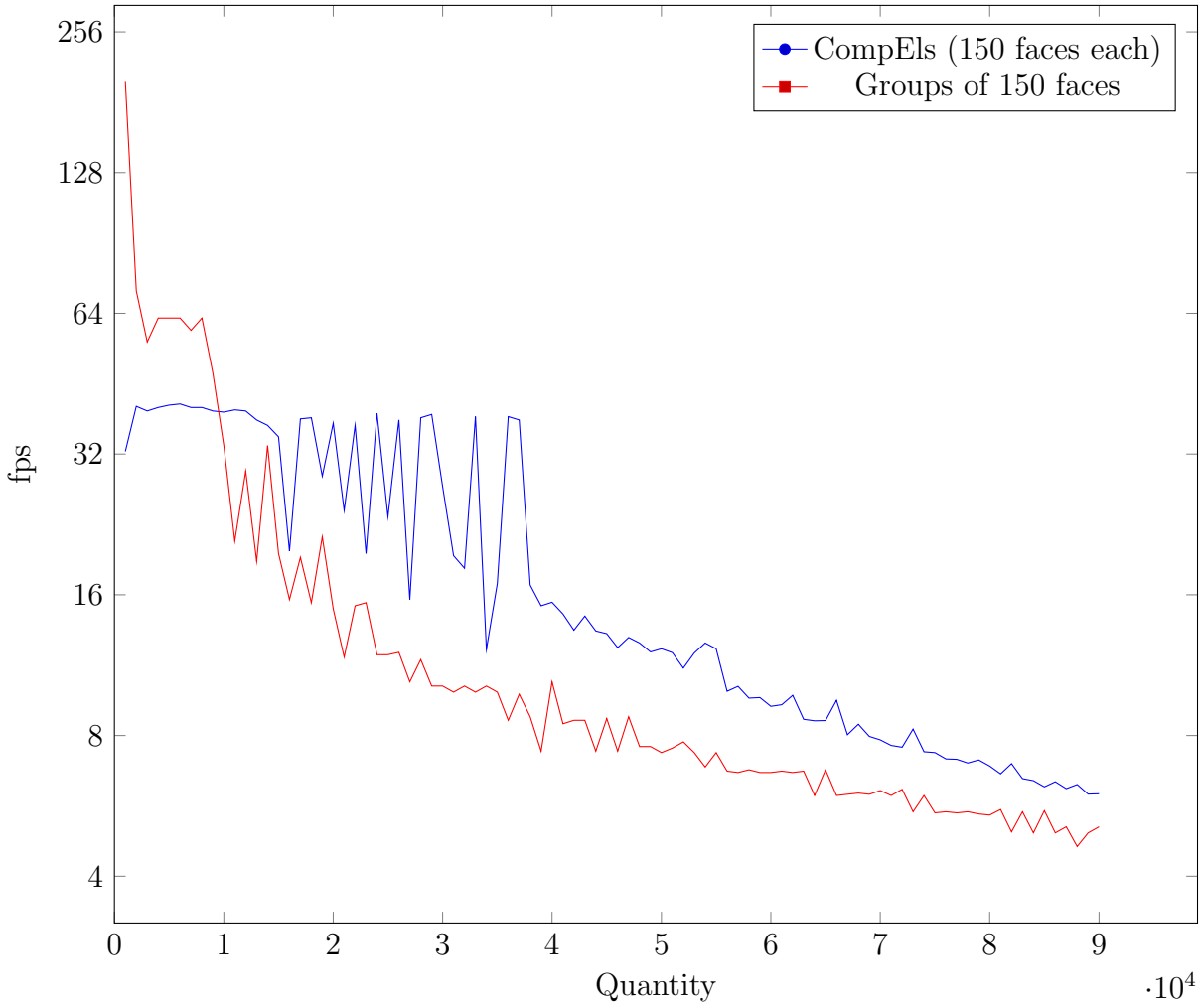


Source: the authors.

This is due to the heavy use of instance-based rendering and of our intersection prevention and back CompEl culling algorithms, which greatly reduces the number of CompEls triangles sent to fragment processing in case of complete composite materials, as can be seen in Figure 33. Rendering the base mesh, subdivided into a similar number of polygons, results in equal or slightly slower rendering times, because less polygons are rejected by the z-buffer algorithm.

One must note, however, that performance can be significantly diminished if CompEl meshes are dense. Similarly to other rendering algorithms, our fps can also vary with the number of CompEls displayed in a single frame.

Figure 32: Performance of our CompEl rendering algorithm with respect to Figure 31. Tests executed in a 800x600 resolution. The large bumps in CompEl rendering performance in the 15.000 to 35.000 range is due to occlusion, that is, a few CompEls inserted that are randomly rendered before the rest, causing the others to not generate any fragments.



Source: the authors.

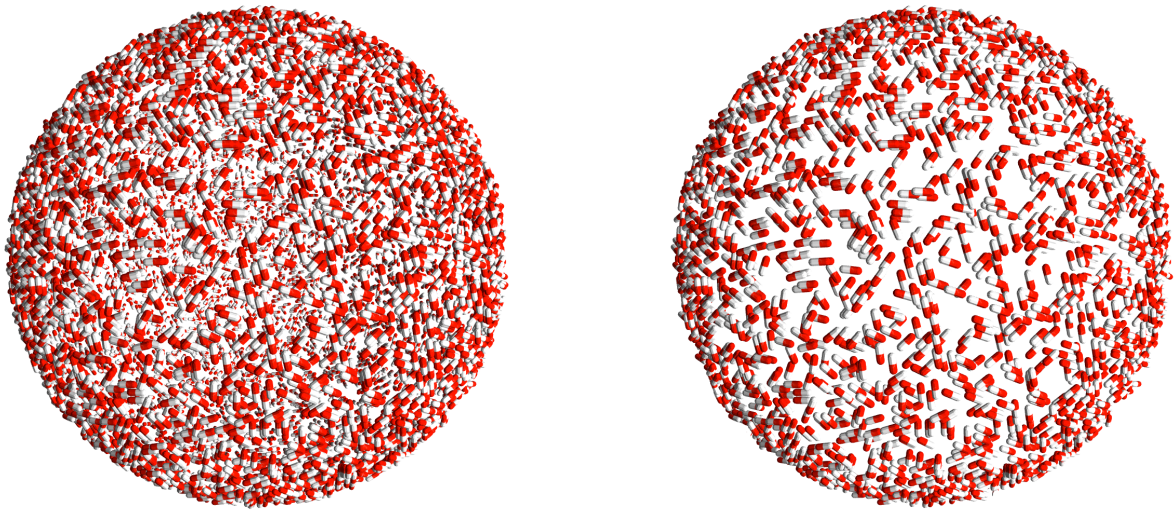
4.3 Temporal continuity

Temporal continuity was of paramount concern throughout our research process. Our CompEl redistribution algorithm greatly limits the number of CompEl insertions and deletions on all basic topological operations performed on the anchor mesh, as can be seen in Table 1, with respect to Figure 34. In our tests, the number of CompEls created and removed during deformation was reduced by more than 70% when compared to simply inserting new CompEls on faces created by edge splits and deleting CompEls on faces destroyed with edge collapses. Their numbers oscillate according to the desired CompEl frequency and the evolution of the surface's area, but are otherwise largely independent from the insertion or removal of faces and vertices.

Figure 33: Back CompEl culling.

(a) Model rendered without culling show several CompEls that would be invisible on complete materials.

(b) Model rendered with culling removes a considerable amount of geometry from fragment generation.



Source: the authors.

Additionally, visual continuity is ensured by the seamless insertion and removal of CompEls on lower layers of the surface. Should there be too few elements to completely cover the surface, CompEls are inserted and removed behind a preexisting elements on an adjoining face. Newly inserted elements sprout seamlessly from inside the sculpted object. This makes it seem that the CompEl was only initially hidden instead of recently inserted. Similarly, excess CompEls are first pushed from sight towards the interior of the shape or behind another element before disappearing.

4.4 Intersection prevention

Intersection between solid composing elements is a common artifact that can break the illusion of a real composite material. In our system, the only way in which we limit intersections is through optical deformations by squishing CompEls along the *eye* vector. Figure 35 shows that this simple strategy is able to completely remove intersections from a rendering perspective.

Though arguably ideal for static viewports, applying the squish operator with $s = 100\%^{-}$ (i.e.: as close as possible to but still less than 100%) may generate animation artifacts. As the camera rotates, CompEls that were behind other CompEls may suddenly pop to the front. This is a direct consequence of a CompEl's center of mass becoming closer to the canvas than another's. It is also theoretically possible, though not observed in our tests, to experience z-fighting between squished CompEls. Both of these problems can be minimized by allowing some small intersection through small reductions of s. This

Figure 34: Example sculpture created from a sphere. Left: pills used as CompEls show their relative frequency and texture distribution. Right: the final result. Sculpture time: 12 min. e = 797. For this example, a single layer is used.

Source: the authors.

causes the transition to spread through multiple frames. It greatly reduces the visual artifacts while still allowing for very little intersection.

4.5 Variability

Frequently, the user will want to use more than a single "token" object to construct its agglomerate material. Though the user can include and use multiple separate meshes when sculpting, they are likely to face memory limitations before conveying every variation. Procedural variability is, thus, a very important concept to address.

Our CompEl models are displayed using traditional instance-based rendering. This means that we use a single CompEl model, and that the graphics card performs the numerous mesh instantiations of this model. This alleviates memory requirements in the graphics card and allows us to render a large number of objects in real time. Variability can be introduced directly from the benefits of instancing. Given a very small number of input models, we are able to synthesize a possibly large number of different CompEls.

The first source of diversification are textures. Increasing the number of different textures applied to CompEls can increase variability without increasing the number of models that need to be stored in the graphic card. Different textures are stored and made available in an array. The one used can be picked on the fly using the randomization seed. Figure 36 shows the result of texture variation on a single mesh.

We can promote variability by adding randomized deformations and orientations per instance. This can cause each object to look unique and the scene more natural,

Table 2: CompEl creation and destruction for the sculpture in Figure 34. The number of CompEls is shown as a function of the required density e.

	Quantity			
Variable	Initial	Created	Removed	Final
Edge Splits				41263
Edge Flips				13573
Edge Collapses				30122
Vertices Ring join				22
Area	12.52	N/A	N/A	8.78
Vertices	1026	41270	30197	12099
Faces	2048	82526	60384	24190
CompEls				
$e = 3986$	48882	27469	41844	34507
$e = 797$	8911	66	3855	5122
$e = 341$	3328	14	1128	2214

Source: the authors.

while still maintaining the likeness of the base model. Figure 37 shows how adding different orientations and per-vertex displacement noise to texture variation can create different pebbles.

Finally, our system also supports animated models. Those can be useful should the user wish to create "live" scenes, such as the butterflies on a flower sculpture (such as the one in Figure 30). Should animation be present on a given CompEl information, each instance spawns with a randomized starting frame.

Combining several of these differentiation options cause each CompEl to look alive and unique, and not a mere replica of its peers.

4.6 Final considerations

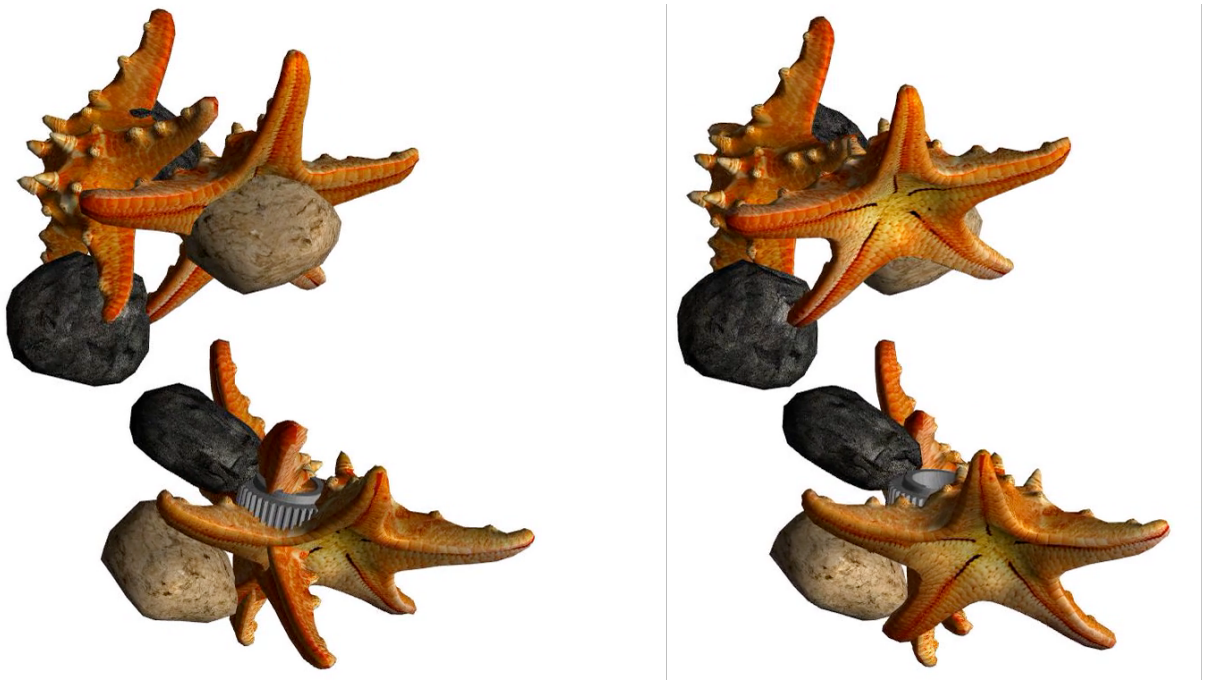
The results we present in this section reveal an efficient strategy to the virtual sculpture of composite objects. Despite not conveying as good an element distribution as previous strategies, we are the first to show to be able to sculpt composites on grand scales with temporal continuity, in real time with adequate distributions. We consider this strategy not as a direct competitor to state of the art composite extraction algorithms, but as the means of producing previews in real time, during the actual sculpting session. This preview can better guide the artist towards the intended shape, when more involved algorithms can be employed to produce the final render.

The next chapter concludes this work with an overall discussion of our contributions, the limitations of our approach and the intended future work.

Figure 35: Intersection prevention between solid elements using only optical deformations.

(a) Undeformed assembly of CompEls reveal intersections between elements.

(b) A squish operator with $s = 100\%^{-}$ completely removes intersections between CompEls.

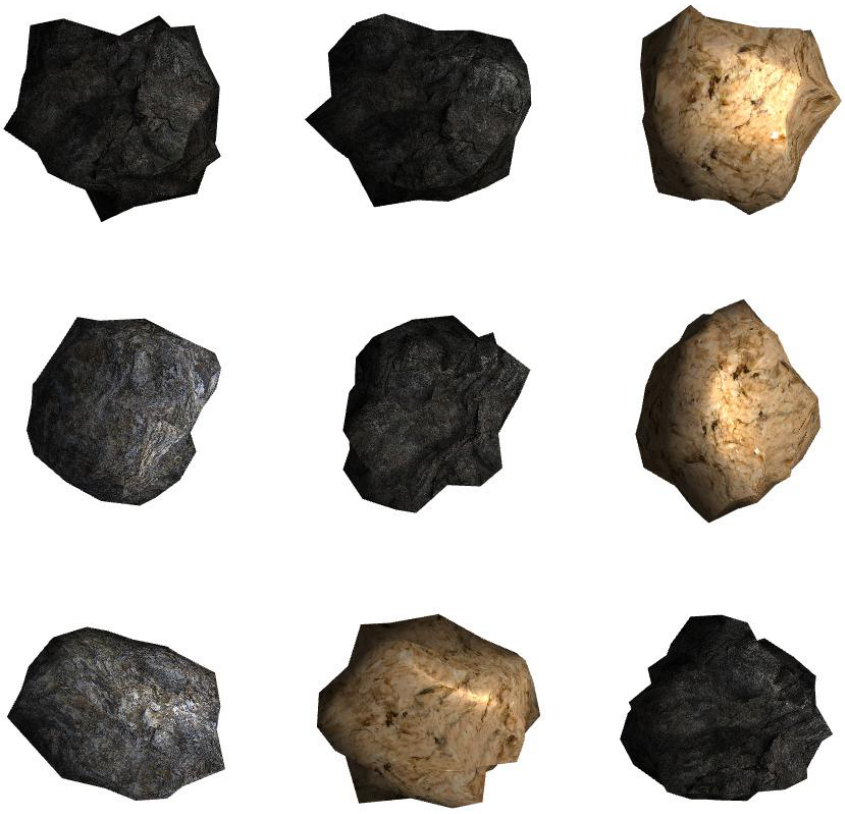


Source: the authors.

Figure 36: The same rock mesh can be used with different textures to generate different CompEls.

Source: the authors.

Figure 37: Adding variation by per-vertex noise and per-instance orientation to a single rock mesh.



Source: the authors.

5 CONCLUSION

In this thesis, we presented a new sculpting framework that maintains high quality and temporally coherent sampling of agglomerate materials. Our algorithms minimize CompEl displacement, insertion and removal, while still complying with the deformation requested by the user. Rather than focusing on costly physical interaction between CompEls, we chose to anchor our elements to the surface. Elements, thus, move about each other only as the space around them warps.

Our sculpting framework is build on top of a system that greatly favors a uniform vertex and face distribution on the support shape. That property favors a local CompEl sampling algorithm, while still producing the illusion of randomness and that of a full volume. This allows for faster computations in the CPU when compared to physical simulations or global samplings. We further reduce computations by keeping elements only on the outermost layers of the shape being sculpted. Object insertion and removal is performed in the innermost populated layers and behind other CompEls, which eliminates any popping effects from such operations. Our results show that this method allows for the deformation of agglomerate materials with tenths of thousands of elements in real time with relatively high quality.

Our method also produces rendering optimizations in the graphics pipeline. We can reduce object intersections with visual deformations. This technique displays elements as if they were entirely separate and non-overlapping with negligible computation overhead. We further use the underlying surface's normal at the CompEl's position to determine if it is front-facing or back-facing. If it is back-facing we remove it from the rendering pipeline before fragment generation, saving GPU computation time. Finally, we use the graphics card to produce element randomization, reducing the uniformity of elements.

We believe that a real-time, approximate solution to the virtual creation of composite shapes interests both industry and academia. Our framework allows users to experience the sensation of an agglomerate throughout the sculpting session, which can aid the creative process. Later, if necessary, a more involved technique can be employed to export the final shape or sequence. This strategy is similar to using z-buffer rendering for keyframe posing when defining animations, saving the costly ray-tracing for the final rendering.

5.1 Limitations

Despite solid results, interactive virtual sculpting of composite shapes remains an open field. We proceed to point out some of the limitations of our approach. Firstly, our CompEls are rendered as meshes. Having too great a number of elements with too

many faces can reduce the number of frames per second. Our rendering optimizations can alleviate the problem to an extent, but the limitation is still present. Secondly, our per instance GPU deformation, used to reduce object intersection, can cause popping effects when the user rotates the camera. This is due to the deformation performed being necessarily view-dependent. Reducing the deformation intensity can reduce the popping in exchange for *some* intersection being present. Furthermore, our approach only removes intersections from a rendering perspective. Based on actual element placement, CompEl intersection still occurs. Finally, our sampling method is entirely stochastic and cannot provide hard guarantees against over or under sampled regions. The problem can be minimized in two ways: first, by using similar vertex and CompEl densities, which allows our framework to piggyback the good sampling and element distribution of the underlying sculpting framework; second, by using approximatively round CompEls with diameter similar to that of d, which promotes good face coverage regardless of element orientation.

5.2 Future work

Future work focuses on solving current limitations and on finding new uses for our CompEl framework. Regarding the former, layered rendering algorithms can be employed to increase object count. Closer CompEls should be rendered in full detail, while farther elements can be efficiently displayed with simpler techniques. Here, simplified meshes or impostors could be employed. Moving even farther, surface textures on the support shape could be used to give the impression of agglomerates whilst saving render times. Some form of view-dependent, GPU-only element sorting could also take place. Explicit information of incremental camera rotations could greatly reduce the number of possible swaps. Finding an efficient sorting algorithm could reduce overdraw and CPU / GPU band limitations. Storing and modifying data directly in the GPU would be even better of a improvement in this direction. The system could also benefit from some form of limited physical simulation parallel to the sculpting process and during idle moments. This could take the form of a spring system that favors good CompEl distribution.

Our CompEl framework can be used to efficiently track any kind of information that can be placed on the surface of a mesh and that is subject to tangential displacement along it's surface, such as handles or touch locations. Additional uses could include tracking touches on multi-user environments. This could allow a user to fix errors or to remove entire operation sequences with minimal effect to his collaborators' work.

REFERENCES

Abdrashitov, Rinat; Guy, Emilie; Yao, JiaXian; Singh, Karan. Mosaic: Sketch-based Interface for Creating Digital Decorative Mosaics. *Proceedings of the 4th Joint Symposium on Computational Aesthetics, Non-Photorealistic Animation and Rendering, and Sketch-Based Interfaces and Modeling*. New York, NY, USA: ACM, 2014, SBIM '14, p. 5–10. URL http://doi.acm.org/10.1145/2630407.2630409.

Aghdaii, Nima; Younesy, Hamid; Zhang, Hao. 5-6-7 Meshes. *Proceedings of Graphics Interface 2012*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2012, GI '12, p. 27–34. URL http://dl.acm.org/citation.cfm?id=2305276.2305282.

Angelidis, Alexis; Cani, Marie-Paule; Wyvill, Geoff; King, Scott. Swirling-Sweepers: Constant Volume Modeling. *Graph. Models*, v. 68, n. 4, p. 324–332, 2006. Special Issue: Pacific Graphics 2004.

Blender Foundation. Blender. 2017. URL http://www.blender.org.

Bojsen-Hansen, Morten; Li, Hao; Wojtan, Chris. Tracking Surfaces with Evolving Topology. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, v. 31, n. 4, p. 53:1–53:10, 2012.

Boyton, Kristen. The Magic of Gaudi – Barcelona, Spain. 2017. URL http://ourwanderland.com/magic-gaudi-barcelona-spain/.

Cohen, Jonathan; Manocha, Dinesh; Olano, Marc. Simplifying Polygonal Models Using Successive Mappings. *Proceedings of the 8th Conference on Visualization '97*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997, VIS '97, p. 395–ff. URL http://dl.acm.org/citation.cfm?id=266989.267108.

de Goes, Fernando; James, Doug L. Regularized Kelvinlets: Sculpting Brushes based on Fundamental Solutions of Elasticity. *SIGGRAPH*, 2017.

de Groot, Erwin; Wyvill, Brian; Barthe, Loic; Nasri, Ahmad; Lalonde, Paul. Implicit Decals: Interactive Editing of Repetitive Patterns on Surfaces. *Computer Graphics Forum*, v. 33, p. 141–151, 2014.

Dekkers, Ellen; Kobbelt, Leif. Geometry Seam Carving. *Comput. Aided Des.*, v. 46, p. 120–128, 2014. URL http://dx.doi.org/10.1016/j.cad.2013.08.024.

Dewaele, G; Cani, MP. Interactive global and local deformations for virtual clay. *Graphical Models* , v. 66, n. 6, p. 352–369, 2004.

Dischler, J.-M.; Maritaud, K.; Lévy, B.; Ghazanfarpour, D. Texture Particles. *Computer*

Graphics Forum, v. 21, n. 3, p. 401–410, 2002. URL http://dx.doi.org/10.1111/1467-8659.t01-1-00600.

Emilien, Arnaud; Vimont, Ulysse; Cani, Marie-Paule; Poulin, Pierre; Benes, Bedrich. WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds. *Proceedings of SIGGRAPH*, 2015.

Endara, Miguel. Benjaman Kyle. 2017. URL http://miguelendara.com/art/benjamankyle/.

Eyiyurekli, Manolya; Breen, David. Interactive free-form level-set surface-editing operators. *Computers & Graphics-UK*, v. 34, n. 5, p. 621–638, 2010.

Ferley, E; Cani, MP; Gascuel, JD. Resolution adaptive volume sculpting. *Graphical Models* , v. 63, n. 6, p. 459–478, 2001.

Gain, J; Marais, P. Warp sculpting. *IEEE Transactions On Visualization And Computer Graphics*, v. 11, n. 2, p. 217–227, 2005.

Gain, James E.; Dodgson, Neil A. Preventing self-intersection under free-form deformation. *IEEE Transactions On Visualization And Computer Graphics*, v. 7, p. 289–298, 2001.

Galyean, T. A.; Hughes, J. F. Sculpting - An Interactive Volumetric Modeling Technique. *SIGGRAPH*, v. 25, p. 267–274, 1991.

Grünbaum, Branko; Shephard, G C. *Tilings and Patterns*. New York, NY, USA: W. H. Freeman & Co., 1986.

Gustavson, Stefan. Simplex Noise Demystified. 2005. URL http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf.

Hale, J. G. *Texture Re-Mapping for Decimated Polygonal Meshes*. Tech. rep., Edinburgh University, 1998.

Hernandez, Raul Fernandez. Dynamic Subdivision Sculpting. 2011. URL http://farsthary.files.wordpress.com/2011/10/dynamic-subdivision-sculpting-final.pdf.

Hsu, William M.; Hughes, John F.; Kaufman, Henry. Direct manipulation of free-form deformations. *SIGGRAPH*, p. 177–184, 1992.

Koniaris, Charalampos; Cosker, Darren; Yang, Xiaosong; Mitchell, Kenny. Texture Mapping Techniques for Volumetric Mesostructure. *Journal of Computer Graphics Techniques (JCGT)*, v. 3, n. 1, p. 18–59, 2014. URL

http://jcgt.org/published/0003/01/02/.

Lachaud, Jacques-Olivier; Montanvert, Annick. Deformable meshes with automated topology changes for coarse-to-fine three-dimensional surface extraction. *Medical Image Analysis*, v. 3, n. 2, p. 187 – 207, 1999.

Lagae, Ares; Dutré, Philip. An Alternative for Wang Tiles: Colored Edges Versus Colored Corners. *ACM Trans. Graph.*, v. 25, n. 4, p. 1442–1459, 2006. URL http://doi.acm.org/10.1145/1183287.1183296.

Lagae, Ares; Kaplan, Craig S.; Fu, Chi-Wing; Ostromoukhov, Victor; Deussen, Oliver. Tile-based Methods for Interactive Applications. *ACM SIGGRAPH 2008 Classes*. New York, NY, USA: ACM, 2008, SIGGRAPH '08, p. 93:1–93:267. URL http://doi.acm.org/10.1145/1401132.1401254.

Land, Michelle D. Bridging Environmental Conservation and Animal Rights. 2016. URL https://earthdesk.blogs.pace.edu/2013/04/22/bridging-environmental-conservation-and-animal-rights/.

Li, Hongwei; Wei, Li-Yi; Sander, Pedro V.; Fu, Chi-Wing. Anisotropic Blue Noise Sampling. *ACM SIGGRAPH Asia 2010 Papers*. New York, NY, USA: ACM, 2010, SIGGRAPH ASIA '10, p. 167:1–167:12. URL http://doi.acm.org/10.1145/1866158.1866189.

Lorensen, William E.; Cline, Harvey E. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH*, v. 21, p. 163–169, 1987. URL http://doi.acm.org/10.1145/37402.37422.

Ma, Chongyang; Wei, Li-Yi; Lefebvre, Sylvain; Tong, Xin. Dynamic Element Textures. *ACM Trans. Graph.*, v. 32, n. 4, p. 90:1–90:10, 2013. URL http://doi.acm.org/10.1145/2461912.2461921.

Medeiros, Esdras; Ingrid, Lis; Pesco, Sinésio; Silva, Claudio. Fast Adaptive Blue Noise on Polygonal Surfaces. *Graph. Models*, v. 76, n. 1, p. 17–29, 2014. URL http://dx.doi.org/10.1016/j.gmod.2013.10.004.

Meng, Johannes; Papas, Marios; Habel, Ralf; Dachsbacher, Carsten; Marschner, Steve; Gross, Markus; Jarosz, Wojciech. Multi-scale Modeling and Rendering of Granular Materials. *ACM Trans. Graph.*, v. 34, n. 4, p. 49:1–49:13, 2015. URL http://doi.acm.org/10.1145/2766949.

Mercier, Olivier; Beauchemin, Cynthia; Thuerey, Nils; Kim, Theodore; Nowrouzezahrai, Derek. Surface Turbulence for Particle-Based Liquid Simulations. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2015)*, v. 34, n. 6, 2015.

Ostromoukhov, Victor; Donohue, Charles; Jodoin, Pierre-Marc. Fast Hierarchical

Importance Sampling with Blue Noise Properties. *ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM, 2004, SIGGRAPH '04, p. 488–495. URL http://doi.acm.org/10.1145/1186562.1015750.

Pastor, O.M.; Freudenberg, B.; Strothotte, T. Real-time animated stippling. *Computer Graphics and Applications, IEEE*, v. 23, n. 4, p. 62–68, 2003.

Peytavie, Adrien; Galin, Eric; Grosjean, Jérôme; Mérillou, Stéphane. Procedural Generation of Rock Piles Using Aperiodic Tiling. *Computer Graphics Forum*, 2009. URL https://hal-unilim.archives-ouvertes.fr/hal-01250531.

Pilgway. 3D Coat. 2017. URL http://www.3d-coat.com.

Pixologic. Sculptris. 2017a. URL http://www.sculptris.com.

Pixologic. ZBrush. 2017b. URL http://www.pixologic.com.

Robertson, Barbara. The Matrix Resolution. *Computer Graphics World*, 2003. URL http://www.cgw.com/Publications/CGW/2003/Volume-26-Issue-12-December-2003-/The-Matrix-Resolution.aspx.

Rohmer, Damien; Hahmann, Stefanie; Cani, Marie-Paule. Real-Time Continuous Self Replicating Details for Shape Deformation. *Computers and Graphics*, 2015. URL https://hal.inria.fr/hal-01152928.

Schwarz, Michael; Stamminger, Marc. On Predicting Visual Popping in Dynamic Scenes. *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*. New York, NY, USA: ACM, 2009, APGV '09, p. 93–100. URL http://doi.acm.org/10.1145/1620993.1621012.

Sederberg, Thomas W.; Parry, Scott R. Free-form deformation of solid geometric models. *SIGGRAPH*, v. 20, p. 151–160, 1986. URL http://doi.acm.org/10.1145/15886.15903.

Simpson, I.; Wood, L. *The Encyclopedia of Drawing Techniques*. A quarto book. Headline, 1987.

Sorkine, O.; Cohen-Or, D.; Lipman, Y.; Alexa, M.; Rössl, C.; Seidel, H.-P. Laplacian Surface Editing. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. New York, NY, USA: ACM, 2004, SGP '04, p. 175–184. URL http://doi.acm.org/10.1145/1057432.1057456.

Stanculescu, Lucian; Chaine, Raphaëlle; Cani, Marie-Paule. Freestyle: Sculpting meshes with self-adaptive topology. *Computers & Graphics-UK*, v. 35, n. 3, SI, p. 614–622, 2011.

Stanculescu, Lucian; Chaine, Raphaëlle; Cani, Marie-Paule; Singh, Karan. Sculpting multi-dimensional nested structures. *Computers & Graphics*, v. 37, n. 6, p. 753 – 763, 2013. Shape Modeling International (SMI) Conference 2013.

Sumner, Robert W.; Schmid, Johannes; Pauly, Mark. Embedded Deformation for Shape Manipulation. *ACM SIGGRAPH 2007 Papers*. New York, NY, USA: ACM, 2007, SIGGRAPH '07. URL http://doi.acm.org/10.1145/1275808.1276478.

Tang, Min; Manocha, Dinesh; Tong, Ruofeng. Fast continuous collision detection using deforming non-penetration filters. *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. New York, NY, USA: ACM, 2010, p. 7–13.

Teller, Seth. Line - Line Closest Points. 2006. URL http://graphics.lcs.mit.edu/~seth/geomlib/linelinecp.c.

Vanderhaeghe, David; Barla, Pascal; Thollot, Joelle; Sillion, Francois X. Dynamic Point Distribution for Stroke-based Rendering. *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, EGSR'07, p. 139–146. URL http://dx.doi.org/10.2312/EGWR/EGSR07/139-146.

Vanhoey, Kenneth; Sauvage, Basile; Larue, Frédéric; Dischler, Jean-Michel. On-the-fly Multi-scale Infinite Texturing from Example. *ACM Trans. Graph.*, v. 32, n. 6, p. 208:1–208:10, 2013. URL http://doi.acm.org/10.1145/2508363.2508383.

von Funck, Wolfram; Theisel, Holger; Seidel, Hans-Peter. Vector Field Based Shape Deformations. *ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM, 2006, SIGGRAPH '06, p. 1118–1125. URL http://doi.acm.org/10.1145/1179352.1142002.

Wachtel, Florent; Pilleboue, Adrien; Coeurjolly, David; Breeden, Katherine; Singh, Gurprit; Cathelin, Gaël; de Goes, Fernando; Desbrun, Mathieu; Ostromoukhov, Victor. Fast Tile-based Adaptive Sampling with User-specified Fourier Spectra. *ACM Trans. Graph.*, v. 33, n. 4, p. 56:1–56:11, 2014. URL http://doi.acm.org/10.1145/2601097.2601107.

Wang, Lili; Shi, Yulong; Chen, Yi; Popescu, Voicu. Just-in-Time Texture Synthesis. *Computer Graphics Forum*, v. 32, n. 1, p. 126–138, 2013. URL http://dx.doi.org/10.1111/cgf.12003.

Wei, Li-Yi. Tile-based Texture Mapping on Graphics Hardware. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. New York, NY, USA: ACM, 2004, HWWS '04, p. 55–63. URL http://doi.acm.org/10.1145/1058129.1058138.

Ying, Xiang; Li, Zhenhua; He, Ying. A parallel algorithm for improving the maximal

property of Poisson disk sampling. *Computer-Aided Design*, v. 46, p. 37 – 44, 2014. URL http://www.sciencedirect.com/science/article/pii/S0010448513001565. 2013 SIAM Conference on Geometric and Physical Modeling.

Zwicker, M.; Pauly, M.; Knoll, O.; Gross, M. Pointshop 3D: An Interactive System for Point-Based Surface Editing. *SIGGRAPH*, p. 322–329, 2002.