Sketch-Based Modeling and Adaptive Meshes

Emilio Vital Brazil[a], Ronan Amorim[a], Mario Costa Sousa[a], Luiz Velho[b], Luiz Henrique de Figueiredo[b]

[a] *Department of Computer Science, University of Calgary, Canada*
[b] *IMPA – Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil*

Abstract

We present two sketch-based modeling systems built using adaptive meshes and editing operators. The first one has the capability to control local and global changes to the model; the second one follows geological domain constraints. To build a system that provides the user with control of local modifications we developed a mathematical theory of vertex label and atlas structure for adaptive meshes based on stellar operators. We also take a more theoretical approach to the problem of sketch-based surface modeling (SBSM) and introduce a framework for SBSM systems based on adaptive meshes. The main advantage of this approach is a clear separation between the modeling operators and the final representation, thus enabling the creation of SBSM systems suited to specific domains with different demands.

Keywords: Sketch-Based Modeling; Adaptive Mesh; Geometric Modeling

1. Introduction

Sketches are the most direct way to communicate shapes: humans are able to associate complex shapes with few curves. However, sketches do not have complete shape information, and the information sketches do provide is often inexact; thus, ambiguities are natural. On the other hand, to create, edit, and visualize shapes using computers, we need precise mathematical information, such as a function formula or a triangle mesh. The problem of how to model shapes using sketches can be formulated as how to fill the missing information about the model. In the last 15 years, sketch-based modeling (SBM) has become a well established research area, encompassing work in different domains, such as computer vision, human-computer interaction, and artificial intelligence [1]. However, this body of work lacks a more theoretical approach on how to build a sketch-based modeling system for a given application. In contrast, we present here two sketch-based modeling systems built on top of the same framework. This framework is tailored for sketch-based surface modeling (SBSM) taking advantage of adaptive meshes.

We advocate that SBSM systems must be suited to each specific application: the specificities of a certain field require suitable mathematical representations for the domain model, and this plays a central role in the characterization of SBSM applications. However, there are common requirements in many SBSM applications that can be abstracted to guide the definition of specific representations for specific domains. These requirements have three main aspects: (1) *dynamic* – the surface will change during the modeling process; (2) *interactive* – the user must be able to see the model changing with interactive response and feedback; (3) *controlled freedom* – some applications have specific modeling rules and the systems must be able to incorporate these rules to guide the user in building a correct model, without losing flexibility.

Adaptive meshes are generally associated with the ability to produce detailed complex models using a smaller mesh. However, our proposed framework is based on adaptive meshes because they can be dynamic and enable rapid updates with local control. Different schemes of adaptive meshes can be used to create a system using our framework; indeed, the choice of the scheme must take into account the final application requirements, such as how to represent features, what changes of topology are allowed, and how smooth the models need to be. Figure 1 shows an instance of a model built within our framework: a 4-8 adaptive mesh adapted to an implicit surface.

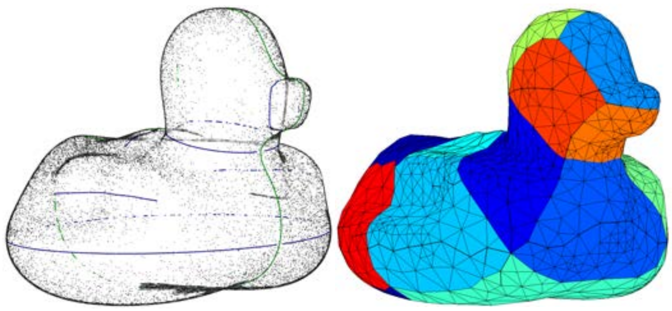


Figure 1: A rubber duck modeled using DASS system: the HRBF implicit surface (left) and the adapted 4-8 mesh (right).

The two sketch-based modeling systems that will be presented here are built using our proposed framework and have major differences. The first system is the *Detail Aware Sketch-Based Surface Modeling* (DASS, Section 5), which approaches a common problem in many SBSM systems: the lack of good control of global and local transformations. We created DASS to allow us to validate our proposed framework, exploring the limitations of a general system without a well defined task. To achieve the required control we developed a method to create atlas structures for adaptive meshes based on stellar operators [2]. The second system is the *Geological Layer Modeler (GLaM, Section 6)*, which is a sketch-based system specialized for geology that aims to help geophysicists to create subsurface models. This system is a good illustration of controlled freedom, where the sketch operators should be restricted to follow geological rules.

2. Related Work

In the past decades there has been a large body of work in sketch-based surface modeling [3, 4, 5, 6, 7]. However, these systems are more concerned with the final results and do not consider the theoretical aspects of the mathematical surface representation used. We discuss below the main works on free-form sketch-based surface modeling that start from scratch under the light of its representations.

There are many ways to represent surfaces in $\mathbb{R}^3$. The most common and general are parametric representations and implicit representations. However, in order to be used in computer graphics and modeling applications, these representations must be more specific and possess practical qualities. As examples we can cite the BlobTree [8], piecewise algebraic surface patches [9], convolution surfaces [10], generalized cylinders, polygonal meshes, subdivision surfaces, among others.

Teddy [3], Fibermesh [4], and Kara and Shimada [11] use triangle meshes as a base representation for their modeling systems. Teddy and Fibermesh start with a planar curve and create an inflated mesh based on the curve's geometry. Teddy supports extrusion and cutting operators that cut a mesh part, then create a new mesh patch, which is merged with the model. Similarly, Fibermesh creates a new mesh based on the input sketches and places it using optimization on differential coordinates, thus enabling the system to keep all previous strokes as constraints. Kara and Shimada also keep a set of 3D curves to define the final model. However, they use curve loops to define triangle mesh patches that have minimum curvature, instead of optimizing across the whole mesh. These patches can be modified using physically-based deformation tools. These three systems are based on the triangle mesh representation and use it to build their modeling operators; as result, their advantages and limitations are directly related with that chosen representation.

Using triangle meshes for modeling purposes has several advantages over other representations. First, triangle meshes are largely used by both academia and industry, and most graphics pipelines are based on triangles, which means that what you see is what you get. Moreover, there is much research on triangle meshes and many techniques have been developed for creating and editing meshes. On the other hand, applying these techniques in sketch-based modeling is not a straightforward task: techniques must be chosen based on the application scope, and these choices will define the limitations of the system. These limitations are noticeable in Teddy and Fibermesh – the latter approaches some drawbacks of the former using optimization on differential representation. Compared with Teddy, in Fibermesh the mesh quality is improved, the topology can be changed, and the construction curves are maintained using differential mesh techniques. However, the need for global optimization to assure mesh quality removes control over global and local editions: editing a small part of the model could affect other parts. Indeed, Nealen et al. [4] and Kara and Shimada [11] raised this issue: Nealen et al. suggested to embed the multi-resolution operator as a solution, whereas Kara and Shimada suggested to improve their method of creating and editing curves.

Parametric surfaces are defined by mapping a planar domain to 3D space. Working with parametric surfaces has some advantages: it is simple to obtain a good triangle mesh that approximates the model, it is relatively easy to map textures to the surface, and it provides continuous normal and curvature information. Cherlin et al. [12] and Gingold et al. [5] use parametric representation to create sketch-based systems. Cherlin et al. introduce two novel parametric surfaces based on sketched curves; Gingold et al. convert sketches to generalized cylinders. However, both approaches have issues with topology change and creating augmentations; these difficulties are mainly caused by the chosen parametric representations. Nasri et al. [13] and Orbay and Kara [7] create their systems based on subdivision surfaces – only being able to deal with set of curves that form closed loops. Heightfield is another example of parametric surface: it gives a 3D point (x, y, z) as a function of 2D coordinates, $z = f(x, y)$. This representation is fast and simple, and is usually enough for most terrains comprising mountains and hills. However, heightfields are not able to represent terrains with more complex geological structures, such as overhanging cliffs or caves. Hnaidi et al. [14] present a sketch-based system to model terrains. The characteristics of the terrain are defined by the user through a set of feature curves representing ridges, river beds, and cliffs. Constraints on these curves define elevation, angle and noise parameters along them. These constraints are then defined for the entire domain by diffusion. When the smooth terrain is ready, details are added by a procedural noise generator. The final terrain is a heightfield that results from combining the smooth terrain with the details.

In contrast with parametric surfaces, implicit surfaces can easily change topology when parameters change. They can also provide a compact, flexible, and mathematically precise representation which is well suited to describe coarse shapes. Implicit surfaces allow global calculations, such as point classification (i.e., whether a point is inside or outside the surface volume) and distance evaluation. They also provide with access to local differential properties, such as normals and curvature. Karpenko et al. [15] introduced variational implicit surfaces as representation to sketch-based surface modeling. Vital Brazil et al. [6] improved this formulation by adding normals as hard constraints. Amorim et al. [16] presented a sketch-based system using Hermite–Birkhoff interpolation to create implicit models applied to geology. Araujo and Jorge [17] provided a set of sketch-based operators adapting the multi-level partition-of-unity implicit model [18]. Schmidt et al. [19] used BloobTrees as a main representation of the ShapeShop system. Bernhardt et al. [20] built the Matisse system based on convolution surfaces. These systems share the main disadvantages known about implicit representations: (1) the standard graphics pipeline is not prepared to handle implicit models; (2) few industrial processes use implicit surfaces, and so the final model must be converted; (3) it is hard to control details. For (1) and (2), almost all systems polygonize the models (e.g., marching cubes), but there are many drawbacks in this approach; e.g., some methods guarantee neither correct topology nor mesh quality.

On the whole, much of this previous work is built on a specific representation and its drawbacks come from that choice. Inspired by that observation, we propose here a simple framework based on adaptive meshes to allow us to mix different representations in one system. This work is a extension of Vital Brazil et al. [21]; besides new results, we include in this version all technical parts of the Detail Aware Sketch-Based Surface Modeling (DASS) system (Section 5), with the mathematical formulations and proofs of the label theory and atlas structure. Moreover, we improve the discussion about the Geological Layer Modeler (GLaM) system (Section 6) with new images and a deeper discussion about the framework and expert feedback. Before presenting these two systems, we give an overview of adaptive meshes in Section 3 and we discuss our framework in Section 4.

3. Adaptive Mesh Overview

An adaptive mesh is a polygonal mesh that has the ability to create and remove vertices, edges, and faces following predefined rules. The creation process is called *refinement* and the deletion process is called *simplification*. An adaptive mesh scheme starts with a base mesh which is refined until it matches a stop criterion. Usually this criterion is associated with a maximum threshold for some error metric. In summary, an adaptive mesh must have a base mesh, criteria for when to apply refinement and simplification, and rules for how to perform refinement and simplification. Since we are working with a dynamic system, we also need an update rule.

Any remeshing scheme can be used to build an adaptive mesh that can be used as core of the proposed framework (Section 4). We chose to study a small set of mesh operators, namely *stellar subdivision operators* and their inverses (Figure 2); these operators are largely studied in combinatorial algebraic topology [22]. We focus on how to create meshes with atlas structures. The concepts of *sequence of meshes* and *level* of an element presented by Velho [23] for stellar operators give the mathematical tools for building our label theory (Appendix A). This theory enables the creation of atlases for adaptive meshes with mathematical guarantees. We use the adaptive 4-8 mesh [2], adopting the dynamic frame work presented by Goes et al. [24]. The 4-8 mesh refinement process only uses the edge stellar operator and the simplification process uses its inverse. However, to be able to convert a generic mesh to a 4-8 mesh, face stellar operators are required [25].

Any dynamic adaptive mesh scheme can be used in the framework proposed in the next section. We chose the 4-8 mesh to build our systems chiefly because it has the following properties: (1) elegant mathematical theory; (2) very small support – if a small part is refined then, except for a relatively short region near the change, the mesh is left untouched (see Figure 3); (3)

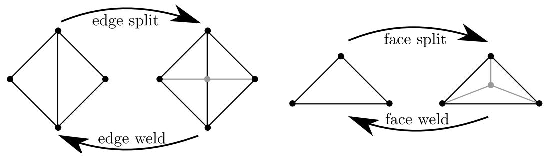


Figure 2: Stellar subdivision operators and their inverses.

simplicity – only stellar operators are used and can be easily implemented using the half-edge data structure [26]. Although the 4-8 subdivision scheme is important in many applications, we do not use in this work. One could use the subdivision scheme to place the vertices; in that case the 4-8 subdivision has several interesting properties [27]. The 4-8 adaptive scheme has a topological uniformity that can be a drawback for some applications: all regular vertices have valence 4 or 8 and this could imply a marked direction bias in the mesh. The choice of the adaptive scheme has to take into account the final application requirements.

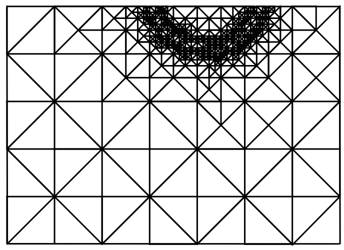


Figure 3: 4-8 local refinement.

4. Framework

The proposed framework enables system designers to build a sketch-based system that is interactive and has controlled freedom. Interactivity means that the system must be able to show how the model changes in interactive time. Controlled freedom means that some applications have specific modeling rules, and the system must be able to incorporate these rules to guide the modeler, but without losing flexibility. Moreover, the framework must be sufficiently general to be applied in different domains with different requirements. We split the framework into three main components: initial shape descriptor, adaptive mesh, and editing operators. Figure 4 illustrates the main information flow between these components.

Figure 4: The framework for Sketch-based surface systems. The arrows depict the information flow.

First of all, we need an initial shape descriptor to be capable to tessellate the coarsest mesh, which is called the *base* *mesh*. For example, it could be the first inflated model of the Teddy or Fibermesh. For most of adaptive meshes, the base mesh must have the same topology of the intended model and must approximate its geometry. Geometry approximation has different meanings depending on the application; as a general rule, it means that when a new vertex is created it can be correctly placed on the surface of the model. For instance, for an implicit surface, the base mesh has to be inside a tubular neighborhood of the surface so that new vertices can be projected onto the surface.

In the proposed framework the main roles of the adaptive mesh is to allow independent geometry representations for the editing operators and to keep the coherence of the modeling process. A positive side effect of using adaptive meshes is to be able to use the base-mesh as a natural parametrization of the surface, as discussed in Section 5.1.

The editing operators are the system parts responsible for all model modifications, such that the edited mesh is still an adaptive mesh. Much of the work of editing adaptive meshes is done by changing the criteria and rules mentioned in the previous section. For instance, if it is a geometric editing the operator can be implemented as a new rule for vertex update and refinement, after which the mesh will be adapted for the new shape. Since the obtained mesh is an adaptive mesh, the editing loop restarts.

We apply this framework to create two very different systems. The DASS system (Section 5) starts with a set of 3D curves in the space and a base mesh. We create an implicit surface that interpolates the curves' points; and the base mesh creates an atlas structure. Together, they are the initial shape descriptor of DASS. In contrast, the GLaM System (Section 6) has three simple initial shape descriptions: one height map, one parametric surface based on boundary curves, and one that is a convex sum of other two. The DASS system uses an small set of editing operators to modify the implicit surface and to create details. On the other hand, using the abstraction of operators, GLaM creates a large variety of complex functionalities by composing operators. Both system use an 4-8 adaptive mesh to build the final model.

5. Detail Aware Sketch-Based Surface Modeling (DASS)

The main goal of DASS system prototype is to allow the user to control local modifications without changing parts of the model outside the region of interest, and keeping details coherent when large deformations are introduced. Hence, we advocate that decomposing the model representation into a base surface that supports different types of properties is a powerful tool for sketch-based surface modeling. Markedly, Blinn [28] introduces the idea of bump-mapping that stores geometric information at two levels: the base geometry and a displacement map which is used to create rendering effects. The same concept is found in [29] and [30]. They use two different types of data: the first one defining the smooth geometry and the second one mapping the first to a parametric space that stores details (similar to a texture mapping).

It is important to remark the difference between our solutions and multi-resolution works [31] and manifold surface modeling [32]: multi-resolution works are concerned with *subdivision schemes* and we use neither subdivision nor multi-scale analysis. Instead, we use a 4-8 mesh, an adaptive mesh which nonetheless can simulate many subdivision schemes [23] (although we do not use it as such). Also, the manifold modeling community approaches the problem of how to build and edit manifold structures starting from a mesh or a subdivision scheme. In contrast, we use the base mesh directly to construct such structure, and we have developed simple rules to ensure correctness of the manifold structure when we apply editing operators.

5.1. Adapted Framework

The DASS system starts with the coarse form defined by an implicit surface; after that, we build a base mesh that has the same topology and approximately the same geometry of the implicit surface. The base mesh induces an atlas and provides a 4-8 base mesh. The atlas is built using a partition of the set of mesh faces, and we use it to edit the model locally. The 4-8 mesh plays two roles in the framework: to build a map between surface and atlas, and to visualize the final surface. After we have all parts, the 4-8 mesh is used to edit details that are saved in the atlas, and the atlas maps details onto the 4-8 mesh. Figure 5 illustrates our framework.

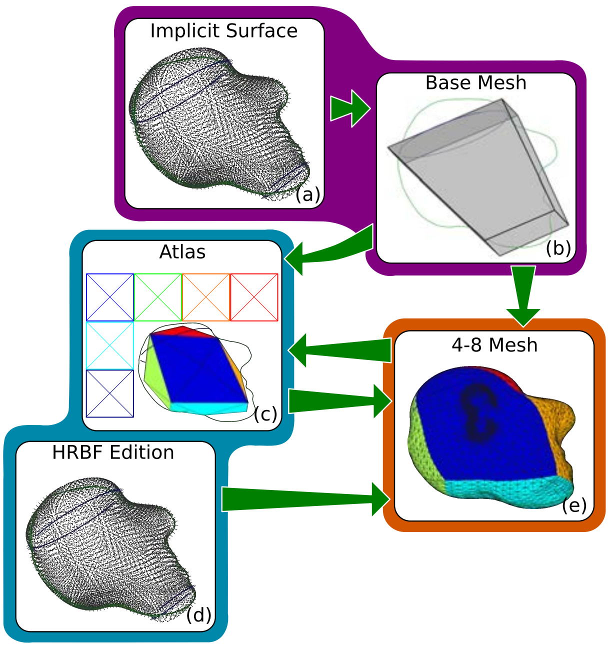


Figure 5: The framework of DASS system. The color boxes are related with the theoretical framework in Figure 4.

The first step in the framework is to obtain a coarse shape of the final model (Figure 5(a)(b)). We use the same implementation described in [6], in which the authors introduce a new representation for implicit surfaces, HRBF, and show how it can be used to support a collection of free-form modeling operations.

After we obtain our implicit surface Ξ, we create the manifold structure to represent our final model S. To handle parameters, we use an atlas $\mathcal{A}$ of S, i.e., $\mathcal{A} = \{\Omega_i, \phi_i\}_{i=0}^{k}$ such that $\Omega_i \subset \mathbb{R}^2$, and $\phi_i : \Omega_i \to S$ are homeomorphisms [33]. However, we have an implicit surface without information about the atlas. One possible way to tackle this problem could be to create a polygon mesh and use one method to obtain a quad mesh [34]. There are many approaches to polygonize implicit surfaces, e.g. [35, 36, 37], but to find the correct topology these approaches depend on user-specified parameters [35, 36], or require differential properties of the surface [37]. In addition, we require interactive time and to obtain a good mesh from an implicit function is an expensive task. Apart from the topology issue, such methods neither guarantee mesh quality nor have a direct way to build an atlas structure. As a result, we have opted to develop a method that is based on our problem and on the desired surface characteristics.

First of all, we observe that there are two different scales of detail to be represented: the implicit surface (which is coarse) and the details (which are finer). The naive approach would be to use the finest scale of detail to define the mesh resolution. However, there are two issues associated with this approach: firstly, we do not know the finest scale a priori; and secondly, if the details appear in a small area of the model, memory and processing time will be wasted with a heavily refined mesh. To avoid these issues we adopted a dynamic adaptive mesh, the semi-regular 4-8 mesh [2] because it enables control on where the mesh is fine or coarse, by using a simple error function.

Returning to the problem of parametrization of our implicit surface, now we wish for more than just a mesh: we need an adaptive mesh. The framework presented by [24] starts with a 4-8 mesh and refines it to approximate surfaces using simple projection and error functions. To obtain a good approximation of the final surface, the 4-8-base-mesh must have the same topology and must approximate the geometry of the final surface. Thereupon our parametrization problem was reduced to the problems of how to find a good 4-8 base mesh and how to construct a good error function.

The parametrization of the implicit surface is built in three parts: base mesh (Figure 5(b)), atlas (Figure 5(c)), and 4-8 mesh (Figure 5(e)). In Section 5.2 we present a base mesh with two roles in our system: inducing an atlas for the surface and creating a 4-8 mesh. We describe a method in Section 5.3 to create an atlas for adaptive meshes based on stellar operators. In Section 5.4 we discuss how build an error function for the 4-8 mesh that is sensitive to levels of detail (LoD).

5.2. *Base Mesh*

The base mesh is the first step to parametrize our surface. This is a crucial piece of our pipeline, because three important aspects of the final model depend on the base mesh: the topology of the final model, the atlas, and the quality of the 4-8 mesh. In the context of sketch-based modeling, it is natural to exploit user input to extract more information about the model and create the base-mesh.

The user handles a simple unit of tessellation element (tesel) which can have the topology of a cube or a torus. This tesel is projected onto the drawing plane enabling its modification to improve the geometric and topological approximation of the model by moving its vertices on the plane, by dividing it creating one more tesel, or by changing its topology. Afterwards, the system creates a tessellation in the space by moving each tesel vertex along the direction normal to the drawing plane. Figure 6 shows the typical steps taken to create the base mesh: the user starts with a bounding box of the sketched lines, then divides tesels, moves vertices, and changes tesel's topology to build a better approximation of the intended shape. Our system defines vertex heights by searching along the normal direction for a point on the implicit surface. Each quad face defines a chart; then this face is triangulated to be used as the 4-8 base mesh.

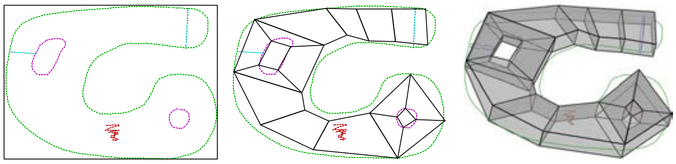


Figure 6: Creating a base-mesh for an implicit surface using the construct lines described in Vital Brazil et al. [6]. Left to right, the first approximation, after a user corrects the topology and improve the geometry, and the final result in $\mathbb{R}^3$.

5.3. *Atlas*

We must construct an atlas to obtain the manifold structure for our model, i.e., a collection of charts c_i formed by open sets $\Omega_i \subset \mathbb{R}^2$, and functions $\phi_i : \Omega_i \to S$ that are homeomorphisms [33]. Specifically for this application, each chart of $\mathcal{A}$ is associated with a height map, which is used to define a displacement along the normal direction. In Section 5.4 we use that height map to define an error function that helps to define the 4-8 refinement.

Figure 7 illustrates the steps to create an atlas for a 4-8 mesh M. After the base mesh is obtained and each of its faces is triangulated, one refinement step is performed and then each base mesh face is associated with a chart (Figure 7(a)). When the mesh is refined to better approximate the geometry, the atlas is updated and the user can draw curves over the M which are transported to the charts; these curves create or modify the height maps (Figure 7(b)). If the mesh resolution is not enough to represent the desired details, M is refined. Usually that happens when the user creates or modifies the height maps (Figure 7(c)).

In Appendix A we discuss the main aspects of a *vertex map* and how to use it to create the atlas structure. In Appendix B we describe how we use the vertex map to sketch over the surface creating the height map.

5.4. *Using 4-8 Mesh*

The 4-8 mesh M has two main roles in DASS system. The first one is to transport points to the atlas, as described in the previous section and in the appendices. The second role is to visualize the approximated final surface. In addition we need to provide a function that samples an edge returning a new vertex,

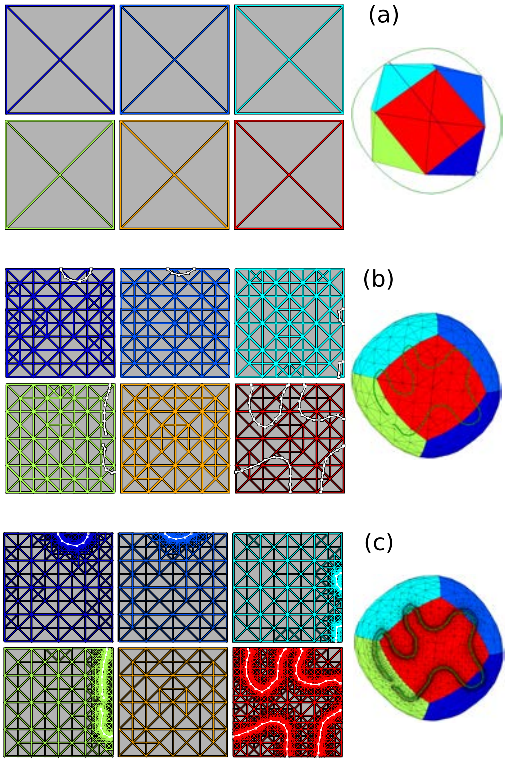


Figure 7: Steps to create an atlas: (a) The atlas is defined after one refinement step of M. (b) M is refined and the user defines an augmentation sketching over the surface, and the sketches are transported to $\mathcal{A}$ to built a height map. (c) M is refined to represent details of the final surface with height map.

and two error functions: one to classify the edges for the refinement step and one to classify the vertices for the simplification step.

To define a new vertex we adopt the naive approach that projects the midpoint of an edge onto the surface: we split an edge $e = \{v_1, v_2\}$ creating a new vertex $v_n = \Pi_S((v_1 + v_2)/2)$; and, as described in Appendix A, if $v_n \in c_i$ we save its local coordinates too. This simple technique achieves good results for our application.

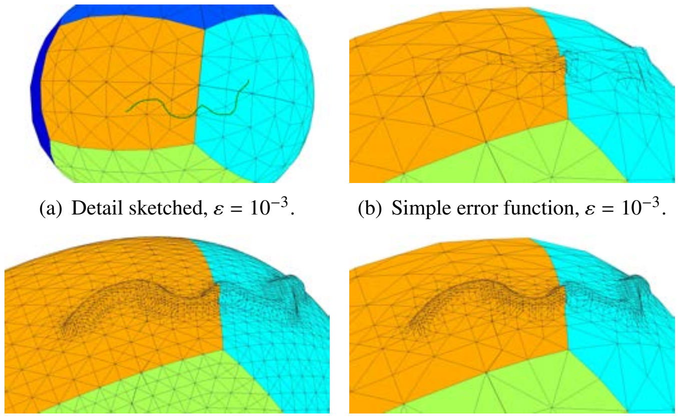
We need to select which edges will be split, to refine the mesh, and which vertices will be removed, to simplify the mesh. In our implementation, this classification is done using two error functions and one parameter. To define our error functions we need to describe how we measure the distance between a point and the surface. First, observe that Π_Ξ is the projection on $\Xi \neq S$, and so Π_Ξ is not enough to define the distance. To project a point p onto S, first we project p onto Ξ, and then, using the atlas information, we apply the displacement function D. More precisely,

$$\Pi_S(p) = \Pi_\Xi(p) \oplus D(\Pi_\Xi(p)),. \tag{1}$$

The distance between p to S is the usual

$$d_S(p) = |p - \Pi_S(p)|. \tag{2}$$

Now we can determine the error functions using the stochastic approach presented by [24]. We first define the error function in faces by taking the average of the distance from the point to n random points on the face. The error function on edges and vertices is the average error on their respectively incident faces. To control mesh adaptation, we define an error threshold $\varepsilon > 0$, and declare that if the edge error is above that threshold the edge should be refined. Observe that ε controls the size of our final mesh. If ε is small we have a good approximation of the surface, but the mesh will have too many vertices, which is computationally expensive (Figure 8(c)). On the other hand, if ε is large, the mesh will be computationally cheap but the mesh will not represent well the final surface details (Figure 8(b)).



(a) Detail sketched, $\varepsilon = 10^{-3}$.

(b) Simple error function, $\varepsilon = 10^{-3}$.

(c) Simple error function, $\varepsilon = 10^{-4}$.

(d) Local error function, $\varepsilon = 10^{-3}$.

Figure 8: Local error control.

It is natural to have an approximation for Ξ that is coarser than for S. We are assuming that Ξ is only the coarse information, whereas S also has details (Figure 8(a) and (c)). However, since details are typically restricted to small surface areas, if we use S to choose ε we could have an expensive mesh without adding any real benefit. Since our application works with two different levels of details, it is natural to use this LoD structure to define the error functions. In our representation the details are encoded in D. We define the LoD at a point p as

$$E(p) = \eta(D(p)), \tag{3}$$

where $\eta : \mathbb{R} \to \mathbb{R}_+$. We implement that using the height maps since they are our details over the surface. Specifically, Equation (3) is rewritten as $E(p) = \max\{2|\nabla h_p|, 1\}$, where ∇h_p is the gradient of the height map evaluated in p.

Now we have all elements to define an error function based on the level of detail at a point over the surface. We define the local error function using Equations (2) and (3); so we have $\Delta(p) = d_{\widetilde{S}}(p)E(p)$. We apply this new definition in the face error calculation and as result we reformulate the edge error and the vertex error functions. In Figure 8 we can observe the difference between using the simple error function and using the local error function. The mesh in Figure 8(b) has 460 vertices but we lost the details of the final surface. If we decrease ε (Figure 8(c)) we reveal the details but the mesh grows ten fold to 4.8k vertices. When we use the local error function (Figure 8(d)) we reveal the details and the mesh size does not grow too much, only to 1.3k vertices.

5.5. *Work-flow and Results*

Our work-flows are based on the presented by Goes et al. [24] to adaptive dynamic meshes. The DASS system has three different work-flows: (1) the user starts the modeling system with a blank page, or changes the current model topology, (2) the geometry of the implicit surface is changed, and (3) the mesh resolution is recalculated (this usually happens when the height maps are changed). Figure 9 shows an overview of the work-flow.

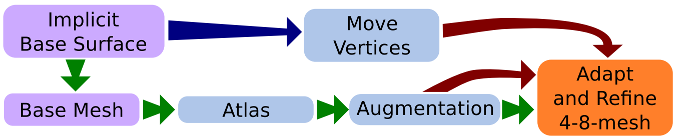


Figure 9: Overview of DASS system work-flows: green arrows are the startup and topological change step sequence, blue arrow are stepped when the implicit surface is edited, and the red arrow is done when the mesh resolution changes.

The user starts the modeling session by drawing construction curves, as described in [6]. Then, the system uses these curves to create samples defining an implicit surface (Figure 10(a)). After that, the user creates a planar version of the base mesh that approximates the geometry and has the same topology of the final model (Figure 10(b)). Thus, the base mesh is transported to 3D space (Figure 10(c)). Then, the base mesh is used to create an atlas structure (Figure 10(d)) for a 4-8 mesh. This mesh is refined creating the first approximation of the final model (Figure 10(e)). The steps described up to now are the common steps for all modeling sessions. They are represented by the green arrows in Figure 9. These steps also are illustrated in Figure 11(a) and (b), and 12(a). When we change the topology we also need to change the base mesh, restarting the process, as illustrated in Figure 11(a) and (b). If there is a predefined height map, the model reaches the end of this stage with one or more layers of detail. For example, in Figure 13(a) we start the model with a height map encoded as a gray image.

After the first approximation for the final surface, the user can modify the implicit surface and create or modify a height map. When details are added on the surface, in almost all cases this implies that the resolution of the mesh is not fine enough to represent the new augmentation. In this case, we must adapt and refine the mesh. In Figures 10(f), 11(c), 12(b), and 13(b): the user sketches a height map over the surface and the mesh is refined to represent the geometry of the augmentation correctly. The user can change the implicit surface at any stage, and if the topology is still the same, the system allows vertices to be moved without adaptation and refinement (in order to obtain a fast approximation). Since details are codified separately, they are moved consistently when implicit surfaces are modified. We illustrate that in Figures 10(g) 13(c), and 12(c), (e) and (f). Specifically, in Figure 12(e) and (f) we can compare good final results preserving the details despite the significant changes of the implicit surface. Sometimes, when only the implicit surface is changed, moving the vertices alone is not enough to reach the desired quality. In such cases, the user can adapt and refine the mesh decreasing the error threshold, as shown in Figure 12(d). Here, the user initializes $\varepsilon = 10^{-3}$, and after some modeling steps, a new threshold of 10^{-4} is chosen.

The modeling session of each model took approximately 10 minutes, from the blank page stage up to the final mesh generation. All the results were generated on an 2.66 GHz Intel Xeon W3520, 12 gigabyte of RAM and OpenGL/nVIDIA GForce GTX 470 graphics. The most expensive step was to create the implicit surface, followed by the creation of the base mesh; on the other hand, processing of the augmentation and minor adjustments in the implicit surface had a minor impact on performance. The bottleneck is the mesh update: if the mesh has too many vertices (around 10k), one refinement step after an augmentation takes about 10 seconds. The final models of space car, terrain, head, and party balloon have 10k, 11k, 7k and 13k vertices respectively.

6. Geological Layer Modeler (GLaM)

We developed a sketch-based system for seismic interpretation and reservoir modeling (Figures 14) based on the framework presented in Section 4. Most of the existing tools for seismic interpretation rely on the automatic extraction of horizons (interfaces between two rock layers) using segmentation algorithms. However, seismic data have a high level of uncertainty and noise which leads to mistakes in the horizon extraction. The main objective of the GLaM system is to enable the experts to directly interpret the geology using their knowledge and fix problems coming from an automatic extraction. The GLaM system enables augmenting, editing, and creating geological horizons using sketch-based operators. We have a seismic reflection volume, a distance volume (computed from the seismic volume), and a complete horizon candidate given as input to our system.

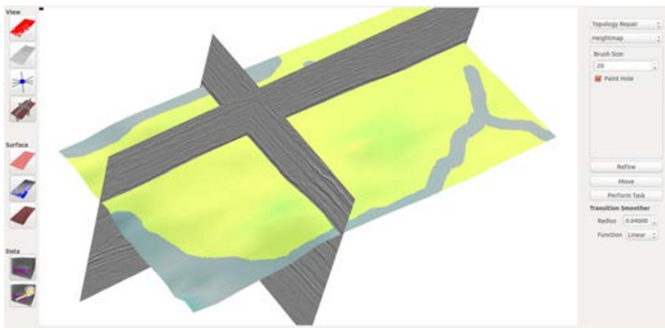


Figure 14: GLaM system interface.

Following our proposed framework (Section 4), the GLaM system has an *initial shape descriptor* and rules to change the adaptive mesh to follow the user's sketches. Compared to the DASS system, the initial shape descriptor is simple. According to our framework, the *initial shape descriptor* must be able to tessellate the base mesh that will be used as a first approximation of the model. In the GLaM system, there are three different ways of creating a horizon: (1) from an input horizon candidate, (2) from user-specified lines that define boundaries of the horizon, or (3) from a combination of two existing horizons. Thus,

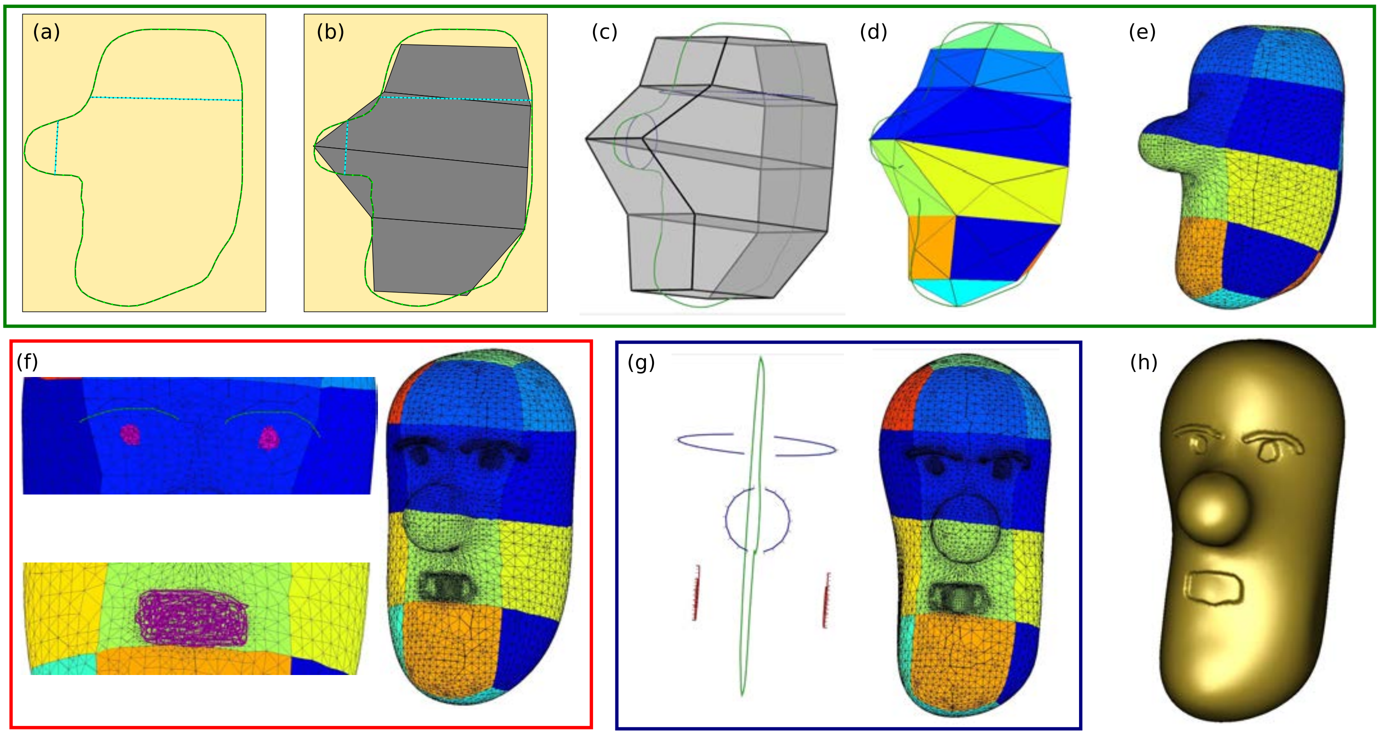


Figure 10: Steps to model a head using DASS.

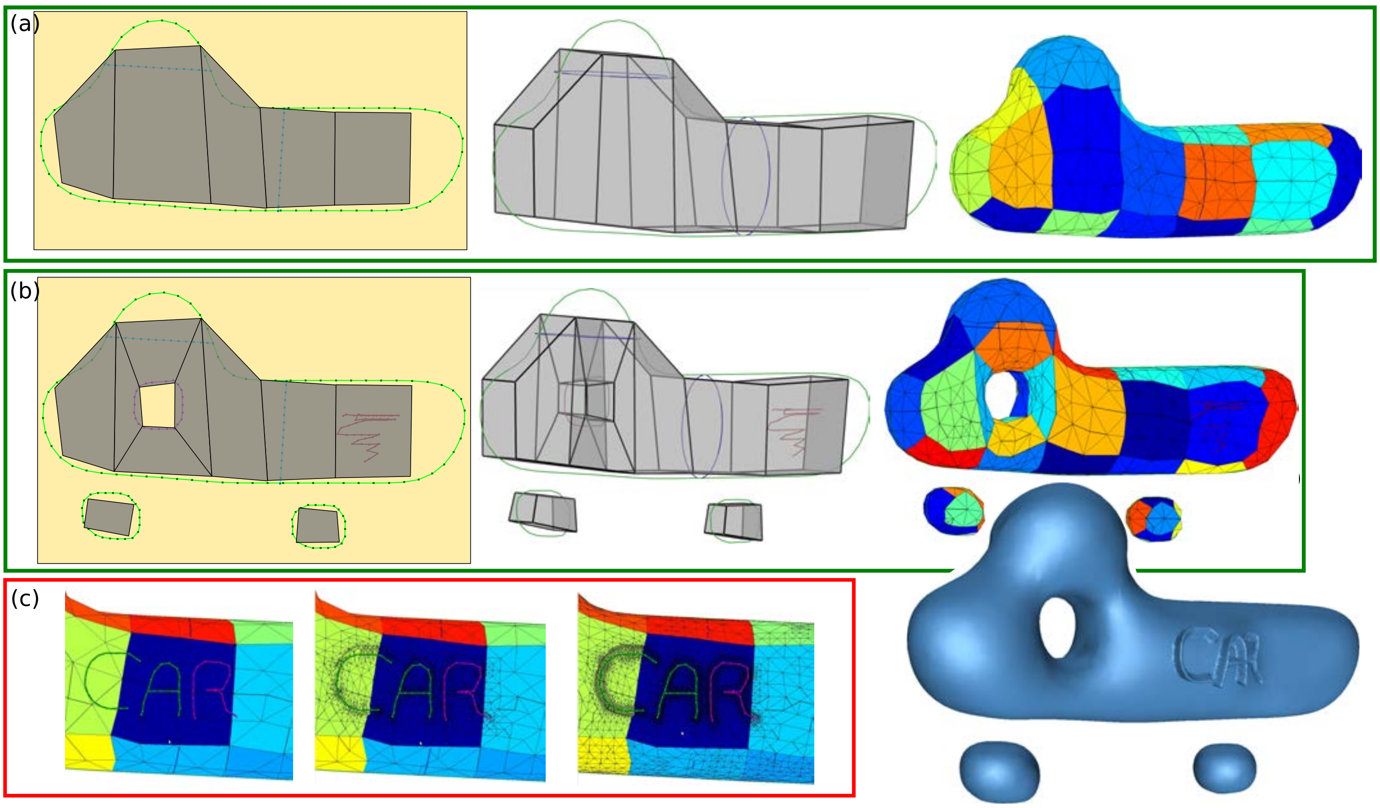


Figure 11: Steps to model a space car using DASS.

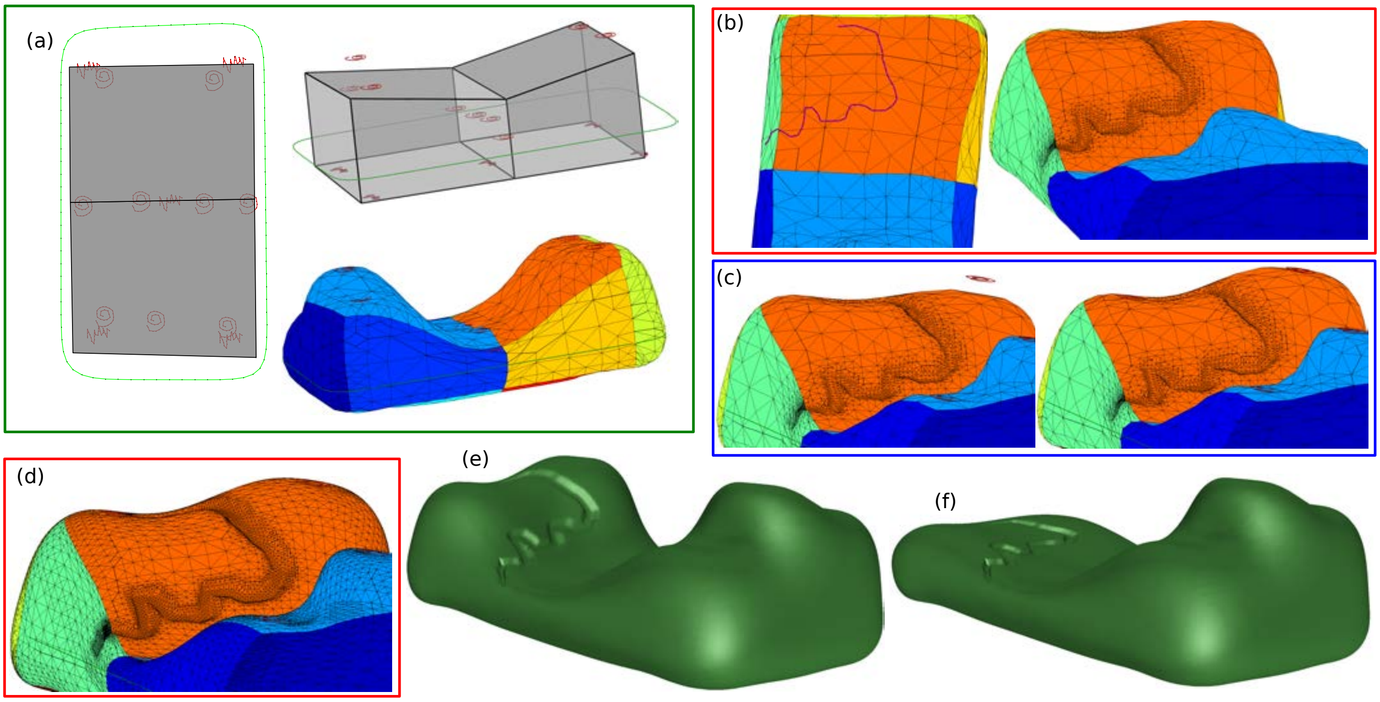


Figure 12: Steps to model a terrain using DASS.

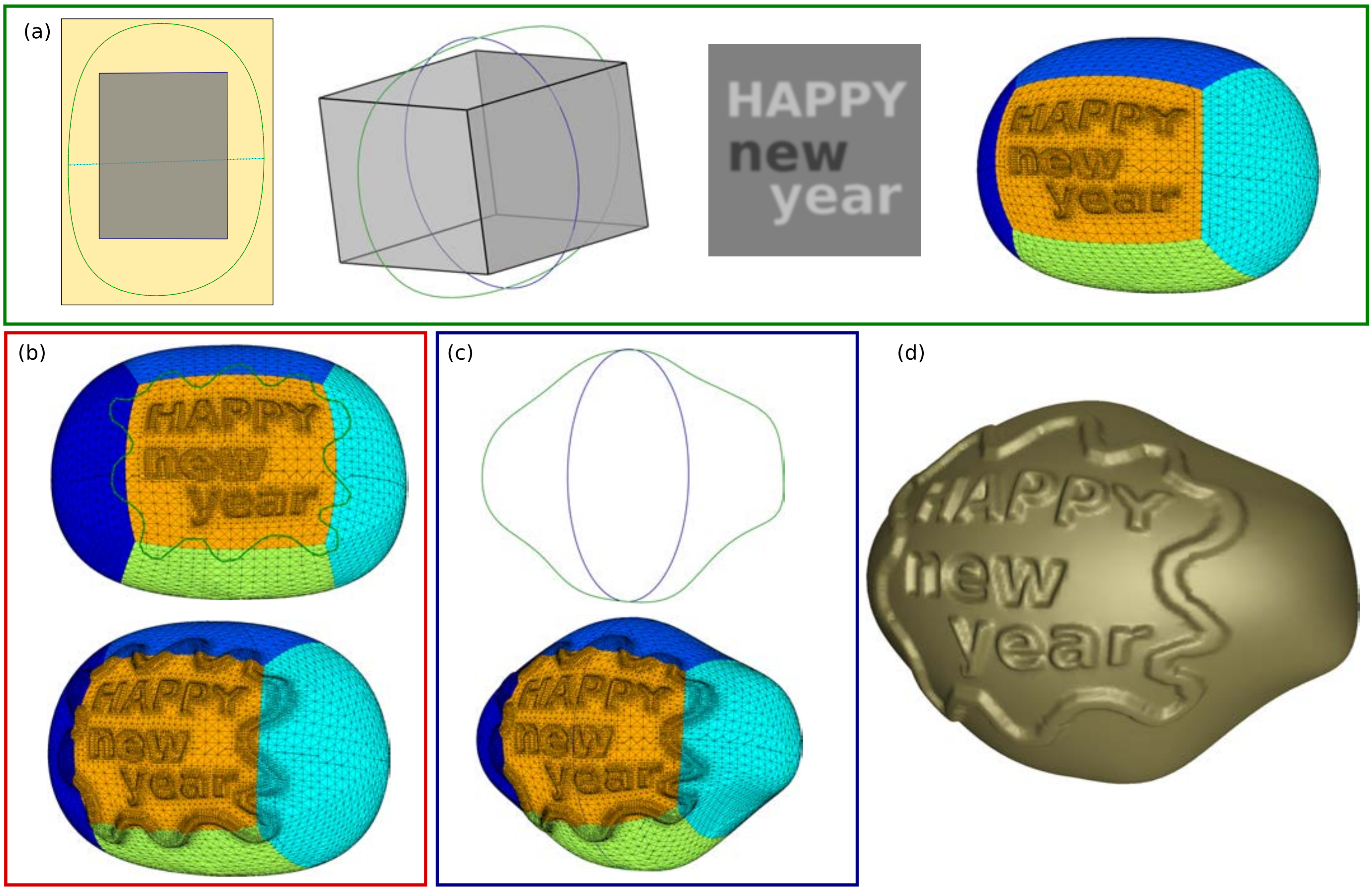


Figure 13: Steps to model a party balloon using DASS.

we have three possible *initial shape descriptors* which are, in our case, 2D parametric representations. The *Initial shape descriptor* of (1) is a heightmap extracted from the input triangle soup; of (2) is a Coons Surface [38], and of (3) is a convex sum of two horizons. The *base mesh* can be easily constructed as a rectangle from the extremities of these 2D parametric *initial shape descriptors*.

Base meshes are sculpted into a final mesh through operators that define the rules of adaptation and refinement of the 4-8 mesh. These operators are based on the initial shape descriptors or are sketch-based. The sketch-based operators of the GLaM system are good examples of the flexibility of surface representations as proposed in our framework. Each sketch-based operator is implemented independently and can have its own internal representation. To perform their deformations, each operator modifies its internal representation and provides rules to adapt and manipulate the 4-8 meshes. Besides the mesh, operators have different inputs such as filtered information from keyboard and mouse containing which surface and face (triangle) have been clicked. The GLaM system enables the combination of different operators to create more complex ones. For instance, a refinement of the mesh may be necessary by several different operators. Instead of implementing the same refinement for all operators, a refinement operator can be implemented and composed with the others.

Since the main purpose of this paper is to discuss the proposed framework we will not give many details about each implemented operator. All technical details of the system can be found in [39]. Following, we overview the main operators of the GLaM prototype to illustrate better the versatility of the proposed framework.

- *Topology Repair Operator* enables the user to create or delete holes on the horizons by texture manipulation. This operator is a good example of combination of simple operators, the first allows for the users modify *hole texture* using brushes like an image, after they are satisfied with the result other two operators are used, one to refine the mesh around the holes and other to remove the vertex creating the final mesh with the desired topology (Figures 15).

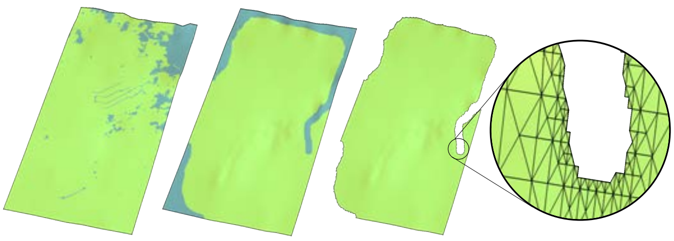


Figure 15: Topology repair operator. Left to right: original mesh, after *hole texture* edition, and final mesh.

- *Feature Augmentation and Horizon Fault Deformation Operators* create deformations using a set of sketched curves. These operators deform only the selected area using a parametric representation based on the distance to strokes to create final effects. The main differences between them are the meaning of the lines and the Horizon Fault operator changes the mesh topology (Figures 16).

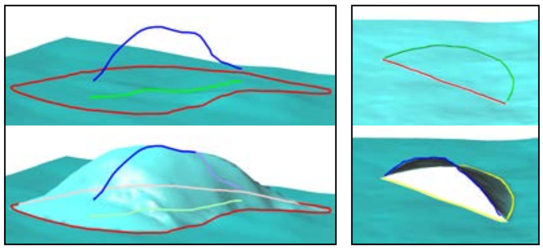


Figure 16: Left: Feature Augmentation and right: Horizon Fault Deformation.

- *Magnetic Operator* is an operator created to improve a common task in traditional horizon extracting work flow, where the experts select a voxel to be used as a seed in a growing segmentation algorithm, resulting in a horizon patch. The magnetic operator uses a pre-segmented volume to snap a hole to the closest horizon patches, having the meaning of many seeds placed at the same time (Figures 17).

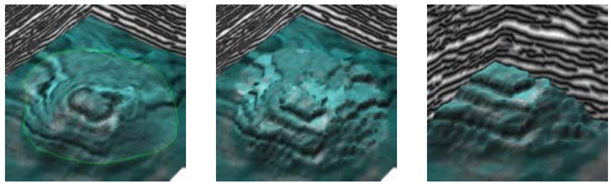


Figure 17: Magnetic Operator with the pre-segmented volume.

- *Horizon Convex Sum and Coons Surface Operators* create new surfaces inside the seismic volume. The first one uses 2 others horizons to create one between them. The latter allows the expert to draw strokes on the seismic data then it uses that to create a Coons surface following the sketches (Figures 18).

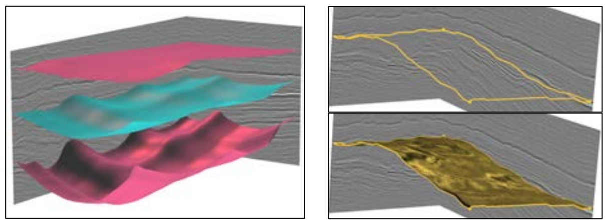


Figure 18: Left:Horizon convex sum creates the surface at middle. Right: strokes and final coons surface.

It is important to remark that each of the presented operators has its own internal representation, such as Coons patch, RBF implicit, and height map. This flexibility along with the proposed framework enables us to build this system following the expert's desiderata. Moreover, the GLaM system received positive feedback from our collaborators in the reservoir modeling domain. The main observations were the usefulness of the horizon fault deformation operator and the magnetic and smooth operators combined. Some improvements were also suggested specially for better fault modeling and navigation. It is important to note that GLaM is an illustrative example of how the proposed framework can be used to create different sketch-based applications.

7. Conclusion and Future Work

We have presented two sketch-based systems to illustrate the flexibility of our framework. The adaptive mesh plays a central hole in this framework enabling rapid updates with local control. This work opens many interesting venues. One of the natural next steps is to use the framework in different domains and applications.

DASS system leaves many interesting open questions. One important example of a problem that demands further research is the base mesh. For instance, we implemented a semi-automatic approach in which the user places the vertices to approximate the geometry and topology, followed by the base mesh creation in the space. This approach achieves good results, but it only allows us to work in a single plane. Since the base mesh is responsible for the topology of the final model, we are restricted to topologies that can be handled in one plane. We plan to explore two approaches for the base mesh problem. Firstly, we intend to transport the actual semi-automatic solution to 3D, letting the user handle boxes directly in space. The main challenge of this approach is developing an effective interface. The other approach is to use a mesh simplification, for instance the method presented by Daniels et al. [40]. Although this approach is automatic, it starts with a dense mesh; we must then exchange the problem of how to find a base mesh for the problem of how to create a mesh with the correct topology.

We developed a theory to construct atlas which is responsible to control the local edition of the model. The label theory developed gives a constructive algorithm with guarantees to create a partition over the set of faces enabling an atlas structure for stellar adaptive meshes. However, there is much more to be done in this problem. We aim to develop tools (mathematical and computational) to handle the scale of the atlas, an interface to control predefined height maps, and algorithms to split the atlas if it has a high level of deformation in relation to the surface.

Acknowledgements

We would like to thank our colleagues for their useful discussions and advice, in particular to Nicole Sultanum. We also thank the anonymous reviewers for their careful and valuable comments and suggestions. This research was supported in part by the NSERC / Alberta Innovates Technology Futures (AITF) / Foundation CMG Industrial Research Chair Program in Scalable Reservoir Visualization and by grants from the Brazilian funding agencies CNPq and CAPES / PDEE.

Appendix A. Building Atlas

In this Section we construct the theoretical framework to build an atlas using a label function over the vertices of a mesh. We work with a general description of adaptive surfaces, based on *stellar subdivision grammars* [23]. Our choice of parametric representation, the 4-8 mesh developed by Velho [2], is an example of application of this grammar. The atlas defined using vertices of the mesh has the following advantages: it is compact and simple; it naturally classifies edges as inner and boundary; and it is suitable to work with dynamic adaptive meshes.

Appendix A.1. Vertex-Map

As aforementioned we need an adaptive mesh to represent the high-frequency details. However, when we do one refinement step in a mesh, new elements (vertices, edges, faces) are created; then, we need to update the atlas. We propose a solution to construct and update an atlas using the natural structure of adaptive surfaces, using a simple label scheme for 4-8 mesh. Each vertex is labeled as inner vertex of a specific chart or as a boundary; that means if we have N charts there are $N+1$ possible labels. The $4-8$ mesh uses stellar operators (Figure 2), subsequently, we developed rules to update the atlas when these operators are used.

First of all we formalize the concept of the *regular labeled mesh*. After that we use these definitions to build an atlas with guarantees for adaptive surfaces that uses stellar subdivision operators.

Definition 1. *A mesh $M = (V, E, F)$ is* k-labeled *if each vertex $v \in V$ has a label $L(v) \in \{0, 1, 2 \ldots, k\}$, i.e., if there is $L : V \rightarrow \{0, 1, 2 \ldots, k\}$. L is called* k-label function. *If $L(v) = i \neq 0$, then v is an* inner-vertex *of the chart c_i; if $i = 0$, v is a* boundary-vertex.

Definition 2. *A face $f \in M$, is* regular k-labeled *or* rk-face *if there is $v \in f$ with $L(v) \neq 0$ and $\forall\ v_1, v_2 \in f$ such that $L(v_1) \neq 0 \neq L(v_2) \Rightarrow L(v_1) = L(v_2)$. A mesh is* regular k-labeled *(or* rk-mesh*) when all their faces are rk-faces. The function $L : V \rightarrow \{0, 1, 2 \ldots, k\}$ that produces a rk-mesh is called a* regular k-label *or* rk-label.

Observe that an edge in a regular k-labeled mesh has vertices with the same label or one of them has label 0. If the edge has at least one vertex v such that $L(v) = i \neq 0$; we call it an *inner-edge* of the chart c_i or $L(e) = i$; if it has the two vertices labeled as zero it is a *boundary-edge* or $L(e) = 0$.

Proposition 3. *A regular k-label function induces a partition on the set of faces.*

Proof. Let $M = (V, E, F)$ be a rk-mesh. Define the set $F_i = \{f \in F | \ \exists \ v \in f$ such that $L(v) = i\}$, $i \in \{1, 2, \ldots, k\}$. By definition 2 every $f \in F$ has at least one v with $L(v) \neq 0$ then:

$$\bigcup_{i=1}^{k} F_i = F,$$

and if there is more than one $v \in f$ such that $L(v) \neq 0$ then all such vertices will have the same value of L, i.e., the face belongs to only one F_i, so we conclude:

$$F_i \cap F_j = \varnothing \quad \text{if} \quad i \neq j.$$

$\square$

This proposition allows us to define a collection of charts over a *rk*-meshes. We say that a face f is in the chart c_i ($L(f) = i$) if there is at least one $v \in f$ such that $L(v) = i$. However for our application it is not enough to have a static map because our mesh is adaptive. Hence we need rules to assign a L value to the new vertices created by the refinement step.

We study how to update the atlas after applying one of the stellar operators described in Section 3: i.e., edge and face split, and their inverse edge face weld (Figure 2). Observe that, the stellar subdivision operators (split) add only one vertex, thus to update the atlas we only need rules to label the new vertex v_n.

- Face Split – when the face f is split we define:

$$L(v_n) = L(f) \tag{A.1}$$

- Edge Split – when the edge e is split we define:

$$L(v_n) = L(e) \tag{A.2}$$

Proposition 4. *A stellar subdivision step using the previous rules on a rk-mesh M produces M' that is a rk-mesh too.*

Proof. First, case we split a face f we create a new vertex v_n and 3 news faces (f_1, f_2, f_3), since M is a *rk*-mesh the equation (A.1) is well defined and $L(v_n) = i \neq 0$. To proof that f_1, f_2, f_3 are *rk*-faces, we observe that $v_n \in f_1 \cap f_2 \cap f_3$ then they have at least v_n with $L(v_n) \neq 0$. And, since f is a *rk*-face for all $v \in f$, $L(v)$ is 0 or i, and for $j = \{1, 2, 3\}$, $v \in f_j \Leftrightarrow v = v_n$ or $v \in f$, we conclude if $v \in f_j \Rightarrow L(v) = 0$ or $L(v) = i$, i.e, f_j is a *rk*-face.

The edge split creates four new faces $f_j, j = 1, 2, 3, 4$. Note that the operator edge split subdivides two faces. Lets name these faces *west-face* (f^w) and *east-face* (f^e); and their opposite vertex as v_e and v_w respectively. i.e., $v_* \in f^*$ and $v_* \notin e$.

If e is an inner-edge then for at least one of its vertices $L(v) = i \neq 0$. Since e is in f^w and f^e we have $L(f^w) = L(f^e) = i$ it implies that if $v \in f^w \cup f^e$ then $L(v) = i$ or $L(v) = 0$. As a result when we split a inner-edge we have $L(v_n) = i$ and $v_n \in \bigcap_j f_j$ and $v \in f_j \Rightarrow v \in f^w \cup f^e$ or $v = v_n$, then f_j is a *rk*-face.

If e is a boundary-edge and f^w and f^e are *rk*-faces t $L(v_e) \neq 0$ and $L(v_w) \neq 0$. Since $v_w \in f_j$ or $v_e \in f_j$ we have one $v \in f_j$ such that $L(v) \neq 0$, then we conclude that f_j is *rk*-face. $\square$

The simplification step of an adaptive mesh is very important to our application, because when the user changes the sketches the mesh is dynamically updated that implies that the two steps (refinement and simplification) are done. Starting with a *rk*-mesh (level 0) and perform n refinement steps, then to any $m \leq n$ simplification steps we have a *rk*-mesh. It is easy to see because when a refinement step is done we do not change the value of the vertices of the current level j, thus when we do the inverse operator to simplify only vertices of level $j + 1$ are deleted so then the L function over faces is well defined in level j.

To create a *rk*-mesh using our base-mesh, i.e., to create the M_0, we label all vertices of the base-mesh as boundary ($L(v) = 0$) and split each face, the new vertex added is labeled with a new value not 0. After that each face of the base-mesh generates a new chart into the atlas, i.e., if the base mesh has k faces the atlas has k charts. In Figure A.19 we illustrate the process to create a mesh M_0 that is a $r2$-mesh and three refinement steps.

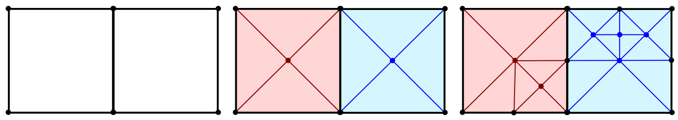


Figure A.19: Creating a $r2$-mesh and refinements. Left to right: the base-mesh, M_0 which is $r2$-mesh, and after 3 refinement steps: M_3. Black elements are boundary ($L(\cdot) = 0$), blue elements are into chart c_1 ($L(\cdot) = 1$), and red elements are into chart c_2 ($L(\cdot) = 2$).

Appendix A.2. Creating a Manifold Structure

Now we have a partition over the surface and we know how to refine and simplify the mesh respecting this partition. However, we do not have all elements of an atlas, we need to define open sets Ω_i and homeomorphisms ϕ_i. First of all, we overload the notation for chart; $c_i \in \mathcal{A}$ has two meanings, the first one is a set of faces, edges and vertices, used in previous section. The second one is the parametric space $[0, 1]^2 \subset \Omega_i$; more precisely, we say a point of M belongs to a chart c_i if we can write this points in Ω_i coordinates and its coordinates are in $[0, 1]^2$. At this point all vertices v of M have at least two geometrical information, its coordinates in $\mathbb{R}^3$ and, its coordinates in at least one Ω_i. The notation v^i is used to be clear when we are using v in coordinates of Ω_i, how to recover this information we will discuss later. We start an atlas setting the four vertices of the base-mesh face $f_i = \{v_1, v_2, v_3, v_4\}$ to be the boundary of c_i, i.e., the local coordinates in Ω_i of these vertices are: $v^i_1 = (0, 0)$, $v^i_2 = (1, 0)$, $v^i_3 = (1, 1)$, $v^i_4 = (0, 1)$.

Since M is an adaptive mesh and now it has two geometrical aspects, its coordinates in $\mathbb{R}^3$ and in $\mathcal{A}$, we need rules to update this information. When we split an edge $e = \{v_1, v_2\}$ we get its middle point v_m and project it on S and if $e \in c_i$ then $v^i_m = (v^i_1 + v^i_2)/2$. A projection $\Pi_S(p)$ of a point p on a surface S is well defined if it is in the tubular neighborhood of S. We are assuming that Π_S is well defined for all points on a edge in M. That is true when the vertices of the base mesh start close to S.

To build the homeomorphisms we also will use the $\Pi_M(p)$, the projection of $p \in S$ on M, and again we are supposing that the mesh approximates well the surface. If a point $p^i \in c_i$ then there is a face $f^i = \{v^i_1, v^i_2, v^i_3\}$ such that p^i is a convex combination of its vertices. More precisely $p^i = \sum_{k=1}^{3} \alpha_k v^i_k$ with $\alpha_k > 0$, $\sum_{k=1}^{3} \alpha_k = 1$. So then we define:

$$\phi_i(p^i) = \Pi_S\left(\sum_{k=1}^{3} \alpha_k \phi_i(v^i_k)\right).$$

Specifically when we split an edge e, which belongs to c_i, $e^i = \{v_1^i, v_2^i\}$ we have:

$$\phi_i(v_n^i) = \Pi_S\left(\frac{\phi_i(v_1^i) + \phi_i(v_2^i)}{2}\right). \quad \text{(A.3)}$$

Proposition 5. *For all i, j and $v \in V$ such that $v \in c_i$ and $v \in c_j$ holds $\phi_i(v^i) = \phi_j(v^j)$.*

Proof. We proof that proposition by induction in all levels of refinement of M. When we start the charts c_i and c_j all edges that are in their boundary belongs to the base mesh, if $v \in c_i$ and $v \in c_j$ then $\phi_i(v^i) = \Pi_S(v) = \phi_j(v^j)$, by construction. Now suppose the Proposition 5 is true for all v with level less or equal the current level. Observe that by (A.1) and (A.2) a boundary vertex v is created only when a boundary edge is split, consequently by (A.3) and induction hypothesis holds:

$$\begin{aligned}\phi_i(v^i) =& \Pi_S\left(\frac{\phi_i(v_1^i) + \phi_i(v_2^i)}{2}\right)\\ =& \Pi_S\left(\frac{\phi_j(v_1^j) + \phi_j(v_2^j)}{2}\right) = \phi_j(v^j).\end{aligned}$$

□

To define the inverse of ϕ_i we use the projection Π_M, the idea is to project the point on the mesh, identify which face it is projected and use the barycentric coordinates to define it coordinates in Ω_i. More precisely, let $\Pi_M(p) = \sum_{k=1}^{3} \alpha_k v_k$, with $\alpha_k > 0$, $\sum_{k=1}^{3} \alpha_k = 1$ and $f = \{v_1, v_2, v_3\}$ where $L(f) = i$, then we have:

$$\phi_i^{-1}(p) = \sum_{k=1}^{3} \alpha_k v_j^i. \quad \text{(A.4)}$$

Since we are supposing that M is close to S we have ϕ and ϕ^{-1} well defined, i.e., $\phi_i \circ \phi_i^{-1}(p) = p$ and $\phi_i^{-1} \circ \phi_i(p^i) = p^i$ for all $p \in S \cap \phi_i(c_i)$ and $p^i \in c_i$.

To build the height maps consistently we need to know how to write inner-points of c_i in Ω_j coordinates when c_i and c_j are neighbors, i.e., we need be able to write a point $p^i \in c_i$ in Ω_j coordinates when c_i and c_j have common vertices. Since we started our chart with quadrangle domains we use the approach develop by Stam [41] to convert p^i to p^j. The author recovers the relative affine coordinates of Ω_i to Ω_j, he achieves that by matching commons edges of c_i and c_j.

Appendix B. Sketching over the Surface

To enable the users to augment the model we freeze the camera and they draw polygonal curves over the surface. These strokes are transported to atlas $\mathcal{A}$ where they are used to define the height map, we name these projected curves as *height curves*. To transport the curves to $\mathcal{A}$ we project the curve points directly on M, identifying which face they were project, and use their barycentric coordinates to transport them to the correspondent c_i. If the line segment pq starts in the chart c_i and ends in the chart c_j then to guarantee continuity we write $p^i q^i$ and find its point that is over the boundary of c_i and add this point to the height-curve. We do the same thing to the segment $p^j q^j$. In Figure B.20 we show the result of this process.

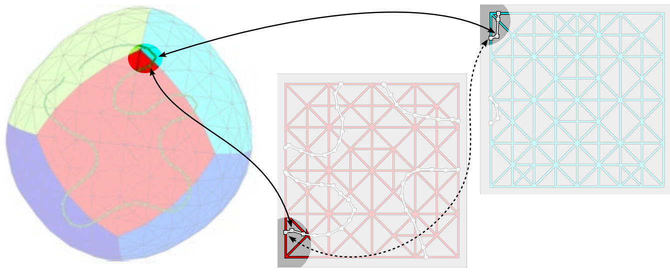


Figure B.20: Sketch over surface and the curve transported to $\mathcal{A}$. The two solid arrows show points on M that are transported to $\mathcal{A}$, the dashed arrow shows points that are created in the chart boundaries to guarantee hight-curve continuity.

After all, we have a height map h_u^i for each chart c_i that can be sketched by the user. We can compose this height map with another, such as a gray depth image h_d^i, for example to obtain a final height at $p \in M$ adding the heights, $h_p = h_d^i(p^i) + h_u^i(p^i)$. Then, we have $D(p) = h_p N_p$ where N_p is it normal at p. Thus we complete the formulation of the final surface: $S = \Xi + D(\Xi)$; specifically, for all $p \in M$ we have $\tilde{p} = p + h_p N_p$.

References

[1] Olsen L, Samavati FF, Costa Sousa M, Jorge J. Sketch-based modeling: a survey. Comp & Graph 2009;33(1):85–103.

[2] Velho L. A dynamic adaptive mesh library based on stellar operators. Journal of graphics, gpu, and game tools 2004;9(2):21–47.

[3] Igarashi T, Matsuoka S, Tanaka H. Teddy: a sketching interface for 3D freeform design. In: SIGGRAPH '99. ACM; 1999, p. 409–16.

[4] Nealen A, Igarashi T, Sorkine O, Alexa M. Fibermesh: designing freeform surfaces with 3D curves. ACM Trans Graph 2007;26(3):41–50.

[5] Gingold Y, Igarashi T, Zorin D. Structured annotations for 2D-to-3D modeling. ACM Trans Graph 2009;28(5):148.

[6] Vital Brazil E, Macêdo I, Costa Sousa M, de Figueiredo LH, Velho L. Sketching variational Hermite-RBF implicits. In: SBIM '10. 2010, p. 1–8.

[7] Orbay G, Kara LB. Sketch-based surface design using malleable curve networks. Comp & Graph 2012;6(8):916 –29.

[8] Wyvill B, Guy A, Galin E. Extending the CSG tree-warping, blending, and boolean operations in an implicit surface modeling system. CGF 1999;18(2):149–58.

[9] Sederberg TW. Piecewise algebraic surface patches. Computer Aided Geometric Design 1985;2(1–3):53–9.

[10] Bloomenthal J, Shoemake K. Convolution surfaces. In: SIGGRAPH '91. ACM; 1991, p. 251–6.

[11] Kara LB, Shimada K. Sketch-based 3D-shape creation for industrial styling design. IEEE Comput Graph Appl 2007;27(1):60–71.

[12] Cherlin JJ, Samavati F, Costa Sousa M, Jorge JA. Sketch-based modeling with few strokes. In: SCCG '05. New York, NY, USA: ACM; 2005, p. 137–45.

[13] Nasri A, Karam WB, Samavati F. Sketch-based subdivision models. In: SBIM '09. ACM; 2009, p. 53–60.

[14] Hnaidi H, Guérin E, Akkouche S, Peytavie A, Galin E. Feature based terrain generation using diffusion equation. CGF 2010;29(7):2179–86.

[15] Karpenko O, Hughes JF, Raskar R. Free-form sketching with variational implicit surfaces. CGF 2002;21:585–94.

[16] Amorim R, Vital Brazil E, Samavati F, Sousa MC. 3d geological modeling using sketches and annotations from geologic maps. In: Proceedings of the 4th Joint Symposium on Computational Aesthetics, Non-Photorealistic Animation and Rendering, and Sketch-Based Interfaces

and Modeling. SBIM '14; New York, NY, USA: ACM. ISBN 978-1-4503-3018-3; 2014, p. 17–25. doi:10.1145/2630407.2630411.

[17] Rodrigues De Araujo B, Jorge J. A calligraphic interface for interactive free-form modeling with large datasets. In: Computer Graphics and Image Processing, 2005. SIBGRAPI 2005. 18th Brazilian Symposium on. 2005, p. 333–40. doi:10.1109/SIBGRAPI.2005.2.

[18] Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel HP. Multi-level partition of unity implicits. ACM Trans Graph 2003;22(3):463–70. doi:10.1145/882262.882293.

[19] Schmidt R, Wyvill B, Costa Sousa M, Jorge JA. ShapeShop: Sketch-based solid modeling with blobtrees. In: SBIM '05. 2005, p. 53–62.

[20] Bernhardt A, Pihuit A, Cani MP, Barthe L. Matisse: Painting 2D regions for modeling free-form shapes. In: SBIM '08. 2008, p. 57–64.

[21] Vital Brazil E, Amorim R, Costa Sousa M, Velho L, de Figueiredo L. A sketch-based modeling framework based on adaptive meshes. In: Graphics, Patterns and Images (SIBGRAPI), 2014 27th SIBGRAPI Conference on. 2014, p. 17–24. doi:10.1109/SIBGRAPI.2014.16.

[22] Lickorish WBR. Simplicial moves on complexes and manifolds. In: Proceedings of the Kirbyfest (Berkeley, CA, 1998); vol. 2 of *Geom. Topol. Monogr.* Geom. Topol. Publ., Coventry; 1999, p. 299–320 (electronic).

[23] Velho L. Stellar subdivision grammars. In: SGP'03. Eurographics Association; 2003, p. 188–99.

[24] de Goes F, Goldenstein S, Velho L. A simple and flexible framework to adapt dynamic meshes. Comp & Grap 2008;32(2):141–8.

[25] Velho L, Gomes J. Variable resolution 4-k meshes: Concepts and applications. Computer Graphics Forum 2000;19(4):195–212. doi:10.1111/1467-8659.00457.

[26] Kettner L. Using generic programming for designing a data structure for polyhedral surfaces. Computational Geometry 1999;13(1):65 – 90.

[27] Velho L, Zorin D. 4-8 subdivision. Comput Aided Geom Des 2001;18(5):397–427. doi:10.1016/S0167-8396(01)00039-5.

[28] Blinn JF. Simulation of wrinkled surfaces. In: Proc. of SIGGRAPH '78. ACM; 1978, p. 286–92.

[29] Krishnamurthy V, Levoy M. Fitting smooth surfaces to dense polygon meshes. In: SIGGRAPH '96. ACM; 1996, p. 313–24.

[30] Lee A, Moreton H, Hoppe H. Displaced subdivision surfaces. In: Proc. of SIGGRAPH '00. ACM; 2000, p. 85–94.

[31] Zorin D. Modeling with multiresolution subdivision surfaces. In: SIGGRAPH 2006. ACM; 2006, p. 30–50.

[32] Grimm C, Zorin D. Surface modeling and parameterization with manifolds. In: ACM SIGGRAPH 2006 Courses. SIGGRAPH '06; New York, NY, USA: ACM; 2006, p. 1–81.

[33] do Carmo MP. Differential geometry of curves and surfaces. Englewood Cliffs, N. J.: Prentice-Hall Inc.; 1976.

[34] Maximo A, Velho L, Siqueira M. Adaptive multi-chart and multiresolution mesh representation. Computers & Graphics 2014;38(0):332 –40. doi:http://dx.doi.org/10.1016/j.cag.2013.11.013.

[35] Bloomenthal J. An implicit surface polygonizer. San Diego, CA, USA: Academic Press Professional, Inc.; 1994, p. 324–49.

[36] Velho L. Simple and efficient polygonization of implicit surfaces. Journal of graphics, gpu, and game tools 1996;1(2):5–24.

[37] Stander BT, Hart JC. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In: SIGGRAPH 2005 Courses. ACM; 2005,.

[38] Salomon D. Curves and Surfaces for Computer Graphics. Berlin, Germany: Springer-Verlag; 2006. ISBN 0-387-24196-5 , 0-387-28452-4 (e-book).

[39] Amorim R, Vital Brazil E, Patel D, Costa Sousa M. Sketch modeling of seismic horizons from uncertainty. In: SBIM'12. Eurographics Association; 2012, p. 1–10.

[40] Daniels J, Silva CT, Shepherd J, Cohen E. Quadrilateral mesh simplification. ACM Trans Graph 2008;27:9.

[41] Stam J. Flows on surfaces of arbitrary topology. ACM Trans Graph 2003;22:724–31.