DAG Amendment for Inverse Control of Parametric Shapes

ÉLIE MICHEL, LTCI, Télécom Paris, Institut Polytechnique de Paris, France
TAMY BOUBEKEUR, Adobe, France

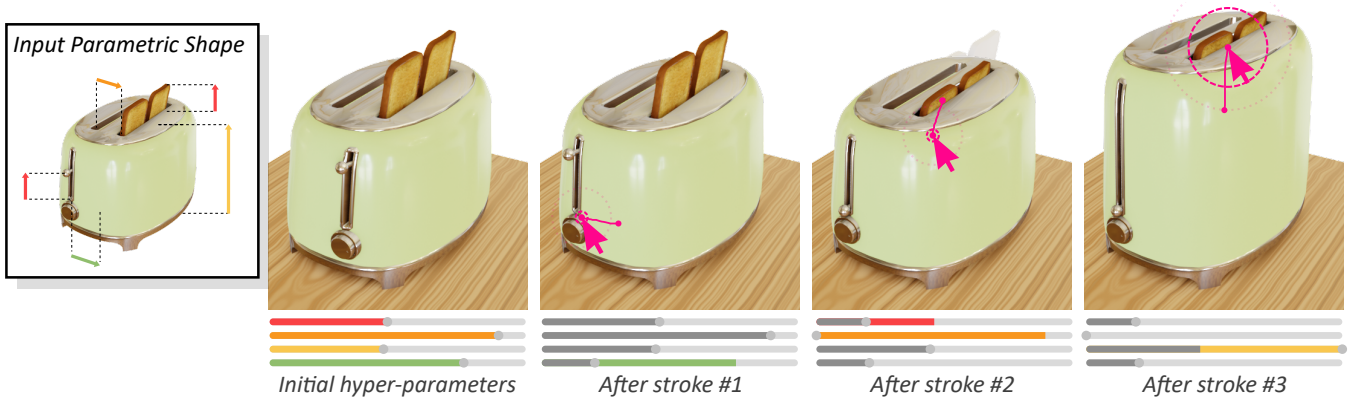

Fig. 1. Our method infers without any manual setup how to update the hyper-parameters of a parametric shape to comply with an intent expressed as a brush stroke on its visualization. This enables a more direct and intuitive interaction process than tunning individual sliders, at no extra cost for the shape's designer.

Parametric shapes model objects as programs producing a geometry based on a few semantic degrees of freedom, called hyper-parameters. These shapes are the typical output of non-destructive modeling, CAD modeling or rigging. However they suffer from the core issue of being manipulated only indirectly, through a series of values rather than the geometry itself. In this paper, we introduce an amendment process of the underlying direct acyclic graph (DAG) of a parametric shape. This amendment enables a local differentiation of the shape w.r.t. its hyper-parameters that we leverage to provide interactive direct manipulation of the output. By acting on the shape synthesis process itself, our method is agnostic to the variations of the connectivity and topology that may occur in its output while changing the input hyper-parameters. Furthermore, our method is oblivious to the internal logic of the DAG nodes. We illustrate our approach on a collection of examples combining the typical nodes found in modern parametric modeling packages – such as deformation, booleans and surfacing operators – for which our method provides the user with inverse control over the hyper-parameters through a brush stroke metaphor.

CCS Concepts: • **Computing methodologies** → **Shape modeling**; *Shape analysis.*

Additional Key Words and Phrases: Parametric Design, Direct Manipulation, Shape Modeling, Inverse Control, Shape Differentiation

ACM Reference Format:
Élie Michel and Tamy Boubekeur. 2021. DAG Amendment for Inverse Control of Parametric Shapes. *ACM Trans. Graph.* 40, 4, Article 173 (August 2021), 14 pages. https://doi.org/10.1145/3450626.3459823

Authors' addresses: Élie Michel, LTCI, Télécom Paris, Institut Polytechnique de Paris, 19 place Marguerite Perey, Palaiseau, 92120, France, elie.michel@telecom-paris.fr; Tamy Boubekeur, Adobe, 9 Rue de Milan, 75009, Paris, France, boubek@adobe.com.

0730-0301/2021/8-ART173 $15.00
https://doi.org/10.1145/3450626.3459823

1 INTRODUCTION

A *parametric shape* is a shape driven by a few input values that we call *hyper-parameters*. Such a shape is the result of a process commonly referred to as *non-destructive modeling*, where the shape's *designer* intentionally leaves a few hyper-parameters publicly available to the shape's *end-user* (Figure 1).

Mathematically, a parametric shape is a function $\mathcal{F}$ mapping the hyper-parameters $\boldsymbol{\pi}$ to a static geometry $G \subset \mathbb{R}^3$ called an *instance* of $\mathcal{F}$. The set of hyper-parameters is a subset Π of $\mathbb{R}^n$. In our case, G is represented as a 3D surface mesh. We assume a certain degree of regularity of this function since it is intended for human interaction.

The core problem of parametric shape manipulation by the end user is that it does not occurs in the 3D space but rather in the hyper-parameter space, which is roughly a list of sliders in a UI. Yet, the *intent* of the end-user is often more naturally expressed in the 3D space. This mismatch results in a trial and error loop that is dampening the creation process.

In some contexts like animation, this issue is such a deal-breaker than riggers are asked to equip shapes with *manipulators*, which are handles lying in the 3D space and whose transform drives some hyper-parameters. But creating these requires extra time and skills on top of the design of the parametric shape itself.

In the workflow we target (Figure 2), a technical designer first builds an object using non-destructive modeling tools, and simply exposes some hyper-parameters without minding manipulators, thus defining $\mathcal{F}$. Lastly, the end-user edits the hyper-parameters to customize the object. In between, our DAG amendment automatically modify $\mathcal{F}$ so that besides the hyper-parameters sliders the end user may directly manipulate the shape in the 3D view.

Animation set-up is also a use case complying with the aforementioned mathematical definition. The *set-up* of a geometry consists in

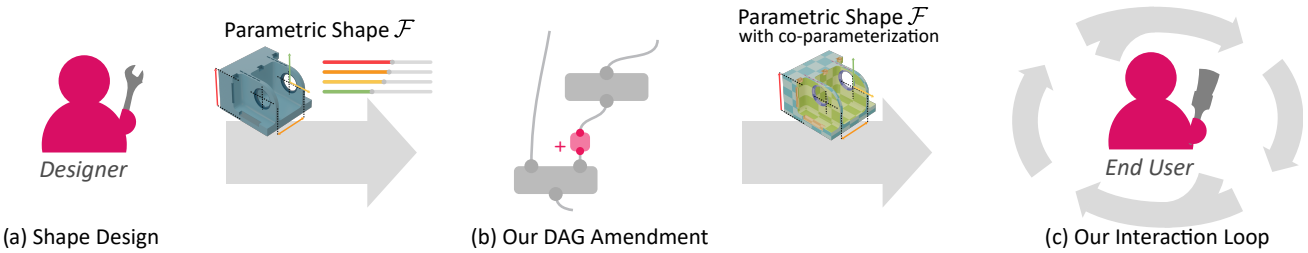

Fig. 2. A common creation workflow separates (a) the designer of a parametric object from (c) its end user. The former handles technical issues for the latter to be fully dedicated to more artistic and intuition based matters (e.g. animation, staging). Our method improves this end interaction without extra effort from the designer by (b) automatically inserting a few nodes to the parametric shape's graph representation (DAG) produced by the designer.

attaching it (*skinning*) to a coarse control shape (often called *skeleton*) whose topology and degrees of freedom are carefully designed (a process called *rigging*) to restrain its deformation space to plausible poses only. This design process is slightly different because one first creates one particular instance and only then transforms it into a parametric shape, whereas a non-destructive modeling workflow directly creates a shape that is parametric.

Nevertheless these two processes result in the same nature of objects, namely parametric shapes described as a Direct Acyclic Graph (DAG) as shown in Figure 11, and since our contribution lies in the manipulation of parametric shapes rather than their design, it applies to any of these cases without loss of generality.

In this paper, we introduce a method for interpreting user inputs expressed in the 3D viewport as changes in the hyper-parameter space without any extra controller setup. Our key contributions are:

- an amendment operator for the parametric shape graph yielding a co-parametrization which associates points across hyper-parametric variations and thus makes it possible to measure point-wise shape jacobians efficiently;
- a non-linear filtering mechanism acting on the the resulting jacobians to both regularize and sparsify the shape optimization, fostering hyper-parameters whose behavior comply with the scale of the user brush.

Our approach is **(i) automated** – no extra effort is required from the shape's designer; **(ii) flexible** – it is possible to locally override the automated process whenever it is needed, and falling back to other methods remains possible at any time; **(iii) non-invasive** – it can fit into existing parametric shape engines without requiring to rewrite the content of generation operations.

Although we tried to remain agnostic in the underlying DAG engine – in particular we do not require it to be automatically differentiable – we make the assumptions that the operations (a) process only mesh-based data (b) can transmit extra attributes of the kind of texture coordinate from their input to their output and (c) label the output geometry with a duplicate index (that we denote j) when they duplicate input geometry.

2 RELATED WORK

Parametric shapes are a natural way to represent 3D objects in a space of lower dimension and higher meaningfulness than for instance raw vertex positions. Many works output objects that are in effect parametric shapes.

The object of (auto-)rigging methods is to transform a given static geometry into a parametric shape. In this context, there is often a distinction made between the *kinematic parameters*, which are the raw degrees of freedom of a coarse skeleton or control cage and the *rig space* made of higher-level semantic values on which the animator has control (called respectively α and β in [Capell et al. 2005]). It is the latter, publicly exposed to the end user, that we call hyper-parameters.

Although kinematic parameters can be estimated using geometric analysis, based for instance on path finding [Tsao and Fu 1984; Wade and Parent 2000] or Reeb graphs [Aujay et al. 2007; Hilaga et al. 2001; Lazarus and Verroust 1999; Pascucci et al. 2007; Tierny et al. 2006], determining semantic hyper-parameters requires a domain specific prior.

Some works address the problem of fitting an existing rig space to a new input geometry [Ali-Hamadi et al. 2013; Avril et al. 2016; Baran and Popović 2007; Li et al. 2010], typically for motion re-targeting. Others pick parts from different examples like Frankenrigs [Miller et al. 2010]. For more examples [Rumman and Fratarcangeli 2016] surveys auto-skinning methods.

The other way of injecting prior knowledge is through the use of machine learning techniques, that have been applied to most common use cases of rigging like human bodies [Anguelov et al. 2005; Liu et al. 2019; Loper et al. 2015; Osman et al. 2020], faces [Blanz and Vetter 1999; Li et al. 2017; Vesdapunt et al. 2020] or even generic shapes [Xu et al. 2019]. [Holden et al. 2015] learns how to place semantic manipulators to reach a given kinematic pose. Generally a subset of the hyper-parameters gives the morphologic identity of the character and the reminder is related its posing and animation.

In the context of machine learning, this lower dimensional abstract space is often called a *latent space* or *embedding space*. Its *decoder*, that generates a geometry given a particular point of the latent space, is an example of parametric shape. Latent spaces often have more dimensions than a human designer can handle though, so [Chiu et al. 2020] and [Abdrashitov et al. 2020] try to reduce these dimensions, bringing it even closer to our conception of hyper-parameter.

The aforementioned body of work focuses on parametric shapes representing organic shapes, but another common use case for them is CAD modeling. [Mitra et al. 2012] surveys methods that can extract a semantic construction graph from the auto-similarities

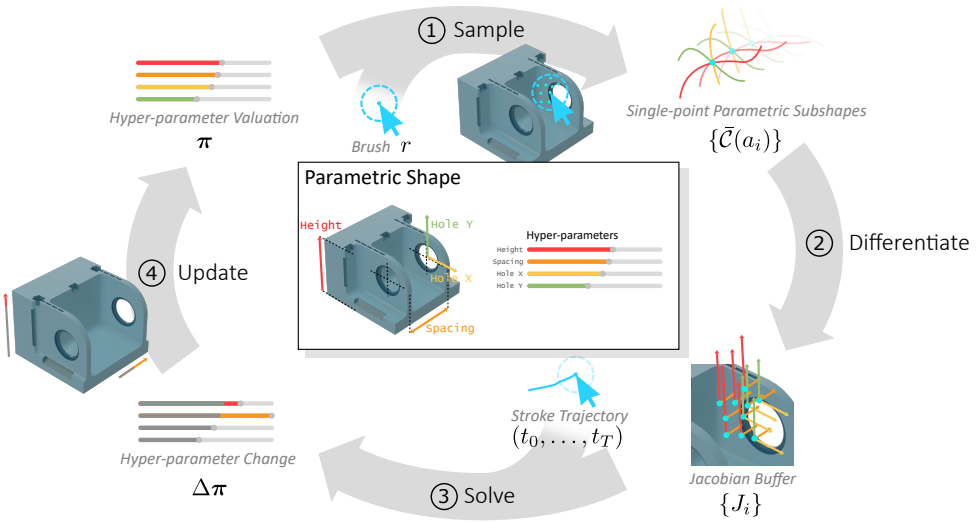

Fig. 3. Overview of our interaction loop. At the beginning of a stroke, points are sampled around the user cursor to extract co-parameters a_i and so single-point parametric shapes $\bar{C}(a_i)$. Each of these is differentiated to measure their jacobians, which are then provided to the solver. Confronting jacobians to the trajectory of the cursor, the solver determines the update to apply to the hyper-parameters.

that a given static geometry presents [Kalojanov et al. 2016; Liu et al. 2015]. More generally, inverse procedural modeling [Aliaga et al. 2016] intends to represent existing geometries as generation programs. However we only consider the ones that can be evaluated in interactive time; inverse control of slower procedural generators raises challenges of a different kind [Talton et al. 2009, 2011].

CSGNet [Sharma et al. 2018] learns a CSG tree from a static geometry using reinforcement learning. [Ganin et al. 2018] considers that the input to a renderer is a *program* rather than a *model* and trains its machine learning model so. [Jones et al. 2020] learns shape generation programs (i.e. parametric shapes).

Hyper-parameter space is sometimes called latent space, embedding space, rig space [Hahn et al. 2012], design space [Talton et al. 2009], animation space [Merry et al. 2006], abstract parameters [Capell et al. 2005].

Inverse Control. The most common case of inverse control in computer graphics is Inverse Kinematics (IK), that originates from robotics [Saab et al. 2013] and has been extensively studied for skeletal animation [Aristidou et al. 2018]. Although their announced scope is often limited to trees of rigid transforms, the methods proposed in the IK literature may be applied to more complex mesh deformations like human face posing [Lewis and Anjyo 2010], the main input requirement being to have access to the jacobian matrix of the action of hyper-parameters onto a point of the mesh. With this jacobian at hand, methods have been developed to solve robustly the inverse problem [Deo and Walker 1992] and account for boundaries of the hyper-parameters [Baerlocher and Boulic 1998; Raunhardt and Boulic 2007]. Our focus being on interactive design, we are interested in online solutions.

The main issue we tackle in this paper consists in defining a reliable way to measure this jacobian matrix even when the connectivity of the geometry changes and so a vertex' index cannot be used to identify a point.

Interactive mesh deformation. Deforming *raw* geometries that are not the output of an underlying parametric shape requires extra prior knowledge. Some methods try to maximize rigidity [Igarashi et al. 2005; Levi and Gotsman 2015], sometimes based on examples [Sumner et al. 2005; Wampler 2016]. Linear variational methods [Botsch and Sorkine 2008] deform the input by solving a linear system capturing the intrinsic properties of the mesh, enriched with direct control constraints coming in the form of vertex handles. Non-linear methods [Botsch et al. 2006; Sorkine and Alexa 2007] further develop this concept, to better preserve volumes and cope with large handle motions. Alternatively, linear blend skinning [Baran and Popović 2007; Jacobson et al. 2011] offers a scalable framework where no system is solved at run time, and the bulk of the shape analysis yielding the handles influence is located at the initial per-vertex weight computation.

We want to provide such direct control capabilities but our case differs significantly, as our a priori is the space of possible embeddings a parametric shape can undergo through variations of its hyper-parameters. This is somehow an extreme case of structure-aware shape processing [Mitra et al. 2014], although such method usually couples the user deformation (change of the hyper-parameters) with the extraction of symmetries [Kurz et al. 2014; Wu et al. 2014] or of coarse control structures from the geometric analysis of one [Bokeloh et al. 2012; Gal et al. 2009] or many similar shapes [Gadelha et al. 2020].

Improving interaction with parametric shapes has been explored by [Kelly et al. 2015] who automatically places the hyper-parameter

controllers in the 3D view, but the controllers themselves must have been hand-designed first.

Shape correspondence. We will see in Section 4 the need to identify points across multiple meshes of potentially varying connectivity, which is commonly referred to as *shape correspondence* or *cross-parameterization* [Kilian et al. 2007; Kraevoy and Sheffer 2004; Schreiner et al. 2004]. This consists in mapping each point from a shape to points that have the same *semantic* but in other shapes. [van Kaick et al. 2011] surveys a large variety of shape correspondence works, and more recent work even try to match dissimilar shapes [Hecher et al. 2018]. But this field focuses generally on offline registration of a small number of geometries, while we have to register a continuous infinity of meshes. Some works build correspondences for large amount of objects. [Mahmood et al. 2019] addresses the lack of consistent parameterization among datasets of human bodies, but is hand tuned for this very use case. [Leimer et al. 2017] creates a parametric shape by registering together a whole collection of shapes. Unfortunately these methods are offline and resource intensive. Furthermore there are no geometric features guaranteed that we may rely on to in general. For all these reasons, we adopt a quite different approach. Establishing a shape correspondence is a semantic operation, so we leverage the implementation of the parametric shape – the DAG – because its structure carries semantic information beyond what the output geometry shows.

Optimizations in hyper-parameter spaces. There are other cases of optimization in hyper-parameter spaces than IK. Such a process can be found in parametric architecture [Yang et al. 2011; Zhao et al. 2013], in particular to find values for which a form is constructible [Whiting et al. 2009]. This is also a way to make hand-crafted animation and physical phenomena coexist [Hahn et al. 2012], or even to determine hyper-parameters from reference photographs [Debevec et al. 1996]. Indeed, the machine learning literature contains a number of such examples, like the work by Zhang et al. [2020] for pose estimation. A close example to our work is [Umetani 2017]. Even though their parametric shape is a bit ad-hoc, they experience the need for a consistent parameterization, both for feeding 3D objects into a deep learning pipeline and for providing inverse control of the latent space by simply dragging vertices.

DAG Rewriting. Automatically editing the program that generates a geometry is used in the field of procedural generation. [Barroso et al. 2013] proposes to rewrite the rule set of a shape grammar (which may also be represented as a DAG as shown by [Patow 2012]). [Lipp et al. 2019] transforms edits applied by the user on a particular instance of a generator into edits of the program that generates a procedural shape. [Lienhard et al. 2017] proposes automatic grammar rewriting, thus synthesizing programs that are the interpolation of other input programs. [Mathur et al. 2020] assists the creation of generative programs by transforming hand selections into semantic queries.

These work are different from our case because they provide ways for the designer to modify the parametric shape's program whereas our method is geared towards the end user of an existing program.

We focus on DAGs representing imperative generation programs, but other paradigms can be used, leading to different workflows, such as [Krs et al. 2020] which proposes a powerful combination of imperative, declarative and example-based ways of modeling shapes.

3 OVERVIEW

Following a common painting metaphor, we model the user input as a series of brush strokes. Each of these strokes must be interpreted as a modification $\Delta\boldsymbol{\pi}$ of the hyper-parameters. This grounds the interaction loop shown in Figure 2c and detailed in Figure 3. Our loop follows the usual approach of inverse control problems, namely getting a differential information (Jacobian matrices $\{J_i\}$) in order to locally inverse the function $\mathcal{F}$ (the solver). For the solving part we can draw from the IK literature, however this literature takes for granted the access to the Jacobians, which is not obvious in our case.

We will first focus on how to theoretically define and practically measure the Jacobians telling the influence of the hyper-parameters over the part of the geometry where the stroke starts (Section 4). This is the source of the automatic step b in Figure 2, interleaved between the design and the use of the parametric shape. Then Section 5 shows how we use this differential information to compute $\Delta\boldsymbol{\pi}$ and details the choices we have made compared to other such solving contexts. We then show results in Section 6 and finally discuss the current limitations and many prospects of our method in Section 7.

Terminology. Here are the key terms we use along this paper.

A *Parametric shape* $\mathcal{F}$ is a function mapping input values $\boldsymbol{\pi}$ called *hyper-parameters* to 3D surface meshes. An *instance* of the parametric shape is this 3D surface mesh for a fixed value of the hyper-parameters.

A *Single-point parametric subshape* takes as input the same hyper-parameters than the original parametric shape, but only outputs a single point from the corresponding instance. It may be undefined for some values of the hyper-parameters, otherwise returns point that has the same *meaning*.

The *parameter* of a 3D point of an instance designates a 2D coordinate that indexes this point and is often used for texture mapping.

The *co-parameter* a of a single-point parametric subshape is a coordinate that indexes this subshapes among all the other ones. By extension, the co-parameter of a point of an instance is the co-parameter of the single-point parametric subshape that this point is an instance of.

4 CO-PARAMETERIZATION

4.1 Co-parameter definition

Measuring the Jacobian of the parametric shape at a given point means to tell for each hyper-parameter how this point's position changes when the hyper-parameter is subject to an infinitesimal change. The most straightforward way to do so is the finite difference method, that consists in evaluating the position of the point for two close enough values of the hyper-parameter and measuring their difference. However, the function $\mathcal{F}$ returns a set of many points – an infinity of points – with no way to *recognize* one among them.

More formally, our function $\mathcal{F} : \boldsymbol{\pi} \longmapsto G$ does not have a definition of differential nor jacobian, so the problem is not in the choice

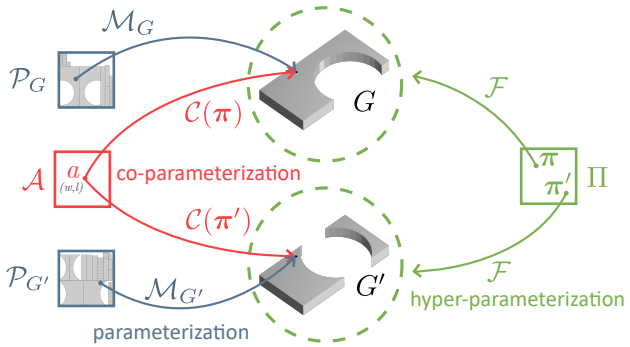

Fig. 4. For our interaction loop to work, we need to be able to *recognize* a point after a change of the hyper-parameters $\boldsymbol{\pi}$. We model this using three notions of parameterization. $\mathcal{M}_G : \mathcal{P}_G \to G \subset \mathbb{R}^3$ is a parameterization as meant in parameterized surfaces. The parametric shape $\mathcal{F} : \Pi \to \{G\}$ itself is a higher-order parameterization. Since $\mathcal{M}_G$ is not enough because in general it is different for each $\boldsymbol{\pi}$, we introduce $C : \Pi \to (A \to \mathbb{R}^3)$ which outputs parameterizations consistent among all the geometries resulting from $\mathcal{F}$.

of differentiation scheme; other ones – e.g. auto-differentiation – would suffer from the same issue. We will thus introduce the notion of *co-parameterization* of $\mathcal{F}$, a way to extract single-point parametric subshapes $\boldsymbol{\pi} \mapsto x \in \mathbb{R}^3$, which can be differentiated.

The usual way to identify a point on a geometry is to *parameterize* it. Importantly, this must not be confused with our hyper-parameterization (see Figure 4). It consists in defining for a fixed geometry G a bijection $\mathcal{M}_G : \mathcal{P}_G \longrightarrow G$ mapping to each point of G a *parameter* from a set $\mathcal{P}_G$. Such a parameter can typically be a unique texture coordinate or – in the case of meshes – a face index together with barycentric coordinates. There are in general many different ways of parameterizing a given geometry.

The problem in our case is that this mapping $\mathcal{M}_G$ may depend on $G = \mathcal{F}(\boldsymbol{\pi})$, and so on the hyper-parameter $\boldsymbol{\pi}$. As a consequence, it is of no use to *recognize* a point after $\boldsymbol{\pi}$ changed. Hence the need for a collection $\mathcal{C}$ of *consistent* parameterizations, each associated with a different $\boldsymbol{\pi}$ but all sharing the same parameter set $\mathcal{A}$:

$$\mathcal{C}(\boldsymbol{\pi}) : \mathcal{A} \longrightarrow \mathcal{F}(\boldsymbol{\pi}) \subset \mathbb{R}^3$$

The strength of this second order function $\mathcal{C}$ is that it may be uncurried because $\mathcal{A}$ does not depend in $\boldsymbol{\pi}$, so

$$C : \Pi \longrightarrow (\mathcal{A} \longrightarrow \mathbb{R}^3)$$

becomes

$$\tilde{C} : \Pi \times \mathcal{A} \longrightarrow \mathbb{R}^3$$

and may even be curried back with its arguments swapped:

$$\bar{C} : \mathcal{A} \longrightarrow (\Pi \longrightarrow \mathbb{R}^3)$$

We call $\bar{C}$ a *co-parameterization* of the parametric shape $\mathcal{F}$ and $\mathcal{A}$ its *co-parameter set*. It plays a role similar to the surface parameterization but in the space of parametric shapes. With these notations, for each $a \in \mathcal{A}$ the function $\bar{C}(a)$ is a differentiable object, for which using for instance finite differences makes sense. We call

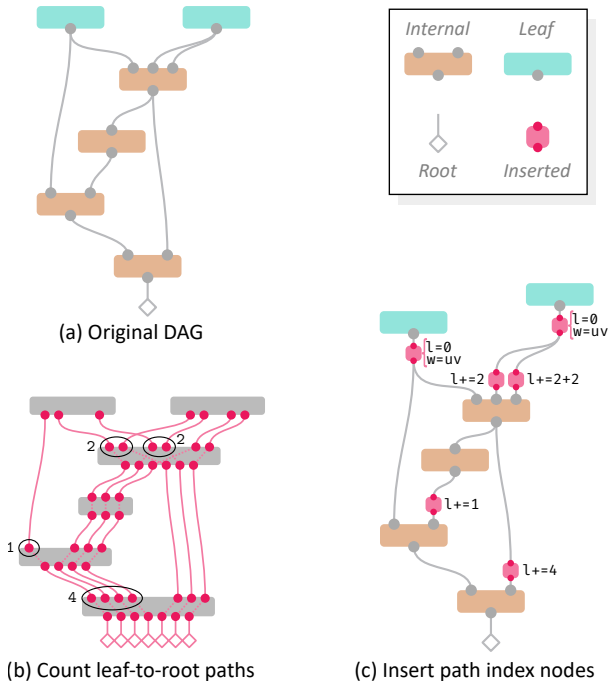

Fig. 5. We identify points across different outputs of a DAG (modeling the implementation of the parametric shape) using two attributes attached to face corners. w is a copy of the parameterization (UV) at the leaf the face corner comes from. l is a unique index of the leaf-to-root path that generated the face corner. We first count the number of paths flowing through each input of each node (b.). We then automatically insert nodes (c.) to first initialize l to 0 after each leaf and offset l before any input by the number of paths flowing through previous inputs of the same node. As a result, each face corner of the output geometry is labeled with a unique path index.

this a *single-point parametric subshape* of $\mathcal{F}$ (the output of step 1 in Figure 3).

So to determine the influence of the hyper-parameters on a point p_i sampled on the geometry $G = \mathcal{F}(\boldsymbol{\pi})$, we actually consider its co-parameter $a_i = C(\boldsymbol{\pi})^{-1}(p_i)$ and evaluate the jacobian $J_i(\boldsymbol{\pi})$ of $\bar{C}(a_i)$ at $\boldsymbol{\pi}$. The co-parameter a_i of a point p_i is thus the way to "recognize" it after a change of the hyper-parameters. We will discuss in the next section how to build this co-parameterization in practice.

4.2 Automatic DAG Amendment

We assume in this section that the geometry produced by the parametric shape $\mathcal{F}$ is a 3D surface mesh. We will automatically modify $\mathcal{F}$ so that the geometries it produces have each of their face corners labeled with their co-parameter. Thus, sampling a 3D point p_i onto the output mesh also provides its co-parameter $a_i = C(\boldsymbol{\pi})^{-1}(p_i)$ by interpolating the co-parameters of the corners of the face that p_i belongs to (Figure 7a).

Without loss of generality, we can model the implementation of $\mathcal{F}$ as a DAG whose nodes are mesh processing operations. Hyper-parameters affect the behavior of individual nodes, but the connectivity of this graph remains static. Our automatic modification of $\mathcal{F}$

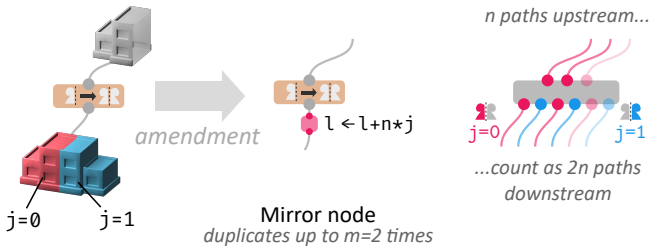

Fig. 6. A node that duplicates geometry up to m times and has n incoming paths is considered downstream as being traversed by $n \cdot m$ paths. Assuming there is a way to infer the index j of the duplicate a face belongs to, the path index l is replaced by $n \cdot j + l$.

consists in inserting new nodes into this graph. It is non-invasive in the sense that it does not require to bring any change to the internal logic of individual nodes.

The co-parameter attribute a that we intend to create at each face corner must be:

unambiguous There must not be two points sharing the same value of a.

interpolable within a face. In order to infer any point's co-parameter from the value at the corners of its face.

consistent across possible values of the hyper-parameters. To ensure the continuity of the single-point parametric sub-shapes $\boldsymbol{\pi} \mapsto \mathbb{R}^3$ that we extract.

We split a into a real component w and an integer component l. The real component is technically no different from a texture coordinate, which is also a real vector attached to face corners. The integer component must be constant across a given face in order to ensure interpolability, so it may in practice be attached to faces rather than corners.

The attribute l of a face contains the index of the data flow path that generated it (Figure 5.b). This is consistent information since the connectivity of the DAG never changes once the shape has been modeled. Disambiguating faces generated through the same path is ensured by the real component w that is given by a regular parameterization of the leaf of this path.

Construction. We first insert a node after each leaf of the DAG. This node initializes w by copying the canonical texture coordinate output by the leaf. When the leaf node generates meshes of constant connectivity, any fixed automatic parameterization (auto UV unwrapping) can be used. When the node is a primitive shape (sphere, cylinder, box, etc.), its canonical parameterization works. Practical mesh-based parametric shape engines support forwarding face corner attributes through their internal nodes like any other texture coordinate, so w is hence defined at the output of the DAG.

To produce the integer part, we first initialize it to $l = 0$ after each leaf (in the same node that initializes w). Then, before the k-th input of an internal node, we add a node that increments l by $\sum_{i<k} n_i$ where n_i is the number of paths going through input i (see Figure 5).

The goal of the index l is to disambiguate cases where w overlaps. Counting paths is a way to address cases caused by nodes that combine several input meshes, like a boolean operation (difference, fusion, intersection). The other major source of overlap is duplication nodes (mirror, copy and transform, scatter, etc.). To include this into our framework, we multiply the number n of paths flowing into a duplication node by the maximum number m of duplicates it may produce (Figure 6). If the duplication index j has a finite number of m possible values (for a mirror, $j \in \{0, 1\}$), we insert after the duplication a node that replaces l with $n \cdot j + l$.

If j may take an arbitrary large value, we add an extra dimension to the integer index l to store it, promoting it to an integer vector. Since the real component w is typically in $[0, 1]^2$ the first two dimensions of l are emulated by offsetting w in order to alleviate memory usage.

Thus, each face corner of the output of the DAG is uniquely and consistently identified by its path index l and leaf parameter w. Our process is summarized by pseudo-code in Appendix B.

4.3 Sampling and differentiation

At this point we are able to define what the jacobian of a point p_i sampled on the geometry $G = \mathcal{F}(\boldsymbol{\pi})$ means. When a stroke starts, we sample K such points by casting rays from the viewport and intersecting them with G. The hit information is used to not only find the 3D intersection point p_i but also its co-parameter $a_i = (w_i, l_i)$.

The extent of the brush may cover areas at very different depths, but we assume that the user intent has a limited depth of field, affecting either the foreground or the background but not both at the same time. To match this, we select the closest sample to the center of the brush, and discard all the points that are too far from its unprojected world space location.

To measure the k-th column of the jacobians J_i, we evaluate the parametric shape with the k-th hyper-parameter slightly changed. The step of differentiation is set to $\delta_k = 10^{-5} \cdot (\alpha_k - \beta_k)$, where $[\alpha_k, \beta_k]$ is the range of possible values of this hyper-parameter. Within the new geometry G' that this produces, we look for points that have their co-parameter equal to a_i. For each possible value of l_i, we build a mesh where coordinates are the w attribute of face corners from G'. We then project w_i onto this mesh to find the face index and barycentric coordinates of the new position p'_i of the i-th sample with respect to G'. The k-th column of J_i is thus $(p'_i - p_i)/\delta_k$ (see Figure 7).

If the nearest neighbor of w_i is too different, we assume that the point p_i has no equivalent in the new geometry G'. This happens for instance for points at the edge between the operands of a boolean operation. In such a case, we set the k-th column of J_i to zero to prevent changing this hyper-parameter, provided we do not know its influence.

5 SOLVING

The solver is provided with the jacobians $J_i \in \mathbb{R}^{3\times n}$ measured at the K points p_i sampled within the brush of radius r when the stroke started as well as the trajectory $(t_0, \ldots, t_T)$ of the stroke. The solution $\Delta\boldsymbol{\pi}$ returned by the solver must ensure the following properties:

exactness The points originally lying inside the brush must still be inside the brush at the end of the stroke.

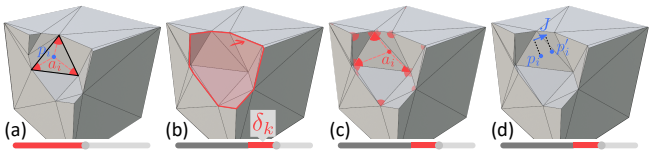

Fig. 7. To evaluate a column of the jacobian at a sample point p_i, (a) we use its co-parameter a_i interpolated from the face corners, then (b) vary the hyper-parameter by δ_k and (c) look for the new point p'_i whose co-parameter equals a_i. (d) The column of the jacobian w.r.t. this hyper-parameter is $(p'_i - p_i)/\delta_k$.

sparsity The hyper-parameter update must have an amplitude as low as possible; the user does not expect a single stroke to apply too significant changes.

continuity The hyper-parameter update must be continuous along the trajectory, i.e. adding a new way point t_{T+1} close to t_T must not suddenly change $\Delta\boldsymbol{\pi}$.

speed A result must be found at interactive frame rate. The user should not feel that hyper-parameters are changing while she is not moving the mouse.

5.1 Inversion

At the first order, we know that for each of the single-point parametric subshapes $\bar{C}(a_i)$ that we sample – denoted simply $\bar{C}_i$ below – we can approximate the new location of the point using the jacobian $J_i = J_{\bar{C}_i}(\boldsymbol{\pi})$ computed at step 2 of Figure 3:

$$\bar{C}_i(\boldsymbol{\pi} + \Delta\boldsymbol{\pi}) \simeq \bar{C}_i(\boldsymbol{\pi}) + J_i \cdot \Delta\boldsymbol{\pi} \tag{1}$$

The stroke trajectory is expressed in the viewport, so we compose equation 1 with a function $Proj : \mathbb{R}^3 \to \mathbb{R}^2$ mapping the world space to the screen space. Since $\bar{C}_i(\boldsymbol{\pi}) = p_i$ is the point that was clicked on, it is mapped to t_0 – the beginning of the stroke. To fulfill the objective of exactness, we want the new position $\bar{C}_i(\boldsymbol{\pi} + \Delta\boldsymbol{\pi})$ of this point to match the new position t_T of the user's cursor:

$$t_T = t_0 + J'_i \cdot \Delta\boldsymbol{\pi}$$

where $J'_i = J_{Proj} \cdot J_i$ is the jacobian of the composition with the projection. The jacobian J_{Proj} of this projection is detailed in Appendix A.

This is a typical problem of inverse kinematics which can be solved with a damped least square method [Baerlocher and Boulic 2004; Deo and Walker 1992]. Such a method finds the solution $\Delta\boldsymbol{\pi}$ that has a near minimal L_2 norm while avoiding discontinuities at singularities (where the rank of J'_i changes):

$$\Delta\boldsymbol{\pi} = J'^{+}_i \cdot \Delta t$$

where $\Delta t = t_T - t_0$ and J'^{+}_i is a singularity robust pseudo-inverse of J'_i.

Domain boundaries. In order to account for the boundaries of the domain Π of allowed hyper-parameters, we use the active-set method shown in Algorithm 1, inspired from the Prioritized Inverse Kinematics presented in [Baerlocher and Boulic 1998]. We iterate

Algorithm 1: Our solver uses an active-set method to account for hyper-parameter boundaries. `Diag(active_set)` returns a diagonal matrix whose j-th coefficient is 1 iff $j \in$ `active_set` in order to *freeze* hyper-parameters that are no longer in the active set.

Input: Jacobian matrix J, target move Δt
Output: An update $\Delta\boldsymbol{\pi}$ of the hyper-parameters

```
active_set ← {0, ..., n − 1};
Δπ ← (0, ..., 0);
repeat
    J⁺ ← PseudoInverse(J · Diag(active_set));
    δπ ← J⁺ · (Δt − J · Δπ);
    Δπ ← Δπ + δπ;
    for j ∈ active_set do
        if IsOutOfBounds(Δπ_j) then
            active_set ← active_set \ {j};
            Δπ_j ← Clamp(Δπ_j);
        end
    end
until δπ is null;
```

resolution steps and projections onto the domain, and *freeze* hyper-parameters affected by the projection to their clamped values for the remaining steps. Freezing is done by setting the corresponding column of the jacobian to zero. To avoid breaking the continuity of the solution, we add to the `IsOutOfBounds` test of Algorithm 1 a maximum distance to the hyper-parameter update that was returned at the previous execution of the function (i.e. for $\Delta t = t_{T-1} - t_0$). We also initialize $\Delta\boldsymbol{\pi}$ to the previously returned solution.

We are thus able to handle a point-wise constraint and fulfill the requirements listed above. We see in the next section how we combine multiple such constraints over the extent of the brush.

5.2 Jacobian buffer filtering

The variations of a single point may not be representative of those of the patch of surface surrounding it, so we sample multiple points within the extent of the brush and average their jacobians. This is still fast because the bottleneck is the evaluation of the parametric shape which is common to all samples (see Section 6.1).

The second motivation for filtering the jacobian buffer is that the L_2 norm, minimized above, is not the most appropriate way to model sparsity. Indeed, we rather need to limit the number of hyper-parameters that have a non-zero update i.e., the L_0 norm. For instance, when two hyper-parameters have a similar influence over the dragged points, we want to use only one of them rather than applying a small change to both.

Hence we refine the user intent with the following model. (i) All other things being equal, we want to foster hyper-parameters that show less variation within the extent of the brush. And (ii) we want to favor hyper-parameters whose influence over the dragged area would change notably if the brush radius would be increased. Intuitively, this corresponds to making the assumption that the user chose the maximal brush radius fitting their intent, as illustrated in

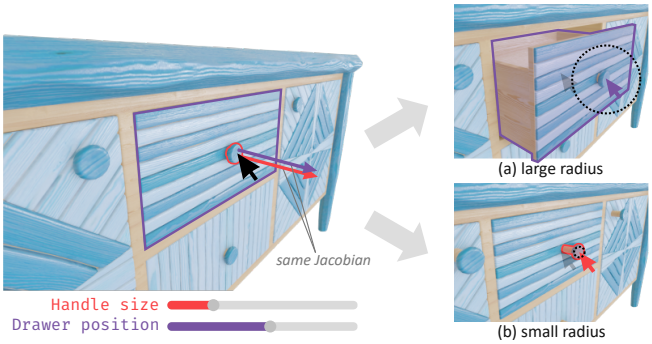

Fig. 8. Both hyper-parameters of this scene have the very same influence on the drawer's handle. Yet our jacobian buffer filtering enables to distinguish the intent behind the choice of a large (a) or small (b) brush (the dotted circle).

the drawer example in Figure 8. We inject extra knowledge about the use case by setting some columns of J_i to zero, thus ignoring the influence of the hyper-parameter over the i-th point.

For objective (i), we compare the coefficients of variation v_k (standard deviation over mean) of the norms of the columns of the J_i within the brush. We discard hyper-parameters such that $\frac{\min v_{k'}}{v_k}$ is lower than a threshold $\lambda_v \in [0, 1]$.

Among the remaining hyper-parameter, we address (ii) by measuring a contrast factor c_k which is the ratio of the average norm of the k-th column of J_i inside of the brush over the one outside of the brush. We foster hyper-parameters that have a high contrast factor, so if $\frac{c_k}{\max c_{k'}}$ is lower than a threshold $\lambda_c \in [0, 1]$, the k-th hyper-parameter is discarded.

Thus a larger brush is more likely to affect hyper-parameters whose influence has lower frequencies and a pickier brush will affect hyper-parameters with faster variations in the Jacobian buffer. The thresholds translate a global trade-off between L_2 and L_0 sparsities, which would depend on the kind of object that is manipulated. Empirically, L_2 is more important for organic shapes while L_0 is more critical for mechanic ones. In practice, we use $\lambda_v = 0.2$ and $\lambda_c = 0.75$. A high value for λ_c favors sparsity in the modified hyper-parameters, while a high value for λ_v favors regularity in the hyper-parameter selection i.e., ignoring noisy hyper-parameters.

Single Direction. At an extreme edge of this trade-off, we add the possibility to keep only one hyper-parameter. We consider that the beginning of the stroke is more meaningful than the end, because the jacobian information that we have is only valid for small variations of $\boldsymbol{\pi}$, so we pick the one hyper-parameter based on the direction at the beginning of the stroke only, $\Delta t = t_1 - t_0$. For each column j'_k, we look at the cosine similarity (c_{sim}) between j'_k and Δt, as well as the norm $\|j'_k\|_2$. We favor columns with high norm in order to reduce the L_2 norm of the output $\Delta\boldsymbol{\pi}$. On another hand, the higher the cosine similarity, the more exact the solution. Hence we pick hyper-parameter $\tilde{k}$ based on:

$$\tilde{k} = \underset{k}{\operatorname{argmin}}\, c_{\text{sim}}(j'_k, \Delta t) + \lambda \cdot \left\|j'_k\right\|_2$$

with $\lambda = 1/2$ in practice.

6 RESULTS

We implemented our method as an add-on for the Blender open source package. Its direct manipulation capabilities are illustrated on a few examples in Figure 9. In particular, we can observe that examples (a) and (b) exhibit changes of connectivity while the last edit in example (b) shows that clicking in an area not affected by any hyper-parameter induces, as expected, a null update. These examples are available as animations in the supplementary video. Our DAG automatic amendment (Section 4.2) is exemplified in Figure 11.

6.1 Performances

For all the examples illustrating this paper, the execution time of the DAG amendment is negligible, boiling down to a few milliseconds each time the graph topology is updated. Hence, we focus here on the runtime performance of our system during interaction.

Figure 10 gives execution time measurements on five scenes. The bulk of the computation is located at the beginning of the brush stroke since the finite differences require many evaluations of the input parametric shape $\mathcal{F}$. Then, when the stroke sees its extent evolving under the user input event, the overhead introduced by the solver is negligible compared with the time required to evaluate $\mathcal{F}$, which is needed anyway to display the current state of the parametric shape.

The overall jacobian evaluation time is only indirectly related to the number of vertices in the geometry and rather depends on the complexity of the DAG and its nodes' logic. The time needed to retrieve the position of the points from their co-parameters depends on the number of vertices, but since they are grouped by path index l the relation is not strictly proportional. For instance, the table in example (c) has twice as much vertices as the curtain in example (d), but this complexity is mostly concentrated in the legs. The average position evaluation time is 11.3ms, lower than for the curtains, but it has a much wider standard deviation. It peaks around 27ms when points are sampled on the legs but goes below 1ms when dragging elements of the plate.

Performances were measured with 64 sampled jacobians. This count linearly affects the initial sampling of co-parameters, the evaluation of positions from their co-parameters and the filtering of the jacobian buffer. Other elements are not modified. Empirically 64 is a high number of samples in the sense that the output jacobian is already robust enough for an intuitive interaction at lower values. In practice we use 32 samples, which was way enough for all our examples.

6.2 Ablation study

To assess the symbiosis of the elements composing our approach, we study here the influence of three of them over the whole system: sample discarding, outbound sampling and path indexing.

Figure 12 illustrates the importance of discarding sample points after unprojection. Even if they are close to the center of the brush in screen space, the drawers are not on the same plane than the likely area of focus of the user so they should not get affected by the stroke.

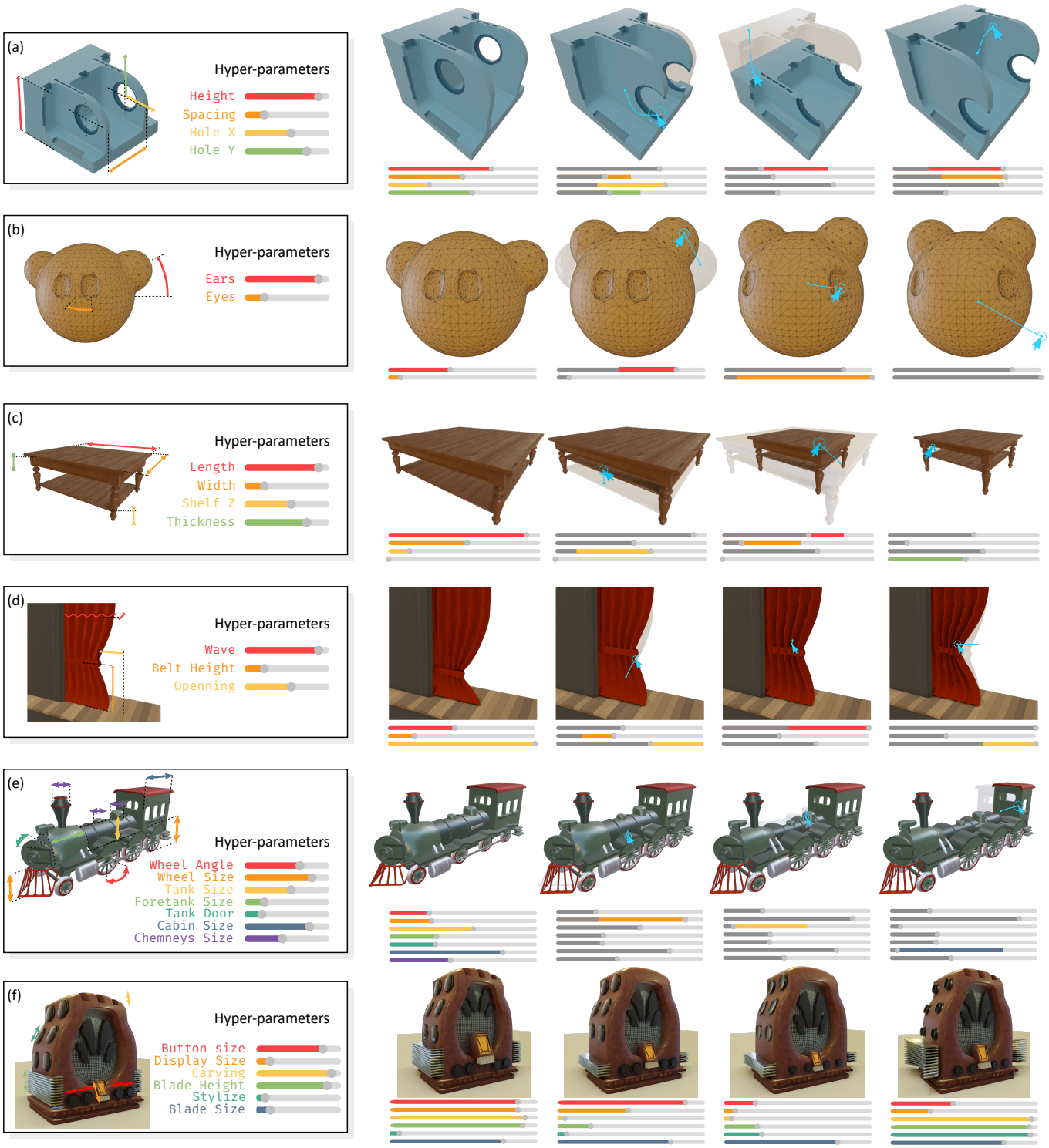

Fig. 9. Examples of sequences of edits using our method on various scenes. Corresponding DAG amendments can be found in the supplementary material.

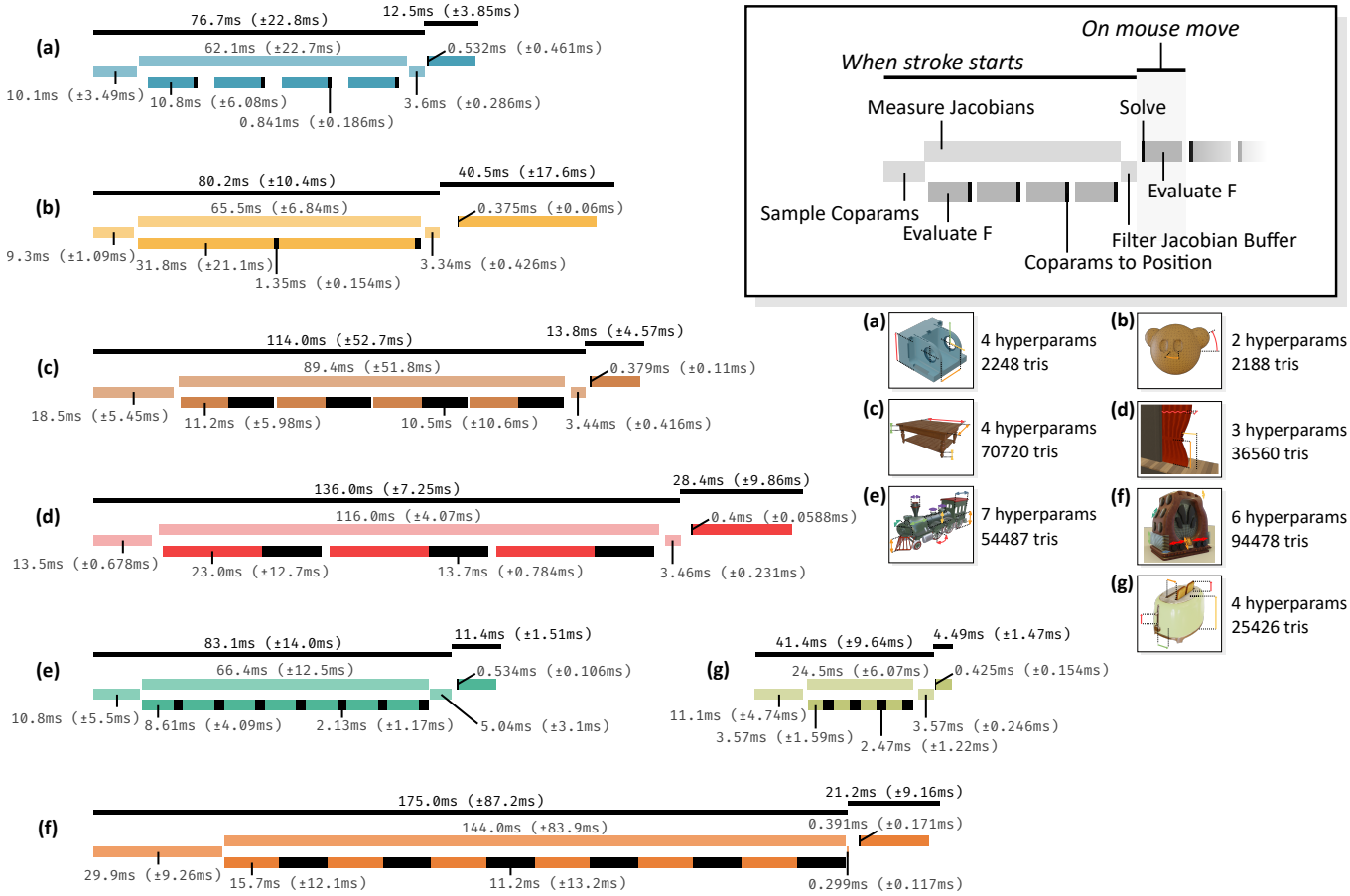

Fig. 10. Detailed profiling breakdown on several example scenes with varying complexity of DAG and output geometry. All examples are given for 64 sample points. The time needed to evaluate $\mathcal{F}$ does not depend on our method but on the parametric shape engine that we have built onto, and its standard deviation is due to caching mechanisms.

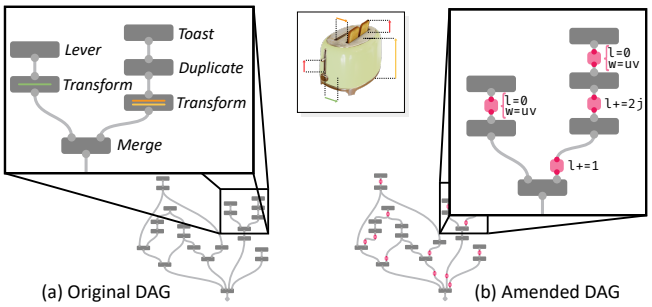

Fig. 11. Preview of the DAG amendment applied on the example of Figure 1. Small pink nodes in (b) are the node we insert. A more detailed version of this figure can be found in the supplementary material.

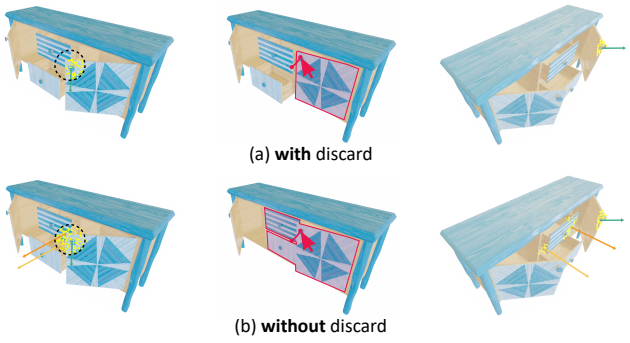

Fig. 12. Interaction is better localized when we discard samples far from the center of the brush once unprojected in world space (a) than when keeping all points (b). Middle column shows the consequences of a stroke. Right-hand column shows the same interaction under another viewpoint.

In the absence of samples outside of the brush (Section 5.2), the only way to change the size of the handle in the drawer example of Figure 8 would be to first change the drawer position all the way to its boundary then change the handle and finally move the drawer back to the desired location. Our method makes this same change possible in a single stroke.

Path indices generated by our DAG amendment ensure that there is not two points with the same co-parameter in the output geometry. Without so, if p_i and p_j share the same co-parameter, there is a risk that a row of the jacobian is set to $p'_j - p_i$ instead of $p'_i - p_j$, where p' is the new location of the point p after a slight change of an hyper-parameter. This leads to jacobians totally unrepresentative of the influence of hyper-parameters.

7 DISCUSSION

7.1 Properties

As it stands, our method allows intuitive interaction with a parametric shape directly in the 3D view. In particular, a single mouse event can yield multiple hyper-parameters to be updated concurrently. The seminal parametric shape may also be exposed with various alternative control spaces easily, by simply masking/exposing a subset of its hyper-parameters, making it easy to "publish" the shape for various application scenarios. Moreover, our DAG amendment is non intrusive since we only insert new nodes, which is an automation of a process that a parametric shape creation tool exposes to the user anyways.

Our approach opens the possibility to apply the many works that have been carried out on inverse kinematic to parametric shapes that are generated by complex graphs including operations that drastically affect a mesh connectivity like boolean operations. Not only do we give sense to the notion of jacobian of a point of the surface but also we propose a filtering scheme to adapt their raw value to the needs of intuitive direct manipulation.

Our approach is agnostic of the dimension of the interaction space. We have focused mainly on screen based interaction, but any other input device such as VR handles could be used as well. In this case, the projection of manipulation-space sample points onto the geometry at Step 1 of the interaction loop becomes a nearest neighbor search rather than a ray casting.

Implementation Guidelines. To integrate our method to an existing shape engine, the latter must expose a way to insert a non destructive operation on texture coordinates before/after existing operations. The implementation must list for each available operation the number of duplicates it may create and a mean to retrieve the duplicate index j. The interaction loop expects that the host software provides the user input, a way to query the geometry attributes at sample points on the screen and a way to evaluate the DAG programmatically.

User Feedback. We presented the tool to 19 users whose proficiency with 3D software ranges from absolute beginner to professional, asked them to reach a target configuration of the parametric shape, then collected their feedback on scales from 1 to 5. Users were able to manipulate almost all the hyper-parameters they wanted (only 1/4 felt blocked and it was at most on a single hyper-parameter) and felt confortable with completing the task (63% found it rather easy). In the majority of cases (63%), they used our brush exclusively or felt back only a few times to the sliders (resp. 42% and 21%). Professional users used to hand designed manipulators were sometimes frustrated not to be able to target for certain a given hyper-parameter, but we recall that such manipulators require extra work when originally creating the parametric shape, which our method does not. On average, users were leaning towards our brush rather than the sliders and would be likely to use it in their usual 3D software. More extensive results are available in the supplementary material.

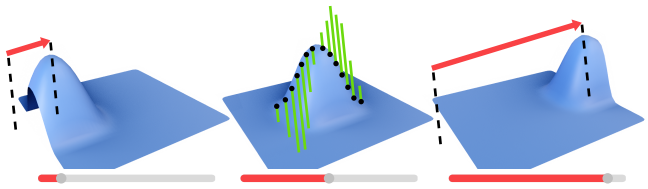

Fig. 13. **Limitation:** The only operation of this parametric shape consists in moving a vertex and its neighborhood. The hyper-parameter defines which vertex is selected rather than how to move it, so the Jacobians of single-point subshapes (lines shown in the middle figure) do not match the intuitive influence of the hyper-parameter.

7.2 Limitations

Co-parameterization. Our proposed model of co-parameterization relies on the practical ability of the DAG nodes to forward extra attributes on face corners. Although this can be seen as a restriction, it is a very reasonable assumption provided that production-ready parametric shape engines usually need this feature in order to conserve texture coordinates.

Some nodes though introduce overlaps in UVs even when there was none in their input (e.g. some smoothing algorithms). Some other nodes are simply not able to assign face corner attributes to their output (e.g. a convex hull node). Discrete hyper-parameters like the number of repetition of a duplication operation are not handled by our approach as is because it makes the single-point subshapes non differentiable.

It is nonetheless always possible for the shape's designer to manually overcome these cases by adding extra nodes dedicated to fixing the w attribute. The supplementary material shows an example where a continuous box proxy is used to override the value of w after a duplication.

Unintuitive jacobians. When an hyper-parameter acts on the selection from a geometry that gets affected by an operation rather than the way the operation itself moves points, the jacobians of single-point subshapes may no longer match the intuition of the end-user (see Figure 13).

Homogeneity. Measuring the norm of an hyper-parameter update $\Delta\boldsymbol{\pi}$ is ill-defined because hyper-parameters are in general not homogeneous to each other, namely they are expressed in different units. This is why our jacobian buffer filtering takes care of only comparing affine invariant properties (coefficient of variation, contrast factor), but it remains a problem to properly define the objective of sparsity of $\Delta\boldsymbol{\pi}$ in the presence of diverse units.

First order. We currently only measure first order information about the parametric shape – the jacobians – and do it only once, at the beginning of the stroke. For long strokes, hyper-parameters that have a non linear behavior are thus incorrectly interpreted.

Furthermore, when the evaluation time of $\mathcal{F}$ increases, the delay needed to compute the jacobians starts to be noticeable, between the click and the first update of the hyper-parameters.

7.3 Future prospects

Global sampling. We could try to precompute jacobians before the beginning of the stroke – while the user is changing the view point for instance – to avoid the slight lag when the interaction begins. This might require to use an acceleration structure to find the nearest neighbor of w in the new geometries as there would be more sample points to consider, or would require to store the cooked geometries G', costing memory.

Such a global measure would also enable the use of global criteria in the solver such as the conservation of volume. Sampling points all over the object would also help homogenizing the hyper-parameters; defining what is a "small" or "large" change to them.

More semantic. One of the strengths of our approach is to leverage the semantic information carried by the DAG. One could look for other ways of using it. For instance, the depths of the nodes using a particular hyper-parameter could be used to prioritize some of them during filtering.

Proxies. In cases where limitations occur in the construction of the co-parameterization, hand-tuned workarounds based on geometry proxies are possible. We could explore ways to automate this.

DAG pruning. The restriction $\bar{C}(a)$ of the parametric shape $\mathcal{F}$ to the single point of co-parameter $a \in \mathcal{A}$ may not be affected by all the nodes of the DAG. The graph could hence be pruned while measuring finite differences in order to alleviate its evaluation cost.

Auto-differentiation. Using automatic differentiation can make the nodes of the DAG output a jacobian as part of their computation process. This replaces the time consuming evaluation of finite differences and also enables to update the jacobian buffer at each frame during a stroke. We experimented with auto-differentiation on simple scenarios only. Advanced mesh processing nodes like boolean operations are non trivial to implement using an automatic differentiation framework. So we did not stress test its scalability, especially in terms of complexity in memory.

Other surface representation. Our practical construction of the co-parameter $a = (w, i)$ focuses on mesh-based representation of 3D surfaces, but the overall approach and the notion of co-parameterization is more general. We show for instance in the supplementary material an experiment with signed distance fields enriched so that they also return a co-parameter.

7.4 Conclusion

Our method leverages the information provided by the parametric shapes when seen as programs – described in general as graphs of operations – to make inverse control available to them in an intuitive brush-based interaction loop. Our approach may pave the way for more advanced uses of graph-based shape representations, exploring our local differentiation scheme with alternative optimization strategies.

REFERENCES

Rinat Abdrashitov, Fanny Chevalier, and Karan Singh. 2020. Interactive Exploration and Refinement of Facial Expression Using Manifold Learning. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST '20)*. Association for Computing Machinery, New York, NY, USA, 778–790. https://doi.org/10.1145/3379337.3415877

Dicko Ali-Hamadi, Tiantian Liu, Benjamin Gilles, Ladislav Kavan, François Faure, Olivier Palombi, and Marie-Paule Cani. 2013. Anatomy Transfer. *ACM Trans. Graph.* 32, 6, Article 188 (Nov. 2013). https://doi.org/10.1145/2508363.2508415

Daniel G. Aliaga, İlke Demir, Bedrich Benes, and Michael Wand. 2016. Inverse Procedural Modeling of 3D Models for Virtual Worlds. In *ACM SIGGRAPH 2016 Courses (SIGGRAPH '16)*. Association for Computing Machinery, New York, NY, USA, 1–316. https://doi.org/10.1145/2897826.2927323

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 408–416. https://doi.org/10.1145/1186822.1073207

A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. 2018. Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics Forum* 37, 6 (2018), 35–58. https://doi.org/10.1111/cgf.13310

Grégoire Aujay, Franck Hétroy, Francis Lazarus, and Christine Depraz. 2007. Harmonic Skeleton for Realistic Character Animation. In *SCA '07 - ACM-SIGGRAPH/Eurographics Symposium on Computer Animation*, Michael Gleicher and Daniel Thalmann (Eds.). Eurographics Association, San Diego, United States, 151–160. https://doi.org/10.2312/SCA/SCA07/151-160

Quentin Avril, Donya Ghafourzadeh, Srinivasan Ramachandran, Sahel Fallahdoust, Sarah Ribet, Olivier Dionne, Martin de Lasa, and Eric Paquette. 2016. Animation Setup Transfer for 3D Characters. *Computer Graphics Forum* 35, 2 (2016), 115–126. https://doi.org/10.1111/cgf.12816 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12816

P. Baerlocher and R. Boulic. 1998. Task-Priority Formulations for the Kinematic Control of Highly Redundant Articulated Structures. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, Vol. 1. 323–329 vol.1. https://doi.org/10.1109/IROS.1998.724639

Paolo Baerlocher and Ronan Boulic. 2004. An Inverse Kinematics Architecture Enforcing an Arbitrary Number of Strict Priority Levels. *The visual computer* 20, 6 (2004), 402–417.

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. ACM, New York, NY, USA, Article 72. https://doi.org/10.1145/1275808.1276467

Santiago Barroso, Gonzalo Besuievsky, and Gustavo Patow. 2013. Visual Copy & Paste for Procedurally Modeled Buildings by Ruleset Rewriting. *Computers & Graphics* 37, 4 (2013), 238–246. https://doi.org/10.1016/j.cag.2013.01.003

Volker Blanz and Thomas Vetter. 1999. A Morphable Model for the Synthesis of 3D Faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 187–194. https://doi.org/10.1145/311535.311556

Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. 2012. An Algebraic Model for Parameterized Shape Editing. *ACM Transactions on Graphics* 31, 4 (July 2012), 78:1–78:10. https://doi.org/10.1145/2185520.2185574

Mario Botsch, Mark Pauly, Markus H Gross, and Leif Kobbelt. 2006. PriMo: Coupled Prisms for Intuitive Surface Modeling. In *Symposium on Geometry Processing*. 11–20.

M. Botsch and O. Sorkine. 2008. On Linear Variational Surface Deformation Methods. *IEEE Transactions on Visualization and Computer Graphics* 14, 1 (Jan. 2008), 213–230. https://doi.org/10.1109/TVCG.2007.1054

Steve Capell, Matthew Burkhart, Brian Curless, Tom Duchamp, and Zoran Popović. 2005. Physically Based Rigging for Deformable Characters. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '05)*. Association for Computing Machinery, New York, NY, USA, 301–310. https://doi.org/10.1145/1073368.1073412

Chia-Hsing Chiu, Yuki Koyama, Yu-Chi Lai, Takeo Igarashi, and Yonghao Yue. 2020. Human-in-the-Loop Differential Subspace Search in High-Dimensional Latent Space. *ACM Trans. Graph.* 39, 4, Article 85 (July 2020). https://doi.org/10.1145/3386569.3392409

Paul D. Debevec, Camillo J. Taylor, and Jitendra Malik. 1996. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry-and Image-Based Approach. In *Proceedings of the 23th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*.

A. S. Deo and I. D. Walker. 1992. Robot Subtask Performance with Singularity Robustness Using Optimal Damped Least-Squares. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*. 434–441 vol.1. https://doi.org/10.1109/ROBOT.1992.220301

Matheus Gadelha, Giorgio Gori, Duygu Ceylan, Radomir Mech, Nathan Carr, Tamy Boubekeur, Rui Wang, and Subhransu Maji. 2020. Learning Generative Models of Shape Handles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and*

Pattern Recognition. 402–411.

Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. IWIRES: An Analyze-and-Edit Approach to Shape Manipulation. In *ACM SIGGRAPH 2009 Papers (SIGGRAPH '09)*. Association for Computing Machinery, New York, NY, USA, Article 33. https://doi.org/10.1145/1576246.1531339

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. 2018. Synthesizing Programs for Images Using Reinforced Adversarial Learning. (2018). arXiv:1804.01118 [cs.CV]

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-Space Physics. *ACM Trans. Graph.* 31, 4, Article 72 (July 2012). https://doi.org/10.1145/2185520.2185568

M. Hecher, P. Guerrero, P. Wonka, and M. Wimmer. 2018. How Do Users Map Points Between Dissimilar Shapes? *IEEE Transactions on Visualization and Computer Graphics* 24, 8 (Aug. 2018), 2327–2338. https://doi.org/10.1109/TVCG.2017.2730877

Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura, and Tosiyasu L. Kunii. 2001. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 203–212. https://doi.org/10.1145/383259.383282

Daniel Holden, Jun Saito, and Taku Komura. 2015. Learning an Inverse Rig Mapping for Character Animation. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. ACM, Los Angeles California, 165–173. https://doi.org/10.1145/2786784.2786788

Takeo Igarashi, Tomer Moscovich, and John F. Hughes. 2005. As-Rigid-as-Possible Shape Manipulation. *ACM Trans. Graph.* 24, 3 (July 2005), 1134–1141. https://doi.org/10.1145/1073204.1073323

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis. *ACM Trans. Graph.* 39, 6, Article 234 (Nov. 2020). https://doi.org/10.1145/3414685.3417812

Javor Kalojanov, Michael Wand, and Philipp Slusallek. 2016. Building Construction Sets by Tiling Grammar Simplification. *Computer Graphics Forum* 35, 2 (2016), 13–25. https://doi.org/10.1111/cgf.12807

T. Kelly, P. Wonka, and P. Mueller. 2015. Interactive Dimensioning of Parametric Models. *Computer Graphics Forum* 34, 2 (May 2015), 117–129. https://doi.org/10.1111/cgf.12546

Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. 2007. Geometric Modeling in Shape Space. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 64–es. https://doi.org/10.1145/1275808.1276457

Vladislav Kraevoy and Alla Sheffer. 2004. Cross-Parameterization and Compatible Remeshing of 3D Models. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 861–869. https://doi.org/10.1145/1015706.1015811

V. Krs, R. Mech, M. Gaillard, N. Carr, and B. Benes. 2020. PICO: Procedural Iterative Constrained Optimizer for Geometric Modeling. *IEEE Transactions on Visualization and Computer Graphics* (2020), 1–1. https://doi.org/10.1109/TVCG.2020.2995556

C. Kurz, X. Wu, M. Wand, T. Thormählen, P. Kohli, and H.-P. Seidel. 2014. Symmetry-Aware Template Deformation and Fitting. *Computer Graphics Forum* 33, 6 (2014), 205–219. https://doi.org/10.1111/cgf.12344

Francis Lazarus and Anne Verroust. 1999. Level Set Diagrams of Polyhedral Objects. In *SMA'99 Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*. ACM, Ann Arbor, United States. https://doi.org/10.1145/304012.304025

Kurt Leimer, Lukas Gersthofer, Michael Wimmer, and Przemyslaw Musialski. 2017. Relation-Based Parametrization and Exploration of Shape Collections. *Computers & Graphics* 67 (Oct. 2017), 127–137. https://doi.org/10.1016/j.cag.2017.07.001

Zohar Levi and Craig Gotsman. 2015. Smooth Rotation Enhanced As-Rigid-as-Possible Mesh Animation. *IEEE Transactions on Visualization and Computer Graphics* 21, 2 (Feb. 2015), 264–277. https://doi.org/10.1109/TVCG.2014.2359463

J. P. Lewis and K. Anjyo. 2010. Direct Manipulation Blendshapes. *IEEE Computer Graphics and Applications* 30, 4 (July 2010), 42–50. https://doi.org/10.1109/MCG.2010.41

Hao Li, Thibaut Weise, and Mark Pauly. 2010. Example-Based Facial Rigging. *ACM Trans. Graph.* 29, 4, Article 32 (July 2010). https://doi.org/10.1145/1778765.1778769

Tianye Li, Timo Bolkart, Michael J. Black, Hao Li, and Javier Romero. 2017. Learning a Model of Facial Shape and Expression from 4D Scans. *ACM Trans. Graph.* 36, 6, Article 194 (Nov. 2017). https://doi.org/10.1145/3130800.3130813

Stefan Lienhard, Cheryl Lau, Pascal Müller, Peter Wonka, and Mark Pauly. 2017. Design Transformations for Rule-Based Procedural Modeling. *Computer Graphics Forum* 36, 2 (May 2017), 39–48. https://doi.org/10.1111/cgf.13105

M. Lipp, M. Specht, C. Lau, P. Wonka, and P. Müller. 2019. Local Editing of Procedural Models. *Computer Graphics Forum* 38, 2 (2019), 13–25. https://doi.org/10.1111/cgf.13615

Han Liu, Ulysse Vimont, Michael Wand, Marie-Paule Cani, Stefanie Hahmann, Damien Rohmer, and Niloy J. Mitra. 2015. Replaceable Substructures for Efficient Part-Based Modeling. *Computer Graphics Forum* 34, 2 (2015), 503–513. https://doi.org/10.1111/cgf.12579

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. 2019. NeuroSkinning: Automatic Skin Binding for Production Characters with Deep Graph Networks. *ACM Trans. Graph.* 38, 4, Article 114 (July 2019). https://doi.org/10.1145/3306346.3322969

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graph.* 34, 6, Article 248 (Oct. 2015). https://doi.org/10.1145/2816795.2818013

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. 2019. AMASS: Archive of Motion Capture as Surface Shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Aman Mathur, Marcus Pirron, and Damien Zufferey. 2020. Interactive Programming for Parametric CAD. *Computer Graphics Forum* 39, 6 (Sept. 2020), 408–425. https://doi.org/10.1111/cgf.14046

Bruce Merry, Patrick Marais, and James Gain. 2006. Animation Space: A Truly Linear Framework for Character Animation. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1400–1423. https://doi.org/10.1145/1183287.1183294

Christian Miller, Okan Arikan, and Don Fussell. 2010. Frankenrigs: Building Character Rigs from Multiple Sources *(I3D '10)*. Association for Computing Machinery, New York, NY, USA, 31–38. https://doi.org/10.1145/1730804.1730810

Niloy J. Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. 2012. Symmetry in 3D Geometry: Extraction and Applications. In *EUROGRAPHICS State-of-the-Art Report*. https://doi.org/10.1111/cgf.12010

Niloy J. Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. 2014. Structure-Aware Shape Processing. In *ACM SIGGRAPH 2014 Courses (SIGGRAPH '14)*. Association for Computing Machinery, New York, NY, USA, 1–21. https://doi.org/10.1145/2614028.2615401

Ahmed A A Osman, Timo Bolkart, and Michael J. Black. 2020. STAR: A Spare Trained Articulated Human Body Regressor. In *European Conference on Computer Vision (ECCV)*.

Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. 2007. Robust On-Line Computation of Reeb Graphs: Simplicity and Speed. In *ACM SIGGRAPH 2007 Papers (SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 58–es. https://doi.org/10.1145/1275808.1276449

G. Patow. 2012. User-Friendly Graph Editing for Procedural Modeling of Buildings. *IEEE Computer Graphics and Applications* 32, 2 (March 2012), 66–75. https://doi.org/10.1109/MCG.2010.104

D. Raunhardt and R. Boulic. 2007. Progressive Clamping. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 4414–4419. https://doi.org/10.1109/ROBOT.2007.364159

Nadine Abu Rumman and Marco Fratarcangeli. 2016. State of the Art in Skinning Techniques for Articulated Deformable Characters. In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications: Volume 1: GRAPP (GRAPP 2016)*. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 200–212. https://doi.org/10.5220/0005720101980210

L. Saab, O. E. Ramos, F. Keith, N. Mansard, P. Souères, and J. Fourquet. 2013. Dynamic Whole-Body Motion Generation Under Rigid Contacts and Other Unilateral Constraints. *IEEE Transactions on Robotics* 29, 2 (April 2013), 346–362. https://doi.org/10.1109/TRO.2012.2234351

John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. 2004. Inter-Surface Mapping. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 870–877. https://doi.org/10.1145/1186562.1015812

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. 2018. CSGNet: Neural Shape Parser for Constructive Solid Geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Olga Sorkine and Marc Alexa. 2007. As-Rigid-as-Possible Surface Modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing (SGP '07)*. Eurographics Association, Goslar, DEU, 109–116.

Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. 2005. Mesh-Based Inverse Kinematics. *ACM Transactions on Graphics* 24, 3 (July 2005), 488–495. https://doi.org/10.1145/1073204.1073218

Jerry O. Talton, Daniel Gibson, Lingfeng Yang, Pat Hanrahan, and Vladlen Koltun. 2009. Exploratory Modeling with Collaborative Design Spaces. 28, 5 (Dec. 2009), 1–10. https://doi.org/10.1145/1618452.1618513

Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. 2011. Metropolis Procedural Modeling. *ACM Trans. Graph.* 30, 2, Article 11 (April 2011), 11:1–11:14 pages. https://doi.org/10.1145/1944846.1944851

Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. 2006. 3D Mesh Skeleton Extraction Using Topological and Geometrical Analyses. In *14th Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2006)*. Tapei, Taiwan, s1poster.

Yea-Fu Tsao and King-Sun Fu. 1984. Stochastic Skeleton Modeling of Objects. *Computer Vision, Graphics, and Image Processing* 25, 3 (1984), 348–370. https://doi.org/10.1016/0734-189X(84)90200-7

Nobuyuki Umetani. 2017. Exploring Generative 3D Shapes Using Autoencoder Networks. In *SIGGRAPH Asia 2017 Technical Briefs (SA '17)*. Association for Computing

Algorithm 2: Our DAG rewriting algorithm.

```
CountPaths(dag.root);
for n ∈ dag.nodes do
    c ← GetMaxDuplicates(n);
    if IsLeaf(n) then
        InsertAfter(n, MakeInitNode());
    else if c > 1 then
        InsertAfter(n, MakePostDuplicateNode(c));
    sum ← 0;
    for input ∈ n.inputs do
        if input.index > 0 then
            InsertAfter(n, MakeIncrementNode(sum));
        end
        sum ← sum + input.path_count;
    end
end
```

Algorithm 3: The recursive pseudo code of `CountPaths`. It is memoizing the result at each node input in the field `path_count`.

```
Input: Some DAG node n
Output: The number count of path flowing through this
        node
if IsLeaf(n) then
    count ← 1;
else
    count ← 0;
    for input ∈ n.inputs do
        if input.path_count is not defined then
            input.path_count ←
              CountPaths(input.connected_node);
        end
        count ← count + input.path_count;
    end
    count ← count · GetMaxDuplicates(n);
end
```

Machinery, New York, NY, USA, Article 24. https://doi.org/10.1145/3145749.3145758

Oliver van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. 2011. A Survey on Shape Correspondence. *Computer Graphics Forum* 30, 6 (2011), 1681–1707. https://doi.org/10.1111/j.1467-8659.2011.01884.x

Noranart Vesdapunt, Mitch Rundle, HsiangTao Wu, and Baoyuan Wang. 2020. JNR: Joint-Based Neural Rig Representation for Compact 3D Face Modeling. (2020). arXiv:2007.06755 [cs.CV]

L. Wade and R. E. Parent. 2000. Fast, Fully-Automated Generation of Control Skeletons for Use in Animation. In *Computer Animation*. IEEE Computer Society, Los Alamitos, CA, USA, 164. https://doi.org/10.1109/CA.2000.889075

Kevin Wampler. 2016. Fast and Reliable Example-Based Mesh IK for Stylized Deformations. *ACM Trans. Graph.* 35, 6, Article 235 (Nov. 2016). https://doi.org/10.1145/2980179.2982433

Emily Whiting, John Ochsendorf, and Frédo Durand. 2009. Procedural Modeling of Structurally-Sound Masonry Buildings. (2009).

Xiaokun Wu, Michael Wand, Klaus Hildebrandt, Pushmeet Kohli, and Hans-Peter Seidel. 2014. Real-Time Symmetry-Preserving Deformation. *Computer Graphics Forum* 33, 7 (2014), 229–238. https://doi.org/10.1111/cgf.12491

Z. Xu, Y. Zhou, E. Kalogerakis, and K. Singh. 2019. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. In *2019 International Conference on 3D Vision (3DV)*. 298–307. https://doi.org/10.1109/3DV.2019.00041

Yong-Liang Yang, Yi-Jun Yang, Helmut Pottmann, and Niloy J. Mitra. 2011. Shape Space Exploration of Constrained Meshes. In *Proceedings of the 2011 SIGGRAPH Asia Conference (SA '11)*. Association for Computing Machinery, New York, NY, USA, Article 124. https://doi.org/10.1145/2024156.2024158

Jianfeng Zhang, Xuecheng Nie, and Jiashi Feng. 2020. Inference Stage Optimization for Cross-Scenario 3d Human Pose Estimation. *Advances in Neural Information Processing Systems* 33 (2020).

Xin Zhao, Cheng-Cheng Tang, Yong-Liang Yang, Helmut Pottmann, and Niloy J. Mitra. 2013. Intuitive Design Exploration of Constrained Meshes. In *Advances in Architectural Geometry 2012*, Lars Hesselgren, Shrikant Sharma, Johannes Wallner, Niccolo Baldassini, Philippe Bompas, and Jacques Raynaud (Eds.). Springer Vienna, Vienna, 305–318.

A APPENDICES

A. Jacobian of the projector

Let $Proj : \mathbb{R}^3 \to \mathbb{R}^2$ be the projection of the user's view. Usually, this projection is expressed in the form $Proj(X) = \frac{P \cdot X}{[P \cdot X]_w}$ with P an arbitrary projection matrix. In this case, we derive the following Jacobian:

$$J_{Proj}(X) = \frac{1}{[P \cdot X]_w} \left(P - Proj(X) \cdot P_{w,\cdot} \right) \tag{2}$$

where $P_{w,\cdot}$ is the row of P corresponding to the component w and X is a column vector.

B. Pseudo-code of DAG rewriting

Algorithms 2 and 3 describe a base implementation of the method presented in Section 4.2.