ImplicitDrag: Drag-Based Editing of Differentiable Implicits

Henro Kriel[1†], Marzia Riso[1†], Aamen Muharram[2], Siddhartha Chaudhuri[3], Daniel Ritchie[2], Adrien Bousseau[1]

[1] Centre Inria d'Université Côte d'Azur, France
[2] Brown University, USA
[3] Adobe Research, USA

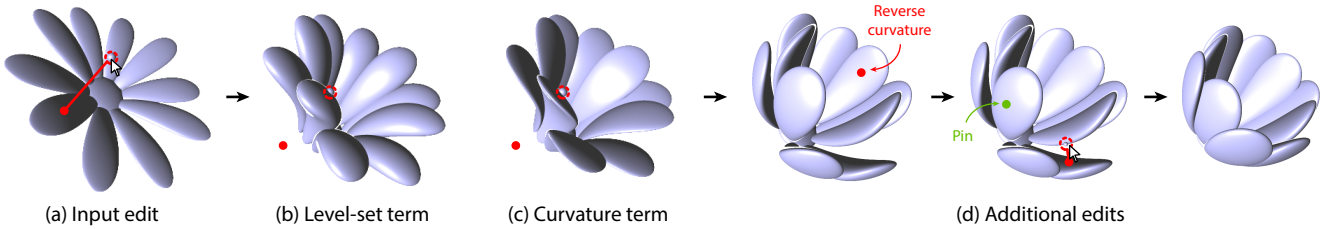


Figure 1: *Parametric shape models often expose multiple, intertwined parameters. For example, this flower has parameters to control the distribution of petals around its center, the opening of the petals, their bending inward or outward, their thickness. Instead of manipulating these parameters individually, users of our approach can express desired shape changes by dragging points of the implicit parametric surface. (a-b) Optimizing the model parameters to bring the implicit surface to the target point effectively closes the flower, but it also inflates the petals. (c) We offer additional control by leveraging the differentiability of implicit surfaces. In this example, constraining the edit to preserve curvature at the dragged point avoids inflation of the petals. (d) Other constraints allow the user to reverse the curvature of the petals and close the flower by dragging some of the petals while using a pin constraint to keep others fixed.*

Abstract

We present a method to perform drag-based editing of parametric shape models that represent classes of shapes as implicit surfaces. These models typically expose multiple parameters to control variations of a shape, but achieving a desired edit often requires tedious trial and error to find appropriate parameter values. Our approach lets users select points on an instance of a parametric model and drag these points to specify desired transformations of the implicit surface. Our method then solves for the model parameters that best satisfy these user-provided constraints. The key challenge we face is that implicit shape models do not offer a consistent parameterization of their surfaces, preventing a direct association between points across shape variations. Our solution to support diverse edits is to constrain the optimization to preserve local differential quantities of the surface around the dragged points, offering users a means to control the orientation and curvature of the surface they drag despite the lack of explicit point correspondences. Importantly, our approach applies to any differentiable parametric implicit surface, including procedural models as well as neural implicits.

Keywords: parametric shape models, implicit surfaces, neural implicit fields, signed distance fields, direct manipulation

1. Introduction

Parametric shape models encode entire shape classes via a limited set of parameters. These models can be created by expert designers in dedicated modeling software [Ble26, GPGC24, PLH*25], or learned from shape collections [PFS*19, JBX*20]. In the former case, the shape is obtained by executing a graph of operators that act on parametric primitives (e.g., boolean operations over cubes, cylinders, spheres, etc). Varying the parameter values of the primitives and operators produces diverse shapes, and designers often define these parameters to correspond to semantically meaningful shape variations. In the latter case, a neural network is trained to reproduce a set of shapes belonging to the same class, each shape being assigned a compact latent code. Varying the latent code values interpolates between the input shapes, although the latent dimensions are generally not semantically interpretable. Once con-

† Equal contribution

© 2026 This is the author version of the paper.

structed, a parametric shape model allows downstream users with limited expertise in shape modeling to control shape dimensions by changing the corresponding parameter values, allowing them to customize a product in an e-commerce configurator [GMR14], to create personalized assets for video games [FDZA25], or to tailor an object for 3D printing [OWM15].

Despite their ability to encode shape classes in a compact, editable manner, parametric shape models can be difficult to explore through changes of their raw parameters. On the one hand, expert-designed or learned models can have tight inter-dependency between their parameters, making it difficult to identify which parameter values to adjust to achieve the intended result. On the other hand, poorly-constrained models can produce broken, undesired shapes for some parameter values. This limitation of parametric models has motivated extensive work on so-called *direct manipulation interfaces*, where users interact with the shape produced by the parametric model, and the system deduces how to update the parameters to produce a new shape that satisfies these user interactions [Gle94]. In particular, drag-based interfaces let users select points of interest over the shape and drag them in space to express desired changes of dimensions [GW91, GKG*22, MB21, RMP*24, CSQ*22].

Our work contributes to this category of methods by introducing a simple solution to perform drag-based editing of parametric *implicit* shape models represented either as procedural graphs over implicit primitives [RMP*24] or as deep neural networks encoding implicit surfaces [PFS*19, LWJ*22, MSLJ23]. Such implicit representations have witnessed renewed interest, both because they support simple boolean and blending operators for manual creation [WGG99, Ado25] and because they behave well under optimization for neural representations. Similarly to previous work [MB21, RMP*24, YBP*24], we assume that we are given parametric implicit models that have already been designed and parametrized by experts, and we do not perform any kind of manual or automatic parameter extraction [KJA*23]. Our contribution builds on this setting by providing a framework that enables non-expert users to manipulate these implicit models through direct geometric edits rather than manually tuning individual parameters.

The main challenge we address is the lack of surface parameterization for implicit models, which prevents the direct identification of a point of interest before and after parameter updates. Prior work proposed to track the dragged points *explicitly* using custom mechanisms for procedural models [MB21, RMP*24] or joint encoding of neural and point based representations for neural models [HAESB20]. In contrast, our key idea is to perform the edit *implicitly* by optimizing the model parameters such that the surface exhibits *similar local characteristics* at the start and end point of the drag. We demonstrate how to apply this idea to both procedural and neural parametric shapes by characterizing points with *differential quantities* of the surface. We compute these quantities from the first and second-order derivatives of the implicit surface [Gol05], which are readily available for procedural and neural models implemented in automatic differentiation frameworks. We also leverage these differential quantities to offer users control over the orientation and curvature of the dragged points (Fig. 1). Furthermore, we demonstrate so-called *floating constraints* that allow users to preserve the local characteristics of a point without having to specify where this point should move—the point simply follows the surface as it evolves due to other edits.

In summary, our contributions are:

1. A simple formulation for drag-based editing of parametric implicit surfaces, irrespectively of their representation.
2. A set of constraints implemented within this formulation to offer users control on the local position, orientation and curvature of parametric implicit surfaces.

2. Related Work

Direct manipulation interfaces have a long history in computer graphics, starting with the seminal work of Gleicher [Gle94] who described how to optimize shape models subject to user-provided constraints on the position and orientation of shape parts. This idea has then been demonstrated on different types of shape representations. When dealing with a single shape, variational mesh editing lets users manipulate a few vertices and propagate the edits to other vertices to best preserve differential quantities over the surface [SCOL*04, NISA07, IMH05, SSP07]. While we also seek to preserve local differential quantities, we exploit these quantities to control high-level parameters of implicit shape models rather than low-level vertex positions. We also stress that we focus on interacting with models that capture *classes* of shapes, which distinguishes us from prior work on editing a *single* implicit surface by warping its scalar field [SWS10] or by fine-tuning its neural implicit field [TMR23, RT25, YBHK21, BMR*26].

To be successful, a drag-based edit must bring the point selected by the user to its desired location. Optimizing for this objective requires a way to identify the point of interest on the shape before and after parameter updates. Mesh-based parametric models offer a trivial solution to this challenge by expressing the point of interest with respect to vertex coordinates [GKG*22, CSQ*22]. But this simple solution prevents the use of boolean operations that can modify the mesh topology as parameters change. This solution also does not apply to implicit functions that lack explicit coordinates over their level-sets. Michel et al. [MB21] and Riso et al. [RMP*24] address this challenge for models composed of parametric primitives and operators by augmenting the corresponding procedural graph with so-called *co-parameters* that uniquely identify the same point over all shape instances produced by the parametric model. While this mechanism offers an elegant solution for tracking points in expert-designed parametric models, it requires a custom parametrization for each type of primitive and operator used. More importantly, the co-parameter strategy does not apply to learned parametric models where the shapes are encoded within a black-box neural network rather than a white-box procedural graph.

Other mechanisms have been proposed to manipulate neural models. Umetani et al. first proposed to optimize shapes within a compact latent space for drag-based edits [Ume17], or to improve aerodynamism [UB18], but they rely on a quad mesh template to obtain a consistent parameterization of all instances of the shape. DualSDF [HAESB20] introduces a dual representation where the shape is represented both as a neural network and as a dense set of spheres. A shared latent space relates these two representations,

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

so that users can edit the shape by dragging spheres, and these edits propagate to the neural implicit by optimizing the shared latent code. Alternatively, Elsner et al. [EIC*22] enforce a Lipschitz-type constrain on the latent code of a neural model to favor the emergence of a small set of point handles that can be manipulated explicitly. Closer to our approach, Berzins et al. [BIK23] optimize the parameters of a neural implicit to achieve a user-specified displacement of the implicit surface. They focus on normal displacements as tangential displacements of an implicit surface are ambiguous in the absence of a unique parameterization [SS11]. We show that preserving differential properties of the surface, such as curvature, offers a simple solution to perform displacements tangential to the surface in areas where these properties are discriminative, i.e., at salient shape features (e.g. Fig. 9a). In the related domain of neural image generation, DragGAN [PTL*23] identifies the same point across variations of an image by comparing the deep features responsible for generating the corresponding pixels. Similar drag-based editing tools have also been proposed for 3D Gaussians, leveraging image diffusion models to preserve realism [DW25, QCL*25]. While DragGAN's strategy might apply to neural representations of shape collections, it is not compatible with procedural graphs, since they don't rely on deep features for surface generation. We instead propose to identify similar points across shape instances by comparing differential quantities that can be computed on any differentiable implicit surface, being procedural or neural.

Finally, Yuan et al. describe an efficient algorithm for differentiable rendering of procedural CSG models [YBP*24], but using this algorithm for direct manipulation requires converting the model to a 3D mesh, performing mesh-based edits using existing software, and rendering the edited mesh from several viewpoints to form target images for the differentiable rendering optimization.

3. Method

Preliminaries. We aim at enabling direct manipulation of a parametric shape model $f(\theta,x)$ that maps a set of parameters θ to an implicit field whose zero level-set $f(\theta,x)=0$ delineates the shape boundary. In what follows, we illustrate our approach on 2D and 3D shapes, i.e., $x\in\mathbb{R}^2$ or $\mathbb{R}^3$. Given a shape produced by some initial parameter values θ_0, users of our method select one or several *source points* $\{p\}$ on the shape and transform them to *target points* $\{q\}$. Our goal is to solve for updated parameter values θ_t such that points $\{q\}$ on shape $f(\theta_t,x)=0$ correspond to points $\{p\}$ on shape $f(\theta_0,x)=0$.

A first condition to satisfy the edits is that the implicit field should take the same values at the target points q as it had at the source points p. For a given point pair (p,q), this condition is encouraged in a least-squares sense by solving for the parameter values that minimize:

$$\underset{\theta_t}{\operatorname{argmin}}\, E_{\mathrm{L}}(q) = \big(f(\theta_t,q) - f(\theta_0,p)\big)^2, \tag{1}$$

which can be solved using gradient descent for differentiable implicit fields. This simple approach—which already exists in the implicit modeling library `libfive` [Kee24]—is sufficient to displace the boundary of the shape. But since the source point p shares the same value $f(\theta_0,p)=0$ as any other point on the zero level-set, Eq. 1 has no incentive to bring p rather than another point to q, i.e., it is insensitive to tangential displacement over the implicit surface. Note that the formulation proposed by Berzins et al. [BIK23] is similar (Eq.5 in their paper), except that they optimize for the normal displacement of the implicit field rather than its value. Another difficulty is that parametric shape models often exhibit redundancy in their parameter sets, such that multiple parameter values can minimize Eq. 1 yet produce different shapes.

We propose to give users additional control on how the shape evolves by augmenting the optimization with terms that favor some solutions over others. Specifically, we constrain the solution such that the surface at the target point, $f(\theta_t,q)$, exhibits similar *local characteristics* as the surface at the source point, $f(\theta_0,p)$. Our key insight is to leverage the differentiability of the implicit field to characterize the local shape via its derivatives. Letting $g\circ f$ denote some differential quantities of the field f, we express different constraints by comparing these quantities at points p and q via loss terms of the form:

$$E_{\mathrm{C}}\big(g\circ f(\theta_t,q), g\circ f(\theta_0,p)\big). \tag{2}$$

We next describe different types of constraints implemented in this manner. Note that similar constraints have been described by Gleicher [Gle94] (Chapter 8) for the direct manipulation of parametric curves. Our originality is to demonstrate the simplicity and effectiveness of such constraints for the manipulation of *implicit surfaces*, for which the source and target points do not share an explicit parameterization. In this section, we illustrate the constraints on 2D parametric distance fields that represent an emoji, modeled as a procedural implicit with 7 parameters, and a letter modeled as a neural implicit with a 32-dimensional latent code. We provide examples on more complex 3D models in Sec. 5.

Orientation constraints. Setting g to the normalized gradient $\bar{\nabla}$ of f allows users to preserve surface orientation throughout the drag:

$$E_{\mathrm{Orient}} = 1 - \big(\bar{\nabla}\circ f(\theta_t,q)\cdot\bar{\nabla}\circ f(\theta_0,p)\big)^2, \tag{3}$$

Users can also add a rotation to the gradient if they want to tilt the surface locally (Fig. 2):

$$E_{\mathrm{Tilt}} = 1 - \big(\bar{\nabla}\circ f(\theta_t,q)\cdot R\,\bar{\nabla}\circ f(\theta_0,p)\big)^2, \tag{4}$$

with R a rotation matrix.

Curvature constraints. Recent work has shown how the curvature of implicit fields can be leveraged for surface regularization [DXW*24, YPW*25, NSS*22, ZWB25] and other geometry processing tasks [DWX*25]. Inspired by this family of work, we constrain curvature magnitudes to maintain local bumps and cavities at the edited points (Fig. 4). We follow Kindlmann et al. [KWTM03] to compute the principal curvature magnitudes κ_1 and κ_2 from the gradient and Hessian of f (see supplemental materials for derivation). Our constraint then compares the curvature magnitudes between source and target points:

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

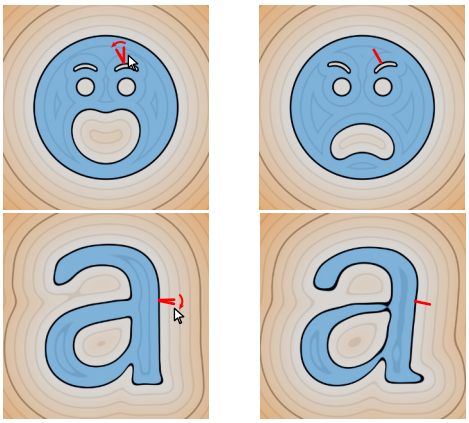


Figure 2: *In these examples, the user keeps the edited points in place but uses an orientation constraint to tilt the eyebrow (top) and to make the letter italic (bottom).*

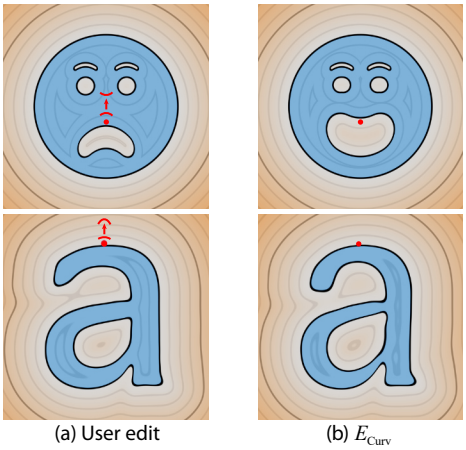


Figure 3: *The curvature constraint can be used to increase or decrease the curvature at a specific point, here to make the emoji smile (top) and the letter more rounded (bottom).*

$$E_{\text{Curv}} = \big(\kappa_1 \circ f(\theta_t, q) - \alpha\kappa_1 \circ f(\theta_0, p)\big)^2 + \big(\kappa_2 \circ f(\theta_t, q) - \alpha\kappa_2 \circ f(\theta_0, p)\big)^2. \tag{5}$$

The optional parameter α allows to increase or decrease the target curvature, as shown in Fig. 3.

Floating constraints. The constraints we have presented so far require users to specify the target position q where the source point p should move. Yet, users might want to preserve the local orientation or curvature at a point p while letting this point move freely. We achieve this control—which we call a *floating constraint* (Fig. 5)—by treating the constraint position x as an optimization variable:

$$\underset{\theta_t, x}{\operatorname{argmin}} E_{\text{C}}\big(g \circ f(\theta_t, x), g \circ f(\theta_0, p)\big). \tag{6}$$

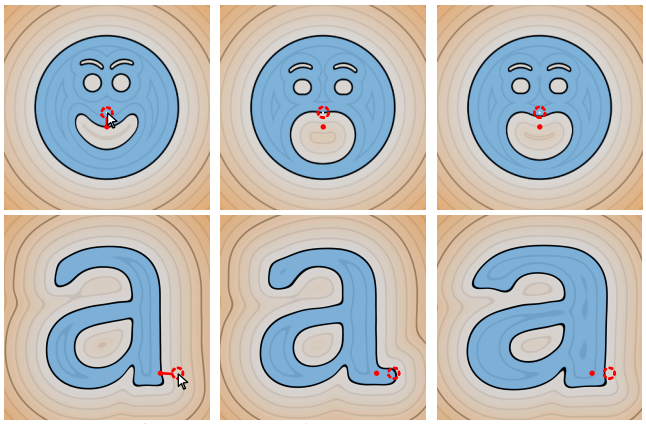


Figure 4: *Optimizing only for the level-set term E_L brings the upper lip of the procedural emoji to the target point, but the shape of the mouth changes significantly (top, a-b). Similarly, the level-set term makes a serif appear on the neural letter "a" when dragging a point to the right (botton, a-b). Adding the curvature term E_{Curv} opens the mouth yet keeps it smiling and makes the letter bold yet keeps it sans-serif (c). Note that the optimization achieves a trade-off between the user constraints when no parameter values can satisfy them all.*

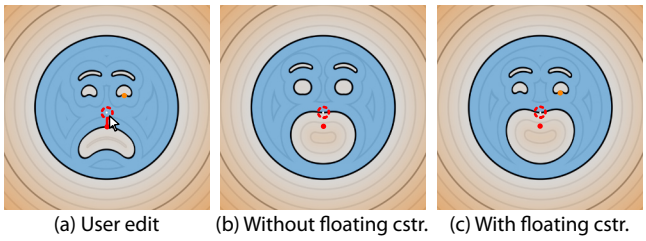


Figure 5: *Setting a floating curvature constraint in the concavity of the squinting eye (orange dot) allows to preserve its shape even though the eyes move slightly apart as the mouth opens.*

Pin constraints. Finally, users can keep parts of a shape unchanged by specifying the above constraints with pairs of points that stay in place, i.e., setting $p = q$ (Fig. 6).

Summary. We combine the different constraints with Eq. 1 to obtain our complete optimization problem:

$$\underset{\theta_t, \{x\}}{\operatorname{argmin}} \sum_{\{p,q\}} E_{\text{L}}(q) + w_{\text{C}} E_{\text{C}}\big(g \circ f(\theta_t, q), g \circ f(\theta_0, p)\big), \tag{7}$$

which we solve using gradient descent, initializing the shape parameters θ_t to θ_0 and the floating constraints $\{x\}$ to the points $\{p\}$ at which the user wants to impose these constraints. Multiple E_{C} terms can be imposed simultaneously on different pairs of points, with the corresponding weights w_{C} determining the relative influence of each type of constraint (we used a value of $w_{\text{C}} = 0.01$ for the results shown in this paper). Note that our approach assumes that the implicit field evolves smoothly during gradient descent,

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

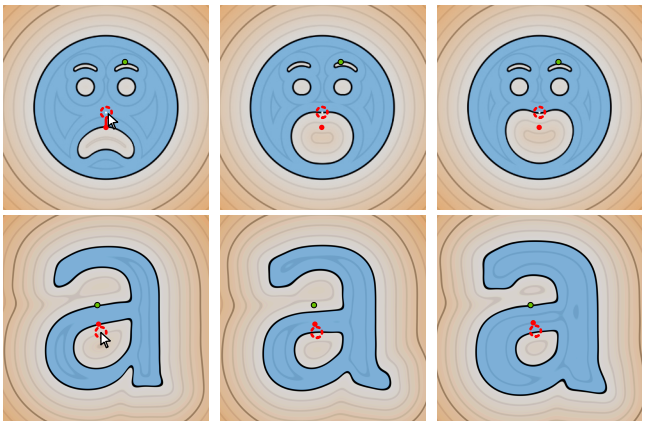


Figure 6: *A pin constraint (green dot) maintains the boundary in place as the user drags other parts of the shape. This constraint allows to keep the eyebrow fixed as the mouth changes (top) and to thicken the letter by pulling the boundary away from the pinned point (bottom).*

such that neither the dragged points nor the floating constraints jump abruptly to distant parts of the shape as the optimization progresses. Also, a source point can only reach a target point if their respective characteristics share parameters. These assumptions hold for moderate displacements, while larger displacements need to be decomposed into smaller steps (Sec. 5).

4. Implementation

We demonstrate our approach on both procedural and neural implicit surfaces. We have implemented a simple interactive tool to perform the diverse edits enabled by our approach. When editing 3D models, the endpoint of the 2D mouse drag is projected from the screen space to a plane parallel to the camera, located at the same depth as the selected point on the model. We provide screen recordings of typical operations in the accompanying video.

Procedural implicit models. We represent procedural implicit models as trees of parametric signed distance fields. We have implemented one 2D model (emoji, parameterized as an interpolation over 7 key facial expressions) and four 3D models with different numbers of parameters (6 for the flower, 11 for the cup and for the snowman, 20 for the house, see supplemental materials for description of the trees and their parameters). Note that in addition to standard CSG boolean operators, the house, flower, snowman and emoji models include differentiable parametrized warping operations, such as bends, twists, smooth booleans and interpolations to illustrate more general function-based modeling [PASS95, WGG99]. The trees are built at runtime using operator overloading in Python. The tree doubles as an abstract syntax tree for a GLSL code generator, which allows us to create sphere-tracing GPU kernels to render the resulting shapes. For optimization, we implemented pointwise evaluation of the trees in JAX [BFH*18], which allows computation of derivatives (in our case, gradients and Hessians) using automatic differentiation. For each user interaction, we optimize and compute the new set of parameters using the Python/JAX backend and send the parameters as uniforms to the GLSL shader for display. We use the Adam optimizer from JAX's optimization library Optax, with 2000 iterations and no additional stopping criteria.

Neural implicit models. We model neural implicit fields using the DeepSDF network architecture [PFS*19], which we adapt to better suit our context. Specifically, we make the field twice differentiable by replacing the ReLU activation functions in the fully connected layers with SIREN activations [SMB*20]. Furthermore, we follow Liu et al. [LWJ*22] and incorporate their Lipschitz regularization term in the training loss to encourage a smooth latent space. We conducted experiments on 2D and 3D shapes. We display the 2D distance fields by evaluating them over a grid in the Python/JAX backend and sending that grid to the GLSL shader. For 3D distance fields, we implemented sphere-tracing as a just-in-time compiled process in the Python/JAX backend, which evaluates the 3D positions on the zero-level set and their corresponding normals according to the current camera orientation. This information is stored into textures and passed to the GLSL frontend for shading and rendering on screen. The neural network needs to be evaluated multiple times during sphere tracing, which increases the computational cost and prevents real-time rendering. To improve visual feedback, we render the model at a lower resolution during user interactions (e.g. rotating or zooming the camera).

For the 2D neural example, we built a custom image dataset consisting of 140 black-and-white renderings of the letter "a" in various fonts, centered within each image. We extracted the signed distance fields from the binary images using the Euclidean Distance Transform from the SciPy library [VGO*20]. For the 3D neural examples, we collected 540 and 818 meshes of tables and vases respectively. We generated these meshes from 3D procedural models provided by GeoCode [PLH*25], tailoring their dataset generation script to our context by fixing the discrete parameters to constants. Using GeoCode's procedural models allowed us to generate dense, high-quality training sets that result in smooth latent spaces. We varied 12 continuous parameters for the table and 23 for the vase to capture significant shape variations.

The training procedure follows the same structure and hyperparameters as the original DeepSDF framework, using a latent code of 32 dimensions. We generate for each shape a set of training samples (p, d), where p is the point location and d is the corresponding ground truth distance to the zero-level set. Following DeepSDF's training strategy, we sample more points near the zero level-set of each shape and fewer points in the interior and exterior regions, and train each neural model for $200k$ epochs using two different Adam optimizers for the model weights and the latent codes.

5. Results

Procedural and neural models. In addition to Fig. 2 to 6 that illustrate edits on 2D procedural and neural models, Fig. 1 and 9 show different types of edits on 3D procedural models, and Fig. 10 shows similar edits on neural models. In several cases, we compare our results using the curvature term E_{Curv} to results obtained using only the simple level-set term E_{L}, which emulates the drag-based direct editing feature of the software library `libfive` [Kee24]. As

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

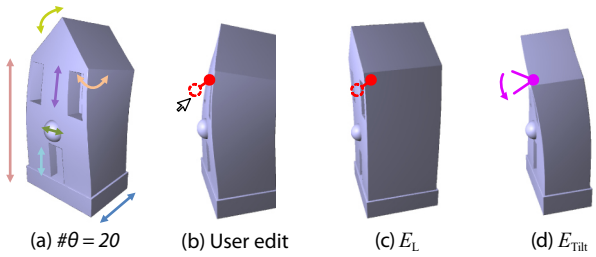


Figure 7: *The house model has 20 parameters, including bending and twisting (a). Dragging a point of the facade downwards does not produce the intended bending, the optimization prefers to thicken the house to bring the facade to the target (b-c). A tilt constraint is more effective to control bending (d).*

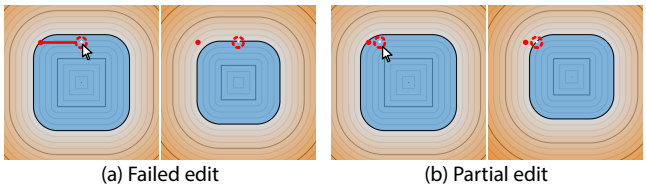


Figure 8: *The optimization can fail to progress when the point to drag is too far from its target. In this example (a), the target point is not influenced by the rounded corner, which prevents the curvature term to have any effect. Instead, the level-set term attracts the top boundary of the shape. The optimization succeeds when the target point is sufficiently close to the source point, such that they are influenced by the same parameters (b-c). Note that we used both the curvature term E_{Curv} and the orientation term E_{Orient} to drag a specific point of the circular arc.*

noted by Riso et al. [RMP*24], this term is not capable of distinguishing displacements tangential to the surface, as is also the case for the normal displacement term used by Berzins et al. [BIK23]. We observe this limitation in Fig. 9(a, center). In contrast, our curvature term allows to distinguish the source and target points in this case (Fig. 9a, right).

Fig. 7 illustrates a challenging edit on a more complex procedural model (house, 20 parameters), inspired by the model shown by Riso et al. [RMP*24] (see Fig. 11 in their paper). Attempting to bend the house forward by dragging a point of the facade only works partially because another point of the facade reaches the target and satisfies the level-set term E_L before the intended bending is achieved. Adding an orientation term E_{Tilt} allows to obtain a similar result as shown by Riso et al. in their paper.

These results took between less than a second to a few seconds to converge. We believe that our method could be accelerated by replacing JAX computation with a custom GPU kernel. Note also that JAX incurs an additional cost of several seconds to compile the functions the first time they are called.

Floating constraints. Fig. 9(b,d) demonstrates the power of our floating constraints, which stick to the convex edge of the cup and the concave neck of the snowman to maintain their curvature as other parts are changed. These two results illustrate typical challenges raised by parametric shape models. On the one hand, the multiple parameters of the cup make drag edits ambiguous; our orientation and curvature constraints serve to clarify the user intent. On the other hand, we designed the snowman to illustrate that dragging a single part will not necessarily drag the entire model if the model parameterization is poor. A floating constraint allows the user to keep the parts attached, thus correctly performing a translation of the entire model. Fig. 10(d) shows a similar edit on a neural model, where the thickness and rounded shape of the table top is better preserved by adding a floating constraint.

In theory, floating curvature constraints are free to navigate over the surface along paths of constant curvature. For example, the floating constraints in Fig. 9(b,d) could turn around the edge of the cup or the neck of the snowman. In practice, we only observed this behavior when we let the optimization run beyond convergence, possibly because gradient descent continues to update the constraint position for marginal improvement on the loss.

Limitations. The simplicity of our approach hinges on the assumption that the shape evolves smoothly as the parameters are continuously updated. Fortunately, procedural models are often designed to react smoothly to small parameter changes, a necessary condition to make the parameters easy to adjust by end users. This need for continuity is also the motivation for the Lipschitz regularization of neural models [LWJ*22], from which we benefit.

Our implicit method assumes that the source and target points share some parameters, such that gradient descent can continuously update these parameters from source to target. Fig. 8(a) illustrates a typical failure case where this assumption is violated because the target point is on a flat part that is not influenced by the rounded corner on which the source point lies, preventing the target to have any effect on the curvature term in the optimization. A workaround to this limitation consists in decomposing the desired edit into a succession of smaller steps (Fig. 8 b-c). Generally, the outcome of an edit is inherently influenced by the underlying parameterization. While our optimization explicitly searches for a solution that respects both the level-set objective and the selected geometric local properties, the desired edit can only be achieved if a suitable parameter assignment exists in the parameter space. Otherwise, the optimization converges to the closest feasible solution.

Finally, our approach requires that the shape is sufficiently smooth at the point of interest for computing first and second derivatives. This requirement prevents dragging infinitely sharp features, such as the edge of a perfect cube. Note that such sharp features cannot occur with neural models implemented with SIREN activation functions [SMB*20]. For procedural models, a solution consists in using slightly rounded primitives, as for the edge of the cup in Fig. 9(b). One can even resort to rounded primitives for optimization while keeping sharp primitives for display.

6. Conclusion

Implicit surfaces have gained revived interest for representing classes of shapes procedurally or as deep neural networks. Com-

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

plementary to prior work that leverages the gradient and Hessian of the implicit field to regularize shape reconstruction, we show how these differential quantities can be used as handles for editing parametric implicit surfaces, offering users similar control as direct manipulation interfaces originally developed for explicit shape representations.

Future Work. While we demonstrated our approach on tree-based procedural implicit models, it could apply to other implicit representations, such as convolution surfaces parameterized by a skeleton and smoothing kernel [BS91,ZBQC13], as long as they provide end-to-end differentiability with respect to the procedural parameters. The constraints we have presented in Sec. 3 only characterize the surface in the immediate vicinity of the points of interest. While this behavior proves sufficient for the relatively smooth models we experimented with, future work could extend our approach by measuring differential quantities at multiple scales to discriminate high-frequency details from the overall low-frequency shape [MGB*12]. Recent work on neural field filtering allows to compute such a scale space [MNT*24, YMR*25], but performing the equivalent filtering on procedural implicit fields remains an open challenge.

Acknowledgments

This work was supported by the ANR-NSF grant NaturalCAD (ANR-23-CE94-0003, NSF #2315354) and by software and research donations from Adobe.

References

[Ado25] ADOBE: Project neo. https://projectneo.adobe.com/, 2025. Accessed: 2025-12-20. 2

[BFH*18] BRADBURY J., FROSTIG R., HAWKINS P., JOHNSON M. J., KATARIYA Y., LEARY C., MACLAURIN D., NECULA G., PASZKE A., VANDERPLAS J., WANDERMAN-MILNE S., ZHANG Q.: JAX: composable transformations of Python+NumPy programs, 2018. URL: http://github.com/jax-ml/jax. 5

[BIK23] BERZINS A., IBING M., KOBBELT L.: Neural implicit shape editing using boundary sensitivity. In *Proc. International Conference on Learning Representations* (2023). 3, 6

[Ble26] BLENDER ONLINE COMMUNITY: Blender, 2026. Accessed: 2026-05-29. URL: https://www.blender.org. 1

[BMR*26] BAIERI D., MAGGIOLI F., RODOLÀ E., MELZI S., LÄHNER Z.: Implicit-ARAP: Efficient handle-guided neural field deformation via local patch meshing. In *Proc. Annual Conference on Neural Information Processing Systems* (2026). doi:10.52202/085713-4158. 2

[BS91] BLOOMENTHAL J., SHOEMAKE K.: Convolution surfaces. In *SIGGRAPH* (1991), p. 251–256. doi:10.1145/122718.122757. 7

[CSQ*22] CASCAVAL D., SHALAH M., QUINN P., BODIK R., AGRAWALA M., SCHULZ A.: Differentiable 3d cad programs for bidirectional editing. *Computer Graphics Forum (proc. Eurographics) 41*, 2 (2022). doi:10.1111/cgf.14476. 2

[DW25] DONG J., WANG Y.-X.: 3dgs-drag: Dragging gaussians for intuitive point-based 3d editing. In *ICLR* (2025). 3

[DWX*25] DONG Q., WEN H., XU R., CHEN S., ZHOU J., XIN S., TU C., KOMURA T., WANG W.: Neurcross: A neural approach to computing cross fields for quad mesh generation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 44*, 4 (2025). doi:10.1145/3731159. 3

[DXW*24] DONG Q., XU R., WANG P., CHEN S., XIN S., JIA X., WANG W., TU C.: Neurcadrecon: Neural representation for reconstructing cad surfaces by enforcing zero gaussian curvature. *ACM Transactions on Graphics (Proc. SIGGRAPH) 43*, 4 (2024). doi:10.1145/3658171. 3

[EIC*22] ELSNER T., IBING M., CZECH V., NEHRING-WIRXEL J., KOBBELT L.: Intuitive shape editing in latent space, 2022. URL: https://arxiv.org/abs/2111.12488, arXiv:2111.12488. 3

[FDZA25] FUKAYA K., DAYLAMANI-ZAD D., AGIUS H.: Intelligent generation of graphical game assets: A conceptual framework and systematic review of the state of the art. *ACM Computing Surveys 57*, 5 (2025). doi:10.1145/3708499. 2

[GKG*22] GAILLARD M., KRS V., GORI G., MĚCH R., BENES B.: Automatic differentiable procedural modeling. *Computer Graphics Forum (proc. Eurographics) 41*, 2 (2022), 289–307. doi:10.1111/cgf.14475. 2

[Gle94] GLEICHER M.: *A Differential Approach to Graphical Interaction*. Ph.d. thesis, Carnegie Mellon University, November 1994. 2, 3

[GMR14] GANDHI A., MAGAR C., ROBERTS R.: How technology can drive the next wave of mass customization. *Business Technology Office* (2014), 1–8. 2

[Gol05] GOLDMAN R.: Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design 22*, 7 (2005), 632–658. doi:10.1016/j.cagd.2005.06.005. 2

[GPGC24] GONZALEZ J. F., PIETRZAK T., GIROUARD A., CASIEZ G.: Facilitating the parametric definition of geometric properties in programming-based cad. In *Proc. ACM Symposium on User Interface Software and Technology (UIST)* (2024). doi:10.1145/3654777.3676417. 1

[GW91] GLEICHER M., WITKIN A.: Differential manipulation. In *Graphics Interface* (1991). 2

[HAESB20] HAO Z., AVERBUCH-ELOR H., SNAVELY N., BELONGIE S.: Dualsdf: Semantic shape manipulation using a two-level representation. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020). 2

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 24*, 3 (2005). doi:10.1145/1073204.1073323. 2

[JBX*20] JONES R. K., BARTON T., XU X., WANG K., JIANG E., GUERRERO P., MITRA N. J., RITCHIE D.: Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 39*, 6 (2020). doi:10.1145/3414685.3417812. 1

[Kee24] KEETER M.: libfive: Infrastructure for solid modeling. https://libfive.com/, 2024. Accessed: 2025-12-20. 3, 5

[KJA*23] KODNONGBUA M., JONES B., AHMAD M. B. S., KIM V., SCHULZ A.: Reparamcad: Zero-shot cad re-parameterization for interactive manipulation. In *SIGGRAPH Asia Conference Papers* (2023). doi:10.1145/3610548.3618219. 2

[KWTM03] KINDLMANN G., WHITAKER R., TASDIZEN T., MOLLER T.: Curvature-based transfer functions for direct volume rendering: methods and applications. In *IEEE Visualization* (2003). doi:10.1109/VISUAL.2003.1250414. 3

[LWJ*22] LIU H.-T. D., WILLIAMS F., JACOBSON A., FIDLER S., LITANY O.: Learning smooth neural functions via lipschitz regularization. In *ACM SIGGRAPH Conference Proceedings* (2022). doi:10.1145/3528233.3530713. 2, 5, 6

[MB21] MICHEL E., BOUBEKEUR T.: Dag amendment for inverse control of parametric shapes. *ACM Trans. Graph. 40*, 4 (July 2021). doi:10.1145/3450626.3459823. 2

[MGB*12] MELLADO N., GUENNEBAUD G., BARLA P., REUTER P., SCHLICK C.: Growing least squares for the analysis of manifolds in

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

scale-space. *Computer Graphics Forum 31*, 5 (2012), 1691–1701. doi:10.1111/j.1467-8659.2012.03174.x. 7

[MNT*24] MUJKANOVIC F., NSAMPI N. E., THEOBALT C., SEIDEL H.-P., LEIMKÜHLER T.: Neural gaussian scale-space fields. *ACM Transactions on Graphics 43*, 4 (2024). doi:10.1145/3658163. 7

[MSLJ23] MARSCHNER Z., SELLÁN S., LIU H.-T. D., JACOBSON A.: Constructive solid geometry on neural signed distance fields. In *SIGGRAPH Asia Conference Papers* (2023). doi:10.1145/3610548.3618170. 2

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: FiberMesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007). doi:10.1145/1276377.1276429. 2

[NSS*22] NOVELLO T., SCHARDONG G., SCHIRMER L., DA SILVA V., LOPES H., VELHO L.: Exploring differential geometry in neural implicits. *Computers & Graphics 108* (2022), 49–60. doi:10.1016/j.cag.2022.09.003. 3

[OWM15] OEHLBERG L., WILLETT W., MACKAY W. E.: Patterns of physical design remixing in online maker communities. In *Proc. ACM Conference on Human Factors in Computing Systems (CHI)* (2015), p. 639–648. doi:10.1145/2702123.2702175. 2

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11* (1995), 429–446. doi:10.1007/BF02464333. 5

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). 1, 2, 5

[PLH*25] PEARL O., LANG I., HU Y., YEH R. A., HANOCKA R.: Geocode: Interpretable shape programs. *Computer Graphics Forum 44*, 1 (2025). doi:10.1111/cgf.15276. 1, 5

[PTL*23] PAN X., TEWARI A., LEIMKÜHLER T., LIU L., MEKA A., THEOBALT C.: Drag your gan: Interactive point-based manipulation on the generative image manifold. In *ACM SIGGRAPH 2023 Conference Proceedings* (2023), SIGGRAPH '23, Association for Computing Machinery. doi:10.1145/3588432.3591500. 3

[QCL*25] QU Y., CHEN D., LI X., LI X., ZHANG S., CAO L., JI R.: Drag your gaussian: Effective drag-based editing with score distillation for 3d gaussian splatting. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers* (New York, NY, USA, 2025), SIGGRAPH Conference Papers, Association for Computing Machinery. doi:10.1145/3721238.3730600. 3

[RMP*24] RISO M., MICHEL E., PARIS A., DESCHAINTRE V., GAILLARD M., PELLACINI F.: Direct manipulation of procedural implicit surfaces. *ACM Trans. Graph. 43*, 6 (Nov. 2024). URL: 10.1145/3687936, doi:10.1145/3687936. 2, 6

[RT25] RUBAB F., TONG Y.: Interactive Stroke-based Neural SDF Sculpting. In *High-Performance Graphics - Symposium Papers* (2025), The Eurographics Association. doi:10.2312/hpg.20251169. 2

[SCOL*04] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proc. EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing* (2004). doi:10.1145/1057432.1057456. 2

[SMB*20] SITZMANN V., MARTEL J. N., BERGMAN A. W., LINDELL D. B., WETZSTEIN G.: Implicit neural representations with periodic activation functions. In *Proc. NeurIPS* (2020). 5, 6

[SS11] STAM J., SCHMIDT R.: On the velocity of an implicit surface. *ACM Transactions on Graphics 30*, 3 (2011). doi:10.1145/1966394.1966400. 3

[SSP07] SUMNER R. W., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 26*, 3 (2007). doi:10.1145/1276377.1276478. 2

[SWS10] SUGIHARA M., WYVILL B., SCHMIDT R.: Warpcurves: A tool for explicit manipulation of implicit surfaces. *Computers and Graphics 34*, 3 (2010). doi:10.1016/j.cag.2010.03.008. 2

[TMR23] TZATHAS P., MARAGOS P., ROUSSOS A.: 3d neural sculpting (3dns): Editing neural signed distance functions. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2023). 2

[UB18] UMETANI N., BICKEL B.: Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2018). doi:10.1145/3197517.3201325. 2

[Ume17] UMETANI N.: Exploring generative 3d shapes using autoencoder networks. In *SIGGRAPH Asia Technical Briefs* (2017). doi:10.1145/3145749.3145758. 2

[VGO*20] VIRTANEN P., GOMMERS R., OLIPHANT T. E., HABERLAND M., REDDY T., COURNAPEAU D., BUROVSKI E., PETERSON P., WECKESSER W., BRIGHT J., VAN DER WALT S. J., BRETT M., WILSON J., MILLMAN K. J., MAYOROV N., NELSON A. R. J., JONES E., KERN R., LARSON E., CAREY C. J., POLAT İ., FENG Y., MOORE E. W., VANDERPLAS J., LAXALDE D., PERKTOLD J., CIMRMAN R., HENRIKSEN I., QUINTERO E. A., HARRIS C. R., ARCHIBALD A. M., RIBEIRO A. H., PEDREGOSA F., VAN MULBREGT P., SCIPY 1.0 CONTRIBUTORS: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272. doi:10.1038/s41592-019-0686-2. 5

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum* (1999). doi:10.1111/1467-8659.00365. 2, 5

[YBHK21] YANG G., BELONGIE S., HARIHARAN B., KOLTUN V.: Geometry processing with neural fields. In *Advances in Neural Information Processing Systems* (2021), vol. 34. 2

[YBP*24] YUAN H., BOUSSEAU A., PAN H., ZHANG C., MITRA N. J., LI C.: Diffcsg: Differentiable csg via rasterization. In *ACM SIGGRAPH Asia (Conference track)* (2024). doi:10.1145/3680528.3687608. 2, 3

[YMR*25] YALDIZ M. B., MEHTA I., RAGHAVAN N., MEULEMAN A., LI T.-M., RAMAMOORTHI R.: Spectral prefiltering of neural fields. In *Proc. SIGGRAPH Asia Conference Papers* (2025). doi:10.1145/3757377.3763901. 7

[YPW*25] YIN H., PLOCHARSKI A., WLODARCZYK M. J., KIDA M., MUSIALSKI P.: FlatCAD: Fast Curvature Regularization of Neural SDFs for CAD Models. *Computer Graphics Forum (Proc. Pacific Graphics)* (2025). doi:10.1111/cgf.70249. 3

[ZBQC13] ZANNI C., BERNHARDT A., QUIBLIER M., CANI M.-P.: Scale-invariant integral surfaces. *Computer Graphics Forum 32*, 8 (2013). doi:10.1111/cgf.12199. 7

[ZWB25] ZHANG Y., WANG S., BESSMELTSEV M.: Variational neural surfacing of 3d sketches. In *Proc. SIGGRAPH Asia Conference Papers* (2025). doi:10.1145/3757377.3763900. 3

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.

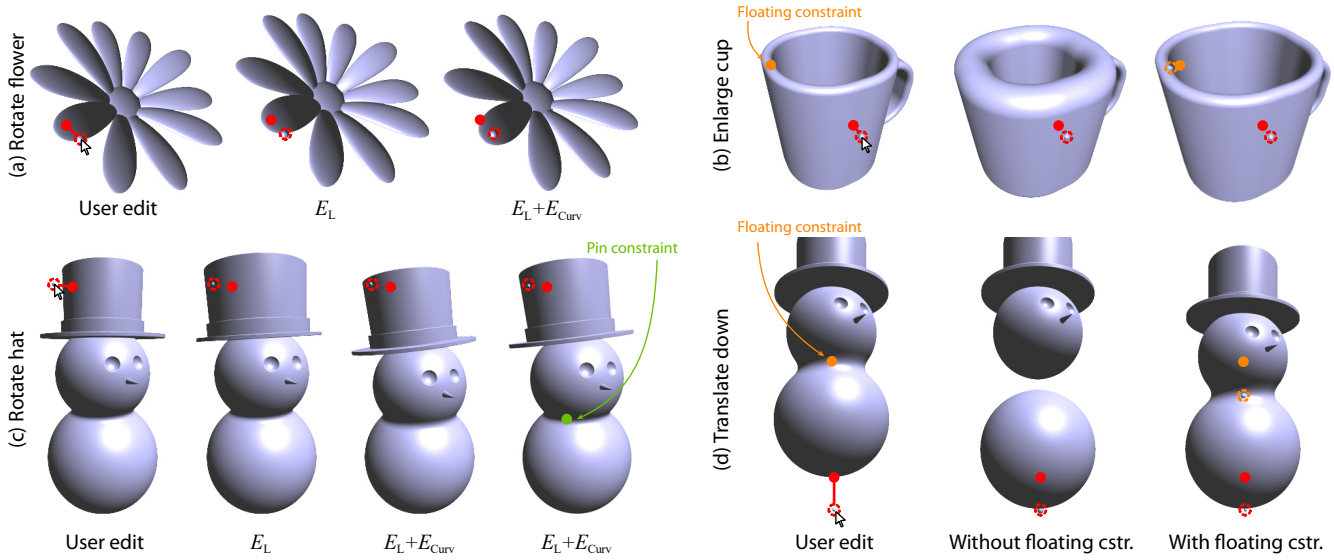


Figure 9: *Results of our method on procedural parametric shape models. (a) Attempting to move the flower petal tangentially to its surface does not produce any change when using only the level-set term. Adding the curvature term makes the flower rotate to follow the drag. (b) The cup can vary in width and thickness. Setting a floating curvature constraint on its edge allows to maintain its thickness. (c) Similarly, the hat of the snowman can vary in radius and in tilt angle. Dragging a point of the hat outward using the level-set term increases its radius. Using the curvature term tilts the hat to keep the radius unchanged, but also lowers the head of the snowman. Adding a pin constraint on the neck keeps the head at the same height. (d) The creator of the snowman model did not constrain the body to stay attached to the head. Adding a floating curvature constraint on the neck allows the user to keep the head attached when translating the body down.*

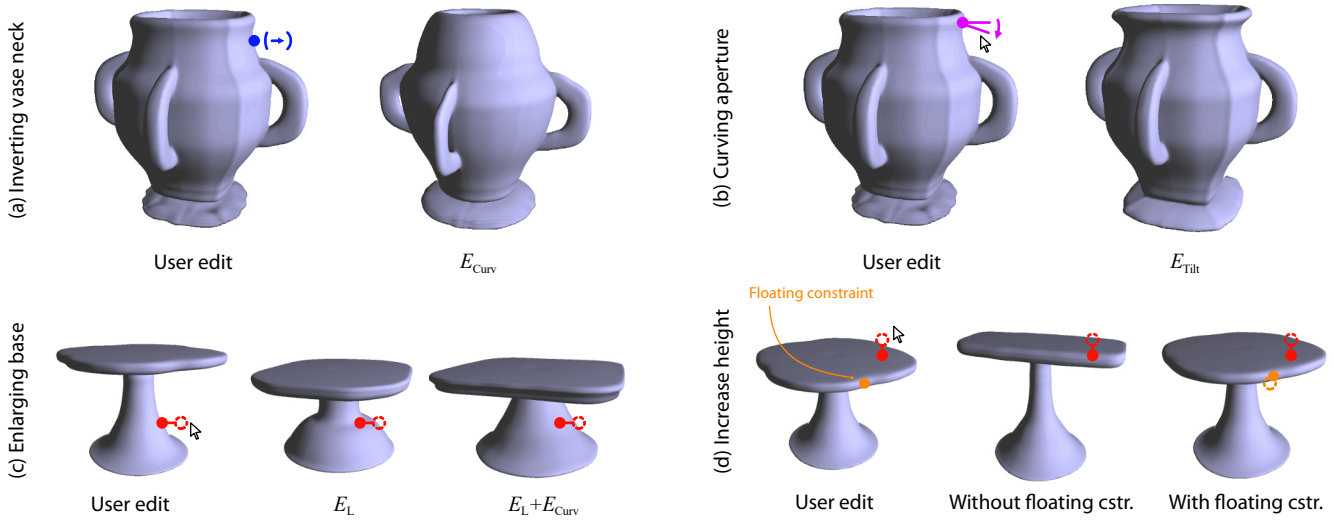


Figure 10: *Results of our method on neural shape models. (a) In addition to disambiguating drag-based edits, our curvature term can also be used to achieve a target curvature, here to change the top part of the vase from convex to concave. (b) The orientation constraint offers local control to tilt the surface near the edge of the vase. (c) Dragging a point on the foot of the table using the level-set term increases the foot radius but introduces a discontinuity in its profile. Adding the curvature term allows to enlarge the foot while preserving its profile. (d) Attempting to raise the table by dragging a point on its top triggers a change of its shape, making the top thicker and rectangular. Adding a floating curvature constraint along the top edge better preserves its shape.*

© 2026 The Author(s).
Proceedings published by Eurographics - The European Association for Computer Graphics.