# Semantic Shape Editing with Parametric Implicit Templates
## *Supplementary Material*

## 1 OUR PARAMETRIC IMPLICIT TEMPLATES

While our method requires a parametric implicit template that has minimal requirements as mentioned in the main paper, our example templates are designed by using a few analytical primitives and their combinations using simple mathematical operations like addition, subtraction, minimum, maximum, vector norm etc. To ease the design of such templates, we use a directional graph representation.

Our templates represent implicit functions $f_\Theta : \mathbb{R}^3 \rightarrow \mathbb{R}$ that map a 3D point in space to a value. We represent the implicit function computation as Directed Acyclic multi-graphs (DAG) $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ composed of nodes $\mathcal{N}$ and arcs $\mathcal{A}$. Nodes $\mathcal{N}$ have inputs and outputs, and correspond to basic mathematical operations without side effects (e.g., addition, subtraction, minimum, maximum, vector norm). Arcs $\mathcal{A}$ connect outputs of source nodes to inputs of target nodes. Graphs are parameterized by a set of interpretable parameters $p \in \mathbb{R}^n$ that control the shape. These parameters can be bounded by $p_{min}$ and $p_{max}$ and are the same parameters that the user can interact with to edit the template. Given a parameter vector $p \in \mathbb{R}^n$, and a 3D point $x \in \mathbb{R}^3$, the shape can be retrieved as the roots of the implicit function $f_p$:

$$f_p(x) = \mathcal{G}(x, p) \quad \text{with} \quad x \in \mathbb{R}^3, p \in \mathbb{R}^n$$
$$\text{s.t.} \quad p_{min} \leq p \leq p_{max}.$$

Figure 1 shows an example of a *bowl* primitive comprising three analytical primitives (two spheres and a plane). The parameters $\Theta$ controlling the template are the two radii $r1, r2$ and the plane normal $pn$. The bowl region can be expressed as the intersection of one sphere with the negative of the other subsequently sliced into half using the cutting plane. The graph in Figure 1 shows the computation graph to calculate the value of the implicit function $f_\Theta(x)$ for an arbitrary point in space $x$.

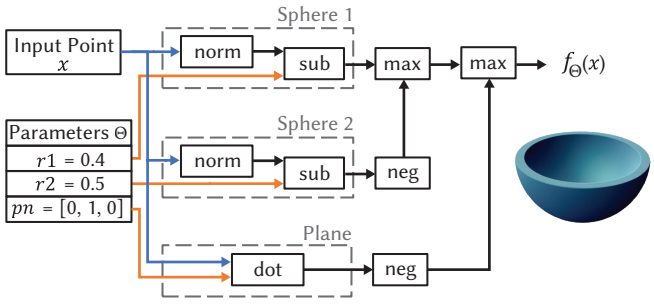


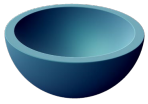

Fig. 1. An example of the analytical computation graph that represents an implicit shape of a *bowl*.

The graph is an intermediary representation, convenient to manipulate by artists, that can be translated to GLSL code for real-time visualization, a PyTorch graph for auto-differentiation, or more languages given the application. The conversion from our procedural graphs to PyTorch code or GLSL code is straightforward as these languages already implement all basic operations of our nodes $\mathcal{N}$, and data types going through arcs $\mathcal{A}$. Figure 2 shows examples of translations we are able to produce from our graph representation.

GLSL

```
uniform float r1 = 0.4;
uniform float r2 = 0.5;
uniform vec3 pn = vec3(0.0, 1.0, 0.0);

float implicit_shape(vec3 x) {
  float plane = dot(x, pn);
  float sphere_int = length(x) - r1;
  float sphere_ext = length(x) - r2;
  return max(max(sphere_ext, -sphere_int), -plane);
}
```



PyTorch

```
r1 = torch.nn.Parameter(torch.tensor([0.4]))
r2 = torch.nn.Parameter(torch.tensor([0.5]))
pn = torch.nn.Parameter(torch.tensor([0.0, 1.0, 0.0]))

def implicit_shape(x):
    plane = torch.sum(x * pn, dim=1, keepdim=True)
    sphere_int = torch.linalg.norm(x, dim=1, keepdim=True) - r1
    sphere_ext = torch.linalg.norm(x, dim=1, keepdim=True) - r2
    return torch.max(torch.max(sphere_ext, -sphere_int), -plane)
```

Fig. 2. Translation of the *bowl* graph defined in Figure 1

### 1.1 Template Design

Designing such a parametric implicit template from scratch using only basic operations is tedious and time consuming. We propose an *interactive design tool* inspired from *Shadertoy* [Quilez and Jeremias 2013] that reduces some of this burden by showing a real time visualization of the shape graph. We also implement a sub-graph mechanism that allows the artist to encapsulate and reuse some parts of the graph. By default, sub-graphs for basic primitives (e.g., cube, spheres, cylinders) and modifiers (e.g., rigid transformation, morph, twist) are available. See Figure 1 where the computation graph features sub graphs for two spheres and a plane. Simple templates can be authored by artists just by assembling sub-graphs and connecting them together. Shapes used in this paper can be authored in 10 minutes to a day by an artist with no prior experience with our tool.

Figure 3 shows a few example templates we use in the paper and their associated parameters. In the supplementary material, we provide an html interface which allows manipulating our templates and parameters from their glsl representation.

While designing shapes, artists have full control on the parameterization and thus can encode high-level knowledge directly in the procedural graph by carefully choosing parameters and their bounds. For example, symmetry constraints can be achieved by sharing the same parameter for different parts of a shape. Figure 1 (right) of the main paper shows the parameterization of two of our procedural graphs.

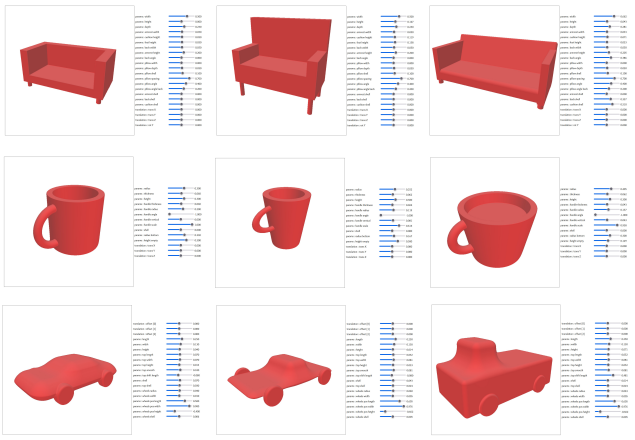


Fig. 3. Exemplar templates and their parameters. Our simple templates are powerful enough to represent many different instances of a shape category. An html interface is provided as supplementary material to browse templates and edit their parameters interactively.

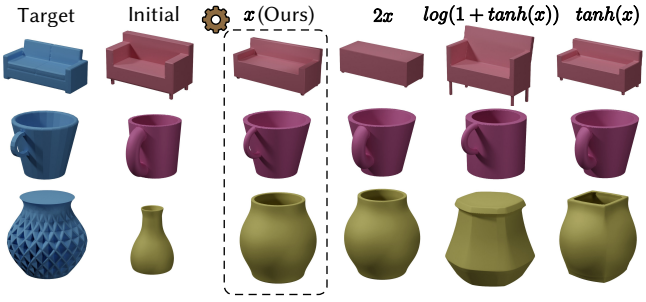



Fig. 4. **Fitting with modulated target.** We run our template fitting scheme while modulating SDF values of the target shape with different functions. In all cases (three last columns), the fit is imperfect but still close to the target shape (blue) thanks to the low weighting applied to volume samples whose implicit values are far from a true SDF.

## 2 FITTING TO MODULATED TARGET

To further demonstrate the robustness of our approach to *severely damaged implicits*, we illustrate results of our fitting strategy for modulated targets $\{h(d(p))\}$ in place of $\{d(p)\}$ (see Fig. 4). As one can see, while the end results differ, the optimized implicit surfaces still overall fit the target surfaces, indicating that the use of our volume samples is indeed harmless for the final fitting. For lack of a better guess, we still choose $\{d(p)\}$ as default targets, as artists *usually* reweigh their implicits in order to obtain a normalized gradient on their 0-set (i.e., so that *near* $\mathcal{S} := f_{\Theta_S}^{-1}(\{0\})$, the implicits *usually* approximate local SDFs). This is particularly important for rendering reasons, as efficiently sphere-tracing an implicit surface $f_{\Theta_S}^{-1}(\{0\})$ requires knowledge of the local Lipschitz constant of $f$ [Galin et al. 2020].

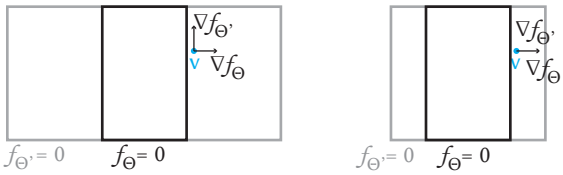



Fig. 5. An example scenario with a large edit where $f_\Theta$ (black) and $f_{\Theta'}$ (grey) are implicits with their 0-level sets as solid lines. In the large edit case (left), the correspondence of the mesh point $v$ with the base representation changes from the vertical surface on its left to the horizontal surface above it, resulting in tracking a different $\nabla f_{\Theta'}$.

## 3 MOTIVATION FOR PROGRESSIVE TRACKING

The intuition behind the need for quasi-static editing and infinitesimal updates can be explained through a simple example scenario of a signed distance function . For a large edit such as shown in Figure 5, we notice that the closest level-set might change, hence enforcing the coupling conditions is not really desired.

Consequently, our iterative algorithm can split an edit into intermediate ones by linearly interpolating between source and target template parameters, then used as in Algorithm 1 of the main paper.

## 4 CONVERGENCE OF TRACKING UPDATES

Starting from $v = v^0$, we define a sequence of iterates $v^{k+1} = v^k + d_v^k$, where $d^k$ depends only on $v^0$ and $v^k$ through Equation (8) of the main paper. Using taylor expansion in Section 5.1.2 of the main paper, we can show that,

$$\left(\frac{\nabla f_{\Theta'}(v^k) + \nabla f_{\Theta'}(v^{k+1})}{2}\right)^T \cdot d^k = f_v - f_{\Theta'}(v^k) \tag{1}$$

Using the update rule from Equation (8), the error accumulated for the $0^{th}$ order condition is quantified as

$$f_{\Theta'}(v^{k+1}) - f_v = \left(\frac{\nabla f_{\Theta'}(v^{k+1}) - J_v \cdot g_v}{2}\right) d^k \tag{2}$$

Note that we are trying to quantify an error measure using only the $0^{th}$ order condition from equation 4. Expanding, $\nabla f_{\Theta'}(v^{k+1})$ further till second order derivatives repeatedly,

$$f_{\Theta'}(v^{k+1}) - f_v = \left(\frac{\nabla f_{\Theta'}(v^0) - J_v \cdot g_v}{2}\right) d^k + \sum_{i=0}^{k-1} (d^i)^T \nabla^2 f_{\Theta'}(v^i) d^{k-1} \tag{3}$$

Observe that using the updates from Eq. 8 gives a lower order error than simple first order updates such as: $\nabla f_{\Theta'}(v^k)^T d^k = f_v - f_{\Theta'}(v^k)$.

Also, $\nabla^2 f_{\Theta'}(v^i)$ is closely related to the shape operator of the zero contour set of $f_{\Theta'}$, and $d_i$ tends to be in the null space of it leading to low values of the summation terms in Equation (3). More-over the error magnitudes of the summation terms are $O(||d^k||^2)$, where as the first term captures the first order error magnitude. The value $f_{\Theta'}(v^{k+1}) - f_v = \left(\frac{\nabla f_{\Theta'}(v^0) - J_v \cdot g_v}{2}\right) d^k$ depends upon the similarity of


Fig. 6. We run the deformation for the same edit and same hyper-parameters while disabling the order-1 coupling from Eq. 5 of the main paper. We observe that when there is only a few number of sub-edit steps taken, using only function value coupling from Eq. 4 of the main paper is not sufficient in case of conflicting edits, as seen on the armrest of the couch which collapses on itself.

$\nabla f_{\Theta'}(v^0), J_v \cdot g_v$ and also the magnitude of $d_k$. Therefore, it is essential to keep the magnitude of $f_{\Theta'}(v^{k+1}) - f_v = \left(\frac{\nabla f_{\Theta'}(v^0) - J_v \cdot g_v}{2}\right) d^k$ low particularly in the beginning steps when the magnitude of $d^k$ is large, which again suggests a quasi static edit approach.

In practice, we show the importance of gradient coupling in Figure 6. We also observe faster convergence with grad coupling than without.

## 5 IMPLEMENTATION DETAILS

### 5.1 Timings for fitting

Sampling volume points around the mesh takes between 2 and 5 seconds. While the time needed to fit a parametric template to a given input mesh depends on its complexity, convergence is usually obtained in between 2 and 4 seconds, for sets of samples from 50k to 300k. Note that we use a non optimized single-threaded Pytorch implementation for the fitting and expect significant speedups with a more efficient implementation.

### 5.2 Evaluation

To generate our results, we run the fitting and deformation in batch mode with no manual tuning of the parameters. The input meshes are normalized to unit box to have a scale close to that of the initial template configuration.

For the comparison with *Spaghetti* [Hertz et al. 2022], we performed the edits using the available UI and manual interaction with the shapes, which can slightly differ from the deformations we produce.

### 5.3 Mesh deformation

The iterative mesh deformation is implemented as described in an alternating local-global algorithm as described by Algorithm 1 of the main paper. While the closed-form solutions for updating our local maps $J_v$ given by Eqs. 13, 17 use pseudo-inverses of $3 \times 3$ matrices, we implement this using a set of local bases for reasons of speed and robustness. We define a source basis corresponding to the source mesh $\mathcal{M}$ at every vertex $v$, $B_v$ and a target basis $B'_v$ corresponding to the current iterate of the deformation $\mathcal{M}'$. While the source bases are computed once parallely for all the vertices before the start of the iterations, the target bases are updated every iteration after the global step using the normals from the global step update. This approach reduces the matrix inversion to a simple full rank $2 \times 2$ inverse, and all the local steps are parallelized. We observe that this approach is faster and numerically more robust.

Depending on the specific use-case, we provide an option to turn on regularization over the transformation matrices controlled by $\sigma_s$. The solution to Eq. 16 can be computed by vectorizing $J_v^k$ and using Sylvester equation multiple times. We implement this in our system ending up with 2 big Cholesky solves in every iteration, one during the global step and one during the local (which isn't local anymore). On the other hand Eq. 17 can be parallelized and gives speed gains. Depending on the exact scenario of use, if the number of iterations $k_{max}$ are high enough, the local parallel approach gives similar results to the exact solution involving the matrix solve. However if a user is interested in a small edit but very smooth solution (defined and controlled by the geometric weights $w_i j$), the *global-global* option can be preferred.

For our results we use $k_{max} = 10$ and number of sub-edit steps $n_e = 40$. Depending on how large the edit is, we choose the number of sub-edit steps with the default value 50. A local-global step for a mesh with around 5k vertices takes 25ms (define as $t$) on a Macbook Pro with an M1 Max processor.

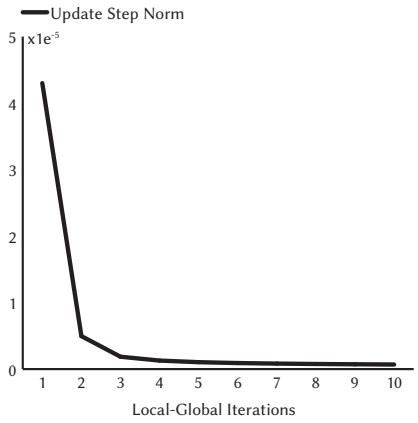



Fig. 7. Variation of the norm of the update step $d(d_v$ concatenated across all $v)$ over the local-global iterations $k$. The step norms at each iteration are averaged across different edit steps for a particular example deformation scenario of doubling the width of a couch using 20 sub-edit steps.

So, depending on $k_{max}$, each sub-edit step takes $k_{max} \times t$ time, and if there are $n_e$ sub-edit steps, the total edit time amounts to a time of $n_e \times k_{max} \times t$. Note that the time of a sub-edit is independent of $n_e$. Depending on how large each edit is, $n_e$ needs to be increased accordingly. In practice, $k_{max}$ can be as low as 2 as seen from Figure 7

In an interactive scenario where a slider is incrementally updated by the drag operation of a user, each slide update can be treated as a

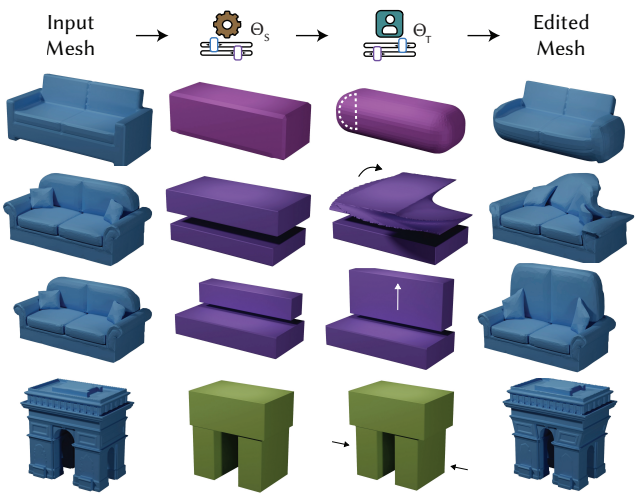



Fig. 8. Editing with simple templates made of boxes. Templates of increasing complexity provide control over the level of details of the deformation to apply. They can easily be tailored to achieve an expected edit.

sub-edit which can be performed in $k_{max} \times t$ time (which amounts to $\approx 2 \times 25 = 50$ms) giving interactive speeds.

### 5.4 Parametric templates

Table 1 provides statistics for all parametric template presented in this paper.

| Graph | Color | # nodes | # params |
|---|---|---|---|
| couch | | 667 | 22 |
| table | | 137 | 11 |
| mug | | 595 | 14 |
| vase | | 877 | 20 |
| airplane | | 1015 | 17 |
| car | | 745 | 19 |
| single box | | 34 | 8 |
| double box | | 75 | 12 |
| three boxes | | 122 | 21 |

Table 1. Statistics for our procedural templates include the number of nodes $\mathcal{N}$ and parameters $p$ for each model.

## 6 ADDITIONAL RESULTS

We present more results in addition to the ones shown in the main paper.

## 7 PERCEPTUAL STUDY

In order to evaluate the perceptual quality of the results of our method compared to those of related work, we conducted a user study through a survey where respondents could choose between two results. For each of the compared methods, a survey form shows around 30 randomly picked editing instances where respondents are asked to choose between one of the two results, or indicate that *Both* or *Neither* results satisfy the edit.

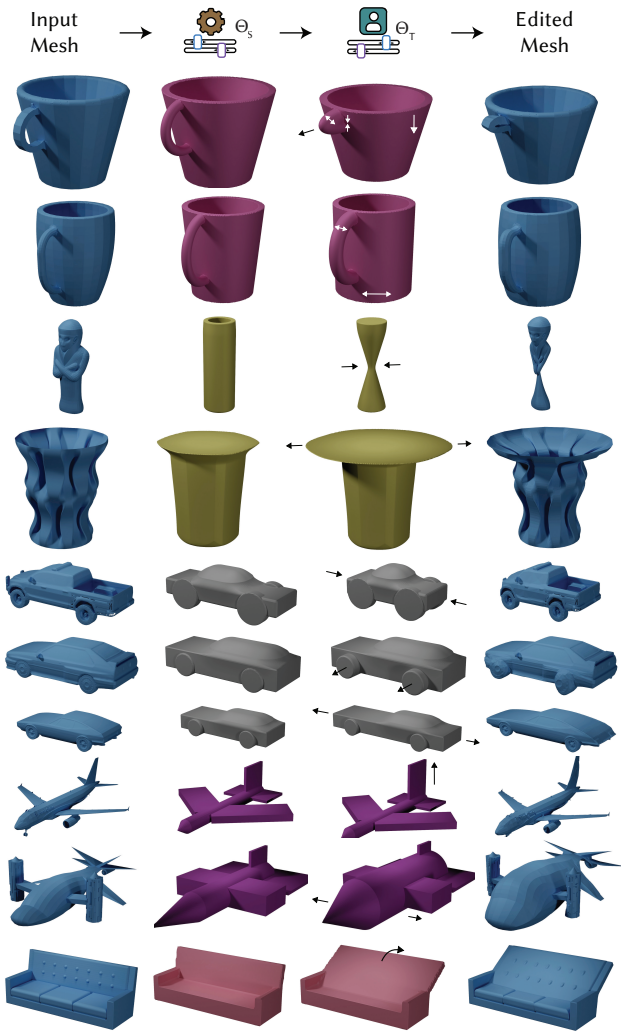



Fig. 9. Applying semantic edits from implicit templates (second column) onto 3D meshes (left). Same colors of templates denote same template used.

Figure 11 shows the results of the study. consisting of 17 responses. As we can see, respondents preferred our results over all three compared methods. Against Neural Cages [Yifan et al. 2020], while our results were mostly preferred, a decent amount of users chose neither results, showing the difficulty of the task. Comparing against Wei et al. [2020], while our results are not as clearly preferred as in other cases, the evaluation instances provided by Wei et al. [2020] do not contain large edits. Against *Spaghetti* [Hertz et al. 2022], our results clearly outpeform the method, which shows that generative methodologies cannot faithfully reproduce input shapes.

## REFERENCES

Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. 2020. Segment tracing using local Lipschitz bounds. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 545–554.

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2022. Spaghetti: Editing implicit shapes through part aware generation. *ACM Transactions*

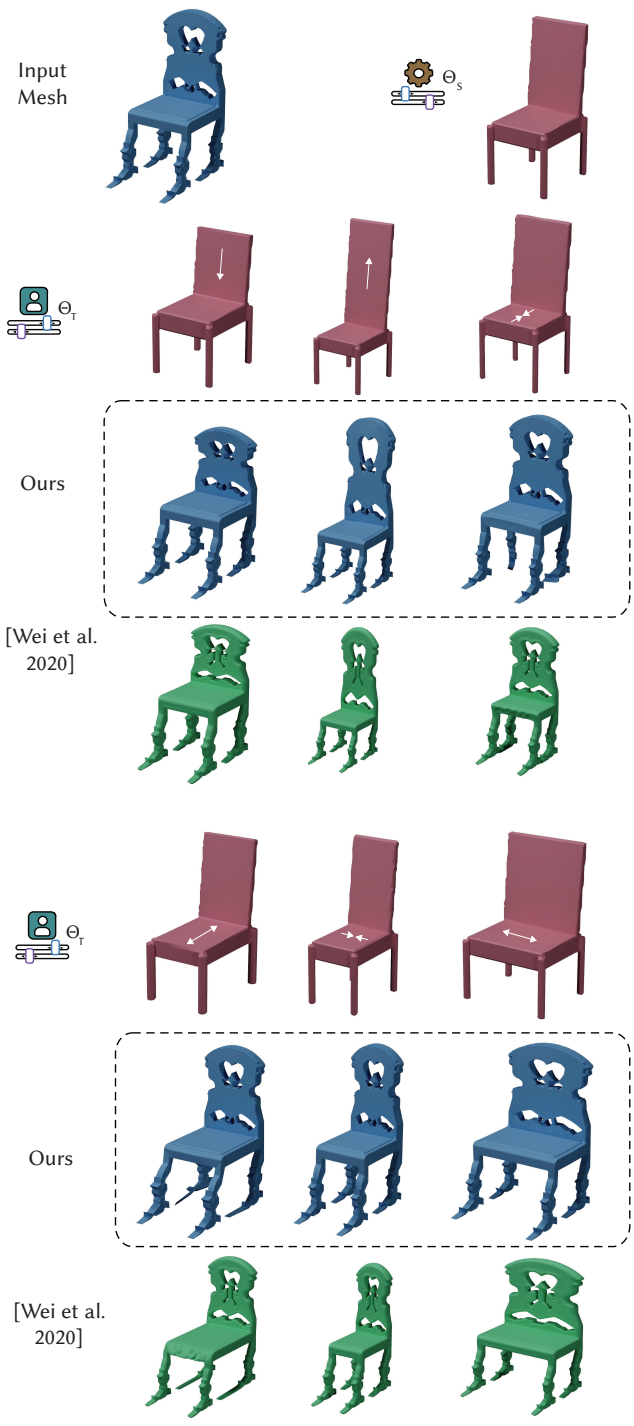



Fig. 12. Comparison to the deformation-based approach of Wei et al. [2020]. Our specific deformation formulation better preserves geometric details, in addition to allowing smooth large scale edits. Note that in this case, we are using the couch template which is able to represent chairs as well.

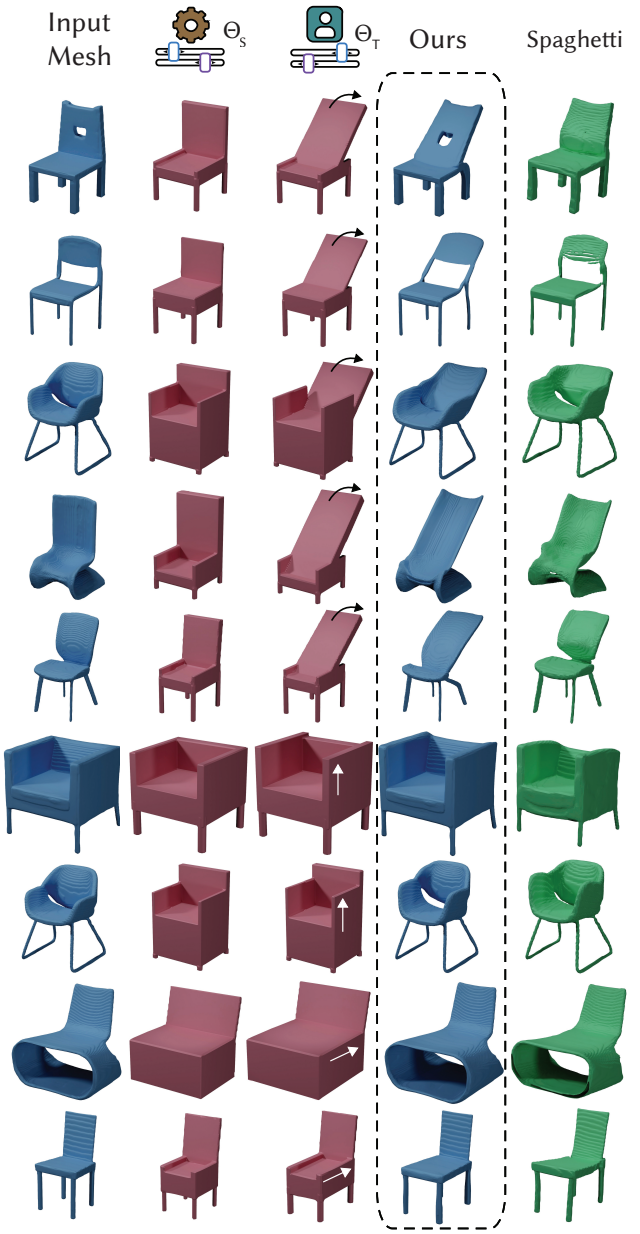
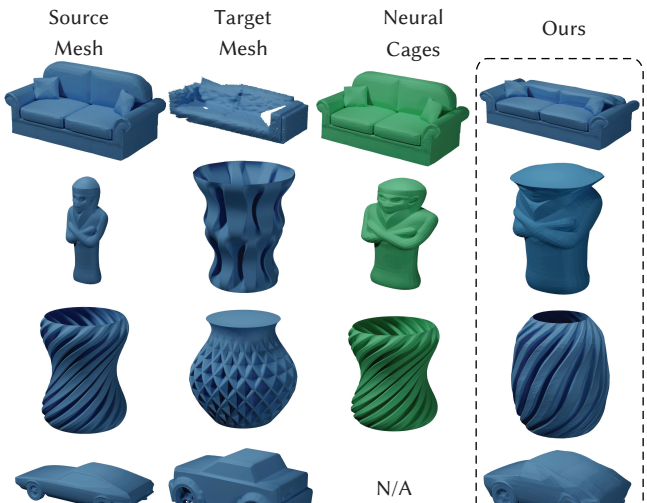



Fig. 13. Comparison to the generated results of *Spaghetti* [Hertz et al. 2022] on samples from their test set. The learned nature of their approach limits expressivity of the model, in particular for large scale edits such as tilting the back. In addition, non edited parts are not well preserved.



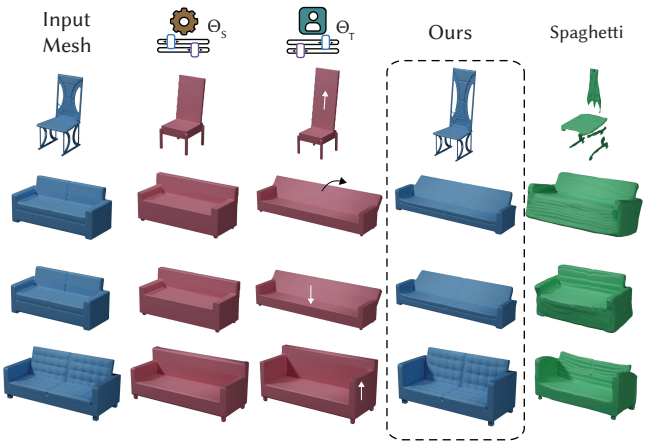



Fig. 14. Comparison to the generated result of *Spaghetti* [Hertz et al. 2022] on new samples. The limited resolution and generalization capability of the neural model damages geometric details and specificities of the input, and prevents generating meaningful or usable edited shapes.

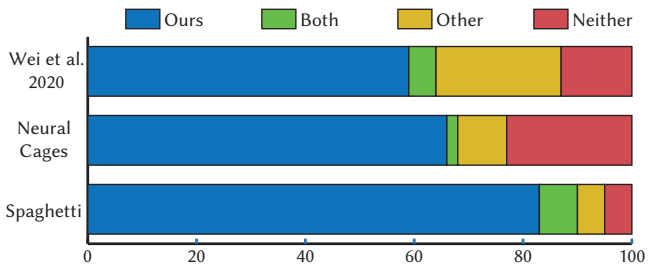



Fig. 11. Results of perceptual study: The $x$-axis indicates the percentage of a choice across instances averaged across responses. For each comparison, we had around 30 instances answered by 17 users. The results of our approach have been consistently preferred by respondents over all three compared methods.

*on Graphics (TOG)* 41, 4 (2022), 1–20. https://doi.org/10.1145/3528223.3530084

Inigo Quilez and Pol Jeremias. 2013. *Shadertoy*. Retrieved May 23, 2023 from https://shadertoy.com

Fangyin Wei, Elena Sizikova, Avneesh Sud, Szymon Rusinkiewicz, and Thomas Funkhouser. 2020. Learning to infer semantic parameters for 3D shape editing. In *2020 International Conference on 3D Vision (3DV)*. IEEE, 434–442. https://doi.org/10.1109/3DV50981.2020.00053

Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural cages for detail-preserving 3d deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 75–83. https://doi.org/10.1109/CVPR42600.2020.00015