Virtual Elastic Objects
—Supplemental Material—

Hsiao-yu Chen[1,3], Edith Tretschk[2,3], Tuur Stuyck[3], Petr Kadlecek[3], Ladislav Kavan[3], Etienne Vouga[1], Christoph Lassner[3]
[1]University of Texas at Austin, [2]Max Planck Institute for Informatics, [3]Meta Reality Labs Research

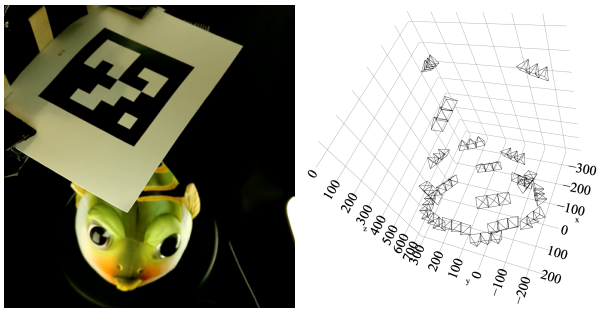
This supplemental material contains additional information on several aspects covered in the main paper.

1. Capture

Table 1. *Purchase links for the PLUSH dataset.* These are Amazon links for all of the items we acquired for our PLUSH dataset (links working as of September 2021).

Object	Baby Alien	Dino Rainbow	Dino Blue	Dino Green	Fish	Leaf	Serpentine	Mr. Seal	Pillow	Pony	Dog	Sponge
Link												

The PLUSH dataset. With the presented method and dataset, we hope to encourage further research in the direction of the reconstruction of elastic objects. To facilitate future work, we will publish the PLUSH dataset (our recordings as well as our reconstructed point clouds). However, since the dataset itself is insufficient for research that involves improvements to data capture, we provide the Amazon links to buy all objects of the PLUSH dataset in Table 1 to enable researchers to use novel capture techniques and compare their results to the presented method.



(a) (b)

Figure 1. *Capture System.* **a)** Top-down view during a capture from the view of one of the three cameras used for tracking the air exhaust nozzle. The ArUco marker is visible prominently. In the background, the 'fish' plushy can be seen during capture. **b)** Typical camera layout during a capture. The three cameras used for tracking are around 60cm away from the capture subject; the remaining cameras roughly form two circles.

Time Synchronization. Ideally, the cameras of the capture system are time-synchronized via gen-lock (generator locking, *i.e.*, a single source controls all camera shutters). However, the cameras we used lack this feature and we found that even within each triplet, the grey-scale and RGB sensor are not gen-locked and drift enough to accumulate an offset of multiple frames over tens of seconds. We use a one-camera-per-controller configuration with no other devices using the USB bus on each recording host, thus ensuring un-blocked transmission of frames to the host and use the received time as frame timestamp. By synchronizing the controllers using the Precision Time Protocol (unfortunately only using software timestamps), we manage to achieve mostly reliable time synchronization.

Camera Layout and Recording Protocol. Figure 1 shows an example view into the capture system during a capture and an overview of the camera layout. We fix the objects using adhesive strips onto a static recording surface. Depending on the object type, we attach fishing line onto extremities to 1) move the extremities during the recording and 2) keep the extremities fixed while moving the air stream over the object surface. Before starting the recording, we move all extremities into a neutral position that allows seeing them on camera from all angles—this helps forming the 'canonical space' reconstruction for NR-NeRF. After starting the recording, we first move all extremities symmetrically (for example, arms and ears all forward, then all backward; fins down, then fins up). After concluding all extremity movement, we fix the extremities in place in the neutral position and start moving the air stream over the object surface. Overall, we end up with recording times between 32s and 67s.

2. 4D Reconstruction

Regularization Losses. We modify NR-NeRF in several ways to improve the result quality. The input videos contain background, which we do not want to reconstruct. We obtain foreground segmentations for all input images via image matting [1] together with a hard brightness threshold. During training, pixels that are marked as background do not use the reconstruction loss but instead use a background

loss on the S point samples along the ray:

$$L_{background} = \frac{1}{S} \sum_{s \leq S} o(\mathbf{x}_s + \mathbf{b}(\mathbf{x}_s, \mathbf{l}_t)). \quad (1)$$

$L_{background}$ clears spurious geometry from empty space. When later extracting point clouds, we need opaque samples on the inside of the object as well. However, we find that $L_{background}$ leads the canonical model to prefer empty space even inside the object. We counteract this with a density loss

$$L_{density} = -\frac{1}{S} \sum_{s \leq S} \mathbf{1}_{0.8 \leq \sum_{s' < s} \alpha_{s'}} o(\mathbf{x} + \mathbf{b}(\mathbf{x}, \mathbf{l}_t)), \quad (2)$$

which raises the opacity of point samples along a ray that are 'behind' the surface, *i.e.* samples whose earlier samples have an accumulative alpha value of at least 0.8. This loss term acts on foreground pixels only. Furthermore, we encourage the space in front of the surface to be empty as well by using:

$$L_{foreground} = \frac{1}{S} \sum_{s \leq S} \mathbf{1}_{0.3 \geq \sum_{s' < s} \alpha_{s'}} o(\mathbf{x} + \mathbf{b}(\mathbf{x}, \mathbf{l}_t)). \quad (3)$$

We get slightly better results by first building a template by pretraining the canonical model on frames with $t \leq 2$ for $50k$ iterations and subsequently tracking the template by only optimizing for the ray-bending network (and its inverse) for all images for $150k$ iterations.

Greyscale Input. Our capture system not only provides RGB streams but also greyscale images. We use these for supervision as well. However, our appearance model $\mathbf{c}(\mathbf{x})$ regresses RGB values and hence renders colored pixels. During training, we optimize for a linear transformation parameterized by $(a_r, a_g, a_b) \in \mathbb{R}^3$ to convert the pixel color (r, g, b) to grey-scale $a = a_r r + a_g g + a_b b$. We share (a_r, a_g, a_b) between all grey-scale input images.

Inverting the Deformation Model. We train a coordinate-based MLP $\mathbf{w}$ to approximate the inverse of the ray-bending network. To that end, we use an ℓ_1 cycle-consistency loss on each point sample $\mathbf{x}$:

$$L_{cycle} = |\mathbf{w}(\mathbf{x} + \mathbf{b}(\mathbf{x}, \mathbf{l}_t), \mathbf{l}_t) + \mathbf{b}(\mathbf{x}, \mathbf{l}_t)|, \quad (4)$$

which encourages $\mathbf{w}$ to regress the negative of $\mathbf{b}(\mathbf{x}, \mathbf{l}_t)$ at $(\mathbf{x} + \mathbf{b}(\mathbf{x}, \mathbf{l}_t), \mathbf{l}_t)$. To focus on the relevant areas, we do not apply this loss term to background pixels. In order to not influence the reconstruction, we do not backpropagate gradients from this loss term into $\mathbf{b}$.

Figure 2. *Point cloud before and after the clean up.* We show the comparison of the reconstructed point cloud before and after the removal of background noise.

Point Cloud Extraction. After training, we need to convert the reconstruction from its continuous MLP format into an explicit point cloud. To achieve that, we cast rays R from all input cameras and extract points that are at or behind the surface and whose opacity exceeds a threshold. Specifically, we define the surface for a ray $\mathbf{r} \in R$ as being located at the point sample $s^*_\mathbf{r}$ with accumulated alpha values $(\sum_{s' < s^*_\mathbf{r}} \alpha_{s'})$ closest to 0.5, the median. We consider all samples $s \geq s^*_\mathbf{r}$ that fulfill $\alpha_s \geq 0.05$ as part of the extracted point cloud. These samples $\mathbf{x}_{s,\mathbf{r}}$ can then be deformed from the canonical model into the deformed state at time t via the inverse network: $\mathbf{x}_{s,\mathbf{r},t} = \mathbf{x}_{s,\mathbf{r}} + \mathbf{w}(\mathbf{x}_{s,\mathbf{r}}, \mathbf{l}_t)$. For time t, we take all these samples from all rays to obtain a deformed point cloud $P_t = \{\mathbf{x}_{s,\mathbf{r},t} | \alpha_s \geq 0.05, s \geq s^*_\mathbf{r}, s \in \mathbf{r}, \mathbf{r} \in R\}$. We thus have a 4D reconstruction in the form of a 3D point cloud's evolving point positions $\{P_t\}_t$, which are in correspondence across time.

The resulting point cloud contains spurious points due to imperfections in the reconstruction. Although there are usually a handful of points close to the object, most of these spurious points occur as clusters far away from the object. The latter we eliminate using the object bounding box, and we remove the former manually. Given that the spurious points have much lower density compared to the object of interest, the cleanup can also be done automatically by removing point cloud cluster under certain density threshold. See Figure 2 for an example comparison before and after cleanup.

3. Simulation

Newton Optimization. We use the standard Newton's method with backward line search to find the equilibrium position $\mathbf{y}$ that solves the variational problem:

$$\arg\min_{\mathbf{y}} \mathbf{E}(\mathbf{y}). \quad (5)$$

Starting with an initial guess $\mathbf{y}_0$, we update the solution iteratively using the update step

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \alpha_l \mathbf{dy}_{k-1} \quad (6)$$

$$\mathbf{dy}_{k-1} = \mathbf{H}^{-1}_{k-1} \nabla \mathbf{E}_{k-1}, \quad (7)$$

where $\mathbf{H}_k$ and $\nabla\mathbf{E}_k$ are the Hessian and gradient of the energy $\mathbf{E}$ evaluated at the current iterate $\mathbf{y}_k$. We iterate over the solution until the norm of the Newton update, $\|\mathbf{dy}\|$, is sufficiently small. We compute the step size α_l using a standard backtracking line search (starting at $\alpha_l = 1$, we halve the step size until $\mathbf{E}(\mathbf{y}_k) < \mathbf{E}(\mathbf{y}_{k-1})$). The value of α_l is not a continuous function of the object material parameters, but in practice, is equal to 1 during almost every Newton step; we therefore treat it as constant when performing automatic differentiation of $\mathbf{y}$.

Initial Guess. To reduce the number of Newton iterations required for each forward simulation and to capture hysteretic effects (where $\mathbf{E}$ has multiple local minima and the equilibrium state of the object depends on the past trajectory of the object), we warm-start the optimization by setting the initial guess $\mathbf{y}_0$ to the equilibrium position from the previous frame, for each consecutive frame sequence used during training and validation. For the first frame in a frame sequence, we choose the object reference pose as our initial guess.

Position Constraints. During forward simulation, certain points on the object boundary are subject not only to external forces, but also to position constraint. For example, some objects were taped to the ground during capture, or had fishing line attached to extremities. Additionally, we do not attempt to model static friction of the PLUSH objects against the ground; we assume friction is sufficiently strong to keep points in contact with the ground fixed in place.

Although one could enforce these position constraint using hard constraints (by removing those points as degrees of freedom in the optimization of $\mathbf{y}$), doing so is problematic in practice due to the presence of the logarithmic term in the Neo-Hookean elastic energy. An abrupt change in the positions of constrained vertices (due to tugging on the fishing line, e.g.) can lead to the inversion of the elements and thus to a negative J, which makes the elastic energy undefined. To prevent this, we treat the constraints as soft instead, using a penalty potential term as described in the main text.

4. Rendering

We can upsample the point cloud generated by the simulation to make it denser, e.g. for visualization. Unlike for rendering, we need to consider forward warping for this case. We first extract points from the canonical model as described in Sec. 3.2 (main paper) to obtain a fuller point cloud $F_c = \{\mathbf{y}_f^c\}_f$. From the simulation, we have a less dense point cloud, namely S_c and S_d. We now seek to replace the forward warping of $\mathbf{w}$: We interpolate the deformation offsets $\mathbf{d}_l^f = \mathbf{x}_l^d - \mathbf{x}_l^c$ with IDW to obtain the dense, deformed point cloud $F_d = \{\mathbf{y}_f^d\}_f$:

$$\mathbf{y}^d = \mathbf{y}^c + \sum_{s\in\mathcal{N}} \frac{w_s}{\sum_{s'\in\mathcal{N}} w_{s'}} \mathbf{d}_s^f, \tag{8}$$

where $\mathcal{N}$ and w_s are defined analogously to Eq. 14 (main paper).

5. Runtime

The upper bound of the run time required for each step of our pipeline is $20h$ for reconstruction on four NVIDIA V100 GPUs, $13h$ for learning the parameters on a AMD Ryzen 5 1600 six-core processor, $5min$ for running a new simulation, and $10min$ for rendering a frame.

References

[1] Shanchuan Lin, Andrey Ryabtsev, Soumyadip Sengupta, Brian Curless, Steve Seitz, and Ira Kemelmacher-Shlizerman. Real-Time High-Resolution Background Matting. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8762—-8771, dec 2021. 1