Supplementary: Approximate Differentiable Rendering with Algebraic Surfaces

Leonid Keselman and Martial Hebert

Carnegie Mellon University, Pittsburgh PA, USA
{lkeselma,hebert}@cs.cmu.edu

1 Video Results

Here we describe additional details about the experiment shown in Fig. 9 of the main paper. Concerning the differentiable renderer: the method, settings and hyperparameters are identical to those in Fig. 7 and 8 and Section 5.3. We simply run the method on different input.

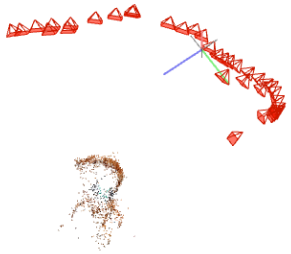
We collected a 14 second video with a Samsung S9 cell phone at 1280 x 720 resolution at 30 Hz. The video contains motion blur, auto-exposure, and clearly visible video compression artifacts, making it unsuitable for some reconstruction methods. We sub-sampled the video down to 6Hz and ran Mask RCNN [3] from Detectron2 [15] with the pre-trained weights `COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml` to detect objects. In our case, the object was detected as part of the teddy bear class, with about 55 viable frames. We ran COLMAP [12] to obtain camera poses for those frames, where COLMAP successfully returned 36 frames with valid camera poses. We ran our SFS pipeline at 160 x 90 resolution to obtain the results shown. Visual examples from this pipeline are shown in Fig. 1. All the methods used their default settings; there was no parameter tuning involved.

1.1 Video Result Analysis

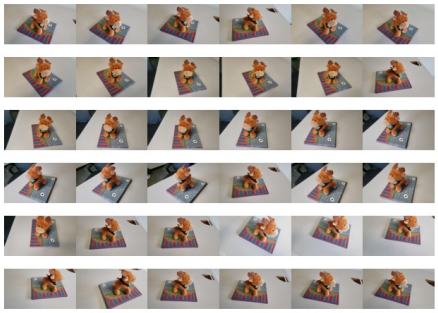
The trajectory shown here only covers about half of the object from a roughly constant elevation. Complicating the reconstruction is that the camera poses are imperfect due to estimation and unmodeled camera distortion. Much more significant is that the Mask RCNN silhouettes used for reconstruction are often extremely under or over segmented.

Despite these issues in the "ground truth" used for optimization, the low degree of freedom of Fuzzy Metaballs allows the model to reasonably recover from the massive artifacts. While the result in the main paper shows the default 50% threshold, to recover some areas, we have to lower our α threshold to 10%.

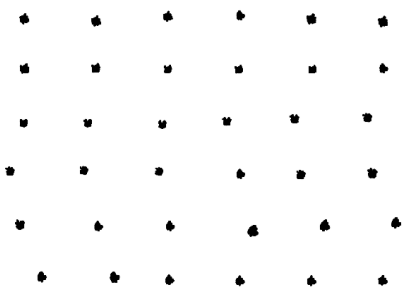
(a) Mask RCNN output for valid frame



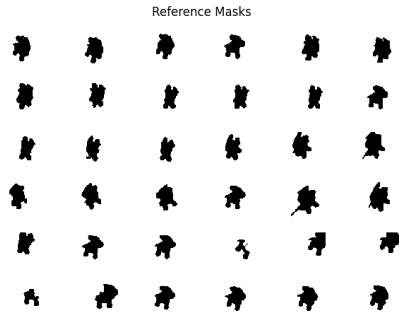
(b) COLMAP estimate of camera poses



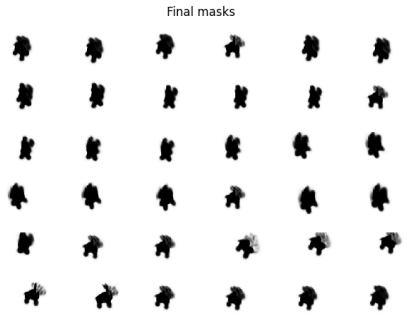
(c) All 36 frames used for SFS



(d) SFS initialization



(e) Mask RCNN Silhouettes



(f) SFS Mask Results

Fig. 1: **Video-based SFS reconstruction**

2 Hyper-parameters

Our proposed method has 5 hyper-parameters described in the paper. Briefly, β_1 prioritizes close hits, β_2 prioritizes hits closer to the camera, β_3 prioritizes hits in front of the camera, and β_4 and β_5 serve as a scale and offset to generate alpha masks. Since our system is fully algebraic, it is possible to perform gradient descent into these hyper-parameters (and the functional form of JAX naturally returns their gradients), but this was not done.

Instead, we optimized them for depth and alpha mask accuracy over a small simulated dataset of the Stanford bunny using standard black-box optimization techniques [1,2,10] before running most of our experiments. We found that the ray-based renderer led to similar optimal hyperparameters across multiple tested resolutions, across a wide range of mixture components, and across our linear, quadratic and cubic methods of intersection computation.

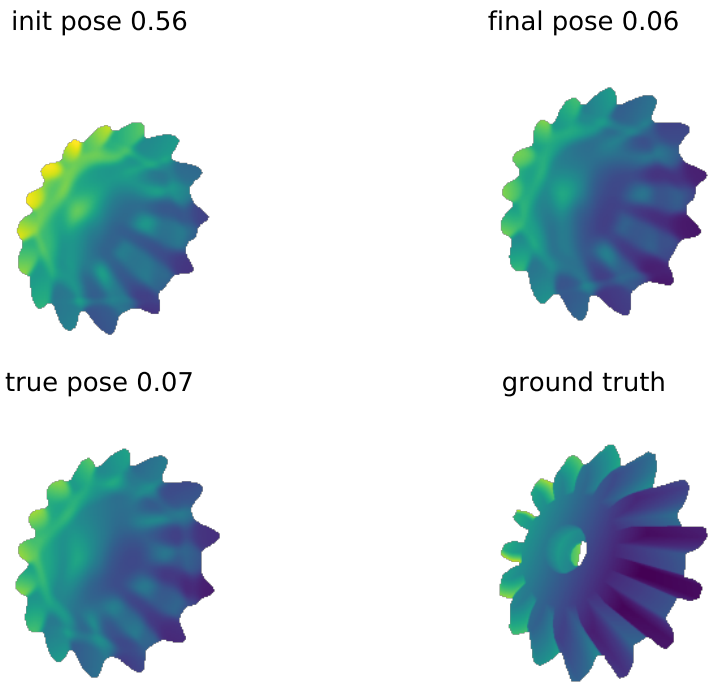


Fig. 2: **Gear Results with Fuzzy Metaballs** Final pose describes the final pose after gradient-based pose optimization, while true pose is rendered view from the ground truth pose. Ground truth is the Blender-generated depth map of the full-fidelity model. The final pose shown here has a rotational error of 23.9 degrees. However, the gear has 15 teeth and hence a 24.0 degree symmetry.

3 Exclusion of `gear` model

The `gear` model was selected because of its interesting geometry from Thingi10k [17]. However, for pose estimation, we exclude its results from the overall average due to symmetry. Our poses are generated with rotations of uniform axis and angle uniformly between -45 and 45 degrees (uniform-axis random spin [13]). The gear model however has 15 teeth and a rotational symmetry of 24 degrees when viewed from one side, as seen in Fig. 2. This can sometimes produce pose errors with no real geometric error.

The model itself is not symmetric, with 15 gears and a back face with 180 degree symmetry. But with a single view, our testing conditions can generate poses which are geometrically correct but produce pose errors. The other model with symmetry, `eiffel`, only has 90 degree symmetry and our testing conditions place all random poses in the same local minima.

We don't use the `yoga` or `plane` models for pose estimation as we only latter added them for the reconstruction experiments. Both models originate from prior differentiable rendering uses in reconstruction [8,16].

4 Pose Estimation Details

We include noise-free results the same seed as the noisy results in the main paper. Summary plots are shown in Fig. 3 and Fig. 4. In the noise-free case, we find that Point-to-Point ICP works better. With noise, Point-to-Plane ICP methods perform better.

4.1 Noise Free

As described in the paper, when ICP methods perform well, they perform extremely well, an order of magnitude better than the differentiable renderers (see the log-scale plot), to fractions of a degree since they have high resolution samples. However, sometimes ICP finds poor local minima and on average our method performs better, even when ICP has a dense point cloud. Despite having a better mean, Fuzzy Metaballs (FM) have a median error that is 8 times higher and a 25th percentile error that is 10 times higher. The increase in robustness from FM is demonstrated in lower 75th percentile errors.

4.2 Noisy Depth Images

With synthetic noise, both differentiable renderer methods are barely affected, while the ICP results see a large degradation in peak performance. Under this experimental condition, Fuzzy Metaballs have the lowest mean, median, 25th and 75th percentile errors (typically by a factor of 2 compared to ICP).

Interestingly, some of the worst case performance of the ICP methods disappears (lower q3 measurements) when noise is added. We hypothesize that this occurs due to a form of symmetry breaking that helps avoid singularities and bad correspondences. Fuzzy Metaballs, being a low fidelity model, experience nearly no degradation in performance when noise is added to depth images.

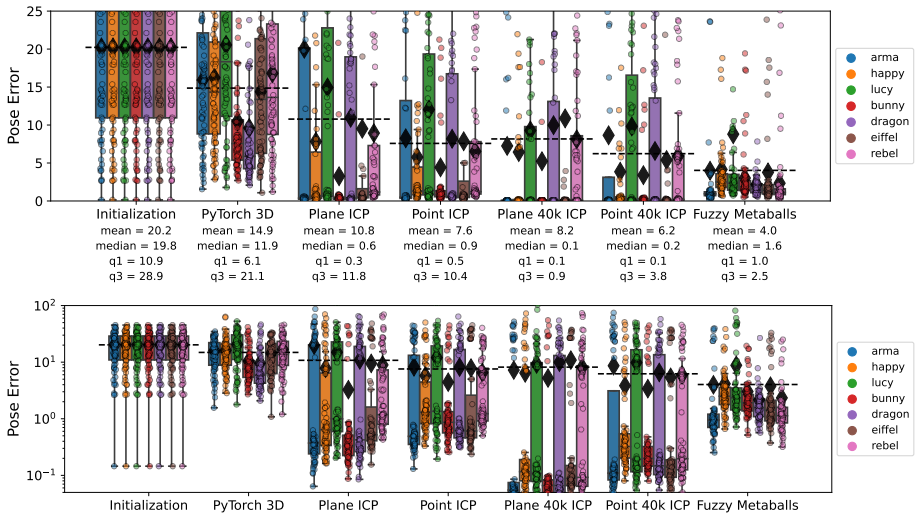


Fig. 3: **Noise Free Pose Estimation** Linear scale plot above and log-scale below. Dashed lines are averages for the method, while the black diamonds show the average for that method and model. Statistics for each method are listed. `gear` model is excluded from statistics.

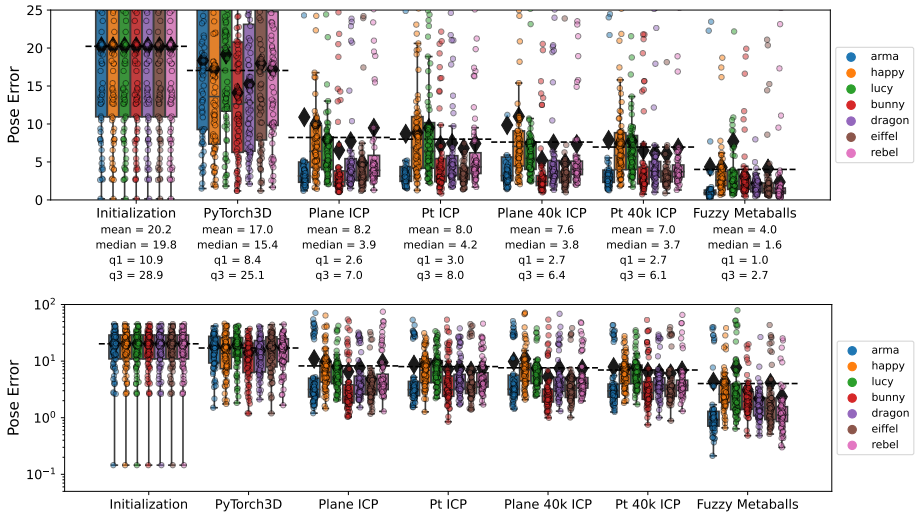


Fig. 4: **Noisy Pose Estimation** Identical visualization to Fig. 3. Linear scale figure is identical to that in the main paper.

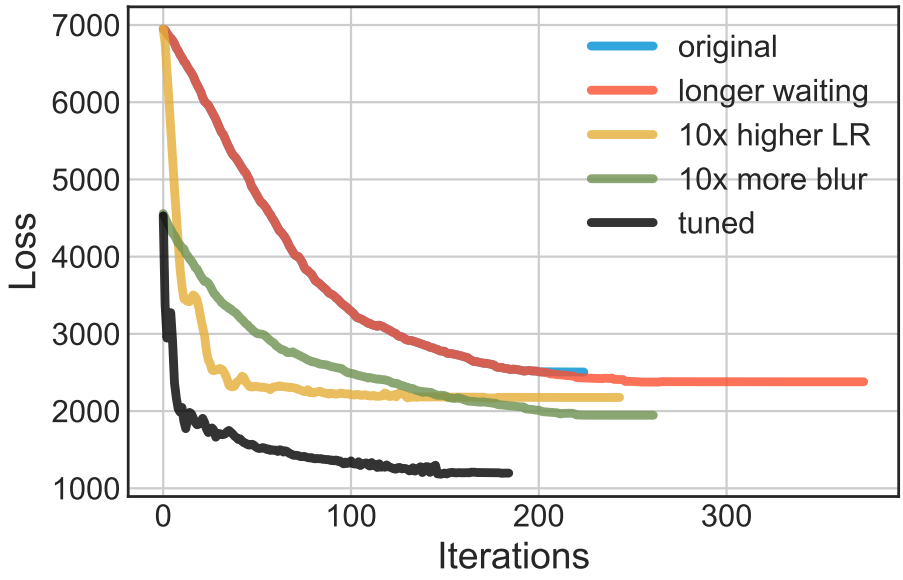


Fig. 5: **PyTorch3D baseline** convergence curves using different hyperparameters for pose estimation. All curves produce roughly equivalent pose errors, significantly worse than FM or ICP.

5 SoftRasterizer performance

One might wonder about why our baseline of PyTorch3D, implementing SoftRas [8], performs so poorly in the pose estimation experiments. Prior work on differentiable rendering [8,7,16] demonstrates their pose optimization experiments with mostly single, visual examples. These often use color images, and provide no baselines results from standard methods. Quantitative results in SoftRas [8] examined solving for rotation uncertainty in images of a colored cube and their resulting rotation errors averaged over 60 degrees. It is perhaps not surprising that our pose estimation experiments, featuring a family of models, simultaneous rotation and translation, while optimizing only depth and silhouettes, might challenging these methods.

We used the hyperparameters from the current PyTorch 3D [11] *Camera position optimization* sample. We tuned learning rates to behave well with our depth + silhouette loss function, and followed an automatic learning rate schedule [5]. As can be seen in Fig. 6, the pose optimization performs reasonably well in reducing image errors. However, the optimized pose still demonstrates visual errors compared to the ground truth pose. Even worse, the optimization perturbs the pose in such a way that the pose error at the end of optimization (16 degrees and 15%) is worse than the pose errors at the perturbed initialization (12 degrees and 8%), despite the significant reduction in loss.

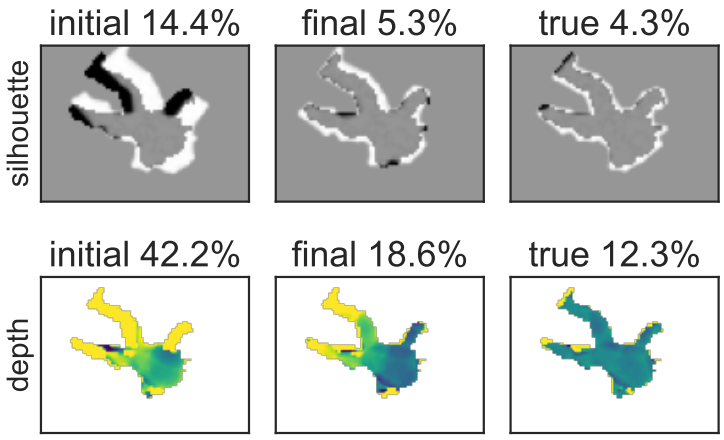


Fig. 6: **PyTorch3D baseline** Visualization of errors seen with pose optimization. Initial is an example pose perturbation of the `arma` model. Final is the result after pose optimization, and true is the result of the ground truth pose. Silhouette error is in percent of pixels that are wrong, while depth error is in average relative depth error. Optimization leads to a reasonable decrease in both.

To check if the hyper-parameters from the PyTorch3D sample was a poor fit, we searched for settings which produced good pose estimation for a single frame. We used CMA-ES [1,2], a fairly common black box method [10]. This type of task-specific hyper-parameter optimization was **never performed** for our Fuzzy Metaballs experiments. We only performed these experiments on an existing baseline to examine how good it might perform in the best case. Convergence curves can be seen in Fig. 5.

All our manual tweaking of PyTorch3D hyper-parameters produced comparable configurations (17-18 degrees of rotation, 15-16 percent translation). The automated optimization found a setting which produced 16 degrees of rotation error and 9 percent of translation error, still worse than the perturbed initialization. However, these settings used a very high learning rate that proved unstable with other frames. Lowering to learning rate resulted in settings with a negligible improvement (2%) to our initial settings. These tests suggest our initial hyper-parameter choices were a reasonably good setting for the baseline method.

Further parameter search with a constraint on learning rate failed to find parameters significantly improved from the defaults. Optimization was over 8 parameters: σ, γ, blurring radius for both depth and silhouette, faces per pixel, learning rate, and multipliers for depth and silhouette loss.

Both the Fuzzy Metaballs and PyTorch3D optimization use an axis-angle, 3 parameter rotation estimation. These is some evidence suggesting PyTorch Autograd for $SO(3)$ might be unstable at times in its native form [14].

5.1 Pulsar performance

Our attempts to test a recent differentiable renderer, *Pulsar*, found it performed very poorly. Not only are there software bugs with the latest PyTorch3D at the time of writing (`0.6.1`) where the code clobbers camera data-structures and requires re-creating them with every call to the render function, but the pose estimation results were very poor.

We used the same settings as the Point Cloud Differentiable Renderer baseline we tested, which provided fair results and produced visually similar outputs. Compared to our base learning rate, reducing it by a factor of two led to flat loss. Increasing it by a factor of two led to divergence and NaNs.

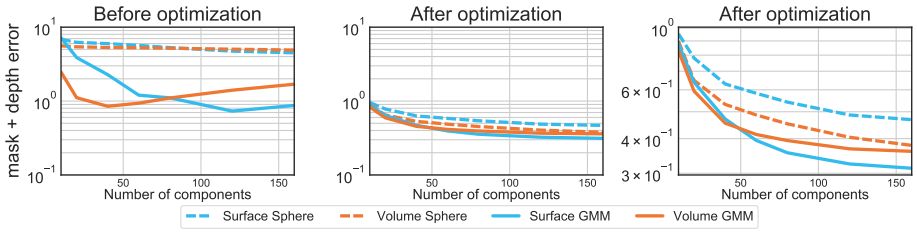


Fig. 7: Optimizing Fuzzy Metaballs from different initializations.

6 Exporting Fuzzy Metaballs

We experiment with exporting fuzzy metaballs as a mesh by running marching cubes [9]. To find an ideal isosurface level, we run optimization to ensure that the centroids of the voxels match the silhouettes over a sample set of views. This leads to results like those in Fig. 8.

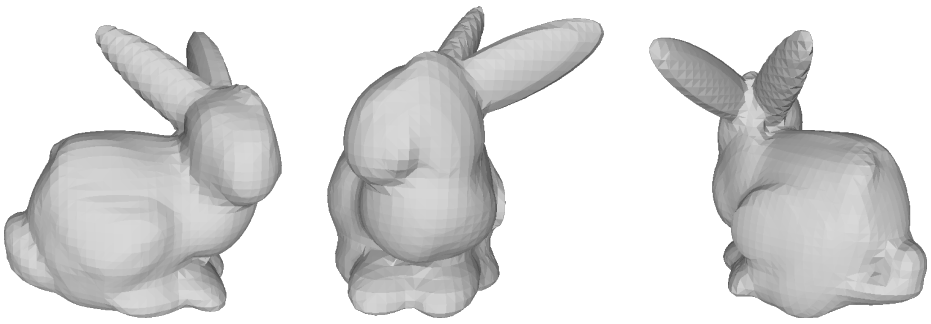
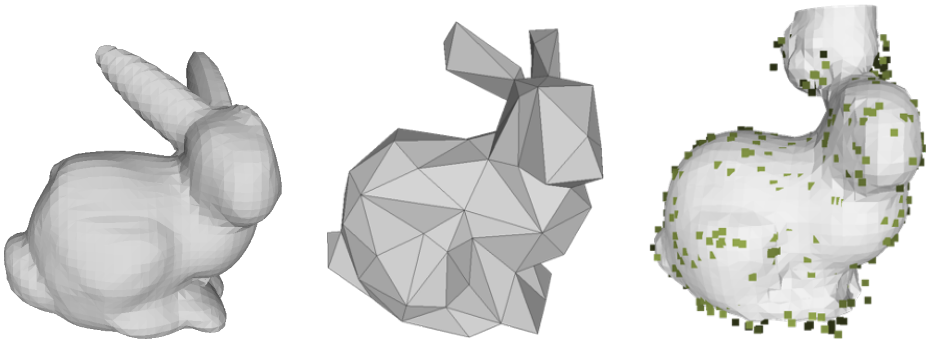


Fig. 8: Mesh extracted from a 40 mixture fuzzy metaball using marching cubes



(a) 40 component Fuzzy Metaballs (400 params)

(b) 170 face, 85 vertex Mesh (810 params)

(c) 430 Points reconstructed [4] (1290 params)

Fig. 9: Equivalent representations visualized, per the experiments in the paper.

7 Fuzzy Metaballs as Surface or Volume GMMs

To understand what classical formulation best matches Fuzzy Metaballs, we try optimizing models with different initializations. We start with both sphere and EM-fit GMM initializations, with surface and volume versions of both. Quantitative results averaged across all 10 models are shown in Fig. 7. Qualitative results for the `Yoga` model are shown in Fig. 14.

At low mixture numbers, Fuzzy Metaballs perform more like a volume GMM, while at high mixture numbers, surface GMMs work better. Often using a GMM as a FM model will produce reasonable results. We use constant hyperparameters from our 40 mixture tuning, and perhaps the out-of-the-box vGMM rendering could improve by adding proper scaling with component number. Finally, all initializations respond very well to optimization, and optimized sphere-initialized models always outperform the models fit with solely with EM.

The Fuzzy Metaballs improve with more components across our entire range of testing. This suggests the asymptotic behavior seen in the **Comparing Representations** section is due to experimental factors of those experiments, and not the representation itself. This is somewhat expected as those experiments use the mesh representation as ground truth and all other formats are sampled.

Lastly, we can see that the fitting process produces no over-fitting as novel and training frames have identical behavior in Fig. 13.

(a) Visualizing normal maps while sweeping β_1 and β_2 demonstrates smoothing.

(b) Sweeping β_4 and β_5 controls the sharpness and extent of the alpha masks.

Fig. 10: Hyperparameter visualization

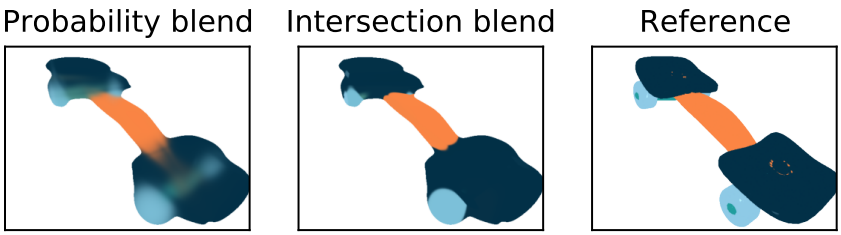


Fig. 11: Rendering Fuzzy Metaball color images of a snakeboard [6] with two forms of blending: one behaves more like a volume where the wheels of the object can be seen, while the other behaves more like a surface with proper occlusion. Shown is a 40 component vGMM with a single color per component. Cartoon-like appearance is from exclusively using ambient lighting.

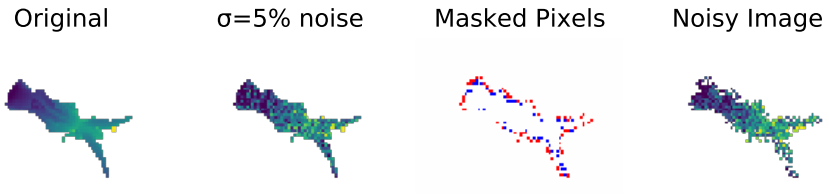


Fig. 12: **Synthetic noise generation**. Gaussian noise is combined with perturbed silhouettes (red pixels are added, blue are removed).

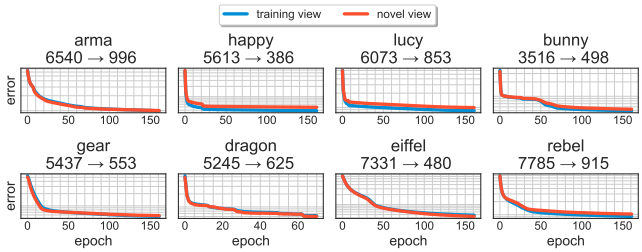
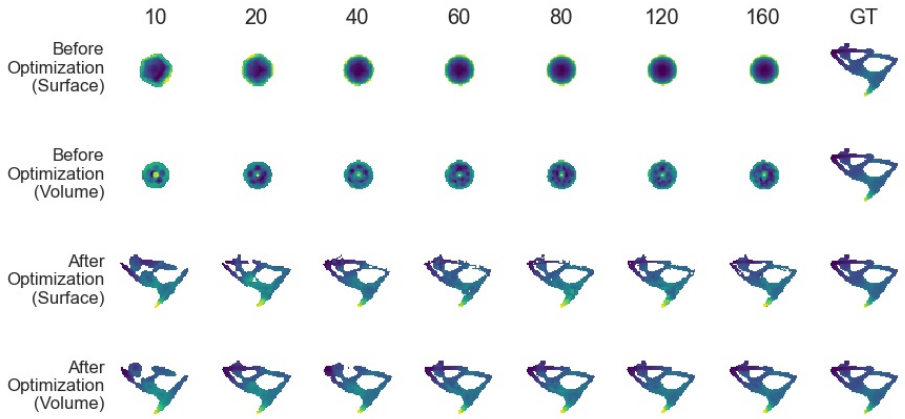
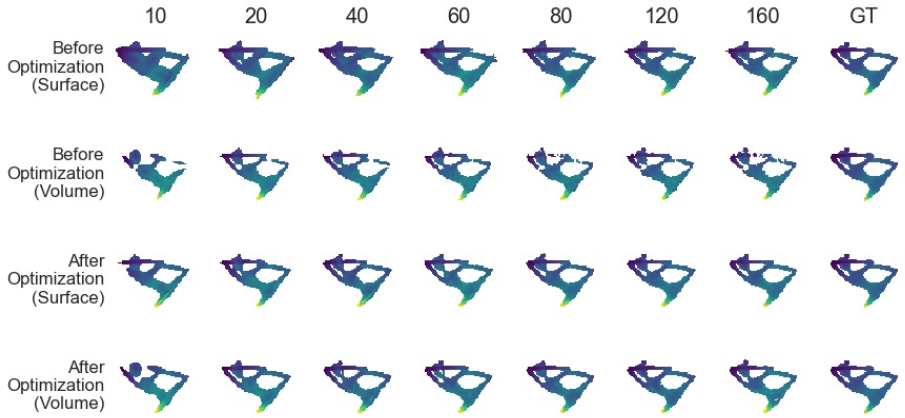


Fig. 13: Optimizing Fuzzy Metaballs from a sphere to a shape. Losses are given for training frames and novel viewpoints, showing no significant difference.



(a) Sphere Initialization



(b) GMM Initializations

Fig. 14: Visual examples of Fuzzy Metaballs at different component numbers, for different initializations, before and after optimization. All images are 60 by 80 pixels and show depth with color coding. Here, unlike the rest of the paper, colors are scaled for maximum contrast, not consistency between images. GT is the ground truth depth map from the mesh rendered by Blender.

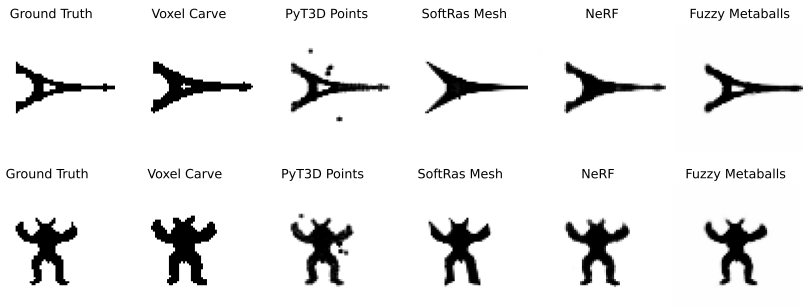


Fig. 15: **Shape from Silhouette Results**. The mesh-based representation cannot change genus from a deformed sphere into the eiffel tower. The point cloud method leaves spurious points. The classic Voxel Carving method is not that precise with 384^3 volume but only 32 views of low resolution 64 x 64 images. NeRF does a good job but might improve with parameter tuning.

References

1. Hansen, N.: The cma evolution strategy: A tutorial (2016)
2. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634 (Feb 2019). https://doi.org/10.5281/zenodo.2559634, https://doi.org/10.5281/zenodo.2559634
3. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask r-cnn. 2017 IEEE International Conference on Computer Vision (ICCV) pp. 2980–2988 (2017)
4. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson Surface Reconstruction. In: Sheffer, A., Polthier, K. (eds.) Symposium on Geometry Processing. The Eurographics Association (2006). https://doi.org/10.2312/SGP/SGP06/061-070
5. King, D.: Automatic learning rate scheduling that really works (Feb 2018), http://blog.dlib.net/2018/02/automatic-learning-rate-scheduling-that.html
6. Kobilarov, M., Crane, K., Desbrun, M.: Lie group integrators for animation and control of vehicles. ACM Trans. Graph. **28** (May 2009)
7. Lassner, C., Zollhöfer, M.: Pulsar: Efficient sphere-based neural rendering. arXiv:2004.07484 (2020)
8. Liu, S., Li, T., Chen, W., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
9. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics **21**(4), 163–169 (1987)
10. Loshchilov, I., Hutter, F.: Cma-es for hyperparameter optimization of deep neural networks (2016)
11. Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., Gkioxari, G.: Accelerating 3d deep learning with pytorch3d. arXiv:2007.08501 (2020)
12. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: European Conference on Computer Vision (ECCV) (2016)
13. Stanfill, B.: Statistical methods for random rotations. Ph.D. thesis, Iowa State University (2014)
14. Teed, Z., Deng, J.: Tangent space backpropagation for 3d transformation groups (2021)
15. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. https://github.com/facebookresearch/detectron2 (2019)
16. Yifan, W., Serena, F., Wu, S., Öztireli, C., Sorkine-Hornung, O.: Differentiable surface splatting for point-based geometry processing. ACM Transactions on Graphics **38**(6), 1–14 (Nov 2019). https://doi.org/10.1145/3355089.3356513
17. Zhou, Q., Jacobson, A.: Thingi10k: A dataset of 10, 000 3d-printing models. CoRR **abs/1605.04797** (2016), http://arxiv.org/abs/1605.04797