SN Computer Science (2023) 4:245
https://doi.org/10.1007/s42979-022-01663-3

ORIGINAL RESEARCH

Towards 3D Scene Understanding Using Differentiable Rendering

Arul Selvam Periyasamy[1] · **Sven Behnke**[1]

Received: 22 June 2022 / Accepted: 30 December 2022 / Published online: 3 March 2023

Abstract
Deep learning methods have achieved significant results in many 2D computer vision tasks. To realize similar results in 3D tasks, equipping deep learning pipelines with components that incorporate knowledge about 2D image generation from the 3D scene description is a promising research direction. Rasterization, the standard formulation of the image generation process is not differentiable, and thus not compatible with the deep learning models trained using gradient-based optimization schemes. In recent years, many new approximate differentiable renderers have been proposed to enable compatibility between deep learning methods and image rendering techniques. Differentiable renderers fit naturally into the render-and-compare framework where the 3D scene parameters are estimated iteratively by minimizing the error between the observed image and the image rendered according to the current scene parameter estimate. In this article, we present StilllebenDR, a light-weight, scalable differentiable renderer built as an extension to the openly available Stillleben library. We demonstrate the usability of the proposed differentiable renderer for the task of iterative 3D deformable registration using a latent shape-space model and occluded object pose refinement using order-independent transparency based on analytical gradients and learned scene aggregation.

Keywords Differentiable rendering · Order independent transparency · Deformable registration

Introduction

In recent years, convolutional neural networks (CNNs) have been fundamental in achieving impressive results in many 2D computer vision tasks [1, 2]. The invariances and inductive biases embodied in the CNNs enabled efficient learning from images. Lately, CNNs are also used in architectures for 3D computer vision tasks. However, the inductive biases inherent to CNNs alone are not sufficient for 3D scene understanding [3, 4]. Thus, it is essential to imbue knowledge about 2D image generation from 3D scenes into models for 3D scene understanding. To this end, one promising area of research is differentiable rendering.

In computer graphics terminology, the process of generating 2D images from 3D scene descriptions is known as rasterization. The principle design choice of the standard rasterization implementation is the assumption that only one 3D face contributes to a 2D pixel. This discrete pixel assumption enables efficient parallelization. An unfortunate consequence of the discrete pixel assumption is that the standard rasterization is not differentiable. Despite the fact that deep learning methods inherit a lot of parallel processing techniques from the graphics pipelines to realize efficient parallelization on GPUs, the standard rasterization and deep learning methods remain incompatible. To address this limitation, lately, approximate differentiable rendering has been proposed. In this article, we present StilllebenDR, which is built on top of the Stillleben [5]. Stillleben enables online synthetic data generation for training deep learning models using OpenGL for efficient rendering. StilllebenDR is built as an extension to the Stillleben library with minimal overhead. In contrast to the well-known differentiable renderers [6–8], StilllebenDR benefits from the optimizations built into the OpenGL library. Thus, StilllebenDR is highly

This article is part of the topical collection "Advances on Computer Vision, Imaging and Computer Graphics Theory and Applications" guest edited by Kadi Bouatouch, Augusto Sousa, Mounia Ziat and Helen Purchase.

✉ Arul Selvam Periyasamy
periyasa@ais.uni-bonn.de

Sven Behnke
behnke@cs.uni-bonn.de

[1] Autonomous Intelligent Systems, University of Bonn, Friedrich-Hirzebruch-Allee 8, 53115 Bonn, Germany

scalable. In our previous work, we used an early version of the StilllebenDR to refine 6D object pose for all objects in a scene using abstract render-and-compare [9]. Here, we present additional use cases for StilllebenDR, namely 3D deformable registration and occluded object pose refinement. Our contributions include:

1. StilllebenDR, a differentiable renderer with PyTorch integration,
2. an end-to-end differentiable pipeline for deformable object registration using a latent shape-space model and differentiable rendering, and
3. a framework for joint object pose optimization and deformable registration to make our pipeline less susceptible to pose initialization errors.

In addition to our conference paper [10] presented at the 17th International Conference on Computer Vision Theory and Applications (VISAPP) we make the following contributions.

Occluded object pose refinement using:

1. order-independent transparency employing analytical gradients, and
2. learned scene decomposition—an efficient alternative to optimization based on analytical gradients.

Related Work

Differentiable Rendering

Rasterization is the process of generating 2D images given the 3D scene description. Libraries like OpenGL [11], Vulkan [12], and DirectX [13] offer optimized rasterization implementations. Although the standard formulation of rendering 3D faces of object meshes into discrete pixels is not differentiable, probabilistic formulations like SoftRas [6], PyTorch3D [7], and DIB-R [8] allow for differentiable rendering. Most of the commonly used differentiable renderers are implemented using CUDA with programming interfaces to neural network libraries like TensorFlow [14] or PyTorch [15].

In this work, we present StilllebenDR, a fast differentiable renderer built as an extension to the openly available Stillleben library [5].

Render-and-Compare

Render-and-compare methods estimate 3D scene parameters—often iteratively—by minimizing the rendered and observed images. Krull et al. [16] trained a CNN to output a similarity score of how well the rendered image corresponds to the observed image and used the Metropolis algorithm to search for scene parameters with the highest similarity score. Moreno et al. [17] demonstrated the applicability of their render-and-compare framework for jointly estimating the shape, pose, lighting, and camera parameters of the scene. In our earlier work [9], we used a render-and-compare framework to refine the 6D pose for all objects in the scene simultaneously by employing an earlier version of StilllebenDR. Li et al. [18] trained a CNN to refine object pose using render-and-compare iteratively.

Deformable Registration

Given a canonical model of an object category, 3D deformable registration aims at deforming the canonical model to match an observed instance while maintaining the geometric structure of the category. Our approach for solving deformable registration is closely related to DeepCPD [19]. DeepCPD uses *coherent point drift* (CPD) to create a low-dimensional shape-space of the object category and employs a CNN to estimate 3D deformation from single-view RGB images. Our pipeline presented in Section "3D Deformable Registration" uses differentiable rendering to estimate a 3D deformation field instead.

Order-Independent Transparency

In contrast to the standard rendering in which along a camera ray only the faces closest to the camera are rendered, *order independent transparency* [20–22] renders all the faces and blends them using alpha compositing. In general, alpha compositing is not to generate natural-looking images. Nevertheless, it is widely used in scientific visualization, computer-aided design (CAD) software, and game engines. In this work, we show a proof-of-concept occluded object pose refinement using order-independent transparency in Section "Occluded Object Pose Refinement Using Order Independent Transparency" (see Figs. 1 and 2).

StilllebenDR

StilllebenDR is built as an extension to the Stillleben library [5], a synthetic data generation pipeline designed to generate data needed for deep learning models on the fly. It provides PyTorch interface and uses OpenGL for rendering and PhysX for physics simulation. OpenGL provides an optimized implementation of a standard rasterization pipeline.

StilllebenDR enables differentiation support with only a minimal overhead to the OpenGL rasterization pipeline. During the rasterization step, a face F constituting of vertices V with colors C is projected on a pixel I. The pixel color I is computed as

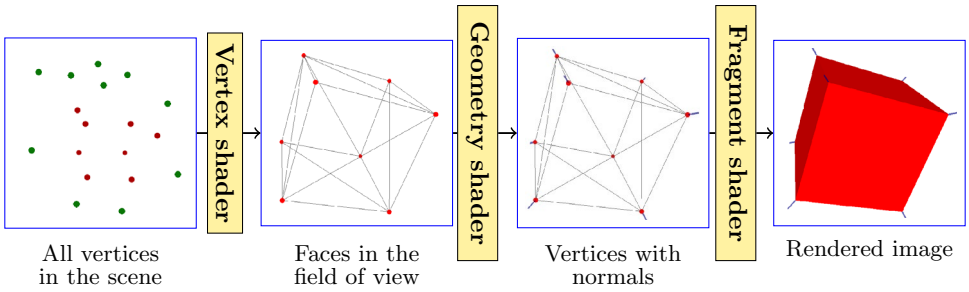

Fig. 1 OpenGL shader pipeline. A typical use-case of a shader pipeline involving the geometry shader. All the valid vertices in the world coordinates are transferred into the camera coordinates and *line primitives* are generated connecting the vertices for a face in the vertex shader. For each vertex, a *line primitive* corresponding to the normal direction is generated in the geometry shader. Faces are rasterized into discrete pixels in the fragment shader

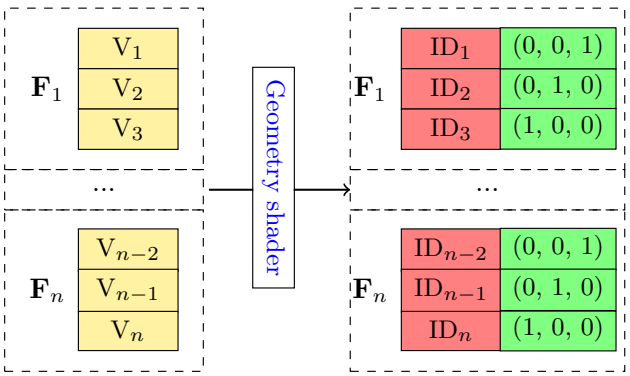

Fig. 2 Geometry shader configuration for barycentric coordinates generation. For each vertex, we generate a global vertex (in pinlk square) and a local vertex index that uniquely corresponds to a vertex for each face (in green square). Global vertex indices are immutable, whereas local vertex indices are mutable

$$I_{\text{rgb}} = \sum_i b_i C_i, \tag{1}$$

where b_i are the barycentric coordinates and $\sum_i b_i = 1$. We simplify the notation I_{rgb} and use I instead. OpenGL allows users to write shaders, specialized light-weight programs that are designed to run at specific stages of a graphics pipeline. Breaking down the graphics pipeline into a sequence of shaders enables parallelization. In addition to the standard RGB-D channels, we utilize the flexibility of the shaders to render additional channels containing information like barycentric coordinates and vertex indices, as shown in Fig. 3. The *vertex shader* and the *fragment shader* are the only two mandatory shaders in an OpenGL pipeline. In the *vertex shader*, vertices in the mesh coordinate are projected into the clip space, the space covered by the camera frustum. In the *fragment shader*, the color for rasterized pixels are computed. To generate barycentric coordinates and vertex indices as additional output channels, we make use of one of the less commonly used shaders, namely the *geometry shader*. The *geometry shader* is invoked at the end of the *vertex shader* and is used to generate additional primitives like vertices, lines, or faces. A common use-case for an OpenGL pipeline consisting of the *geometry shader* is

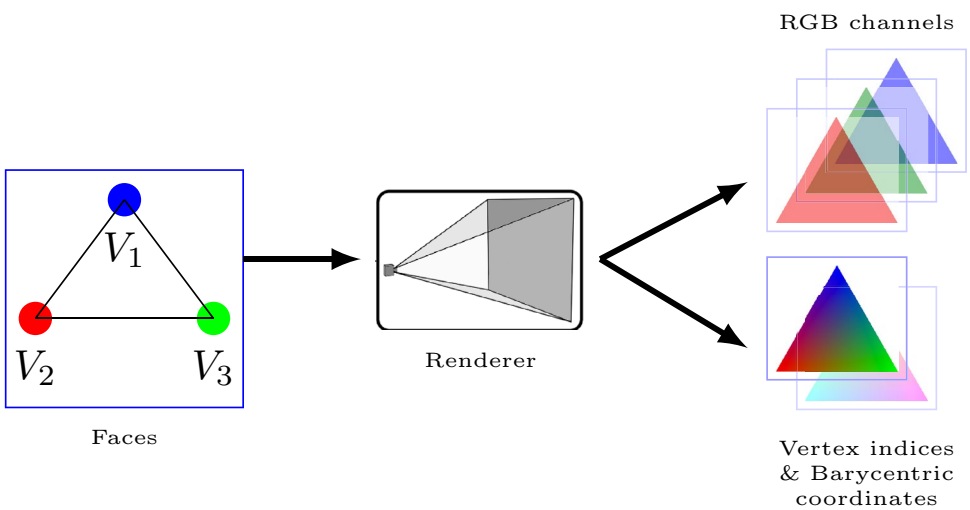

Fig. 3 Forward rendering. In addition to the RGB channels, we also render vertex indices and barycentric coordinates per pixel as separate channels and store them for backward computations. Source: Periyasamy et al. [10]

to visualize vertex normals (see Fig. 1). For each vertex, a line corresponding to the normal direction in the *geometry shader* is generated. We adapt the *geometry shader* to generate barycentric coordinates and vertex indices (see Fig. 2). For each vertex, a global vertex index and local vertex are generated. The global vertex index is marked as immutable and is rendered as constant values. For each face, one of *(0, 0, 1), (0, 1, 0), (1, 0, 0)* mutable vector values are assigned to each vertex uniquely. The vector values are interpolated in the later stages of the shader pipeline using the barycentric coordinates. The immutable global vertex indices and the interpolated local vertex indices reflect the corresponding barycentric coordinates and are rendered as additional output channels.

Given the loss L, computed pixel-wise between the rendered and the observed images, the gradient of the loss with respect to different scene parameters can be decomposed into the gradient of the loss with respect to the rendered image, and the gradient of the rendered image with respect to the scene parameters following the chain rule. For example, the gradient of the loss with respect to the vertex V_i is computed as

$$\frac{\partial L}{\partial V_i} = \frac{\partial I}{\partial V_i} \cdot \frac{\partial L}{\partial I}, \tag{2}$$

where $\frac{\partial I}{\partial V_i}$ is computed automatically by PyTorch autograd. Using the barycentric weights and the vertex indices stored during the forward rendering step, $\frac{\partial I}{\partial V_i}$ is computed as

$$\frac{\partial I}{\partial V_i} = C_i. \tag{3}$$

Similarly, we break down the gradient of the loss function with respect to object pose P as follows:

$$\frac{\partial L}{\partial P} = \frac{\partial I}{\partial P} \cdot \frac{\partial L}{\partial I}. \tag{4}$$

The backward pass of the rendering process is depicted in Fig. 4.

3D Deformable Registration

Deformable registration is crucial for robotic manipulation tasks where robots have to transfer the grasping knowledge from the canonical instance to other instances of the same object category (see Fig. 5).

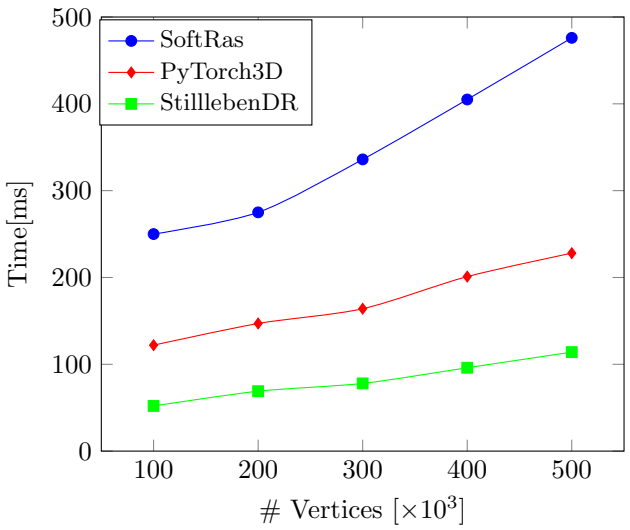

Fig. 5 Runtime comparison between SoftRas [6], PyTorch3D Ravi et al. [7], and StilllebenDR (ours). We report the average time taken by different differentiable rendering approaches to perform forward rendering (1024×1024 pixels) and backward gradient computations. Source: Periyasamy et al. [10]

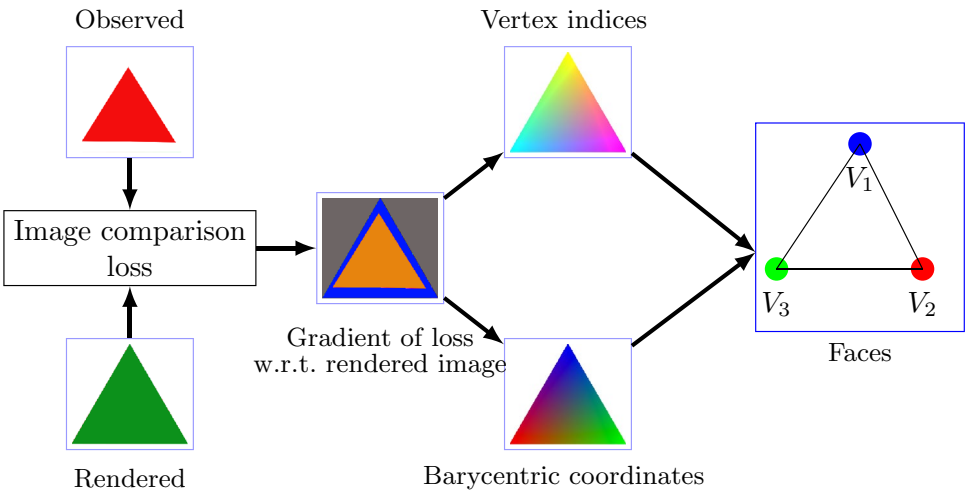

Fig. 4 Backward rendering. The gradient of the image comparison loss function is propagated to the vertices by differentiating through the renderer using the vertex indices and barycentric coordinates information stored during the forward rendering step. Source: Periyasamy et al. [10]

Latent Shape-Space

Given an object category with multiple instances and a canonical model, we use the *coherent point drift* (CPD) registration algorithm to learn a low-dimensional latent shape-space. The deformation τ_i of the canonical model $\mathbf{C}$ to an instance i is modeled as

$$\tau_i(\mathbf{C}_i, \mathbf{W}_i) = \mathbf{C} + \mathbf{G}(\mathbf{C}, \mathbf{C})\mathbf{W}_i. \tag{5}$$

where $\mathbf{W}$ is the deformation field and $\mathbf{G}$ is the Gaussian Kernel matrix defined element-wise as

$$\mathbf{G}(y_i, z_i) = g_{ij} = \exp\left(-\frac{1}{2\beta^2}||y_i - z_i||^2\right), \tag{6}$$

$\mathbf{W}$ is estimated in the M-step of the EM algorithm. Since the shape of $\mathbf{W}_i$ depends on the canonical model $\mathbf{C}$ and not the instance i, we reduce the dimension of $\mathbf{W}_i$ using *principle component analysis (PCA)*. We use latent shape-space of dimension five for all the object categories in our experiments. We refer the reader to Rodriguez et al. [19], and Rodriguez et al. [23] for a detailed explanation of the latent shape-space.

Deformable Registration Pipeline

In Fig. 6, we present a pipeline to deform the canonical model $\mathbf{C}$ to fit the observed image I_{obs} of a novel object instance. We generate the deformation field from the latent shape-space parameters $\mathcal{S}$ as described in Section "3D Deformable Registration" and render the deformed canonical model. We denote the rendered image as I_{rnd} using StilllebenDR. The rendered and the observed images are compared pixel-wise using the image comparison function described in Section "Deformable Registration Pipeline". We implement the image comparison function and mesh deformation generation from $\mathcal{S}$ using PyTorch. The PyTorch autograd engine enables gradient propagation through these steps automatically. The gradient of the mesh parameters with respect to I_{rnd} is provided by StilllebenDR. As shown in Fig. 4, the gradient can be computed only for the faces that are visible in I_{rnd}. However, instead of applying the gradients directly to the mesh parameters, we propagate the gradient to the latent shape-space $\mathcal{S}$ and generate the mesh deformation from $\mathcal{S}$. This step ensures that all vertices deform coherently and maintain the geometry of the object category. We perform the forward rendering and gradient-based $\mathcal{S}$ update iteratively until the image comparison error is negligible.

Image Comparison

Comparing images pixel-wise in the RGB color space is error-prone. Zhang et al. [24–26] demonstrated the effectiveness of the convolutional neural network feature space for image comparison. Inspired by the *learned perceptual image patch similarity metric* (LPIPS) [24], we construct the image comparison function as shown in Fig. 7. We extract the features from the last layer before the output layer of the U-Net model [27] for both rendered and observed images. We normalize the features between – 1 and 1 and aggregate the features along the channel dimension. Finally, we compute *mean-squared error* (MSE) of the aggregated features from the rendered and the observed images.

Experiments

We evaluate the proposed 3D deformable registration on the DeepCPD dataset [19]. The dataset consists of four object categories: bottles, cameras, drills, and sprays (shown in Fig. 8). For each object category, the dataset provides a canonical model and a varying number of instances. Two instances per category are used for testing and the rest are

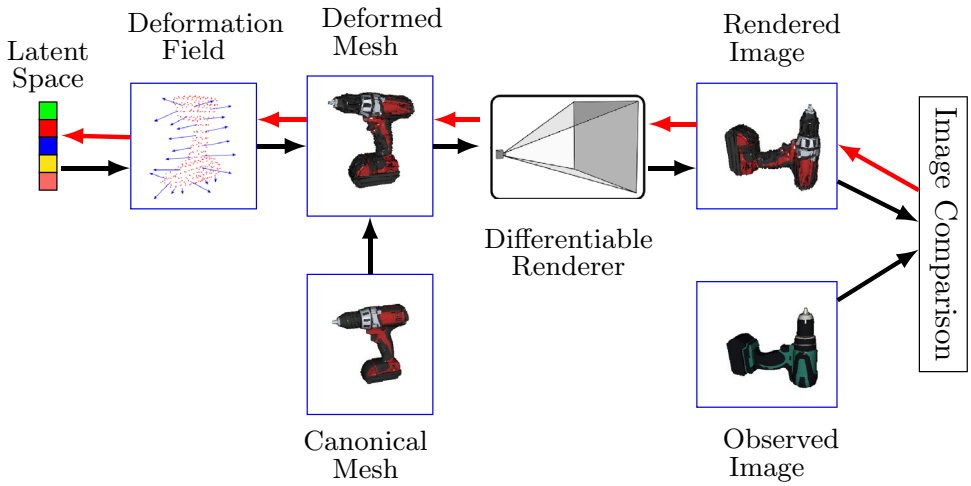

Fig. 6 Proposed deformable registration pipeline. Latent shape-space parameters are optimized to minimize the difference between the rendered image of the deformed mesh and the observed image. The image comparison loss is minimized using gradients obtained by differentiating through the rendering process. Black arrows indicate the forward rendering process and red arrows indicate the backward gradient flow. Source: Periyasamy et al. [10]

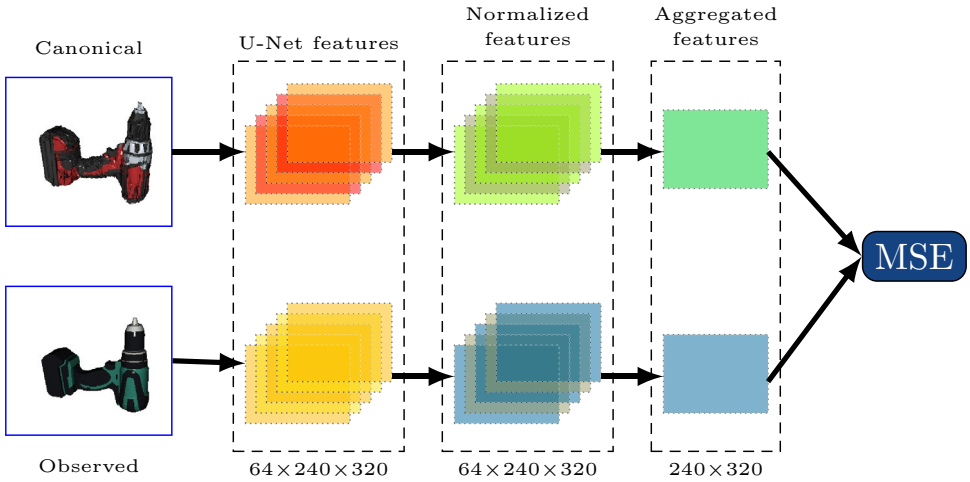

Fig. 7 Image comparison operation. We compare the rendered canonical and the observed image using U-Net features. We normalize the extracted U-Net features and normalize them between – 1 and 1 and aggregate the features along the channel dimension. Finally, we compute the mean-squared error pixel-wise between the aggregated features. Source: Periyasamy et al. [10]

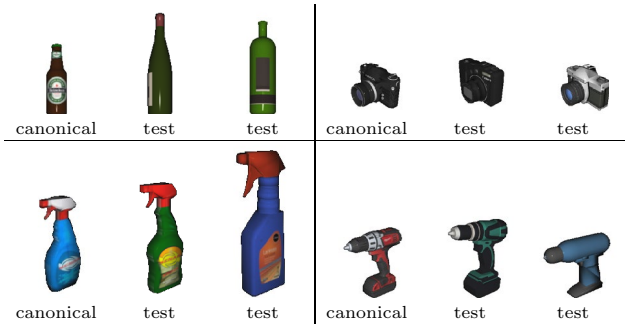

Fig. 8 DeepCPD dataset with canonical instances and exemplary test instances. Source: Periyasamy et al. [10]

used for training. All the instances of an object category are aligned to have a common coordinate frame. Since the proposed method does not involve training a separate model for modeling the deformation, we use the training instances only to train the U-Net model for semantic segmentation. We compare our method with CLS [28] and DeepCPD [19]. CLS works with point clouds and thus needs depth information, whereas DeepCPD is RGB only.

Deformation Registration with Known Poses

We employ the proposed end-to-end-differentiable pipeline for deformable registration iteratively. We use stochastic gradient descent (SGD) with a momentum of 0.9 and exponential weight decay of 0.95. The meshes provided by the DeepCPD dataset are not watertight. Since the goal of our method is to deform the canonical model to fit the observed instance while maintaining the geometry of the object category, our pipeline does not benefit from having vertex colors. Thus, we use uniform red color for all the vertices in the canonical mesh. The tiny invisible holes on the surface of the meshes develop into larger visible holes during the iterative deformable registration process. This results in not only the rendered image looking unrealistic but also the image comparison being harder. To alleviate this issue, we use the *ManifoldPlus* algorithm [29] to generate watertight meshes.

In Table 1, we report the average ℓ_2 distance between subsampled points of the canonical mesh and the test instances. Some qualitative visualizations are shown in Fig. 9. From the visualizations, one can observe that the rendered deformed mesh fits the observed mesh nicely. Our method not only works for objects with simple geometry like *bottles* but also for objects with complex geometry like *drills* and *sprays*. Quantitatively, our method performs only slightly worse than DeepCPD despite not employing any specialized learnable components to model the deformation.

Joint Deformable Registration and Pose Optimization

The accuracy of the deformable registration greatly depends on the quality of the pose estimation. Under the assumption that the exact pose of the observed instance is known, the competing methods CLS and DeepCPD perform slightly better than the proposed method. When the pose estimation is not accurate enough, the accuracy of the deformable registration degrades as well. In contrast to the methods in comparison, our end-to-end-differentiable pipeline can jointly optimize for 6D object pose along with deformable registration. To demonstrate this feature, we randomly sample offsets in the range of $[-0.05, 0.05]$ m for the x and y translation components and [-15°and 15°] for the rotation components. Although our method can optimize z translation along with other pose parameters, optimizing both z translation and vertex position jointly is an ill-posed problem. Thus, we include offsets only for x and y translation components. In our experiments, we observed that the pose parameters require

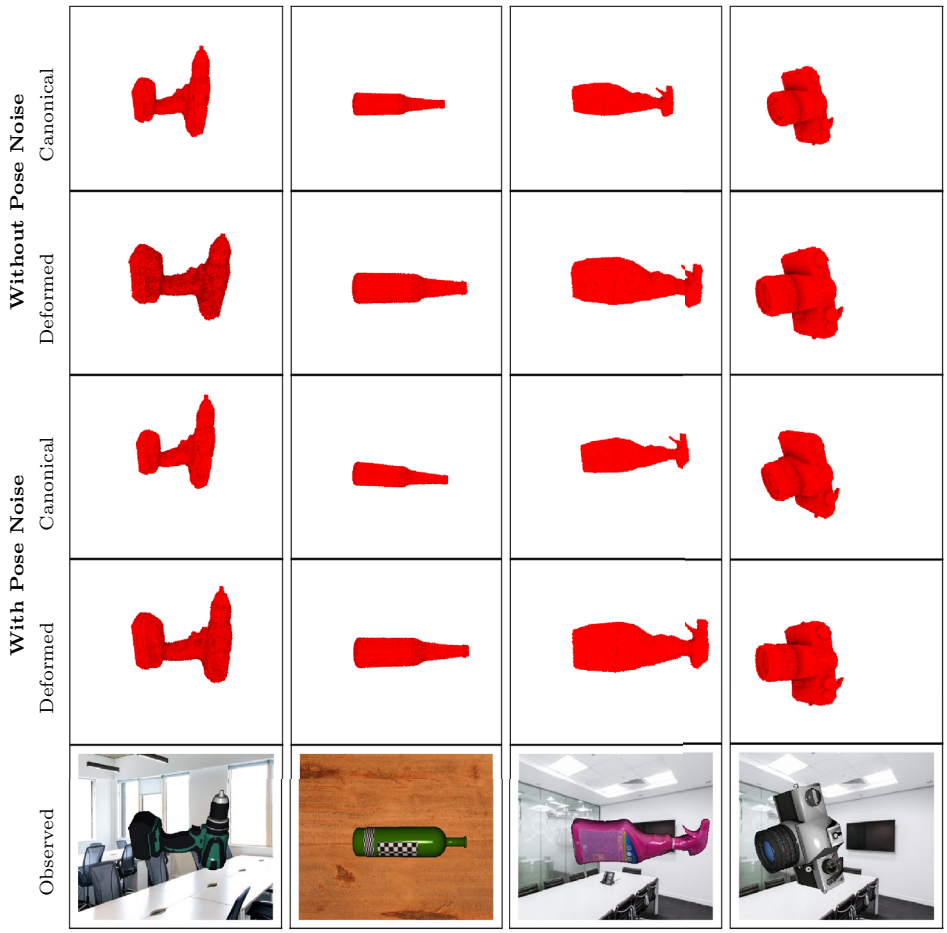

Fig. 9 Visualization of 3D deformation. The canonical mesh is deformed to fit the observed mesh iteratively using differentiable rendering. Source: Periyasamy et al. [10]

Table 1 Comparison of our approach with CLS [28] and DeepCPD [19]

Instance	Ground Truth	Known pose			With pose noise		
		CLS (3D)	DeepCPD (RGB)	Ours (RGB)	CLS (3D)	DeepCPD (RGB)	Ours (RGB)
Camera T1	34.61 (1.97)	**51.93** (10.45)	102.17 (47.89)	122.43 (22.86)	168.54 (357.8)	**105.26** (64.21)	126.65 (28.31)
Camera T2	16.45 (1.61)	**19.87** (4.59)	18.80 (5.11)	66.54 (29.73)	406.45 (492.03)	306.96 (127.89)	**89.65** (33.54)
Bottle T1	23.25 (2.34)	**25.92** (5.18)	45.21 (9.75)	52.63 (19.45)	297.79 (579.49)	227.90 (146.0)	**75.41** (34.23)
Bottle T2	90.42 (28.54)	**72.33** (11.35)	88.35 (18.39)	112.84 (25.78)	852.40 (1818)	289.36 (147.68)	**112.76** (31.76)
Spray T1	29.84 (1.42)	**30.78** (1.89)	47.87 (12.99)	77.74 (26.95)	1035 (406.69)	146.89 (117.57)	**89.59** (33.75)
Spray T2	111.94 (14.29)	**121.19** (19.16)	154.97 (82.34)	151.21 (79.76)	1488 (554.33)	255.69 (167.32)	**178.42** (88.14)
Drill T1	21.18 (0.949)	**28.86** (1.42)	52.71 (23.54)	71.54 (34.56)	232.35 (1325)	92.96 (58.23)	**84.34** (43.56)
Drill T2	63.95 (5.23)	**58.50** (21.51)	119.88 (107.43)	134.21 (89.16)	215.54 (565.48)	262.31 (228.40)	**157.27** (96.36)

The best values for RGB and 3D categories are presented in bold font

fewer updates to converge than the shape parameters. Therefore, we update the shape parameters at a higher frequency than the pose parameters, i.e., we update the pose parameters once per three shape parameter updates. Quantitative results of joint pose and shape optimization are presented in Table 1. The mean error only increases marginally when pose noise is injected, indicating that our method is less susceptible to pose initialization errors than competing methods.

Occluded Object Pose Refinement Using Order Independent Transparency

Render-and-compare frameworks enable joint pose estimation and pose refinement for a single or all objects in a scene [9, 17, 30]. However, occlusion hinders the effectiveness of the render-and-compare framework. To alleviate the issues with occlusion, we present scene peeling. Let us consider the scenario depicted in Fig. 10. In the top row, we show the observed image and the image rendered according to the current pose estimate. In the bottom row, we visualize the scene corresponding to the images in the top row from top view. In the observed scene, the *cup* is lying in front of the *wooden block*. But, due to an erroneous pose estimate, the *cup* is completely occluded by the *wooden block* in the image rendered according to the current pose estimate. In this scenario, a render-and-compare framework using differentiable rendering will not able to refine the pose of the *cup* since the gradients of the loss function exist only for the objects in the rendered image. To address this shortcoming, we introduce depth peeling-based aggregated images for object pose refinement. We render a scene multiple passes by discarding all the faces that are rendered in the previous pass. The first rendering pass is the standard rendering pass resulting in an RGB and a depth image (conventional Z-buffer). In the subsequent passes, we discard all the faces rendered already by ignoring the faces with smaller Z-buffer values than the Z-buffer rendered in the previous pass. In the top row of Fig. 11, we show the resulting RGB images of the first two rendering passes of the scene corresponding to the erroneous pose estimate in Fig. 10.

Peeling Images Aggregation

RGB images rendered during different peeling passes are blended into one aggregated image. Formally, given C_j^i, z_j^i, the color and the Z-buffer value at image from jth pass at pixel i respectively, and C_b, the background color, the aggregated color I at pixel i is defined as:

$$I_i = \sum_i w_j^i C_j^i + w_b^i C_b. \tag{7}$$

The weight w_j is defined as:

$$w_j^i = \frac{\exp(Z_j^i/\tau)}{\sum_k \exp(Z_k^i/\tau) + \exp(\epsilon/\tau)} \tag{8}$$

and $\sum_j w_j^i + w_b^i = 1$, where τ is the temperature parameter and ϵ corresponds to background color. Setting τ to a small value ($1e^{-2}$ in our experiments) allows blending of RGB images from multiple passes evenly, while setting $\tau = 0$ is

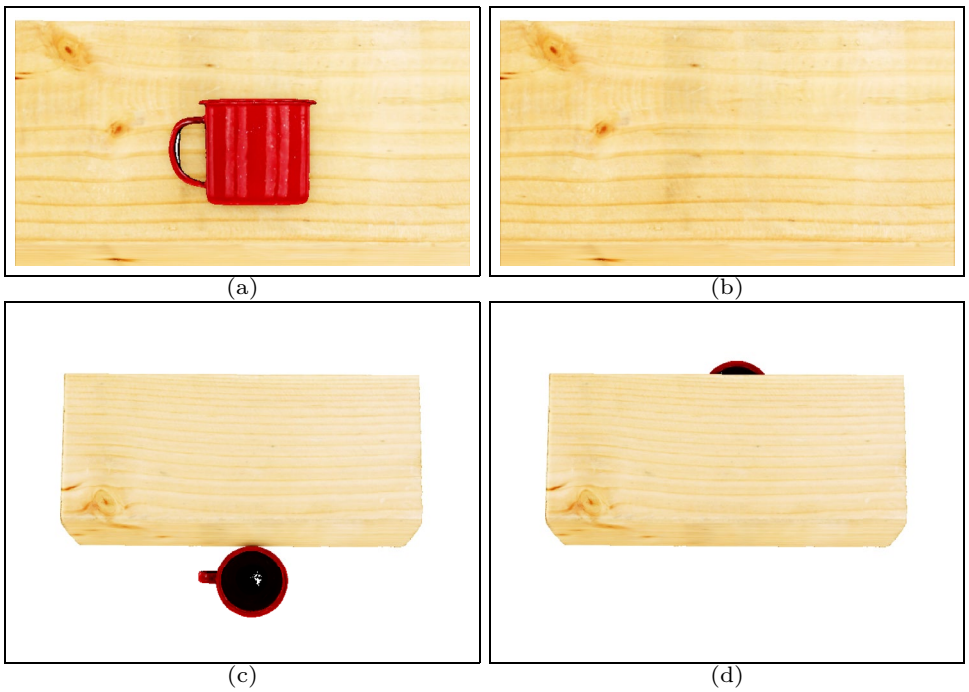

Fig. 10 **a** The observed image. **b** the image rendered according to the current pose estimate. Bottom row: Top view of the scene corresponding to (**a**, **b**) respectively. The *mug* is completely occluded by the *wooden block* due to erroneous pose prediction in the case of the scene corresponding to current pose estimate

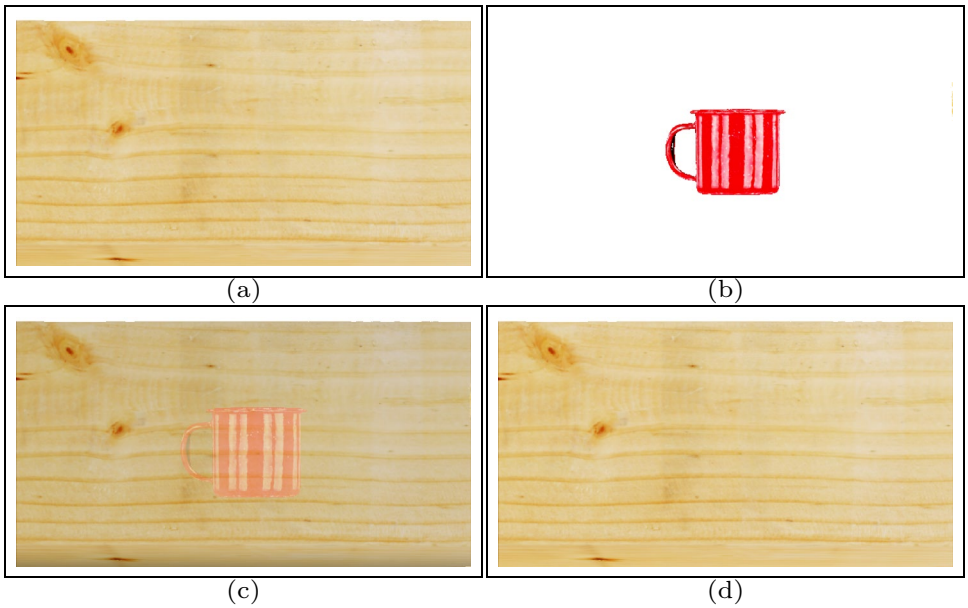

Fig. 11 **a**, **b** RGB images from the first two peeling passes. Bottom row: Aggregated images with different temperature parameter τ. **c** Aggregation with $\tau = 1e^{-2}$. RGB pixels from both the peeling passes are aggregated with nearly equal weighting. **d** Aggregation with $\boldsymbol{\tau} = 0$. Pixels from only the first peeling pass (**a**) are visible. The *mug* is completely occluded by the *wooden block*

equivalent to the standard rendering. The aggregated images with different values of τ are shown in the bottom row of Fig. 11. Figure 11a bottom is generated with τ=$1e^{-2}$. RGB images from the two passes are aggregated with equal weighting. The resulting image is smooth blending of the RGB from the two peeling passes. In contrast, Fig. 11a is the result of an aggressive aggregation where the aggregated image is equivalent to the RGB from the first rendering pass.

Occluded Object Pose Refinement Using Analytical Gradients

Using aggregated images with τ=$1e^{-2}$ for image comparison allows gradients for occluded objects. We perform object pose refinement iteratively to minimize the pixel-wise loss between the aggregated image and the observed image shown in Fig. 10. We decay the temperature parameter by a factor of 0.95 after each iteration. In Fig. 12, we show the aggregated image, and the standard RGB image, the top view of the corresponding scene during the iteration process. The position of *mug* that is completely occluded behind the *wooden block* in the initial iteration is refined to match the observed scene. During the refinement process, we can observe the position of the object *mug* being gradually pulled forward.

Learning Scene Decomposition

While performing occluded object pose refinement using analytical gradients works for simple scenes, applying such an approach for complex scenes is non-trivial. Furthermore, it needs multiple iterative refinement steps to converge. An alternate approach to analytical gradients is to learn scene decomposition. To this end, we propose the *Scene Decomposition Model* depicted in Fig. 13. It takes the observed RGB image, the rendered RGB-D images according to the current pose estimate, and the RGB-D images of the peeling passes as input. RGB image features are extracted using a pre-trained ResNet model. We use the features after the third ResNet block, i.e., from an input RGB image of height H and width W, the features extracted are of dimension $128 \times \frac{H}{8} \times \frac{W}{8}$. The depth images are down-scaled and concatenated with the RGB features. The resulting features are of dimension $515 \times \frac{H}{8} \times \frac{W}{8}$. From these features, the Scene Decomposition Model generates a dense pixel-wise z-estimate for objects in each peeling layer employing ResNet-style convolutional blocks without pooling layers—maintaining the image dimension of $\frac{H}{8} \times \frac{W}{8}$.

Similar to the analytical optimization described in Section "Occluded Object Pose Refinement Using Analytical Gradients", we demonstrate the learned Scene Aggregation Model by optimizing for the z component of the object translation. The model generates dense pixel-wise refined z-estimate for objects in two peeling layers in one forward pass. It is trained using supervised learning to minimize pixel-wise *mean squared error* (MSE) using a custom-made dataset. The dataset consists of two objects, namely *cup*, and *wooden block* placed at random positions in the scene such that one object always occludes the other. We then add noise to the z-component of the objects' translation to simulate erroneous z-estimate. We sample uniform z-estimate noise in the range of [−0.5, 0.5] cm. In addition

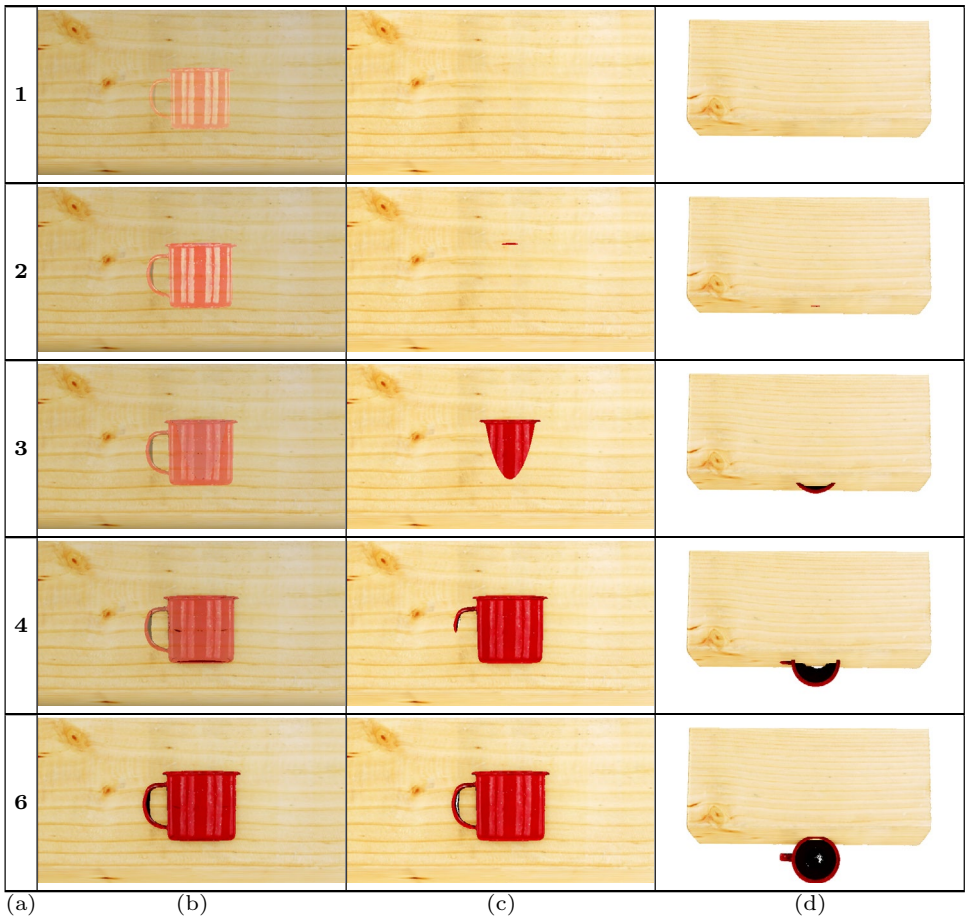

Fig. 12 Occluded object pose refinement using scene peeling. **a** iteration step, **b** aggregated image **c** standard rendered image **d** top view of the corresponding scene. The *mug* is completely occluded by the *wooden block* in the first iteration, whereas it is pulled forward gradually during the refinement process until it matches the observed image

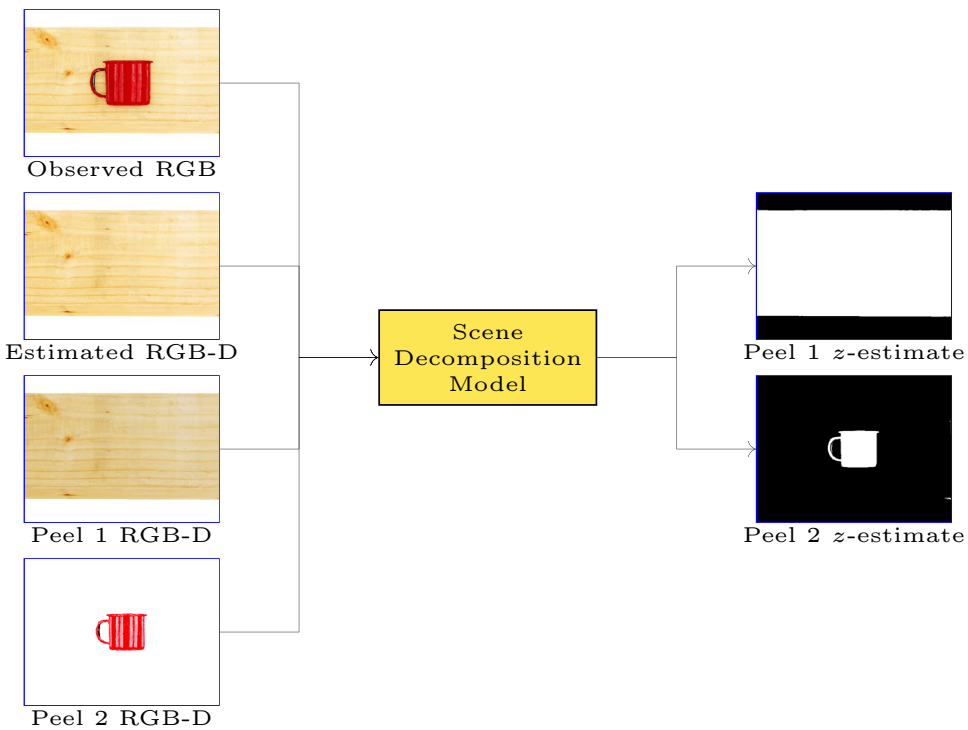

Fig. 13 Scene Decomposition Model. Given the Observed RGB image, the rendered RGB-D image according to the current pose estimate, RGB-D images of the peeling passes, the Scene Decomposition Model generates a dense pixel-wise z-estimate for objects in the peeling layer

to rendering the RGB-D images of objects in the original pose (observed image) and in the erroneous pose (rendered image), we render two RGB-D image peels of objects in the erroneous pose. We use only RGB images of the objects in the original pose for training the model. The model learns the 3D geometry of the scene from the rendered and the peel RGB-D images. We use the Stillleben library to create the dataset on the fly to train the model and train the model for 5000 iterations with a batch size of 32. We create a test set of 1000 scenes for quantitative comparison. To establish a strong competing approach, we create a variant of the Scene Decomposition Model that takes RGB-D observed images instead RGB observed images as input (516 input feature maps instead of 515 feature maps). In Table 2, we present the quantitative comparison of the RGB and RGB-D Scene Decomposition Models. The RGB-D model performs only slightly better than the RGB model despite having access to depth information. This demonstrates the ability of the Scene Decomposition Model to learn the scene 3D geometry only from the RGB images. Moreover, both variants perform better for the *wooden block* object than for the *cup* object. This can be attributed to the simple geometric shape of *wooden block* compared to *cup* consisting of a non-convex surface. Qualitative visualizations of the RGB model are shown in Fig. 14.

Table 2 Quantitative comparison of the RGB and the RGB-D models

Model	$\lvert\Delta z\rvert \leq 0.25$ cm		$0.25\,\text{cm} > \lvert\Delta z\rvert \leq 0.5$ cm	
	Cup	*Wooden block*	*Cup*	*Wooden block*
RGB	1.436	1.044	1.437	1.048
RGB-D	1.333	0.973	1.346	0.969

$\lvert\Delta z\rvert$ represents the magnitude of the z-noise sampled

Mean error values are expressed in $\times 10^{-3}$ m

Conclusion

We presented StilllebenDR and the end-to-end differentiable pipeline for joint 3D deformable registration and pose refinement from single-view RGB images using StilllebenDR introduced in our previous work (Periyasamy et al. [10]). Designed as a light-weight extension to the Stillleben library that uses OpenGL graphics pipeline for efficient rendering enables StilllebenDR to be fast and scalable. Furthermore, we introduced occluded object pose refinement based on order-independent transparency using StilllebenDR employing analytical gradients. Aggregating RGB images from multiple depth peeling passes facilitates gradient flow to objects that are completely occluded. Moreover, we introduced learned scene decomposition, an efficient alternative to iterative analytical gradient-based optimization.

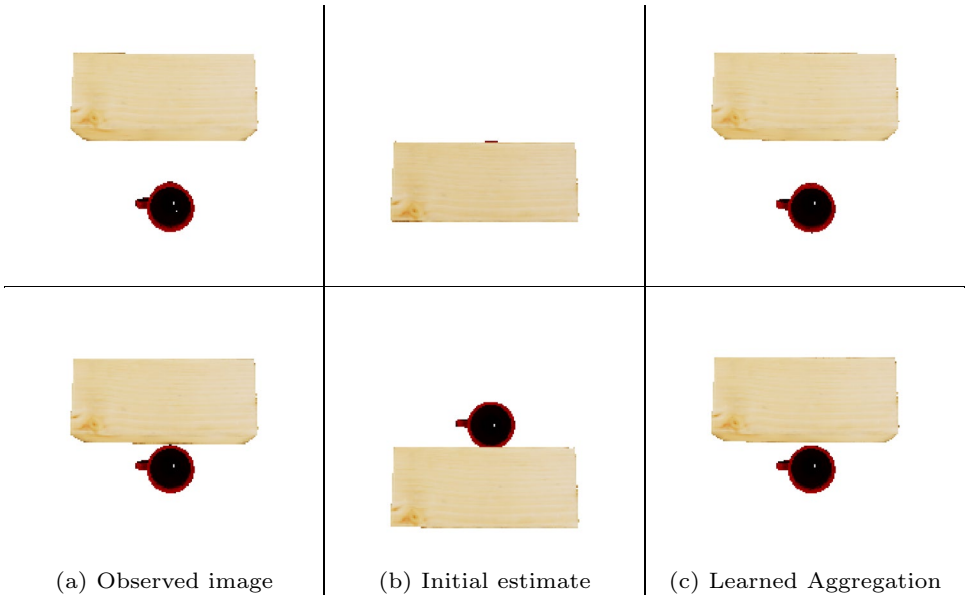

Fig. 14 Exemplar top-view visualizations of learned scene decomposition. The Scene Decomposition Model learns to generate the correct z-estimate for objects in two peeling layers in one forward pass

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability The datasets generated during and/or analysed during the current study are available from the corresponding author upon reasonable request.

Declarations

Conflict of interest Both authors declare that they do not have any conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44.
2. Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press; 2016.
3. Cohen N, Shashua A. Inductive bias of deep convolutional networks through pooling geometry. In: International Conference on learning representations (ICLR), 2017. https://openreview.net/forum?id=BkVsEMYel. Accessed 01 Jan 2023.
4. Xuhong LI, Grandvalet Y, Davoine F. Explicit inductive bias for transfer learning with convolutional networks. In: International Conference on machine learning (ICML), 2018, pp. 80:2825–2834
5. Schwarz M, Behnke S. Stillleben: realistic scene synthesis for deep learning in robotics. In: IEEE International Conference on robotics and automation (ICRA), 2020; p. 10502-10508
6. Liu S, Li T, Chen W, Li H. Soft Rasterizer: a differentiable renderer for image-based 3D reasoning. In: IEEE International Conference on computer vision (ICCV), 2019; pp. 7708–7717 .
7. Ravi N, Reizenstein J, Novotny D, Gordon T, Lo W-Y, Johnson J, Gkioxari G. Accelerating 3D deep learning with PyTorch3D. In: SIGGRAPH Asia 2020 Courses, 2020.
8. Chen W, Ling H, Gao J, Smith E, Lehtinen J, Jacobson A, Fidler S. Learning to predict 3D objects with an interpolation-based differentiable renderer. In: Advances in neural information processing systems, Curran Associates, Inc. 2019; p. 32 Editors: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R.
9. Periyasamy AS, Schwarz M, Behnke S. Refining 6D object pose predictions using abstract render-and-compare. In: IEEE-RAS 19th International Conference on humanoid robots (Humanoids), 2019; pp. 739–46.
10. Periyasamy AS, Schwarz M, Behnke S. Iterative 3D deformable registration from single-view rgb images using differentiable rendering. In: 17th International Conference on computer vision theory and applications (VISAPP), 2022.
11. Segal M, Akeley K. The OpenGL graphics system: a specification (version 1.1). 1999. https://registry.khronos.org/OpenGL/specs/gl/glspec11.pdf. Accessed 20 Dec 2022.
12. Vulkan K. A specification (version 1.0). 2018. https://www.khronos.org/news/press/khronos-releases-vulkan-1-0-specification. Accessed 20 Dec 2022.
13. Microsoft. Directx-specs. 2019. https://microsoft.github.io/DirectX-Specs/. Accessed 20 Dec 2022.
14. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. TensorFlow: a system for Large-Scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016; pp. 265–83 .
15. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison An, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. PyTorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, Curran Associates, Inc., 2019; pp. 32:8024–35. Editors: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R.
16. Krull A, Brachmann E, Michel F, Ying Yang M, Gumhold S, Rother C. Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In: IEEE International Conference on computer vision (ICCV), 2015; pp. 954–62.
17. Moreno P, Williams Christopher KI , Nash C, Kohli P. Overcoming occlusion with inverse graphics. In: European Conference on Computer Vision (ECCV) Workshops, 2016; 3:170-185
18. Li Y, Wang G, Ji X, Xiang Y, Fox D. DeepIM: Deep iterative matching for 6D pose estimation. In: European Conference on computer vision (ECCV), 2018; pp. 683–98.
19. Rodriguez D, Huber F, Behnke S. Category-level 3D non-rigid registration from single-view RGB images. In: IEEE/RSJ International Conference on intelligent robots and systems (IROS), 2020; pp. 10617-10624
20. Bavoil L, Myers K. Order independent transparency with dual depth peeling. 2008. https://developer.download.nvidia.com/SDK/10/opengl/src/dual_depth_peeling/doc/DualDepthPeeling.pdf. Accessed 20 May 2022.
21. Everitt C. Interactive order independent transparency. 2008. https://developer.download.nvidia.com/assets/gamedev/docs/OrderIndependentTransparency.pdf. Accessed 20 May 2022.
22. Maule M, Comba JLD, Torchelsen RP, Bastos R. A survey of raster-based transparency techniques. Comput Graph. 2011;35(6):1023–34.
23. Rodriguez D, Cogswell C, Koo S, Behnke S. Transferring grasping skills to novel instances by latent space non-rigid registration. In: IEEE International Conference on robotics and automation (ICRA), 2018; pp. 1-8
24. Zhang R, Isola P, Efros AA, Shechtman E, Wang O. The unreasonable effectiveness of deep features as a perceptual metric. In: IEEE Conference on computer vision and pattern recognition (CVPR), 2018; pp. 586–95.
25. Zagoruyko S, Komodakis N. Learning to compare image patches via convolutional neural networks. In: IEEE Conference on computer vision and pattern recognition (CVPR), 2015; pp. 4353–61.
26. Appalaraju S, Chaoji V. Image similarity using deep CNN and curriculum learning. 2017. arXiv:abs/1709.08761.

27. Ronneberger O, Fischer P, Brox T. U-Net: convolutional networks for biomedical image segmentation. In: International Conference on medical image computing and computer-assisted intervention (MICCAI), 2015; pp. 234–41.
28. Myronenko A, Song X. Point set registration: coherent point drift. IEEE Trans Pattern Anal Mach Intell (TPAMI). 2010;32(12):2262–75.
29. Huang J, Zhou Y, Guibas L. ManifoldPlus: a robust and scalable watertight manifold surface generation method for triangle soups. 2020. arXiv:abs/2005.11621.
30. Kundu A, Li Y, Rehg JM. 3D-RCNN: Instance-level 3D object reconstruction via render-and-compare. In: IEEE Conference on computer vision and pattern recognition (CVPR), 2018; pp. 3559–68 .

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com