SAIL-VOS 3D: A Synthetic Dataset and Baselines for Object Detection and 3D Mesh Reconstruction from Video Data

Yuan-Ting Hu, Jiahong Wang, Raymond A. Yeh, Alexander G. Schwing
University of Illinois at Urbana-Champaign
{ythu2, jiahong4, yeh17, aschwing}@illinois.edu

Figure 1. Temporal information in videos provides a lot of cues for 3D mesh reconstruction. For instance, as the main character in this scene turns we can accurately reconstruct both the front and back. Unfortunately, existing image-based datasets don’t permit to develop such methods. To alleviate this we introduce the synthetic SAIL-VOS 3D dataset and develop first baselines for this task.

Abstract

Extracting detailed 3D information of objects from video data is an important goal for holistic scene understanding. While recent methods have shown impressive results when reconstructing meshes of objects from a single image, results often remain ambiguous as part of the object is unobserved. Moreover, existing image-based datasets for mesh reconstruction don’t permit to study models which integrate temporal information. To alleviate both concerns we present SAIL-VOS 3D: a synthetic video dataset with frame-by-frame mesh annotations which extends SAIL-VOS. We also develop first baselines for reconstruction of 3D meshes from video data via temporal models. We demonstrate efficacy of the proposed baseline on SAIL-VOS 3D and Pix3D, showing that temporal information improves reconstruction quality. Resources and additional information are available at http://sailvos.web.illinois.edu.

1. Introduction

Understanding the 3D shape of an observed object over time is an important goal in computer vision. Very early work towards this goal [42, 43, 44] focused on recovering lines and primitives like triangles, squares and circles in images. Due to many seminal contributions, the field has significantly advanced since those early days. Given a single image, exciting recent work [21, 31, 37, 76] detects objects and infers their detailed 3D shape. Notably, the shapes are significantly more complex than the early primitives.

To uncover the 3D shape, recently, single view 3D shape reconstruction [10, 16, 41, 48, 55, 56, 77, 81] has garnered much attention. The developed data-driven and learning-based approaches achieve realistic reconstructions by inferring the 3D geometry and structure of objects. All those methods have in common the use of a single input image. However, moving objects and temporal information are not considered. Intuitively, as illustrated in Fig. 1, we expect temporal information to aid 3D shape reconstruction. How can methods benefit from complementary information available in multiple views?

Classical multi-view 3D reconstruction [6, 18, 65] exploits the geometric properties exposed in multiple views. For instance, structure from motion algorithms, e.g., [65], infer the 3D shape using multi-view geometry [24]. However note that the assumption of a static scene in these classical methods is often violated in practice. Moreover, clas-


tactical 3D reconstruction focuses on finding the 3D shape of observed object parts. Nonetheless it is important for methods to predict occluded parts of objects or to infer object extent beyond the observed view [53].

To achieve a more realistic and more detailed reconstruction we aim to leverage temporal information. For this we think ground-truth 3D mesh annotations for video-data are particularly helpful. Unfortunately, this form of annotated data is not available in existing datasets. This is not surprising as acquiring this form of data is very time-consuming and hence expensive. In addition, annotations of this form are often ambiguous due to occlusions. However, following recent work on amodal segmentation [27], we think photo-realistic renderings of assets in computer games could come to the rescue. To study this solution, in this paper, we collect SAIL-VOS 3D, a synthetic video dataset with frame-by-frame 3D mesh annotations for objects, ground-truth depth for scenes and 2D annotations in the form of bounding boxes, instance-level semantic segmentation and amodal segmentation. We think this data serves as a good test bed for 3D perception algorithms.

In order to deal with dynamics of 3D shapes, we propose to take the temporal information into account by developing a baseline model. Specifically, instead of refining a spherical shape we introduce a reference mesh which carries information from earlier frames.

Based on the collected SAIL-VOS 3D dataset and existing image datasets like Pix3D [71], we study the proposed method as well as recent techniques for 3D reconstruction of meshes from images. We illustrate the efficacy of the developed model and the use of temporal information on the collected SAIL-VOS 3D dataset. Using Pix3D [71] we demonstrate generality.

2. Related Work

Single-View Object Shape Reconstruction: Recently, a number of methods [10, 15, 16, 22, 31, 37, 41, 47, 48, 49, 54, 55, 56, 76, 77, 80, 81, 87] have been proposed for shape reconstruction from a single image. These approaches vary by their shape representation, e.g., meshes [77], occupancy grids [82], octrees [73], implicit fields [11, 47, 54] and point clouds [22]. Many of these methods assume that there is only a single shape to be reconstructed. Consequently, these methods usually use a curated datasets like ShapeNet [7], which contains only a single object at a time.

3D shape reconstruction methods that deal with multiple objects [21, 31, 37, 76] usually either involve a detection network [21, 31, 37] like Faster-RCNN [59], which has been very successful on 2D instance-level multi-object detection, or assume instance-level bounding boxes are given [76]. After obtaining the bounding boxes, the 3D shape of each instance is inferred within each bounding box. To generally deal with multiple objects, we also combine detection with shape reconstruction. However, instead of employing a single image, the proposed approach differs in our use of video data for reconstruction.

Multiple-View Object Shape Reconstruction: 3D understanding using multiple images has been studied and used for depth estimation [34, 74], scene reconstruction [51], semantic segmentation for 3D scenes [38], 3D object detection [9, 50, 68], and 3D object reconstruction [12, 85]. We focus on 3D object shape reconstruction literature here as it aligns with our goal. Traditional multi-view shape reconstruction approaches including structure from motion (SfM) [65] and simultaneous localization and mapping (SLAM) [6, 18] estimate 3D shapes from multiple 2D images using multi-view geometry [24]. However, these classical algorithms have well known constraints and are challenged by textureless objects as well as moving entities. Also, classical multi-view geometry does not infer the shape of unseen parts of an object.

Due to the success of deep learning, learning-based approaches to multi-view object 3D reconstruction [5, 12, 17, 26, 30, 33, 52, 72, 75, 79, 85] have received a lot of attention lately. Again, the majority of the aforementioned methods deal with only a single object. Dealing with instance-level multi-view reconstruction remains challenging as it requires not only object detection, but also object tracking as well as 3D shape estimation. Closer to the goal of our approach, SLAM-based methods have been proposed for object-level multi-view reconstruction [13, 26, 64, 70]. However, these methods are challenged by inaccurate estimation of camera poses and moving objects. In contrast, our approach aims at 3D understanding for dynamic objects.

3D Datasets: A number of 3D datasets have been introduced in recent years [2, 3, 4, 7, 8, 14, 20, 28, 36, 39, 46, 48, 62, 66, 69, 71, 82, 83, 84], all of which are commonly used for training, testing and evaluating 3D reconstruction
algorithms. We summarize the properties of several common 3D datasets in Tab. 1. Datasets annotated on real world images [8, 39, 71, 83, 84] or videos [14, 20] are often small compared to synthetic datasets such as SUNCG [69]. Moreover, KITTI [20] and OASIS [8] contain only modal annotations, i.e., only visible regions are annotated. In order to create large scale datasets without labor-intensive annotation processes, synthetic 3D datasets [7, 69, 82] are useful.

Among those, ShapeNet [7] is one of the commonly used synthetic benchmarks for 3D reconstruction. However, the rendered images usually contain uniform backgrounds, making the distribution of rendered images very different from real world images. SUNCG [69] is one of the largest among the synthetic 3D datasets. However, the dataset contains static indoor scenes only, and doesn’t cover outdoor scenarios or dynamic scenes. In fact, except for KITTI [20], which contains dynamic scenes, i.e., moving objects with complex motion, to the best of our knowledge, there is no large scale 3D video dataset with instance-level 3D annotation that captures dynamic scenes with diverse scenarios. Synthetic Datasets: Simulated worlds have been used for computer vision tasks such as optical flow [45], visual odometry [23, 60] and semantic segmentation [63] and human shape modeling [58, 86]. The game engine GTA-V has been used to collect large-scale data, e.g., for semantic segmentation [35, 60, 61], object detection [32], amodal segmentation [27], optical flow [35, 60] and human pose estimation [19].

3. SAIL-VOS 3D Dataset

For more accurate 3D video object shape prediction, we propose the SAIL-VOS 3D dataset: We collect an object-centric 3D video dataset from the photorealistic game engine GTA-V. The game engine simulates a three-dimensional world by modeling a real-life city. The resulting diverse object categories and the obtainable 3D structures make the game engine a suitable choice for collecting a large-scale 3D perception dataset.

As shown in Fig. 2 we collect the following data: video frames, camera matrices, depth data, instance level segmentation, instance level amodal segmentation and the corresponding 3D object shapes. Each instance is assigned a consistent id across frames.

3.1. Dataset Statistics

The presented SAIL-VOS 3D dataset contains 484 videos with a total of 237,611 frames at a resolution of 1280×800. Note that each video may contain shot transitions. There are a total of 6,807 clips in the dataset and on average 34.6 frames per clip. The dataset is annotated with 3,460,213 object instances coming from 3,576 models, which we assign to 178 categories. To assign the categories, we use the SAIL-VOS labels [27]. There are 48 classes that overlap with the categories in the MSCOCO dataset [40]. Fig. 2 (right) shows the distribution of instances among the categories and their respective occlusion rate. Detailed statistics for clip length can be found in the Appendix.

We compare the proposed SAIL-VOS 3D dataset with other 3D datasets in Tab. 1. SUNCG [69], ScanNet [14], OASIS [8] and SAIL-VOS 3D are large scale datasets con-
taining more than 100K images. In OASIS only visible parts of objects are annotated. In SUNCG and ScanNet static indoor scenes are used for collecting the datasets. In contrast, SAIL-VOS 3D contains moving objects, different scenarios (both indoor and outdoor with different lighting conditions and weather), and amodal 3D annotations. Compared to the commonly used ShapeNet [213], SAIL-VOS 3D contains images rendered with cluttered backgrounds and more object categories (178 vs. 55).

We show several annotated frames sampled from SAIL-VOS 3D in Fig. 3 and Fig. 4. Note, the dataset contains meshes despite occlusions, i.e., the entire shape is annotated even if part of it is out-of-view as shown in Fig. 3.

3.2. Dataset Collection Methodology

We collect the cutscenes in GTA-V following the work of Hu et al. [27]. Similarly, we alter the weather condition, the time of day and clothing of the characters to increase the diversity of the synthetic environment. However, different from their method, we are interested in capturing 3D information in the form of meshes, camera data, etc. This requires to extend the approach by Hu et al. [27], and we describe the details for obtaining the data in the following.

**Meshes:** The shapes of objects in GTA-V are all represented as meshes. In order to retrieve the meshes, we use GameHook [35] to hook into the Direct3D 11 graphics pipeline [1] which GTA-V employs. We first briefly highlight the key components of the Direct3D 11 graphics pipeline before we delve into details.

The Direct3D 11 graphics pipeline [1] consists of several stages which are traversed when rendering an image, e.g., the vertex shader stage, where vertices are processed by applying per-vertex operations such as coordinate transformations, the stream-output stage, where vertex data can be streamed to memory buffers, the rasterizer stage, where shapes or primitives are converted into a raster image (pixels), and the pixel shader stage, where shading techniques are applied to compute the color. The intermediate output of stages before the stream-output stage, e.g., the vertex shader stage, can not be directly copied to staging resources, i.e., resources which can be used to transfer data from GPU to CPU. This makes access to vertex data, and hence our approach, more complicated than earlier work: design of the graphics pipeline only permits access to vertex data via the stream-output stage.

In GTA-V, the vertex shader stage processes object vertices by applying the camera matrices as well as the transformation matrices to animate articulated models. Our goal is to retrieve the output of the vertex shader stage which contains the meshes. Since the output of the vertex shader is not directly accessible, we need to retrieve it via the stream-output stage. For this we enable the stream-output stage by using a hook to alter the behavior of the Direct3D function `CreateVertexShader`. We ask it to not only create a vertex shader (it’s original behavior) but to additionally execute `CreateGeometryShaderWithStreamOutput`. This permits to append our own geometry shader whenever the vertex shader is created. Our geometry shader takes as input the output of the vertex shader. It doesn’t change the vertex data and directly writes the vertex data via the stream-output stage to a stream-output buffer. As this buffer is not CPU accessible, we subsequently copy the data to a staging resource so that it can be read by the CPU. We then store the vertex data, i.e., the meshes, using the OBJ file format.

**Camera & Viewport:** To obtain camera matrices we collect camera data from the constant buffers in the rendering pipeline. In each draw call, we store the world matrix, which transforms model coordinates to world coordinates, the view matrix, which transforms the world coordinates to camera coordinates, the projection matrix, which transforms the camera coordinates into image coordinates, as well as the viewport, which defines the viewing region in the rendering-device-specific coordinate.

**Depth & Others:** Along with the 3D shapes and camera matrices, we collect depth by copying the depth buffer from the rendering pipeline. We label shot transitions for each video. For each object in the dataset, we also collect additional information like the 2D segmentation, 2D
4. Video2Mesh for 3D reconstruction

Given a video sequence \((I_1, \ldots, I_T)\) of \(T\) frames, we are interested in inferring the set of meshes \(\mathcal{M}_t\) for the existing objects in each frame \(I_t\). A pictorial overview of our approach, Video2Mesh, is shown in Fig. 5.

To infer the meshes, we first detect objects in each frame \(I_t\). We let the unordered set \(O_t = \{o_i^t\}\) refer to the detected objects, \(i.e., o_i^t\) is the \(i\)th detected object. Note that the set is unordered, \(i.e., o_i^t\) and \(o_j^{t-1}\) may not refer to the same object in frames \(I_t\) and \(I_{t-1}\). To establish object correspondence across time, we solve an assignment problem. This permits to use temporal information during shape prediction. Formally, for every frame \(I_t\), we let \(O_{t-1}\) the previous frame into the ordered tuple \(O_{t-1} = (\tilde{o}_n^{t-1}, \ldots, \tilde{o}_n^{t-1})\), where \(n_t = |O_t|\). Specifically, for object \(o_i^t\) from frame \(I_t\) we let \(\tilde{o}_n^{t-1}\) denote the corresponding object from frame \(I_{t-1}\). If we cannot find a match in the previous frame, a special token is assigned to \(\tilde{o}_n^{t-1}\).

With the objects tracked, we can use the prediction from the previous frame as a reference to predict meshes in the current frame. We detail detection and tracking in Sec. 4.1 and explain the temporal mesh prediction in Sec. 4.2.

4.1. Detection and Tracking

To obtain the set of objects \(O_t\) for frame \(I_t\), we first use Faster-RCNN [59] to detect object bounding boxes and their class. Hence, each object \(o_i^t = (b_i^t, c_i^t)\) consists of a bounding box and class estimate, \(i.e., b_i^t\) and \(c_i^t\). As previously mentioned, there are no correspondences for the detected objects between frames. To obtain those we solve an assignment problem.

Formally, given the detected objects from two consecutive frames, the assignment problem is formulated as follows:

\[
\max_{A_t} \sum_{i,j} A_{t,ij} \left( \text{IoU}(b_i^t, b_j^{t-1}) - L(c_i^t, c_j^{t-1}) \right)
\]

s.t. \(A_{t,ij} \in \{0, 1\}, \sum_i A_{t,ij} \leq 1, \sum_j A_{t,ij} = 1\). (1)

Here, \(A_{t,ij} \in \{0, 1\} \times n_{t-1}\) is an assignment matrix, IoU is the intersection over-union distance between two bounding boxes and \(L\) is the zero-one loss between two class labels. Intuitively, detected objects from different frames will be matched if their bounding boxes are spatially close and their class labels are the same.

Given the assignment matrix \(A_t\), we align the objects of two frames as follows:

\[
\tilde{o}_n^{t-1} \leftarrow \tilde{o}_n^{t-1}, \quad \text{if } A_{t,ij} = 1 \land \text{IoU}(b_i^t, b_j^{t-1}) > 0.5,
\]

\[
\tilde{o}_n^{t-1} \leftarrow \tilde{o}_i, \quad \text{otherwise}.
\]

Here, \(\tilde{o}\) denotes a special token indicating that there were no matches. We don’t match objects if their IoU is too low.

4.2. Temporal Mesh Prediction

Our goal is to predict the set of meshes \(\mathcal{M} = \{m_i^t\}\) for all objects \(o_i^t \in O_t\). In our case, each mesh \(m_i^t\) is a triangular mesh, \(i.e., m_i^t = (V_i^t, F_i^t)\) is characterized by a set of \(K\) vertices \(V_i^t \in \mathbb{R}^{K \times 3}\) and a set of triangle-faces \(F_i^t \subseteq [1, \ldots, K]^3\).

To predict a mesh \(m_i^t\), we deform the vertices of a reference mesh \(\tilde{m}_{i-1}^t\) based on the detected object’s ROI feature \(R_{oi}^t\) extracted from the image frame. The reference mesh \(\tilde{m}_{i-1}^t\) is a class-specific mean mesh \(\bar{M}_{ci}\) if object \(o_i\) is not tracked, and it is the predicted mesh \(\tilde{m}_{i-1}^t\) from the previous frame if object \(o_i\) is tracked. Formally,

\[
\tilde{m}_{i-1}^t = \begin{cases} T_{o_i}^t(\bar{M}_{ci}) & \text{if } \tilde{o}_{i-1}^t = \tilde{o}_i, \\ \tilde{m}_{i-1}^t & \text{otherwise}. \end{cases}
\]
Recall, \( \tilde{o} \) is a special token indicating that there are no matches from the previous frame (see Eq. (2)). We use the transformation matrix \( T_{o,t} \) to align the mean mesh \( M_{c-t} \) with the reference frame of \( o_t \). We parameterize \( T_{o,t} \) using a deep-net, which regresses the transformation matrix from the ROI features of \( o_t \). Specifically, a mapping from \( R_{o,t} \) to a rotation matrix with three degrees of freedom (the three Euler angles) is inferred. We do not consider translation here, as the mesh is centered in a bounding box.

Next, following prior work [21, 77] we refine the vertices of the reference mesh \( \tilde{m}_{t-1} \) by performing L stages of refinement using the MeshRefine procedure summarized in Module 1. Different from prior work, our reference mesh is propagated from the previous frames, i.e.,

\[
\tilde{V}^{(l+1)}_{t,i} = \text{MeshRefine}^{(l)}(R_{o,t}, \tilde{V}^{(l)}_{t,i}),
\]

where \( \tilde{V}^{(l)}_{t,i} \) are the vertices of the reference mesh \( \tilde{m}_{t-1} \). Finally, we use the original reference’s faces \( \tilde{E}_{t-1} \) for the final mesh output, i.e.,

\[
m^i_t = (V^{(L)}_{t,i}, \tilde{E}_{t-1}).
\]

In greater detail, the MeshRefine procedure in Module 1 extracts image-aligned feature vectors for each vertex from an object’s ROI feature \( R_{o,t} \) using the VertAlign operation [21, 77], where \( E^{(l)} \) denotes the number of feature dimensions. Subsequently, a Graph Convolution Network (GCN) is used to learn a refinement offset \( \Delta V \) to update the reference vertices \( \tilde{V} \). Finally, the module returns the updated mesh vertices to be processed in the next stage. In our case, we use \( L = 3 \) and each GCN\(^{(l)} \) consists of three GraphConv layers with ReLU non-linearity.

Importantly, and different from prior work, instead of predicting meshes from a sphere [77], our approach depends on an additional time dependent reference mesh \( \tilde{m}_{t-1} \). This reference \( \tilde{m}_{t-1} \) summarizes information about an object \( o_t \) obtained from the previous frame \( I_{t-1} \). This enables a recurrent formulation suitable for mesh generation from videos by incorporating temporal information.

### 4.3. Training

We train our model using the dataset discussed in Sec. 3 which consists of sequences of image frames, object bounding box annotations, class labels, and corresponding meshes. In the following, we discuss the training details abstractly and leave specifics to Appendix Sec. A.

To train the model parameters for mesh prediction, we use a gradient-based method to minimize the mesh loss based on differentiable mesh sampling [67]. The loss includes the Chamfer and normal distance between two sets of sampled points each from the prediction and ground-truth mesh. Minimizing the mesh loss encourages the predicted meshes to be similar to the ground-truth mesh. The training of bounding-boxes follows the standard Mask-RCNN procedure [25].

Different from standard end-to-end training, our training procedure consist of two stages:

**Stage 1:** We first pre-train the mesh refinement without taking temporal information into account. This is done by setting the reference mesh \( \bar{m}_{t-1} = T_{o,t}(M_{c-t}) \forall o_t^{t-1} \).

**Stage 2:** We fine-tune our model to incorporate the temporal information of the reference mesh following Eq. (3). Note that at training time, we do not unroll the recursion which would be computationally expensive. Instead, we use an augmented ground-truth mesh of \( o_t \) as an approximation. This is done by randomly rotating the mesh and adding Gaussian noise to the vertices. Without the augmentation, the ground-truth mesh will not resemble \( \bar{m}_{t-1} \), which leads to a mismatch at test time.

### 5. Experimental Results

In the following we first discuss the experimental setup, i.e., datasets, metrics and baselines. Afterwards we show quantitative and qualitative results on the proposed SAIL-VOS 3D dataset and the commonly used Pix3D dataset.

#### 5.1. Experimental Setup

We use the proposed SAIL-VOS 3D dataset to study instance-level video object reconstruction. Please see Sec. 3.1 for statistics of this data. We also evaluate on Pix3D [71] to assess the proposed method on real images.

**Evaluation metrics:** For evaluation we use metrics commonly employed for object detection. Specifically, we report the average precision for bounding box detection, i.e., \( \text{AP}^{\text{bbox}} \), using an intersection over union (IoU) threshold of 0.5. We also provide the average precision for mask prediction, i.e., \( \text{AP}^{\text{mask}} \), using an IoU threshold of 0.5.

Additionally, to evaluate 3D shape prediction we follow Gkioxari et al. [21] and compute the average area under the per-category precision-recall curve using F1@0.3 with a threshold of 0.5, which we refer to via \( \text{AP}^{\text{mesh}} \). Note, we consider a predicted mesh a true-positive if its predicted label is correct, not a duplicate detection, and its F1@0.3 ≥ 0.5. Here, F1@0.3 is the F-score computed using precision and recall. For this, precision is the percentage of points sampled from the predicted mesh that lie within a distance of 0.3 to the ground-truth mesh. Conversely, recall is the percentage of points sampled from the ground-truth mesh that lie within a distance of 0.3 to the predic-
Table 2. Performance on SAIL-VOS 3D. Following MeshR-CNN [21] we report box AP, mask AP, and mesh AP.

![Image of Table 2](image)

Table 3. Ablation study of our method. “MS”: mean shape. “T”: estimate T (Sec. 4.2). “Temp.”: temporal prediction (Sec. 4.2).

![Image of Table 3](image)

5.2. SAIL-VOS 3D Dataset

SAIL-VOS 3D consists of 484 videos made up of 215,703 image frames. The data is split into 162 training videos (72,567 images, 754,895 instances) and 41 clips for validation (1,305 images, 12,199 instances). We leave the rest in test-dev/test-challenge sets for future use. Following Hu et al. [27] we use the 24 classes with an occlusion rate less than 0.75 for experimentation. To compute the mean shape, we average the occupancy grids in the object coordinates. For SAIL-VOS 3D, we rescale the meshes such that the longest edge of the ground-truth mesh is 5. For Pix3D, following [21], we rescale the meshes such that the longest edge of the ground-truth mesh is 10.

For more insights, we further evaluate the aforementioned metrics on subsets of the object instances. This includes a split based on objects’ sizes (small – area less than $32^2$, medium, and large – area greater than $96^2$). We also split the objects based on occlusion rate, i.e., an object is heavily occluded if its occlusion rate is greater than 25%, otherwise it’s slightly occluded. Finally, we split based on the length of the video clip, i.e., a clip is classified as long if the number of frames is more than 30, otherwise short.

**Baselines:** In addition to the developed method we study three baselines on the proposed SAIL-VOS 3D data:

*MeshR-CNN [21]* – a single-view multi-object baseline. Its output representation is a mesh but the intermediate result is a voxel representation.

*Pixel2Mesh†* – The original Pixel2Mesh [77] is a single-view single-object baseline. To deal with multiple objects we use Mask-RCNN and attach Pixel2Mesh as an ROI head. We refer to this adjustment via Pixel2Mesh†. We use the implementation by Gkioxari et al. [21] for this extension. We also considered as a baseline Pixel2Mesh++ [78], a multi-view single-object method. The fact that it is a single-object method makes a simple extension challenging as we would need to incorporate a tracking procedure. Moreover, Pixel2Mesh++ assumes camera matrices to be known which is not always feasible.

*Pix2Vox†* – The original Pix2Vox [85] is a multi-view single-object baseline which uses a voxel representation. Similar to Pixel2Mesh†, to enable multiple object reconstruction, we extend the original Pix2Vox by attaching it to a Mask-RCNN. Similar to Pixel2Mesh++ it is non-trivial to extend Pix2Vox to the multi-object setting. It would require to include object tracking.
Input  Predicted Shape

Figure 7. Predicted shapes of the SAIL-VOS 3D trained Video2Mesh model tested on DAVIS.

Figure 8. Visualization of 3D reconstructions from our Video2Mesh method on Pix3D.

Quantitative results: Quantitative results for the SAIL-VOS 3D dataset are provided in Tab. 2. Note that we fix the box and mask head for this ablation study, which ensures a proper evaluation of our mesh head modifications. We use the mean shape to lead to slight improvements (Row 2 vs. Row 1). Having transformation enabled improves results more significantly (Row 3 vs. Row 2), while the temporal prediction leads to further increase in performance (Row 4 vs. Row 3).

Qualitative results on SAIL-VOS 3D: Qualitative results are shown in Fig. 6. We compare to the runner-up approach MeshR-CNN. Due to the use of a mean shape, we observe the 3D reconstructions of our method to be smoother.

Qualitative results on Real Data: We tested a SAIL-VOS 3D trained model (trained on person only) on a real video from the DAVIS dataset [57] and show a qualitative example in Fig. 7. We find that predicted shapes are reasonable even though the model is trained on synthetic data.

6. Transferring to Pix3D

To assess the use of synthetic data we study transferability. Specifically, we use the SAIL-VOS 3D pretrained model and finetune on Pix3D [71] using different amounts of Pix3D training data. Pix3D [71] is an image dataset. Consequently, we cannot study temporal aspects of our model. We use as conditional input the mean shape rather than the prediction from the previous frame. For evaluation we follow Gkioxari et al. [21] and report the AP metrics in Tab. 4. More details regarding experimental setup and implementation are given in Appendix A. Qualitative results of our Video2Mesh method trained on all Pix3D data are provided in Fig. 8.

In the low-data regime where only 10%, 20%, and 50% real mesh data (Pix3D) are available, pretraining on synthetic data is beneficial: the model pretrained on SAIL-VOS 3D outperforms the model without pretraining. Note that depth data may also be a useful cue to facilitate transferability which we didn’t study yet.

7. Conclusion

We introduce SAIL-VOS 3D and develop a baseline for 3D mesh reconstruction from video data. In a first study we observe temporal data to aid reconstruction. We hope SAIL-VOS 3D facilitates further research in this direction.

Acknowledgements: This work is supported in part by NSF under Grant #1718221, 2008387, 2045586, MRI #1725729, and NIFA award 2020-67021-32799, UIUC, Samsung, Amazon, 3M, and Cisco Systems Inc. (Gift Award CG 1377144 - thanks for access to Arcetri).
References


[34] Abhijit Kundu, Yin Li, and James M Rehg. 3d-rccn:


Multi-view 3d models from single images with a convolutional network. In Proc. ECCV, 2016. 2


