Virtual Occlusions Through Implicit Depth

Jamie Watson\textsuperscript{1,3}, Mohamed Sayed\textsuperscript{1,3}, Zawar Qureshi\textsuperscript{1}, Gabriel J. Brostow\textsuperscript{1,3}, Sara Vicente\textsuperscript{1}, Oisin Mac Aodha\textsuperscript{2}, Michael Firman\textsuperscript{1}
\textsuperscript{1}Niantic, \textsuperscript{2}University of Edinburgh, \textsuperscript{3}UCL

https://nianticlabs.github.io/implicit-depth

Abstract

For augmented reality (AR), it is important that virtual assets appear to ‘sit among’ real world objects. The virtual element should variously occlude and be occluded by real matter, based on a plausible depth ordering. This occlusion should be consistent over time as the viewer’s camera moves. Unfortunately, small mistakes in the estimated scene depth can ruin the downstream occlusion mask, and thereby the AR illusion. Especially in real-time settings, depths inferred near boundaries or across time can be inconsistent. In this paper, we challenge the need for depth-regression as an intermediate step.

We instead propose an implicit model for depth and use that to predict the occlusion mask directly. The inputs to our network are one or more color images, plus the known depths of any virtual geometry. We show how our occlusion predictions are more accurate and more temporally stable than predictions derived from traditional depth-estimation models. We obtain state-of-the-art occlusion results on the challenging ScanNetv2 dataset and superior qualitative results on real scenes.

1. Introduction

Augmented reality and digital image editing usually entail compositing virtual rendered objects to look as if they are present in a real-world scene. A key and elusive part of making this effect realistic is occlusion. Looking from a camera’s perspective, a virtual object should appear partially hidden when part of it passes behind a real world object. In practice this comes down to estimating, for each pixel, if the final rendering pipeline should display the real world object there vs. showing the virtual object [53, 24, 36].

Typically, this per-pixel decision is approached by first estimating the depth of each pixel in the real world image [19, 90, 36]. Obtaining the depth to each pixel on the virtual object is trivial, and can be computed via traditional graphics pipelines [33]. The final mask can be estimated by comparing the two depth maps: where the real world depth value is smaller, the virtual content is occluded, i.e. masked.

We propose an alternative, novel approach. Given images of the real world scene and the depth map of the virtual assets, our network directly estimates the mask for compositing. The key advantage is that the network no longer has to estimate the real-valued depth for every pixel, and instead focuses on the binary decision: is the virtual pixel in front or behind the real scene here? Further, at inference time we can use the soft output of a sigmoid layer to softly blend between the real and virtual, which can give visually pleasing compositions [43], compared with those created by hard thresholding of depth maps (see Figure 1). Finally, temporal consistency can be improved using ideas from temporal semantic segmentation that were difficult to apply when regressing depth as an intermediate step.
We have three contributions:

1. We frame the problem of compositing a virtual object at a known depth into a real scene as a binary mask estimation problem. This is in contrast to previous approaches that estimate depth to solve this problem.
2. We introduce metrics for occlusion evaluation, and find our method results in more accurate and visually pleasing composites compared with alternatives.
3. We show that competing approaches flicker, leading to jarring occlusions. By framing the problem as segmentation, we can use temporal smoothing methods, which result in smoother predictions compared to baselines.

Our ‘implicit depth’ model ultimately results in state-of-the-art occlusions on the challenging ScanNetv2 dataset [17]. Further, if depths are needed too, we can compute dense depth by gathering multiple binary masks. Surprisingly, this results in state-of-the-art depth estimation.

2. Related work

Early approaches to occluding virtual assets in real scenes relied on user annotations of object boundaries [53, 65], precluding general real-time use. The typical approach for automated occlusion is to pixel-wise compare a depth map of the real scene with the virtual depth map [7]. The real image depth map can be estimated, e.g. using structured light [24], or from images [19, 90]. When sparse depths are available, they can be densified [36] or used to rescale relative depths to metric depth [1]. Direct estimation of an occlusion mask has been previously performed for segmentation for AR sky replacement [101] or hands grabbing virtual objects [85]. In contrast, our method enables general object compositing. A detailed review of occlusion handling in AR is outlined in [61, 8].

Depth estimation. Depth estimation is a key component of many AR occlusion systems. Depth can be estimated directly if binocular cameras are available at test time [34, 47, 11, 15]. This approach requires specialized hardware, as does depth estimation from structured light, Lidar, or time-of-flight devices [28]. Further, depth from such devices may not be accurate enough for realistic occlusions without further processing [91, 45].

It is attractive to estimate depth directly from color images, for example from a single image [27, 22, 21, 29, 95]. When a sequence of images is available, Multi-View Stereo (MVS) estimates depth for a reference image using one or more source images [25, 79], which assumes that the scene being observed is static. Recent MVS approaches match image pixels [93, 39] or deep features [41, 20] to create a cost volume, which can then be processed using convolutional layers. Other works have improved upon this basic setup by refining the final output [97], through injection of additional metadata [78], by handling occluded pixels between views [59], or with a Gaussian process prior [37].

Alternative approaches have dropped the reliance on supervised data through the use of self-supervision [29, 27, 30, 94]. There have been attempts to improve the quality of depth estimation around depth discontinuities, e.g. using a gradient or normal loss [71, 55, 38] or a learned network to post-process predictions [70]. Higher quality depths can also be computed via an offline optimization on test sequences [30, 60, 10, 14, 51], but this precludes online use.

In contrast to these depth estimation approaches, we directly estimate an occlusion compositing mask. However, in Section 4 we show that our models can also be adapted to predict depths equivalent to state-of-the-art methods without requiring retraining.

Depth via classification. Our implicit depth approach is related to classification-based depth estimation. Xie et al. [95] is an early approach which learned depth via classification, in the context of depth from stereo. In [23], the output domain is divided into discrete bins, and the final output head classifies each pixel as in front or behind each depth bin. Alternatively, [57, 9, 96, 88] classify the probability that the depth lands in a bin itself. Other works have relaxed the requirement for fixed bin centres, allowing them to be adapted on a per-image [5] or per-pixel [6] basis. Bi3D [3] pose stereo depth estimation as a binary classification task, but where a scalar query depth is provided to the network. Also related to depth classification approaches are works which decompose images into two or more layers, e.g. foreground and background [18, 89, 54].

We also frame geometry estimation as classification, but our approach classifies if a per-pixel virtual depth is in front or behind the real world object at that depth. We can therefore estimate a full compositing mask with a single forward pass, without a dense output tensor. We compare to classification approaches and show that our method is superior in terms of accuracy and compute.

Image and video segmentation. Our work is related to object segmentation [31, 99], salient object segmentation [44], and alpha-matting [56, 80, 12]. Like these, we estimate a binary mask, but our mask is conditioned on an input rendered virtual depth map. Similar to video segmentation [26, 68], we encourage temporal consistency across frames to prevent flicker.

Occlusion boundaries and regions. There are works which focus on detecting pixels which become occluded and disoccluded between frames in a video [40, 92, 35, 83, 84]. We differ as we recover the occlusion mask of a virtual object in real input images.

Implicit volumes. Finally, our approach is related to works on implicit volumes e.g. [58, 66, 62, 75, 76, 77, 16, 73, 67], where a 3D shape is represented by a trained multi-layer perceptron (MLP). When evaluated at each location in 3D space, the MLP’s binary output indicates if that location
is inside or outside an object. However, we operate in image space, and predict if a pixel in the real world scene is in front or behind a virtual target’s depth map. Zhu et al. [100] used an implicit function for RGD completion, operating on ray-voxel pairs. Neural radiance fields [63, 4, 64, 98] (NeRFs) are an alternative implicit approach which can estimate depths and color images from novel viewpoints. This can be used for AR effects, but NeRFs are typically not suited to online applications in novel scenes.

3. Method

Our goal is to automatically composite virtual objects into images of real scenes, respecting any real occluding objects that are ‘in the way’. At inference time, we assume we have an RGB image \( I_{\text{real}} \) as input, together with a temporally preceding sequence of RGB source images and corresponding camera intrinsics and poses. We denote the full sequence of \( I_{\text{real}} \) together with source images as \( I_{\text{real}} \). We also assume knowledge of a 3D virtual object that we wish to place in the scene, which can change over time. From the 3D virtual object and camera poses, we extract for each frame a color rendering of the virtual object \( I_{\text{virtual}} \) with an associated virtual depth map \( D_{\text{virtual}} \).

3.1. Our approach

Given the rendering of the asset, the job of an occlusion step is to estimate which virtual pixels should be shown, and which should be hidden, to create the final image \( I_{\text{final}} \). This can be described by a compositing equation [69, 82], using the two images and a per-pixel compositing mask \( C \), so

\[
I_{\text{final}} = CI_{\text{real}} + (1 - C)I_{\text{virtual}}. \tag{1}
\]

This compositing equation is only applied to pixels covered by the virtual object, outside of which we only show \( I_{\text{real}} \).

Traditional occlusion methods, e.g. [70, 90], use a depth map to estimate \( C \). The depth map \( D_{\text{real}} \) is the output of a network \( \psi \), which takes \( I_{\text{real}} \) as input, so \( D_{\text{real}} = \psi(I_{\text{real}}) \). Here the compositing mask \( C \) was formed using the relation

\[
C = [D_{\text{real}} < D_{\text{virtual}}], \tag{2}
\]

where \([\cdot]\) is the Iverson bracket.

Training this network to instead directly predict \( C \) is a potentially attractive alternative solution. However, this direct prediction is not feasible without \( D_{\text{virtual}} \), as the network has no context at inference time of where the virtual object should be positioned in the world and therefore would be unable to produce a plausible mask.

In our approach, we instead use a deep network \( \phi \) to directly estimate \( C \), conditioned on both \( I_{\text{real}} \) and \( D_{\text{virtual}} \) as input, so

\[
C = \phi(I_{\text{real}}, D_{\text{virtual}}). \tag{3}
\]

The final image is then formed using Eqn. 1 from above.

Advantages of our depth informed mask prediction.

We hypothesize that it is easier for our network to directly predict a binary ‘in front vs. behind’ value at each pixel location, compared with existing methods that predict a continuous value to regress the absolute depth.

3.2. Predicting an occlusion map

A natural choice for our network \( \phi \) would be an image-to-image network. This would take the concatenation of \( I_{\text{real}} \) and \( D_{\text{virtual}} \) as input, and then output \( C \). However, at training time such an architecture would need to see both realistic images and realistic virtual depth maps corresponding to the scene. Generating realistic virtual depths for a scene is difficult, as we do not know what the final use case of the system might be, and thus placing virtual objects in a scene automatically at training time is a non-trivial task. We instead take a different approach, and propose an architecture with two parts: (i) a backbone network for image encoding, followed by (ii) a per-pixel MLP (see Figure 2). Our virtual depths are only provided to the per-pixel MLP, meaning our training-time virtual depths do not need to be realistic virtual depth maps.

Backbone network for image encoding. Our backbone network maps the RGB image \( I_{\text{real}} \) to a pixel-aligned feature encoding \( F \) with \( K \) channels per pixel. While we could use any backbone to extract features, for most of our experiments we use a multi-view stereo approach as in [78]. This requires temporally preceding source frames and known camera poses. See Section 3.4 for details.

Predicting the occlusion mask with an MLP. The final prediction of the compositing mask at pixel location \( p \) relies on three inputs:

1. The image features at \( p \), i.e. \( F(p) \). Inspired by [49], we interpolate features from \( F \) at arbitrary sub-pixel locations. This enables us to make final predictions at arbitrary locations and resolutions.
2. The virtual object depth at \( p \), i.e. \( D_{\text{virtual}}(p) \). Again, we can sample \( D_{\text{virtual}} \) at arbitrary sub-pixel locations.
3. The warped previous temporal prediction at \( p \), as described next. This enables the network to use temporal information for more stable predictions.

At location \( p \), we concatenate the above three inputs to make a \( K+2 \)-dimensional feature. This is given to an MLP to produce the final compositing output for that location \( C(p) \). The final layer of the MLP has a sigmoid activation, so \( C(p) \) is continuous \( \in [0, 1] \). This can be interpreted as the probability that \( I_{\text{real}}(p) \) is occluding the virtual object at depth \( D_{\text{virtual}}(p) \).

Temporal stability. Temporally stable occlusions are important for seamless and believable AR immersion as per-frame depth or semantic predictions can vary over time,
leading to visual ‘flickering’. To combat this, we encourage the network to be temporally stable, inspired by [87, 68]. To achieve this, we feed to the MLP the previous prediction for the pixel at location \( p \), \( \hat{C}(p) \), defined as \( C^t(warp(p)) \). warp uses the known relative camera transform to backwards warp the pixels’ locations at time \( t \) to time step \( t - 1 \) using [42]. This warping requires known depth at time \( t \), for which we use the rendered virtual depth. This is in contrast to [87] which does not use geometric information.

**Relationship to prior work.** While previous approaches (e.g. [3]) have trained a network which takes a single query depth as input, our approach is novel as our query depth can vary spatially per pixel, so each pixel can have a different query depth. Methods like [3] would require exhaustive evaluation of all depth values to be able to composite an object. Our approach can be seen as producing ‘implicit depths’, similar to prior work in 3D, e.g. [66].

### 3.3. Training our network

Our goal at training time is to update the weights of our network \( \phi \) (i.e. both feature encoder and MLP) to accurately predict occlusions. We use training datasets, e.g. [17, 72], where we have access to sequences of training images \( I_{real} \) with pixel-aligned ground truth depth maps \( D_{real} \) and associated camera poses and intrinsics. However, these datasets do not come with augmented virtual depths \( D_{virtual} \), so we need to synthesize these at training time.

**Our training samples.** We require training tuples with an image location \( p \), a virtual depth at that location \( D_{virtual}(p) \), and a ground truth label \( y_i \in \{0,1\} \) stating if the virtual depth is in front (0) or behind (1) the real image depth map. Additionally, to encourage temporal stability, we require a previous prediction for this location, \( \hat{C}(p) \).

To generate a training sample, we choose a single training sequence \( I_{real} \), with associated depth map \( D_{real} \). We sample a 2D location \( p \) uniformly in image space, and subsequently sample a training-time feature \( F(p) \). Given the image location \( p \), we have a choice to sample our synthesized virtual depth \( D_{virtual}(p) \) anywhere along \( p \)’s camera ray. Sampling a random depth means we might be far away from the difficult choices. The most difficult choices for depths are when \( D_{virtual} \) is near \( D_{real} \), so we bias a fraction of our training-time samples to come from near the ground-truth depth surface \( D_{real} \). Similar to [76, 77], with probability \( q \) we sample from a Gaussian with mean of the ground truth depth value at pixel \( p \), and variance 0.05. To ensure that we also make sensible predictions away from real surfaces, with probability \( 1 - q \) we sample a training depth uniformly between the minimum and maximum depth in \( D_{real} \). The ground truth depth map determines the label \( y_i \), which, in turn, is used to supervise the network with binary cross-entropy.

**Training for temporal stability.** As stated in Section 3.2, we encourage temporal stability by giving the warped previous prediction as an additional input to our MLP. During training, to avoid the need to run inference for multiple frames, we synthesize a previous prediction from the ground truth label \( y_i \) in a manner similar to [87]. We corrupt \( y_i \) to produce a pseudo previous prediction \( \hat{C}(p) \) by adding random noise, converting the binary labels into...
Table 1. Occlusion and depth scores after converting our masks to depths compared with state-of-the-art prior works. Evaluation is on the ScanNetv2 test set keyframes [20], and follows the evaluation protocol for depth from [20]. Our model is state-of-the-art on both occlusion and depth estimation. * indicates trained on additional data.

<table>
<thead>
<tr>
<th>Method</th>
<th>IoU All↑</th>
<th>IoU Surface↑</th>
<th>IoU Boundary↑</th>
<th>Abs Diff↓</th>
<th>Abs Rel↓</th>
<th>Sq Rel↓</th>
<th>RMSE ↓</th>
<th>δ &lt; 1.05 ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPSNet [41]</td>
<td>46.17</td>
<td>21.68</td>
<td>23.50</td>
<td>.1552</td>
<td>.0795</td>
<td>.0299</td>
<td>.2307</td>
<td>49.36</td>
</tr>
<tr>
<td>MVDensityNet [93]</td>
<td>44.64</td>
<td>21.06</td>
<td>23.22</td>
<td>.1648</td>
<td>.0848</td>
<td>.0343</td>
<td>.2446</td>
<td>46.71</td>
</tr>
<tr>
<td>DELTAS [81]</td>
<td>48.48</td>
<td>23.37</td>
<td>25.51</td>
<td>.1497</td>
<td>.0786</td>
<td>.0276</td>
<td>.2210</td>
<td>48.64</td>
</tr>
<tr>
<td>GPVMVS [37]</td>
<td>46.52</td>
<td>22.43</td>
<td>23.98</td>
<td>.1494</td>
<td>.0757</td>
<td>.0292</td>
<td>.2287</td>
<td>51.04</td>
</tr>
<tr>
<td>DeepVideoMVS, fusion [20]*</td>
<td>53.16</td>
<td>26.49</td>
<td>28.05</td>
<td>.1186</td>
<td>.0583</td>
<td>.0190</td>
<td>.1879</td>
<td>60.20</td>
</tr>
<tr>
<td>SimpleRecon (ResNet)</td>
<td>58.91</td>
<td>31.48</td>
<td>33.03</td>
<td>.0978</td>
<td>.0487</td>
<td>.0151</td>
<td>.1617</td>
<td>69.62</td>
</tr>
<tr>
<td>SimpleRecon [78]</td>
<td>60.52</td>
<td>32.44</td>
<td>34.52</td>
<td>.0871</td>
<td>.0429</td>
<td>.0123</td>
<td>.1460</td>
<td>74.01</td>
</tr>
<tr>
<td>SimpleRecon (ResNet) + Ours</td>
<td>60.14</td>
<td>33.29</td>
<td>36.54</td>
<td>.0988</td>
<td>.0498</td>
<td>.0149</td>
<td>.1595</td>
<td>68.52</td>
</tr>
<tr>
<td>SimpleRecon [78] + Ours</td>
<td>62.61</td>
<td>35.48</td>
<td>38.01</td>
<td>.0862</td>
<td>.0436</td>
<td>.0123</td>
<td>.1426</td>
<td>73.74</td>
</tr>
</tbody>
</table>

Table 2. Our method outperforms depth regression for the occlusion task, regardless of the underlying architecture. All these architectures were trained and evaluated on ScanNetv2 sequences by us, using code published by the authors, and with automated virtual object insertion to evaluate binary occlusion masks.

<table>
<thead>
<tr>
<th>Architecture / Method</th>
<th>IoU All↑</th>
<th>IoU Surface↑</th>
<th>IoU Boundary↑</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleRecon [78]</td>
<td>60.52</td>
<td>32.44</td>
<td>34.52</td>
</tr>
<tr>
<td>SimpleRecon [78] + Ours</td>
<td>62.61</td>
<td>35.48</td>
<td>38.01</td>
</tr>
<tr>
<td>SimpleRecon (ResNet)</td>
<td>58.91</td>
<td>31.48</td>
<td>33.03</td>
</tr>
<tr>
<td>SimpleRecon (ResNet) + Ours</td>
<td>60.14</td>
<td>33.29</td>
<td>36.54</td>
</tr>
<tr>
<td>ManyDepth [94]</td>
<td>55.68</td>
<td>29.18</td>
<td>30.47</td>
</tr>
<tr>
<td>ManyDepth [94] + Ours</td>
<td>56.55</td>
<td>31.56</td>
<td>34.47</td>
</tr>
<tr>
<td>MonoDepth2 [30]</td>
<td>46.06</td>
<td>18.62</td>
<td>23.37</td>
</tr>
</tbody>
</table>

3.4. Implementation details

We train all models and baselines using the Adam optimizer [48] with a batch size of 24 split across 2 GPUs. For speed of convergence, we initialize our backbone network with the weights of a depth regression network, and train for 40k steps, with an initial learning rate of 0.0001 dropping by a factor of 10 after 16k steps and 32k steps. Images are augmented with standard flip and color augmentations, as in [78]. We use $q = 0.25$ for our probability of sampling a virtual depth near the real depth surface. Similar to [30], we supervise the network at 4 output scales, using ground truth depth at a higher resolution than the feature maps, and at test time only use the highest resolution output.

Backbone. Our backbone network is based on [78], where a shallow feature extractor feeds a metadata infused plane sweep volume, followed by a U-Net [74]. See the supplementary material for full details.

MLP. Our compositing mask prediction uses an MLP with three fully-connected layers, with $K+2$ input channels and a single channel final output. Both hidden layers have 128 floats $\in [0, 1]$, simulating the output of a sigmoid. To teach our model to be robust to incorrect previous predictions at inference time, we set $\hat{C}(p) = 1 - \hat{C}(p)$ with probability $p_1$ during training. Additionally we use $\hat{C}(p) = -1$ to indicate the start of a sequence, and set this during training with probability $p_2$. For all experiments we use $p_1 = p_2 = 0.25$.

Regularization around depth discontinuities. Training a model as described in Section 3.3 can give rise to artefacts in predicted compositing masks near depth discontinuities. Due to the inherent ambiguity of occlusions in these regions, our model tends to predict values close to 0.5, which leads to less visually pleasing results in the final compositing. To combat this, during training we locate depth discontinuities in the ground truth depth map using a Sobel filter [46], and apply an L1 regularizer to penalize predictions near 0.5 in these regions. See the supplementary material for details.
dimensions. We use ELU activations after the first two layers, with the final activation being a sigmoid to map our outputs to the range $[0, 1]$. Based on our backbone [78], our full-scale feature map has $K = 64$.

**Timings.** Inference with our backbone (from [78]) takes 64ms and our MLP takes 0.5ms on an A100 GPU.

4. Experiments

**Datasets.** We train and evaluate models using ScanNetv2 [17], with the standard train/val/test split. This allows direct comparison, of occlusions or depths, with most prior depth estimation methods that also train on ScanNetv2. For qualitative comparisons, we also train a model on the synthetic Hypersim [72] dataset. This synthetic data has better aligned edges in the training data, so yields a model with high edge fidelity. In visual occlusion comparisons, we compare that model against Lidar depth sensing [2] and a previous method which does not use a trained model [36].

**Backbone variants.** We present two variants of our model; ‘SimpleRecon + Ours’ which uses the architecture of [78] as a backbone, and a faster lower compute version using a ResNet-18 [32] encoder, a lightweight decoder, and a simple dot-product cost volume, referred to as ‘SimpleRecon (ResNet) + Ours’. We also train a ResNet variant without a cost volume inspired by [30]: ‘MonoDepth2 + Ours’.

4.1. Evaluating virtual asset occlusion

We directly evaluate performance on the task of virtual object insertion. We report scores using the standard segmentation metric, intersection-over-union (IoU), to measure occlusion quality. We compare our variants against state-of-the-art depth estimation on the standard ScanNetv2 test set.

Since rendered virtual assets would add noise to the evaluation process, we use infinite planes that lie ahead of the camera at each frame. These planes are placed at depths $d \in \{0.5m, 1.0m, \cdots, 5.0m\}$ along the look-at vector of the camera, where each plane’s virtual depth map is $D^\text{virtual}_d$. We obtain a ground truth binary occlusion mask, $Y^\text{GT}_d$, for each plane using the ground truth depth map, obtained from depth sensors.

For our method, we compute the probability of occlusion for each depth plane, $C^d$, which we threshold with $\tau$ to produce $Y^{\text{pred}}_d$. We pick $\tau$ for each depth bin using a mini-val set of 100 scans. We compute IoU for the occluded asset fragments, $\text{IoU}^\text{frag}_d$, and the visible parts of the asset, $\text{IoU}^\text{vis}_d$. For depth estimation methods, we obtain the predicted occlusion mask directly by comparing $D^\text{pred}_d$ and $D^\text{virtual}_d$ to compute IoU$^\text{frag}_d$ and IoU$^\text{vis}_d$. We compute IoU All$^d$ for each plane using the harmonic mean of IoU$^\text{frag}_d$ and IoU$^\text{vis}_d$. We average IoUs for each keyframe from [20] and then for each depth plane. As regions near depth boundaries tend to be difficult, following [13] we evaluate IoU Boundary, and separately, regions near the geometry’s surface (IoU Surface).

In Table 2, we combine our method with existing backbones by first training with regression losses [78], and then finetuning with our approach on ScanNetv2. We compare with several recent MVS methods, including SimpleRecon [78], ManyDepth [94], as well as a single frame depth method MonoDepth2 [30]. In all cases, our method improves occlusion scores, most notably in difficult cases near surfaces. Additionally, our lightweight ResNet variant yields strong performance (sometimes exceeding [78]) while operating at a fraction of the compute time (20ms vs 64ms on an A100 GPU). Note that for fair comparison to regression baselines, all results (except those in Table 4) are without our temporal stability contribution.

4.2. Evaluating depth estimation

While our method is focused on estimating occlusions for virtual assets, we can leverage our binary predictions to iteratively refine a binary-searched depth map. We compare against depth estimation methods on the ScanNetv2 test set using the protocol from [20], presenting results in Table 1.

We can convert our binary predictions to depths by making the observation that along each ray from every pixel location $p$ there lies a depth $d$ where the prediction from our network, $C(p)^d$, is at the decision threshold $\tau$. Generally $\tau$ would be 0.5, but we use the best thresholds from Section 4.1. Our optimal estimated depth map $D^\text{pred}$ is one where $C(p)^d = \tau$ for all $p$.

It is time consuming to naively iterate different depth values to find $d$ for which $C(p)^d = \tau$. Instead we binary search along the ray to find the optimal depth, relying on directional predictions that signify if the current depth on the ray is ahead or behind the real depth. We initialize our min. and max. depths to 0.5m and 8m respectively. These are updated each iteration, for each location p, as we search.

We take $M = 12$ binary search steps, achieving an effective granularity of 4096 for each $p$ (see Figure 4). Only the final MLP head is run at each step, with the backbone only run...
Table 3. Ablating our method, showing our contributions lead to better depth and occlusion scores. All are trained equivalently on the ScanNetv2 dataset using the SimpleRecon (ResNet) backbone network.

Table 4. Evaluating temporal stability on ScanNetv2, by comparing predictions on temporally adjacent frames. Our temporal approach leads to significantly less flicker, as seen in the large reduction in the temporal score, without impacting IoU.

4.3. Temporal evaluation

Occlusion systems should exhibit temporal coherence to ensure visually compelling results [36]. In the spirit of [60, 36], we place a fixed virtual AR asset into a scene (here a plane), and keep track of the change in predicted visibility of the groundtruth scene mesh, provided in ScanNetv2, across a window of frames. Specifically, we use an infinite vertical plane at a fixed position in front of the first camera in a sequence, and compute a compositing mask, $C_t^v$, for each subsequent frame. We project scene mesh vertices to the camera and store the visibility prediction from $C_t^v$ for that vertex w.r.t the vertical plane. We tally the number of times the visibility prediction changes for each vertex over 13 frames (i.e. 0.43 seconds). We normalize the count by the number of frames to compute a temporal score. Evaluation is performed on the ScanNetv2 test scenes.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
Method & Temporal Score $\downarrow$ & IoU A $\uparrow$ & IoU S $\uparrow$ & IoU B $\uparrow$ \\
\hline
Regression & 233.1 & 77.84 & 43.44 & 41.29 \\
Ours (w/o temporal) & 235.1 & 79.09 & 44.73 & 42.44 \\
Ours (with temporal) & 164.5 & 79.28 & 44.50 & 42.90 \\
\hline
\end{tabular}
\caption{Evaluating temporal stability on ScanNetv2, by comparing predictions on temporally adjacent frames. Our temporal approach leads to significantly less flicker, as seen in the large reduction in the temporal score, without impacting IoU.}
\end{table}

‘Ours (with temporal)’ results in a significant boost of almost 30% in temporal stability while achieving IoU scores comparable to our non-temporal variant (see Table 4). We also show a qualitative example of our more temporally stable approach in Figure 5. Please see our video for examples.

4.4. Ablation

We validate our approach by training variants of our model with our contributions turned off, and show in Table 3 that these models achieve worse scores. We train a model without our edge-based regularization; without high resolution supervision, where our MLP is supervised at the native model output resolution as in [78]; a non-MVS monocular method; a DORN-style classification network, where we output a classification head with 80 bins each with a BCE-trained sigmoid activation; and a discretized depth-classification loss.

4.5. Qualitative comparisons

Figure 6 shows qualitative results trained on Hypersim [72] on a range of real-world scenes, comparing our approach to alternative state-of-the-art methods. Surprisingly, our method produces visually equivalent, or better-quality, predictions compared to those from on-device Lidar from an iPhone 12 Pro. We also compare to the sparse-point densification approach from [36]. Their approach relies on sparse points as input. We found that ARKit’s SLAM points are too sparse for their method, so we instead randomly sampled 2,000 Lidar depth points for each test frame, and fed these into their publicly available code. Visually, our re-
Figure 6. **Qualitative occlusion comparisons using our own casually captured footage.** We occlude virtual assets (rows 1-2) and a fixed plane at 2m depth (rows 3-4). Our occlusions are typically more realistic than baselines, in particular around soft edges, e.g. leaves. We also avoid catastrophic failures, e.g. around the bars in the final row. Please see the supplementary material for videos.

Figure 7. **Additional qualitative results.** On the right we see a failure mode, where transparency through glass is not handled correctly. This is due to limitations in our training data [72]. Please see our video for more results.

Figure 8. **Moving objects.** While our model was trained on static scenes, we achieve surprisingly robust results on moving objects.

5. Conclusion

We presented a novel approach for inserting virtual objects into real scenes. In contrast to existing depth-based methods, we directly estimate compositing masks. We introduce metrics for evaluating occlusion mask quality, and showed that our approach allows for greater temporal stability than previous methods. Qualitative results highlight that our method produces more realistic object insertions. Results have better edge fidelity than their Lidar-guided predictions. We also found that our predictions are surprisingly good on moving objects, given that our training data comes from static scenes (see Figure 8). More results are shown in Figures 3 and 7, and the supplementary video.

Acknowledgements. Many thanks to Daniyar Turmukhambetov, Jamie Wynn, Clément Godard, and Filippo Aleotti for their valuable help and suggestions.
References


[101] Zhengxia Zou, Rui Zhao, Tianyang Shi, Shuang Qiu, and Zhenwei Shi. Castle in the sky: Dynamic sky replacement