Creating 3D Diorama from Single Image with Deep Learning

Martin Vejbora[1], Elena Šikudová[1]

[1]Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Abstract

Creating 3D scenes is a time-consuming task that requires experience with modeling software. This paper presents a novel approach that combines neural models for panoptic segmentation and monocular depth estimation to construct dioramas. While previous research has explored generating dioramas from single images, to the best of our knowledge, there is no research utilizing deep learning techniques for the task. This paper provides an analysis of existing approaches to diorama generation. We then describe the construction of the diorama, where objects identified by segmentation are separated into distinct images with transparent backgrounds. These images are then placed in a 3D scene, arranged to reflect the estimated depth of each object. We also address several challenges that had to be overcome. Specifically, we employed fine-tuning to address the limitations of the available depth model when applied to outdoor scenes. Our method has been implemented as an add-on for the open-source 3D software Blender, utilizing neural models in the ONNX format for depth and segmentation inferences.

Keywords

deep learning, diorama, Blender, panoptic segmentation, monocular depth estimation

1. Introduction

Creating 3D environments in modeling software can be a repetitive and time-consuming task. However, lower-quality models are usually sufficient for assets in the background and further away from the camera. This is where the use of automated tools can come in handy.

This paper focuses on creating dioramas which are sets of planes placed in a 3D scene to evoke the perception of depth. They are computationally cheap for rendering since they do not utilize any complex mesh, making them suitable for background assets. Dioramas work best when the camera is facing them, moving slightly, and viewing the diorama from slightly different angles. The effect breaks when the diorama is viewed from a side.

Previous works used traditional machine learning techniques to create dioramas, limiting their usage to hazy input images, outdoor scenes, or images with zero or one vanishing point. Moreover, their implementation was either not published or is now outdated and no longer functional, making them impractical to use.

We study the utilization of deep learning to automate the process of creating dioramas. Our implementation uses a pre-trained state-of-the-art model for panoptic segmentation and a competitive model for depth estimation that we fine-tune for outdoor scenes. The selected models are powerful yet small enough to run on standard computers or notebooks. Since research in deep learning has been very progressive in recent years, we pay attention to designing the implementation in order to be able to easily use better models in the future.

We implement our method as an add-on for the free 3D software Blender[1], which supports all three major platforms; Linux, Windows, and Mac. The add-on strives to be easy to use, the user selects an input image, and the add-on automatically creates a diorama from it without a need to do any further manual steps in the process.

Even though the quality of the resulting diorama varies based on the input image, our approach has weaker constraints on the input images than the previous works.

This paper is structured as follows. Section 2 provides an overview of existing work on automatic diorama creation. Section 3 discusses the used framework and models with a focus on fine-tuning the depth model. Then, this section covers the implementation of the add-on and the most significant design choices. Section 4 compares the results of the original and fine-tuned depth models and shows the visual appearance of the diorama. Furthermore, it discusses the strengths and weaknesses of our solution. Final Section 5 summarizes what was achieved and outline potential areas for future work.

2. Related Work

This section looks at existing work that automates creating dioramas or similar 3D entities from single input images.

ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Tatranske Matliare, Slovakia

mvejbora@seznam.cz (M. Vejbora); sikudova@cgg.mff.cuni.cz (E. Šikudová)

0000-0003-4572-4064 (E. Šikudová)

[1]https://www.blender.org/

Based on the research of human depth perception, Assa and Wolf [1] define depth cues, including partial occlusion, texture density analysis, depth of focus, atmospheric scattering, and object height in the visual field. They utilize segmentation to obtain 10-20 major segments per image and smaller patches called superpixels. They estimate relative depth differences among objects by comparing depth clues between superpixels on borders or inside of bigger segments.

Having defined a new viewing point, authors render a novel image that occludes certain parts of the original image. They also use image completion techniques to inpaint the previously occluded areas which become visible. Their approach yields the best results for outdoor scenes with minimal regular patterns or straight lines.

Similarly to the previous approach, Make3D [2] uses segment patches and defines depth cues both within and between these patches. Instead of estimating a depth map, authors build a 3D mesh from planes to represent a scene from an input image. They train a Markov Random Field (MRF) to model relationships between adjacent patches. The MRF infers locations and rotations of segment planes in a three-dimensional space. This inference is conditioned by over 500 local features computed from each patch, along with various relationships computed between patches. These inter-patch relationships involve advanced edge detection or estimation of co-planarity and co-linearity. Since an output of the algorithm is a whole textured mesh, it allows easy synthesizing of novel views.

The research paper called PEEP: Perceptually Enhanced Exploration of Pictures [3] focuses on images with zero or one vanishing point. PEEP maps an image to 5 planes forming a pyramidal frustum to achieve a plausible 3D effect. Similarly to the previous approaches, the first step obtains segmentation patches. These patches correspond to planes in three-dimensional space. Graphcut strategy on patches is used to fit points representing frustum. If we limit ourselves to images with zero or one vanishing point, authors claim their result is visually more plausible even though geometrically less precise than the one created by Make3D [2].

Zhao et al. [4] limit their depth estimation to a single depth cue – atmospheric scattering. They use the Dark Channel Prior dehazing algorithm to compute depth in their research. Authors cluster depth and radiance outputs of the dehazing process obtaining approximately five segments per image. After estimating the depth and segmentation, the position and orientation of segment planes are computed. Segmented alpha planes are placed behind each other to form a resulting diorama.

A significant portion of the article is dedicated to enhancing the visual appeal of segmented images. The authors blend the alpha channel of segment edges to create smoother transitions between planes. Additionally, areas of the photographed scene that were not visible in the original image are filled with inpainting. Prior to the actual inpainting, a few border pixels of the segment edge are removed using erosion to prevent misclassified pixels from affecting the inpainting algorithm. These misclassified pixels often have colors different from the color of the main object within the segment.

The main drawback of the described algorithm is that it can only be applied to hazy images. This limitation comes from the used depth estimation algorithm.

3. Proposed Solution

All the approaches described in Section 2 use some form of segmentation and depth estimation. With recent advancements in neural networks, state-of-the-art solutions for both tasks now use deep learning. However, to our best knowledge, no publicly available research exists where authors would create a diorama using deep learning.

While monocular depth estimation is an unambiguous task, image segmentation is usually categorized into one of three main tasks: instance, semantic and panoptic segmentation. In instance segmentation, the objective is to identify all instances of given object classes and to determine masks for individual objects. Semantic segmentation assigns a label category to every pixel in an image while not distinguishing multiple instances of a class. Panoptic segmentation, proposed by Kirillov et al. [5], unifies semantic and instance segmentation by introducing two types of objects – things and stuff. Things include countable objects like cars or people, where each instance needs a distinct label. Stuff refers to uncountable or amorphous regions like grass or sky, where it is not possible or desired to distinguish individual instances. Similarly to semantic segmentation, panoptic segmentation labels all image pixels which makes it the most suitable for our use-case. A comprehensive survey of methods for all three segmentation tasks can be found in [6].

Our algorithm takes the input image, cuts objects detected by panoptic segmentation into separate images, and places these images relatively behind each other based on their average depth predicted by the monocular depth estimation model. This approach is illustrated in Figure 1.

Re-implementing a state-of-the-art model based on its research paper can be challenging. Also, with transformer neural networks rapidly developing, new state-of-the-art models for datasets like ADE20K [7] or NYUv2 [8]

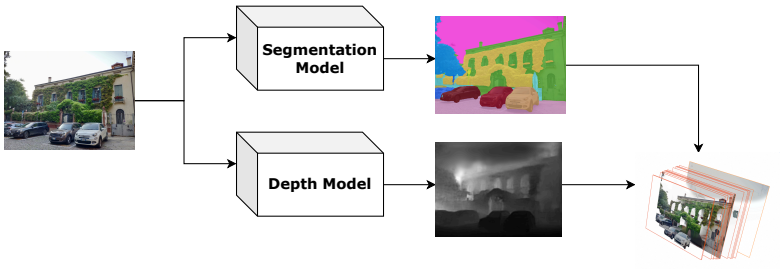


Figure 1: Diagram illustrating how segmentation and depth models are used to generate a diorama from the input image.

appear even multiple times a year. Therefore, better models will likely be available for our tasks in the future, requiring us to re-implement the code again.

For these reasons, we have decided to use a high-level framework called HuggingFace[2] that contains implemented models, including pre-trained weights that can be downloaded from the HuggingFace hub [3]. HuggingFace has a large community, well-documented code, and a lot of online resources. At the time of writing, it had over 80,000 stars on GitHub. Most of its models are implemented in PyTorch, but some also have a TensorFlow or JAX version.

HuggingFace contains very capable models for both of our tasks. The best panoptic model is OneFormer [9], the state-of-the-art model for panoptic segmentation on the ADE20K dataset according to the paperswithcode.com ranking[4] at the time of implementing our add-on. The best depth estimation model from HuggingFace is GLPN [10], ranked 7th on the NYU v2 dataset[5]. We describe the details of these models in the following sections.

3.1. Panoptic Segmentation

Jain et al. [9] introduced **OneFormer**, a model that unifies instance, semantic, and panoptic segmentation tasks. OneFormer achieves state-of-the-art results on all three tasks after training only once, simultaneously.

OneFormer takes two inputs, an RGB image, and a text token. The token determines whether OneFormer executes instance, semantic, or panoptic segmentation. The model's architecture is based on Mask2Former [11], and it consists of three main parts: an encoder-decoder backbone for extracting hierarchical features from the input image, a query module that computes object queries from the input image, and a decoder head that predicts the class and mask for each object query.

To extract multi-scale features from the input image, OneFormer uses Swin [12] backbone encoder and a multi-scale deformable transformer [13] as a pixel decoder. Pixel decoder leverages a deformable attention module that limits attention to a local surrounding, mimicking the inductive bias of convolutions. Like a typical hierarchical decoder, it gradually upsamples the backbone features with the aid of skip connections from the encoder layers of corresponding spatial resolutions. The pixel decoder extracts features at $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, and $\frac{1}{32}$ of the input resolution.

The query formulation module combines the task type input `"the task is {task}"` in a 2-layer transformer with $\frac{1}{4}$ scale features from the pixel decoder to generate query tokens $\mathbf{Q}$. Each query token represents a potential object or segment in the input image. These query tokens are later passed to the transformer decoder, which verifies that they correspond to an actual object, classifies them, and creates a mask for them.

The last part of OneFormer's architecture is the transformer decoder with classification and mask heads. The input of the transformer decoder are object queries $\mathbf{Q}$, which are repeatedly combined with multi-scale features from the pixel decoder. The transformer decoder consists of a masked cross-attention, followed by a self-attention, and a feed-forward network repeated L times for each of the $\frac{1}{8}$, $\frac{1}{16}$, and $\frac{1}{32}$ pixel feature scales. The resulting features are then passed to the classification and mask heads. The classification head predicts a class or `no-object` for each query token. The mask head, on the other hand, computes a binary mask using pixel features at $\frac{1}{4}$ resolution of the original image.

HuggingFace contains versions of OneFormer with Swin backbone [12] pre-trained on the Cityscapes [14], ADE20K [7], and COCO [15] datasets. Figure 2 compares them on an indoor and outdoor scene. We observe that the Cityscapes version yields competitive results on outdoor scenes but does not work at all on indoor scenes

[2] https://huggingface.co/
[3] https://huggingface.co/models
[4] https://paperswithcode.com/sota/panoptic-segmentation-on-ade20k-val
[5] https://paperswithcode.com/sota/monocular-depth-estimation-on-nyu-depth-v2

(Figure 2d). That is due to the dataset's structure containing annotations for 30 classes related only to autonomous driving. On the other hand, models trained on ADE20K and COCO produce comparable results, likely due to the similar structure of both datasets. We choose the COCO version because of its more permissive license.

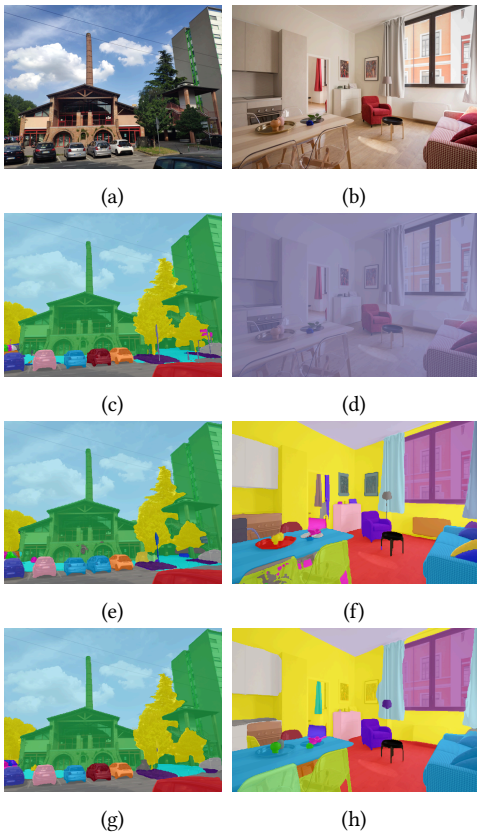


Figure 2: Comparison of panoptic masks from outdoor (left) and indoor (right) scenes obtained with OneFormer trained on Cityscapes (c), (d), ADE20K (e), (f), and COCO (g), (h).

3.2. Monocular Depth Estimation

Global-Local Path Network (GLPN) [10] was introduced in 2022, achieving state-of-the-art results for monocular depth estimation on the NYU v2 dataset [8]. Additionally, the authors argue that GLPN is suitable for real-life applications since it performs well also on images corrupted by methods such as added noise, Gaussian blur, defocus, jpeg compression artifacts, or fog.

GLPN uses a hierarchical transformer encoder from SegFormer [16] to extract features from the input image at four different resolution levels. Each level's encoder block comprises several reduced self-attention and MLP-Conv-MLP modules with residual skip connections. The final component of each encoder block is the patch embedding layer which employs overlapped convolution with stride to reduce the spatial shape of hierarchical features while increasing the number of channels.

A lightweight decoder is connected to the encoder on multiple resolution layers through a Selective Feature Fusion (SFF) module. This module enhances global features with fine details of the local structures that may have been lost in the latter encoder steps. SFFs connect the encoder with the decoder, allowing the decoder to access both the global path from the encoder and the local path through the skip connections. SFF computes a two-channel attention map where the input global features are multiplied by one channel and the local features by the other. These multiplications are element-wise along the channel dimension. Finally, the resulting scaled global and local features are added element-wise.

GLPN applies sigmoid as the last step, which scales the depth output to the range [0, 1]. The result is multiplied by the desired maximal depth in meters, which is specific for each dataset.

HuggingFace offers two versions of GLPN, pre-trained on either NYUv2 [8] or KITTI [17] dataset. A comparison of their inference is shown in Figure 3.

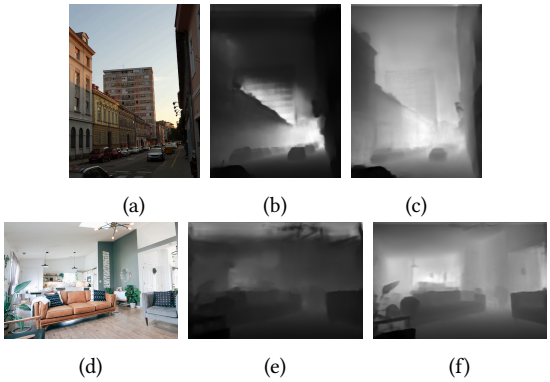


Figure 3: Comparison of inference from outdoor (top) and indoor (bottom) scenes using GLPN trained on KITTI (b), (e) and NYUv2 (c), (f).

We choose the NYUv2 version as it produces more consistent depth maps. The KITTI-trained model produces artifacts, such as a brighter stripe at the top part of the indoor scene in Figure 3e or inconsistent depth estimates of buildings in the left part of the outdoor scene in Figure 3b. The top parts of the higher apartment building and the smaller buildings are estimated to be closer (darker values) than the lower parts of the same real distance. We attribute these artifacts to the structure of the

KITTI dataset, which only contains images captured from a car, so the model does not generalize well on varying scenes. For instance, almost all KITTI images have a sky at the top, and a sky does not have any valid depth values. Thus, the model cannot learn anything there.

3.2.1. Fine-tuning

As shown in Figure 3, the selected model trained on NYUv2 performs well on indoor scenes, and despite never seeing any depth-annotated outdoor scenes, it generalizes surprisingly well on them. However, there are still some inconsistencies. For example, in Figure 3c, the two smaller buildings are estimated to be further away than the high apartment building behind them. To improve the quality of our diorama on outdoor scenes, we decide to fine-tune the model on the DIODE dataset [18], which contains both indoor and outdoor scenes. It contains around 17,000 outdoor images and almost 9,000 indoor images, with all depth maps obtained using a laser scanner. Figure 4 shows an example of RGB images, depth maps, and binary validity masks which mark invalid depth values by black color.

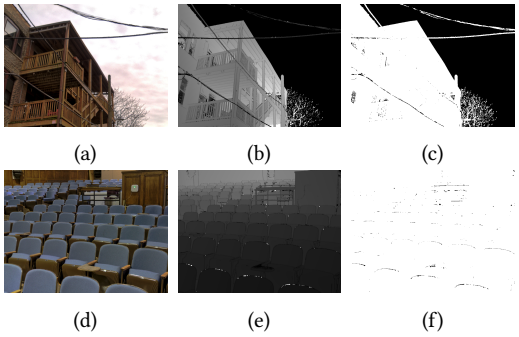


Figure 4: Example outdoor and indoor RGB images (a),(d), corresponding depth maps (b),(e) and validity depth mask (c),(f) from the DIODE dataset [18].

Similarly to the authors of GLPN, we use scale-invariant log scale metric [19] with $\lambda = 0.5$ as the training loss function:

$$L = \frac{1}{n}\sum_i d_i^2 - \frac{\lambda}{n^2}\left(\sum_i d_i\right)^2 \quad (1)$$

where $d_i = \log y_i - \log y_i^*$, y denotes a predicted depth map, y^* a ground truth, n a total number of pixels and i an index of pixel. The authors show this metric is invariant to the global scale of the predicted and ground truth depth maps for $\lambda = 1.0$.

In the following text, we describe the hyperparameters that achieve the best results in our training. We have obtained them after experimenting with various settings.

	Indoor	Indoor & Outdoor	Outdoor
Minimum	0.30	0.30	0.30
Average	3.58	9.55	12.59
Maximum	50.00	300.00	300.00
$Q_{.001}$	0.38	0.39	0.41
$Q_{.01}$	0.61	0.68	0.91
$Q_{.10}$	1.06	1.36	2.37
$Q_{.25}$	1.56	2.30	3.95
$Q_{.50}$	2.48	4.25	7.41
$Q_{.75}$	3.84	9.64	15.48
$Q_{.90}$	6.09	21.47	28.44
$Q_{.95}$	9.03	31.92	39.34
$Q_{.99}$	32.72	61.69	76.10
$Q_{.999}$	39.69	123.44	140.48

Table 1
Analysis of the training split of the DIODE dataset [18], presenting minimum, average, maximum, and quantile values.

First, we need to adjust the **depth range** predicted by the GLPN model. The available pre-trained model outputs values in the 10-meter range as it was trained on the NYUv2 dataset, which has a maximal distance of 10 meters. On the contrary, our DIODE dataset was obtained using a laser scanner with a maximal range of 350 meters. We compensate for this difference by adjusting the final scale, which multiplies the output of the decoder sigmoid. A straightforward choice would be to multiply the result by 350. For example, the authors of the dataset also construct their baseline model [18] to output depth values from 0 to 350 meters. However, we achieve slightly better results using a smaller range, and we argue it is sufficient. When analyzing the official training split of the dataset, we found that more than 99.9% of the depth values of the joint indoor and outdoor parts are smaller than 150 meters. Thus, we can safely use 150 as the maximum depth value without limiting the model too much. We hypothesize that this utilizes the output range better, as well as the slope of the sigmoid for computing gradients. The full analysis of the training split of the DIODE dataset can be found in Table 1.

During training, when we feed the model with images, we use **data augmentation** to improve its generalization capabilities. We limit the augmentation techniques to a subset of those used in the original paper. Specifically, we apply horizontal flipping with a 50% probability and make random adjustments to brightness (±0.2), contrast (±0.2), hue (±0.2), and saturation (±0.3).

In the original research paper, the authors use training images with the **resolution** of 576×448. We have conducted experiments with various resolutions, including the original sizes of NYUv2 (640×480) and DIODE (1024×768) images. However, we have found that changing the resolution does not significantly impact the accuracy. As a result, we use a resolution of 640×480 for

most of our experiments. We attribute this robustness to the fact that the SegFormer [16] backbone in GLPN does not rely on fixed positional encodings concatenated to the input patches. The variable resolution of the images only changes the number of patches but not the encoded value of the input patch.

For most of our experiments, we use a **batch size** of 8, the maximum batch size where the training of 640×480 images fits in the 24GB memory of the NVIDIA Titan RTX GPU that we mainly use.

In the original paper, the authors use the polynomial **learning rate** schedule with a factor of 0.9, which increases the learning rate from 3×10^{-5} to 1×10^{-4} in the first half of training and then decreases it from 1×10^{-4} to 3×10^{-5} in the second half. Accordingly, we employ a learning rate schedule where the learning rate first increases and then decreases. We use a standard PyTorch implementation of the 1cycle learning rate policy with a peak learning rate of 1×10^{-4}.

3.3. Blender Add-on

In this section, we describe the implementation of our Blender add-on, the design decisions we have made, and some adjustments that the add-on does to improve the visual appearance of the final result.

Blender[6] is a powerful open-source software for 3D graphics released under the GNU General Public License (GPL). It supports a wide range of graphics-related tasks, including modeling, still image rendering, and animation creation. Blender is a cross-platform application that can be run on Linux, Windows, and Mac computers. Although Blender is mostly developed in C++, it also provides a Python API, allowing add-ons to be developed.

3.3.1. Model Deployment

Both our HuggingFace models, GLPN and OneFormer, are implemented in PyTorch. While trained models can run in a native PyTorch environment, using it for production has some disadvantages.

Firstly, users of our add-on would need to download the large PyTorch Python module, which can take up several gigabytes of disk space. On a testing Windows machine, the installed PyTorch occupies about 1 GB, and the size increases to 4 GB for the GPU version with CUDA support. Another disadvantage is that PyTorch natively uses eager execution mode, which is convenient for developing models, but it is slower compared to a graph mode, where a computational graph of all operations is constructed before execution, allowing for powerful optimizations.

To solve some of these issues, PyTorch offers TorchScript, a statically typed subset of the Python language that's better suited for deployment. Models implemented in PyTorch can be converted into a computational graph in TorchScript format using the `torch.jit.trace()` or `torch.jit.script()` methods. The TorchScript graph representation can be compiled just before execution and run using PyTorch JIT, which further optimizes models using runtime information. There are also ahead-of-time compilers for TorchScript, such as the TensorRT compiler for NVIDIA GPUs.

Machine learning models can also be deployed in Open Neural Network Exchange (ONNX) format. ONNX was created as a format for interoperability between different frameworks. Both PyTorch and TensorFlow offer methods for converting models to the ONNX format. PyTorch models are converted using the `torch.onnx.export()` method.

Multiple runtimes exist for models in ONNX; the most used one is `onnxruntime`, maintained by Microsoft. It enables models to run on Windows, Linux, and Mac, as well as in a web browser or on mobile devices. With all its dependencies, `onnxruntime` only requires around 600 MB for the GPU version and 150 MB for the CPU-only version. `onnxruntime` also provides options for optimizing models, including quantization which reduces computation precision, making models smaller and faster.

We have chosen to use ONNX in our add-on because it is easy to convert models into this format and allows us to experiment with optimizing inference speed in the future. Using ONNX also enables us to decouple the add-on code from PyTorch, so if a better model becomes available, we can simply convert it to ONNX even if it is implemented in TensorFlow. Then the new model could be used in the add-on without needing to refactor the code or requiring users to install another runtime.

3.3.2. Creating a Diorama

The workflow of the add-on begins with the user selecting an input image. Depth and segmentation models in the ONNX format are then used to perform inferences. An example of the input image, along with the depth map and the panoptic segmentation mask generated inside the add-on, is shown in Figure 5.

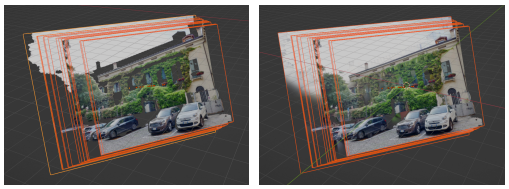
(a) Input image. (b) Segmentation mask. (c) Depth map.

Figure 5: Input image with inferences of deep learning models inside the add-on.

The input image is cut along the segment borders,

[6] https://www.blender.org/

resulting in a set of images where each contains only one object from the panoptic map, and the rest of the pixels are transparent. A 2D plane is spawned in a Blender scene for each segmented image, and the planes are textured with the segmented images. The planes are positioned behind each other, based on the average depth of their segments, and scaled to match the camera's perspective. The more distant planes appear larger, creating a sense of depth, as shown in Figure 6a.



(a) No inpainting. (b) Inpainted holes from the foreground objects.

Figure 6: Comparison of a diorama with and without inpainting.

3.3.3. Cutout Inpainting

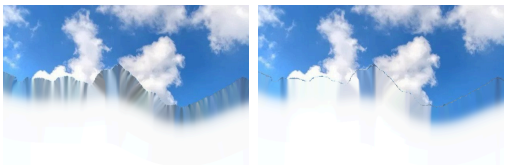
Figure 6a shows that the basic diorama created as described above has artifacts that disrupt the depth perception. The most noticeable artifacts are the holes from foreground objects when the diorama is viewed from an angle. We address this issue with inpainting, which fills in missing parts of the image based on existing parts. We experiment in the add-on with multiple inpainting methods; however, the best results are usually achieved with the inpainting algorithm available in Blender's compositor.

Blender's inpainting algorithm starts at an image edge and gradually spreads the color of the edge to the more distant pixels. However, inpainting performed straight from the edge is prone to artifacts, as shown in Figure 7 where dark pixels from a segmented mountain are inpainted into the sky. Especially at complex boundaries, it happens very often that edges contain pixels from the occluding objects. To avoid these artifacts, we apply erosion to remove pixels from the borders of segments before computing the inpainting. In our add-on, the width of the removed edges is empirically set to 3 pixels. Figure 8 shows a comparison between inpainting with and without erosion, while an example of the final diorama with inpainting applied is shown in Figure 6b.

3.3.4. Depth of the Sky

Another issue that arises in our diorama creation is the incorrect depth estimation of the sky. We can see in Figure 5c that the sky is estimated to be closer than the

Figure 7: Artifacts of inpainting without erosion.



(a) Inpainting without erosion. (b) Inpainting applied after 3-pixel erosion.

Figure 8: Comparison of inpainting with and without erosion.

building as it has darker values in the depth map. This happens because the distance of the sky cannot be learned from real-world datasets. The distance of the sky cannot be measured, and it would effectively need to be infinity compared to other distances in the image. The DIODE dataset [18], which we use for fine-tuning, is not an exception, and it contains masks indicating invalid depth values for the sky in ground-truth maps.

We use a segmentation model already available in our add-on to address this issue. The panoptic mask contains the classified pixels of the sky if it is present in the input image. We then move the plane with the sky segment behind all other segments to create a more realistic representation of the scene.

4. Results and Discussion

In this section, we discuss the performance of our fine-tuned GLPN model and compare its results with the original model trained on the NYUv2 dataset [8]. We also analyze the dioramas created by our solution and discuss its strengths and weaknesses.

4.1. Results of Fine-tuning

We use the official validation split of the DIODE dataset [18], which contains both indoor and outdoor scenes, to compare the two depth models. A challenge with the comparison is that the original pre-trained version of GLPN outputs depth values between 0 and 10 meters, while DIODE's measured depth values go up to 350 meters. To select the right method for the comparison, we hypothesize that the trained GLPN model has

		↓RMSE	↓AbsRel	↓SILog$_{1.0}$	↑ δ_1	↑ δ_2	↑ δ_3
Indoor	NYUv2	2.828	**0.383**	0.189	0.287	0.564	0.703
	Ours	**1.951**	0.403	**0.153**	**0.332**	**0.685**	**0.880**
Outdoor	NYUv2	14.909	0.726	0.339	0.005	0.026	0.099
	Ours	**7.993**	**0.462**	**0.245**	**0.341**	**0.662**	**0.897**
All	NYUv2	9.714	0.579	0.275	0.126	0.257	0.359
	Ours	**5.395**	**0.437**	**0.205**	**0.337**	**0.672**	**0.890**

Table 2
Comparison of GLPN trained on NYUv2 and our fine-tuned version, metrics are computed on the official validation split of the DIODE dataset and ↓ means smaller is better while ↑ means the opposite. From left to right the metrics are root mean square error (RMSE), absolute difference scaled by ground-truth values (AbsRel), scale-invariant log scale loss with $\lambda = 1$ (SILog$_{1.0}$) and percentage of predictions relative to ground truths within threshold $\delta_i = \max(\frac{d}{d^*}, \frac{d^*}{d}) < 1.25^i$ for $i \in \{1, 2, 3\}$.

either a dominant notion of absolute depth or a dominant notion of relative distances. If the dominant notion is absolute depth, the model would correctly estimate the depth of objects closer than 10 meters and return the maximum value for everything further away. On the other hand, if the dominant notion is relative distances, the model would estimate which objects are closer than others correctly, even in outdoor scenes. We can see in Figure 9 that the second option is prevalent. For example, the outdoor scene in Figure 9b shows a building that is further than 10 meters and still not estimated as the maximal depth. Moreover, Figure 9h shows a street nearly a hundred meters long, and the NYUv2 model is able to estimate the relative relations of objects correctly, as well as the direction of the depth gradient on the street, only that it estimated a big depth change in the first few meters and then a smaller change further away.

Based on this observation, we suggest comparing the models using the scale-invariant log scale metric with $\lambda = 1.0$ (SILog$_{1.0}$) [19]. It is invariant to the scale of the predicted and ground truth depth maps which allows for fair comparison of models trained on datasets with different ranges. This is in contrast to training where we used SILog with $\lambda = 0.5$ to jointly learn relative depth relations together with absolute depth values.

Table 2 provides results for several metrics, including SILog$_{1.0}$. We can see that our fine-tuning significantly improved SILog$_{1.0}$ on outdoor scenes. This result is also supported by the observations from Figure 9. The ground-truth map in Figure 9a shows that the right part of the building is further away than the left part; however, the NUYv2-trained model estimates the whole building as approximately the same distance in Figure 9b while our fine-tuned model estimates it more correctly in Figure 9c. On the other hand, SILog$_{1.0}$ on indoor scenes improved by a smaller margin which supports the selection of this metric for comparison since we know that the original model was already well-trained for indoor scenes. Again, the indoor scene in Figure 9 supports the similarity of indoor SILog$_{1.0}$ metrics by showing that both models estimate the depth similarly.

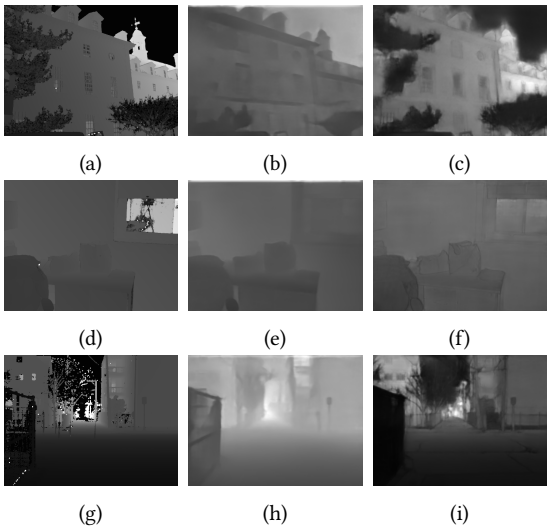


Figure 9: Comparison of depth maps estimated with the original GLPN model trained on NYUv2 (b, e, h) and our version fine-tuned on DIODE (c, f, i). Input images and ground truth values (a, d, g) are from the validation split of DIODE [18].

To assess if improvement in SILog$_{1.0}$ metric from Table 2 is statistically significant, we computed the Wilcoxon signed-rank test between the metric values obtained from the original and fine-tuned models. The p-values were computed separately for indoor and outdoor scenes as well as for the indoor and outdoor scenes together. The tests on all three sets confirmed that the improvement in SILog$_{1.0}$ achieved through our fine-tuning is statistically significant on the level of $\alpha = 0.001$.

Moreover, we can see from Table 2 that other metrics also improved with fine-tuning, but comparing them is not fair due to the different output scales. Interestingly, the original model achieved better absolute relative difference (AbsRel) on indoor scenes. We hypothesize that

this indicates that the original model works slightly better on near objects, as AbsRel penalizes the errors in depth estimation more for close objects by dividing the estimation difference by the ground-truth values. Since more than 95% of values in the training split's indoor part are smaller than 10 meters (as shown in Table 1), errors from not estimating distances beyond 10 meters are not significant.

4.2. Evaluation of Dioramas

Comparing dioramas presents a challenge due to the absence of a definitive ground truth. Therefore, in this section, we focus on outlining the strengths and weaknesses of our approach.

Our solution for creating dioramas works decently for various types of scenes. Particularly, the effect is enhanced if there are multiple objects in the foreground that can pop up from the background. We have also identified that it works well for outdoor scenes with a clear depth separation between objects, as shown in Figures 10b and 10d.

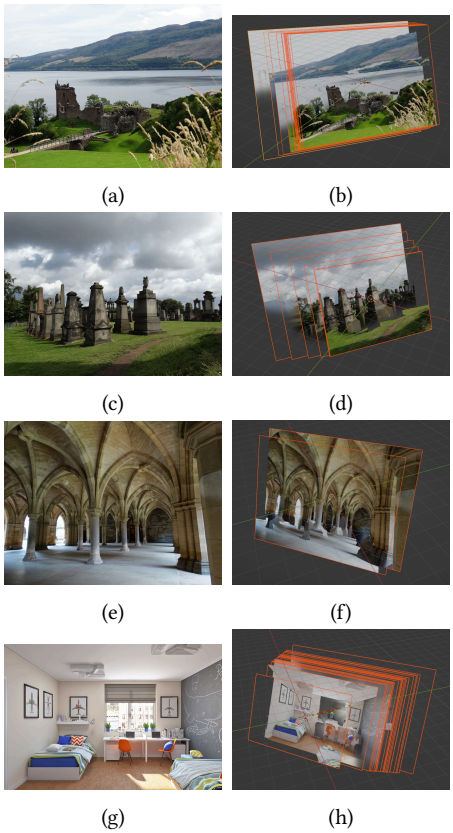


Figure 10: Examples of created dioramas.

However, our solution has some limitations related to the used deep learning models or to the method of constructing the diorama. Firstly, an object with varying depth, such as a wall that is partially in the foreground and partially in the background, has to be placed in a single depth-plane in our diorama, which can result in an incorrect position relative to other objects in the scene, as shown in Figure 10h. The rear wall is segmented together with the side walls and therefore placed incorrectly in front of the chairs and tables. This is a general limitation of all plane-based dioramas. In some cases, rotating the planes in the diorama according to the depth gradient of each segment could improve the issue. However, this would not always help, as with the walls of the room in our example.

To improve the diorama from Figure 10h, we would need first to split the walls into separate segments. This brings us to the next issue related to the definition of the panoptic segmentation task. Models are not trained to segment instances of objects from the `stuff` category, such as walls, roads, or vegetation, so all instances from one category end up in the same plane in our diorama. Figure 10f shows an example of this issue, depicting an ancient tomb where our segmentation model marks almost everything as a single wall object.

Lastly, our solution struggles with segmentation of objects with complex shapes, as seen in Figure 11, where tree branches without leaves are segmented together with the surrounding sky pixels. This limitation is due to the resolution of the used panoptic model and thus, it may be improved with better models in the future.

Figure 11: Imprecise segmentation of detailed object.

Despite these limitations, our solution is usable in many cases, and we believe it can be already helpful in a graphics workflow. Compared to previous works described in Section 2, our solution is less restrictive in terms of the general input image requirements. For example, dioramas created by Assa and Wolf [1] work mostly on outdoor scenes without regular patterns and straight lines. PEEP [3] is limited to images with zero or one vanishing point as it is fitting frustums to the images, and Zhao et al. [4] restrict their solution to hazy images only.

5. Conclusion and Future Work

We started this paper by providing an overview of existing methods for automatic diorama construction and identifying their limitations. Then, we decided to use recent deep-learning models to overcome some of those limitations. We reviewed the fundamentals of current deep learning models for monocular depth estimation and panoptic segmentation and selected a suitable, well-developed framework with transformer-based models. We chose a state-of-the-art model for panoptic segmentation and a competitive model for depth estimation which we fine-tuned to improve performance on outdoor scenes. Furthermore, we investigated ways of deploying deep learning models and we selected the ONNX format as the most suitable for future updates.

Even though the resulting add-on has some limitations, using deep learning to create dioramas is a promising approach. Overall, we believe our implementation is already a useful tool for creating dioramas in Blender, and we expect to continuously improve it.

One option for future work is to focus on improving all the small adjustments made to the cutout images, as the visual quality of dioramas depends on them significantly. For example, the current inpainting method simply spreads the color of the edge pixels into the holes. Our algorithm would benefit from a more advanced inpainting algorithm, such as one based on deep learning.

While we showed in this paper that separate depth and segmentation models can be used for generating dioramas. There is still an open question for future research if deep learning can be used for an end-to-end solution.

The quality of dioramas is closely linked to how users perceive 3D information from it, which is inherently subjective. Thus, conducting a user study to compare our method with prior research would be beneficial.

Acknowledgments

The work was supported by grant number SVV-202-09/260699.

References

[1] J. Assa, L. Wolf, Diorama construction from a single image, Computer Graphics Forum 26 (2007) 599 – 608.

[2] A. Saxena, M. Sun, A. Y. Ng, Make3d: Learning 3d scene structure from a single still image, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (2009) 824–840.

[3] M. Agus, A. J. Villanueva, G. Pintore, E. Gobbetti, PEEP: Perceptually Enhanced Exploration of Pictures, in: M. Hullin, M. Stamminger, T. Weinkauf (Eds.), Vision, Modeling & Visualization, The Eurographics Association, 2016.

[4] L. Zhao, M. Hansard, A. Cavallaro, Pop-up modelling of hazy scenes, in: V. Murino, E. Puppo (Eds.), Image Analysis and Processing — ICIAP 2015, Springer International Publishing, Cham, 2015, pp. 306–318.

[5] A. Kirillov, K. He, R. Girshick, C. Rother, P. Dollar, Panoptic segmentation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[6] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, D. Terzopoulos, Image segmentation using deep learning: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 44 (2022) 3523–3542. doi:10.1109/TPAMI.2021.3059968.

[7] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, A. Torralba, Scene parsing through ade20k dataset, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[8] N. Silberman, D. Hoiem, P. Kohli, R. Fergus, Indoor segmentation and support inference from rgbd images, in: A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, C. Schmid (Eds.), Computer Vision – ECCV 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 746–760.

[9] J. Jain, J. Li, M. Chiu, A. Hassani, N. Orlov, H. Shi, Oneformer: One transformer to rule universal image segmentation, CoRR abs/2211.06220 (2022). arXiv:2211.06220.

[10] D. Kim, W. Ga, P. Ahn, D. Joo, S. Chun, J. Kim, Global-local path networks for monocular depth estimation with vertical cutdepth, CoRR abs/2201.07436 (2022). arXiv:2201.07436.

[11] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, R. Girdhar, Masked-attention mask transformer for universal image segmentation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 1290–1299.

[12] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, B. Guo, Swin transformer: Hierarchical vision transformer using shifted windows, 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (2021) 9992–10002.

[13] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, J. Dai, Deformable DETR: deformable transformers for end-to-end object detection, CoRR abs/2010.04159 (2020). arXiv:2010.04159.

[14] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, The cityscapes dataset for semantic urban scene understanding, in: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[15] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft coco: Common objects in context, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), Computer Vision – ECCV 2014, Springer International Publishing, Cham, 2014, pp. 740–755.

[16] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, P. Luo, Segformer: Simple and efficient design for semantic segmentation with transformers, in: Neural Information Processing Systems (NeurIPS), 2021.

[17] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, Vision meets robotics: The kitti dataset, International Journal of Robotics Research (IJRR) (2013).

[18] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter, G. Shakhnarovich, DIODE: A Dense Indoor and Outdoor DEpth Dataset, CoRR abs/1908.00463 (2019).

[19] D. Eigen, C. Puhrsch, R. Fergus, Depth map prediction from a single image using a multi-scale deep network, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, volume 27, Curran Associates, Inc., 2014.