Chair of Remote Sensing Technology
TUM Department of Aerospace and Geodesy
Technical University of Munich

# Reconstructing Development Mules

**Czychon Lukas**[1,✉], **Körner Marco**[1]
[1]Department of Aerospace and Geodesy, Technical University of Munich (TUM), Arcisstr. 21, 80333 Munich, Germany
✉ lukas.czychon@tum.de
March 31, 2021

**Abstract** — This project tries to synthesize color and depth images with Blender that have a similarity with pictures of development mules. Therefore, a process gets defined and implemented in Python with the Blender Python API. Additionally, the generated images get tested on conventional approaches such like Agisoft's Metashape and Meshroom implement it, as well as on SGDepth, a deep monocular approach.

## 1 Motivation

When car manufacturers develop a new car, they do not always want everybody to know the design of it beforehand. Unfortunately, not all the tests for such a new design can be performed with in labs or the company's perimeter, therefore they have to test them in the *wild*, e.g. in a city. To prevent onlookers and sophisticated computer vision algorithms from deriving new geometry features, they use special textures to confuse. Those tactics and textures are not new and have already been employed in the first and second world war. There, they were called "*Dazzle camouflage*" and used by the U.S. Marine to obfuscate the direction a ship travels.

## 2 Introduction

This project *"Reconstructing Development Mules"* is part of the module *"Project Photogrammetry and Remote Sensing"*. Its aim is the reconstruction of development mules, by training a deep learning algorithm with in Blender synthesized images. To accomplish this goal, the project was split up into the following tasks:

1. training image syntheses
2. research depth estimation approaches
3. test these approaches with the generated images
4. further train & evaluate approaches

Section 3 and section 4 provide information on the realization of the first task, while section 5 and section 6 deal with the later task respectively.

## 3 Theory

In order to generate training and testing images, a concept as seen in Figure 1 was developed and later in section 4 implemented. The main idea behind this workflow was the breakdown of a typical scene in which such a development mule can usually be seen. To reduce the amount of work need to create a believable scene, the complexity was reduced by only focusing on the car and large objects (buildings) in its vicinity. This generation should first start with the placement of the main object, the vehicle. In a next step this model gets a material applied, depending on the later use of the image. Here, a camouflage/dazzle pattern or a single color could be used. Following this, the scenery gets placed around the car model in the center. As a last step, before the rendering, the camera gets positioned in a way that it has an unobstructed view of the vehicle. Finally, two images, one colored and a depth image get rendered.

## 4 Implementation

Blender was chosen as some pre-knowledge on its usage existed, and it provided a scripting interface. The aforementioned image syntheses workflow is implemented with Python API in Blender as described in subsection 4.3. But before that can happen, we first have to prepare the used Assets (subsection 4.1) and arrange a scene/`.blend`-file (subsection 4.2).

### 4.1 Asset Preparation

Since most of the models used are not modeled by myself, but rather come from 3D model sharing websites like blendswap. These models were usually prepared by there original creators for rendering

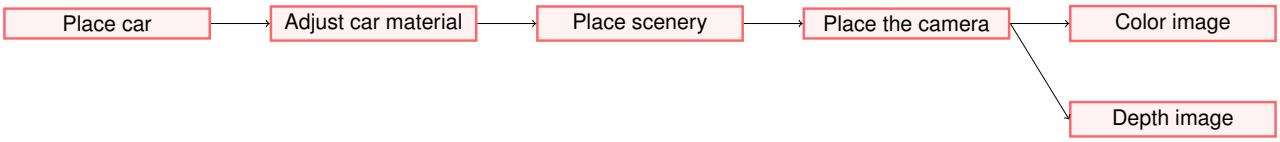



**Figure 1** Training Image Synthesis Workflow

and therefor the individual `.blend`-files contain several other objects that are not needed. Further, the model has to be prepared by the following steps, to be a usable asset:

1. Open the `.blend`-file with Blender that contains the model that you want to convert to an asset.
2. Only select the parts of the model that you want to use for the asset and place them into a new collection.
3. To work around the bug, seen in Figure 2, an object is only allowed to have a single material. To achieve this multiple materials can be joined in to a single one, by baking them into a texture.[1]
4. Set the origin to the center of mass and then move its $z$ position to $0$. So that when creating an instance and setting all its coordinates to $0$, it's above the ground.
5. Add a custom property with the name `inst\_id` and an appropriate value for its type in the *Object Data Properties* tab in the *Properties* panel, to be able to later derive an instance image.
6. Mark the collection as an *Asset Library* in the context menu by right-clicking on the collection.

**Figure 2** Example of texture mesh up caused by a bug in `bpycv` (`https://github.com/DIYer22/bpycv/issues/19`)

Figure 3, shows results of the three assets *BMW 3 Series Coupe*, *Brick apartments* and *VW T1 Van* that where achieved with the aforementioned preparation process.

**(a)** BMW 3 Series Coupe[5] **(b)** Brick apartments[2] **(c)** VW T1 Van[3]

**Figure 3** Some prepared assets

## 4.2 Blender Scene

The previously prepared assets (subsection 4.1 – Asset Preparation) now have to be placed in the following collection structure, so that the script in the next section (subsection 4.3 – Blender Automation):

```
. Scene Collection
|- Assets
|   |- Vehicles
|   |   |- BMW 3 Series Coupe
|   |   |- VW T1 Van
|   |   |- ...
|   |- Buildings
|   |   |- Brick Apartments
|   |   |- Citrohan House
|   |   |- ...
|- ...
```

**Figure 4** Collcetion structure allowing programmatic access to prepared assets

## 4.3 Blender Automation

Blender allows automation over a Python API, this interface gets used to implement the process motioned in section 3. This process is made up of five steps that are executed sequentially. The first step *Place car* and second step *Adjust car material* are

[1] Video tutorial by Grant Abbitt on texture baking in Blender: `https://www.youtube.com/watch?v=9airvjDaVh4`

performed by creating an instance of one of the vehicles models found in the *Vehicles*-collection that was prepared in subsection 4.2 and then manipulating the node structure of the material. The selection and manipulation is influenced by a seed for a random number generator that later allows the regeneration of the same scene. Following, is step three *Place scenery*, here a random number between five and ten buildings get placed. Each of the buildings gets randomly chosen from the *Buildings*-collection and randomly placed in a radius form $-95\,\mathrm{m}$ to $-35\,\mathrm{m}$ and $35\,\mathrm{m}$ to $95\,\mathrm{m}$ from the origin of the scene where the car is placed. As a penultimate step (*Place the camera*), the camera gets randomly positioned in a radius form $-25\,\mathrm{m}$ to $-10\,\mathrm{m}$ and from $10\,\mathrm{m}$ to $25\,\mathrm{m}$ from the origin of the scene. A `Track` constraint is then applied to the camera, so that it always looks at the vehicle in the center of the scene. The placements the different radii are illustrated in Figure 5 and have the advantage to allow neglecting occlusion test and the following camera repositions. Finally, the last step *Render color and depth image* gets done with the help of the `bpycv` python package that works together with Python API and renders additionally an instance and a depth image.

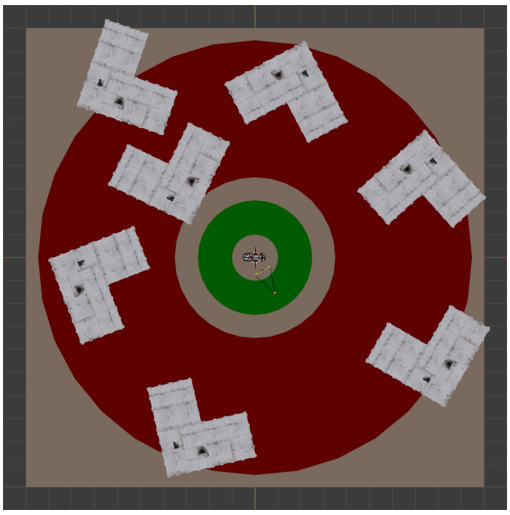
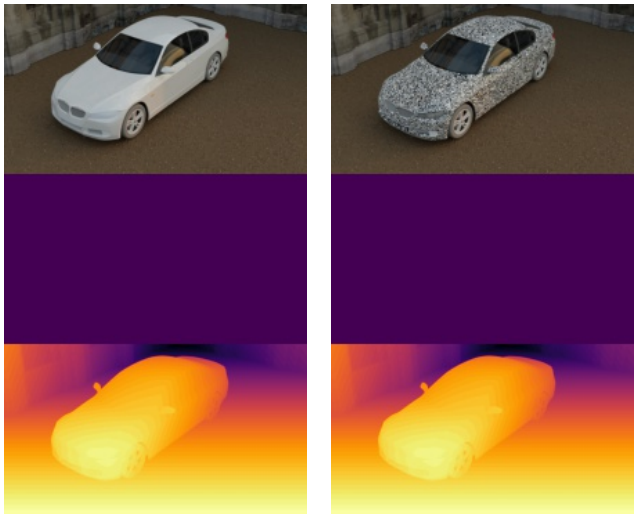


**Figure 5** Radii of placements

The resulting script was then enhanced with `argparse` to use command line arguments to either provide file and directory locations or arguments, such as the number of images to generate or a starting seed for regeneration. It could then be executed by calling it from the command line like `blender Blends/better-synthesis.blend -b -P Scripts/synthesize\_images.py -- -o Renders/test/second/ -n 3`. Arguments before `--` are dedicated for the Blender application, while arguments after it are consumed by the script. Figure 6 shows common result of the generated images. The one on the left Figure 6a uses a single color material, while the one on the right Figure 6b used a random camouflage material.



**(a)** Single color material **(b)** Camouflage material

**Figure 6** Example of generated images

## 5 Experimental Setup

The experiment conducted, tests the usefulness of generated images by using the images as input data into several depth estimation approaches. First a simple scene is constructed with a view scattered buildings and a vehicle. Then the camera gets animated on a path around the vehicle with keyframes. This scene then gets rendered several times, where each time the material of the car gets changed. Figure 7 shows the four materials used.

**(a)** Single color

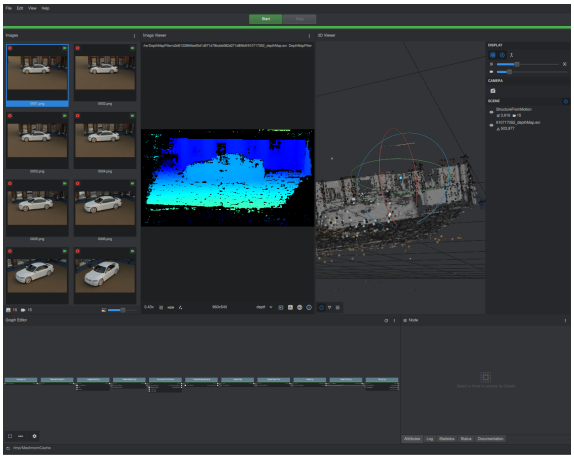
**(b)** Dazzle texture

**(c)** Checker texture

**(d)** Random noise texture (camouflage)

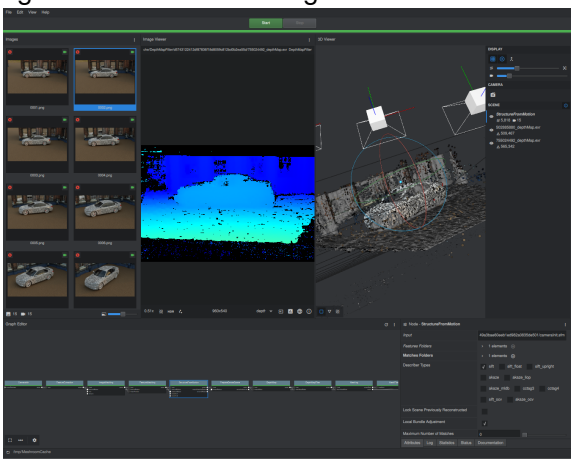
**Figure 7** Vehicle with different tested materials

The generated color images are then feed into the depth estimation approaches. Then the resulting depth images are compared with the generated ones to evaluate their usefulness.



**(a)** Single color material

fig:results:meshroom:single



**(b)** Camouflage material

**Figure 8** Two results from Meshroom

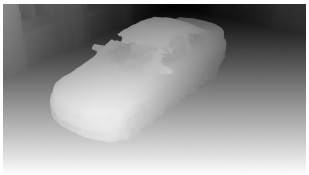
# 6 Qualitative Results

The previously stated experiment (section 5) gets performed on three different estimation approaches. The first targeted a conventional photogrammetry pipeline, while the other two targeted a deep monocular and stereo estimation approaches respectively.

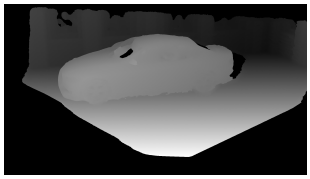
Additionally, to Meshroom, a subset of images (*single color* and *camouflage*) were test with Agisoft's Metashape, a commercial product. Due to a missing license and expertise these test were performed by Stephan Götzer, a fellow student.

## 6.1 Conventional approach

Meshroom, an open source solution based on the Alicevision, a photogrammetric computer vision framework was used. Here at least four images were needed to produce depth images for each. Figure 8 shows the result from using 15 images for each run. It should be noted that the application was able to handle the images with the camouflage material, a random noise texture a lot better and more accurate as compared to the single color material.



**(a)** Depth image calculated from multiple images with the camouflage material



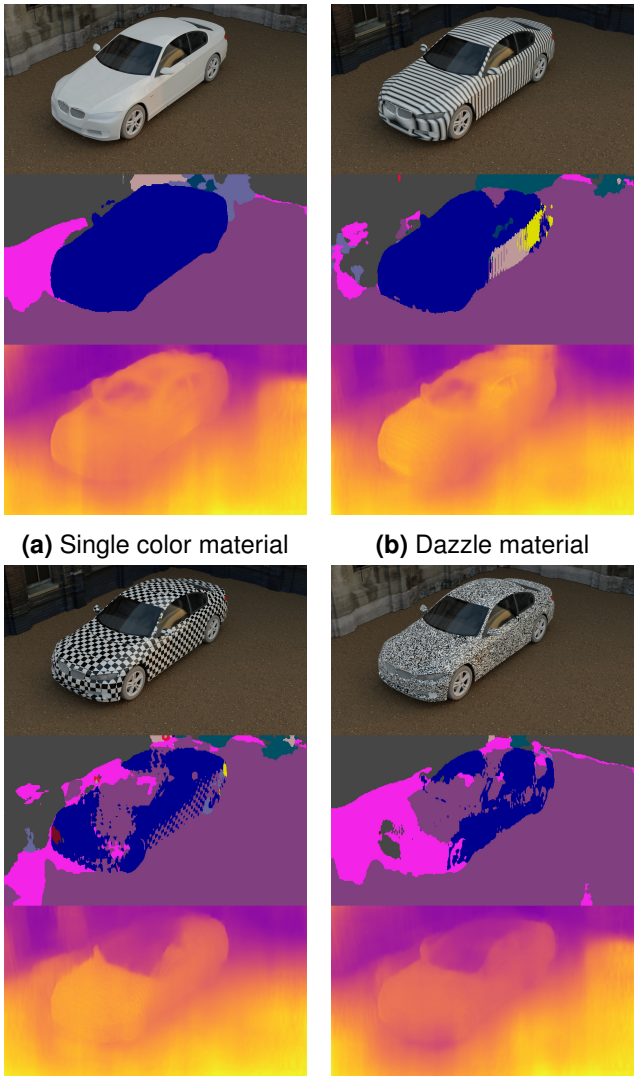
**(b)** Reconstructed 3D model

**Figure 9** Results of the conventional approach in Agisoft's Metashape

### 6.2 Deep monocular approach - SGDepth

*Self-Supervised Semantically-Guided Depth Estimation* (SGDepth) tries to solve the problem imposed by dynamic objects in self-supervised depth estimation by semantic guidance. Both tasks the semantic segmentation and the depth estimation get trained simultaneously. It was imagined by Klingner et al. in “Self-Supervised Monocular Depth Estimation: Solving the Dynamic Object Problem by Semantic Guidance”.

Comparing the results in Figure 10 from the four different scenarios, it can be noted that in both of the dazzle material and the checker material scenario artifacts of the material can be seen in the segmentation result. Both the checker material and the camouflage material are leading to the most errors in segmentation. Solely for the single color material is SGDepth not having troubles. The depth estimation results for the single color material and the dazzle material are better compared to the results of the other two, were parts of the roof couldn’t be correctly estimated.



**(a)** Single color material **(b)** Dazzle material

**(c)** Checker material **(d)** Camouflage material

**Figure 10** Results of the deep monocular approach SGDepth

### 6.3 Deep stereo approach – Highres

Next to the deep monocular estimation approach, a deep stereo estimation approach was tested as well. Unfortunately, the results were unusable due to missing and miss-configured camera calibration files.

## 7 Conclusions

This project developed and investigated an image generation process, that is able to produce development mules lookalikes and their depth images, it further tested these images by handing them to deep and conventional photogrammetry approaches.

The qualitative results were able to show that the generated images with materials similar to the ones used by car manufactures indeed have an effect on segmentation and depth estimation approaches. Here the test with the deep stereo approach should be investigated and repeated with better inputs. Further, the image generation process can now be used to improve the deep depth estimation approaches, in order to accomplish that it has to be enhanced with a sequence generation as compared to the currently implemented single image generation.

In recent versions of Blender (2.9 and following) a *Asset Manager* and *Geometry-Nodes* became available, but were at the time of writing still in active development and only basic features were available for testing.[1] Both of the features proved to be useful in the future, but currently lacked a good documentation and an integration in the Python API. The *Asset Manager* could provide the option to use more and different models, since integration with asset folders would be easier. But there would also be the disadvantage that the `.blend`-file is not self-contained anymore and multiple files have to be maintained. The *Geometry-Nodes* yields even more benefits for the project, it enhances Blender with an easy to define procedural generation workflow similar to SideFX’s Houdini. This allows in the future to

procedurally generate more believable and complex scenarios without manual intervention.
The source code is published on GitHub `https://github.com/codepuree/VPF-Reconstructing-developmpment-mules` that should allow the further development the project.

## Acknowledgments

Firstly I would like to thank my supervisor Mr. Körner, for his continued effort in helping me, staying on track with frequent bi-weekly meetings. There he not only helped me to formulate the research question, but also provided additional insights with keywords for the literature research.
Secondly, I also want to tell my gratitude towards Stephan Götzer, a fellow student who helped me with testing some of my generated images with the Agisoft's Metashape software suite.

## References

[1] Blender Developers, K. Joanna, R. James, CrossMind Studio, B. Carlo, S. Francesco, F. Dalai, B. Christian, Blender Studio team, and SouthernShotty. [Online; accessed February 25, 2021]. Feb. 2021. URL: `https://www.blender.org/download/releases/2-92/`.

[2] Evolix777. *Brick appartments*. [Online; accessed January 12, 2021]. Mar. 2020. URL: `https://blendswap.com/blend/24616`.

[3] Jay-Artist. *1950's VW Van Cycles*. [Online; accessed January 12, 2021]. May 2012. URL: `https://blendswap.com/blend/5358`.

[4] M. Klingner, J.-A. Termöhlen, J. Mikolajczyk, and T. Fingscheidt. "Self-Supervised Monocular Depth Estimation: Solving the Dynamic Object Problem by Semantic Guidance". In: 2020.

[5] mikepan. *BMW 3 Series Coupe*. [Online; accessed January 12, 2021]. May 2011. URL: `https://blendswap.com/blend/2832`.