2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

A Photorealistic Terrain Simulation Pipeline for Unstructured Outdoor Environments

M. G. Müller[1,2*], M. Durner[1], A. Gawel[2], W. Stürzl[1], R. Triebel[1], R. Siegwart[2]

***Abstract*— Suitable datasets are an integral part of robotics research, especially for training neural networks in robot perception. However, in many domains, suitable real-world data are scarce and cannot be easily obtained. This problem is especially prevalent for unstructured outdoor environments, in particular, planetary ones. Recent advances in photorealistic simulations help researchers to simulate close-to-real data in many domains. Yet, there exists no high-quality synthetic data for planetary exploration tasks. Also, existing simulators lack the fidelity required for generating planetary data, which is inherently less structured than human environments. Synthetic planetary data requires careful modeling and annotation of many different terrain aspect and details, such as textures and distributions of rocks, to become a valuable test-bed for robotics. To fill this gap, we present a novel simulator specifically designed for the needs of planetary robotics visual tasks, but also applicable for other outdoor environments. Our simulator is capable of generating large varieties of (planetary) outdoor scenes with rich generation of meta data, such as multi-level semantic and instance annotations. To demonstrate the wide applicability of this new simulator, we evaluate its performance on typical robotics applications, i.e. semantic segmentation, instance segmentation, and visual SLAM. Our simulator is accessible under `https://github.com/DLR-RM/oaisys`.**

I. INTRODUCTION

Modern robotic systems heavily rely on the availability of rich training data to achieve an advanced level of functionality for autonomous operation. This is particularly true when using neural networks for robotic vision tasks. While many general computer vision applications can leverage a large range of established datasets of the past decade [1]–[4], there are fewer examples in robotic research, the most prominent ones being autonomous driving and indoor service robotics [5], [6]. This is mostly attributed to the circumstance that manual labeling of data is time-consuming, especially where rich annotations are required, such as for pixel-wise scene segmentation. Furthermore, some applications even lack larger amounts of suitable and accessible raw data, e.g. planetary exploration, where data are limited to past successful missions [7].

Photorealistic simulations have received a considerable attraction over the recent years [8], [9]. Not only is it possible to produce vast amounts of labeled data, but also complete software-in-the loop simulations can be performed, including realistic physics [10]. Rapid advances are made in the major areas of robotics research [10]–[12]. However, since most current simulators focus on the applications of autonomous driving and indoor service robotics, they mainly provide

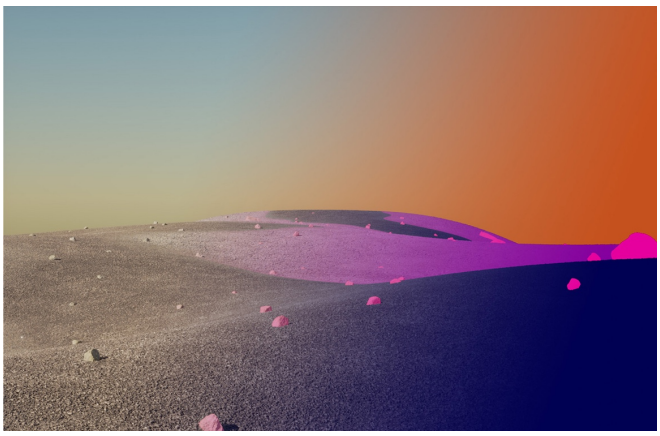

Fig. 1: Example image from the simulator OAISYS. Left, RGB image and on the right side the corresponding semantic image.

structured environments. Even though other landscapes can be used in these simulators, the environments have to be modeled manually. Creating a large variety of environments, as is needed for training, is very resource intensive and requires expert knowledge. Furthermore, the fidelity of current simulators is limited as they typically build on simulation engines not specifically designed for robotics research. Hence, while these simulators produce photorealistic data, they fall short in the creation of crucial meta-data, such as multi-level semantic and instance segmentation. One special case is the texturing of individual elements such as ground planes. Here, entire elements are often assigned a single label. Yet, especially in outdoor environments, a distinction within different regions of the surface may be required.

This paper addresses the specific use-case of planetary robotics, but is not restricted to it. Currently, there exists neither sufficient real data nor any suitable simulators. Planetary environments are characterized by different soil types and distributions of natural objects. Since these environments lack vegetation and human artifacts, which other simulators focus on, additional tools are required to realistically model the natural features of these environments. Nevertheless, we are interested in a simulator that combines existing and new features to be applicable to additional use cases.

Therefore, we present OAISYS (**O**utdoor **A**rtificial **I**ntelligent **SY**stems **S**imulator), a simulator for unstructured outdoor environments that specifically considers the requirements of planetary robotics. OAISYS is built upon the open source engine Blender [13], without requiring expert knowledge in rendering pipelines. Besides highest rendering qualities, key novelties of the simulator include a parametric creation of new environments and the generation of high-fidelity meta-data by providing multiple material channels instead of a single channel with object identification num-

[1]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Germany
[2]Autonomous Systems Lab, Swiss Federal Institute of Technology (ETH Zürich), Switzerland
*Corresponding author: `marcus.mueller@dlr.de`

bers (ObjectID channel). This enables merging of multiple textures to form surface elements while retaining individual annotations, and even assigning multiple labels to the same elements. Furthermore, a parametric set-up allows for a massive generation of novel and diverse data, including random deformation of the base mesh, mixing and merging of surface textures, scattering large amounts of objects, and providing many different lighting conditions.

To demonstrate the capabilities of the OAISYS, we evaluate it on three common use-cases in planetary robotics. First, we employ visual odometries used for navigation of (autonomous) mobile robots. Then, we consider semantic terrain segmentation, which is required for mobile robots to identify traversable regions. Finally, instance segmentation of stones is shown, which is important for robotic sampling.

Thus, our main contributions are:

- A simulator for unstructured outdoor environments
- A novel method to auto-generate planetary environments
- Multi-level semantic labels based on material instead of ObjectID
- A fast method to scatter objects on a base mesh
- An evaluation of the simulated data on three robotic tasks

II. Related Work

The last decade brought a variety of datasets for robotic vision tasks, specifically semantically annotated data [1], [4], [14]–[18]. These are further divided into indoor datasets [4], outdoor datasets in human environments [1], [19], datasets in unstructured outdoor environments [16]–[18], and combinations of it [15]. Most relevant for our work are datasets in unstructured environments similar to planetary bodies. Xue et al. [16] present a small-scale dataset, suitable for terrain classification with top-down views. Valada et al. [17] and Wigness et al. [18], on the other hand, provide larger amounts of data from varied perspectives. However, the recorded data contain mostly vegetation and landscaped environments, which do not much resemble planetary bodies. Meyer et al. [20] and Wagstaff et al. [7] present planetary(-like) robotic image data. However, these either contain no semantic annotations [20] or unsuitable classes for terrain classification [7].

Possibilities to overcome the gap in available data include the creation of synthesized image, e.g., by combining backgrounds and objects from different data [14], [21], and (photorealistic) simulations [10]–[12], [22]–[25]. While the fusion of different data is a promising avenue to generate relevant data from limited amounts of real data, current sets focus on urban driving scenarios [14], [21]. Hence, simulated data are an attractive choice to generate rich annotated data of arbitrary environments. Furthermore, Johnson-Roberson et al. [23] and Ros et al. [24] show that semantic segmentation models trained on these synthetic data can generalize well to real environments. However, like most real datasets, no current simulator features environments and fidelity needed for planetary robotics.

While generic simulators allow creation or inclusion of such environments [10]–[12] and also output of semantically annotated images, these simulators lack the granularity to easily produce rich meta-data, such as multi-layered semantic annotations and instance segmentations. In addition, the creation of a new and natural environment through the realistic distribution of objects is typically a manual process where individual objects and textures need to be placed in an environment. The simulator presented in [25] offers higher granularity. However, it focuses on indoor scenes, and does not offer added features for automatic planetary surface generation and multi-level labels. PANGU [26], [27] and SurRender [28] are simulators which can randomly generate environments and scatter rocks on them. However, they do not provide any semantic annotations.

The most popular simulation frameworks for visual data are based on game engines (Unreal [8] and Unity [9]) or 3D creation suites like Blender [13]. Game engines have the great advantage that they can render data in real-time, which also enables hardware-in-the-loop simulations. The advantage of 3D creation suites are integrated rendering engines that tend to produce outputs of higher visual quality. In contrast to game engines that typically allow for only a single semantic output (e.g., either semantic segmentation or instance segmentation), 3D creation suites enable multiple outputs.

Our approach is based on Blender [13], and overcomes the drawbacks of existing datasets and simulators with respect to planetary data. It enables generation of the highest quality planetary visual data with rich annotations. Additional modules further allow automatic generation of realistic object distributions, such as rocks, and parametric creation of ground surfaces and textures. Although designed for planetary tasks, it can also be used for any other unstructured outdoor environments and can be easily extended.

III. Outdoor Artificial Intelligent Sytems Simulator (OAISYS)

The basic idea of the simulator is to use a main mesh as the ground surface, which is called the stage. This stage is modified in geometry, texture, and arbitrarily added assets to create a large number of different environments. These can then be rendered from different perspectives and under different lighting conditions.

After loading the main mesh into the simulator, it is deformed by a modifier using random noise as deformation

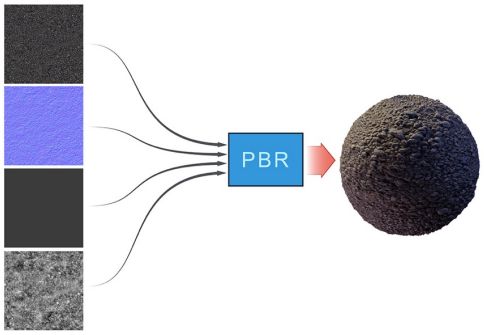

Fig. 2: Example of a PBR material. In this case, 4 texture channels (diffuse, normal, reflection, displacement) are used to create the material on the right side, which is applied to a sphere. Note that height displacement modifies the underlying geometry of the mesh.

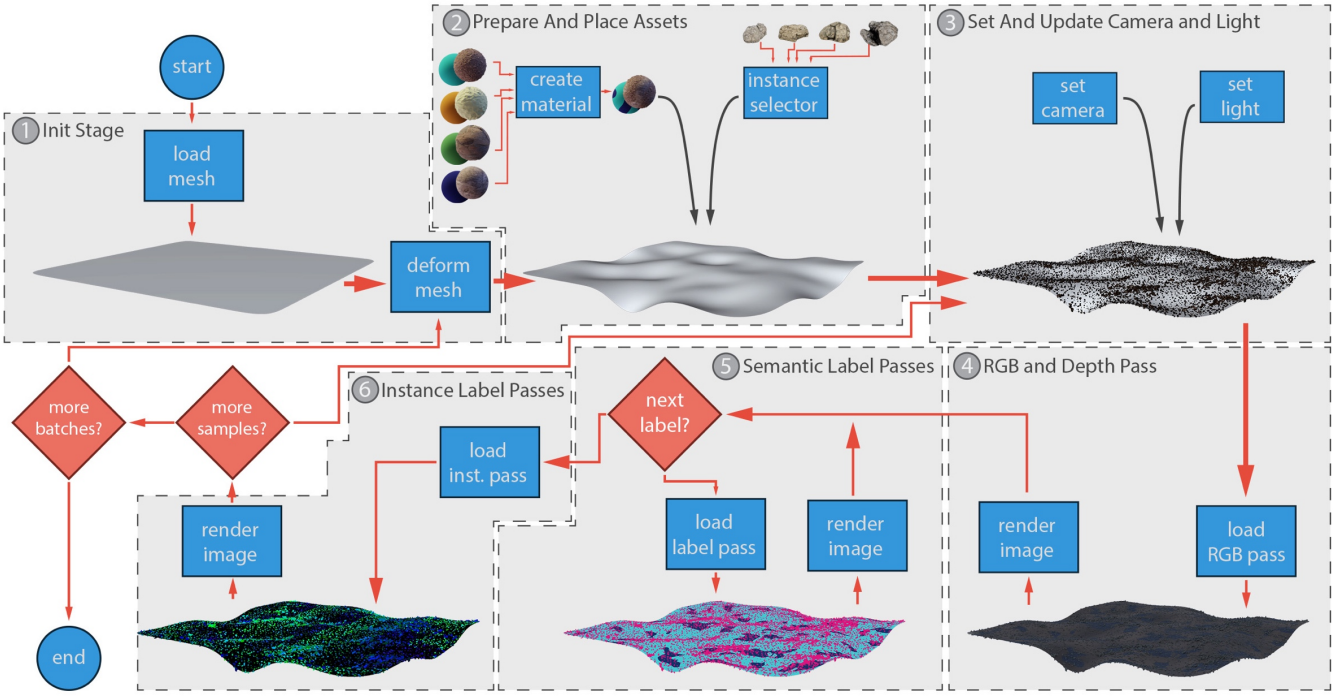

Fig. 3: Basic Flow Diagram of OAISYS. First, the main stage mesh is loaded and deformed ①. Afterwards, the terrain materials are created randomly and mesh assets are picked randomly ②. In ③, the camera and light are set up. In ④, the RGB pass of all assets is loaded and rendered. Afterwards, the semantic labels are loaded and rendered ⑤. This procedure is repeated on as many semantic levels as desired. In ⑥ the instance labels are loaded and rendered. Afterwards, either more samples of the sample asset configuration are processed or a new batch ends. The process is completed when the number of desired batches is reached.

parameters. With that, the simulator already provides auto-generated modifications for the displayed terrain. Afterwards, the simulator loads a list of terrain textures, which each belongs to one terrain class. To achieve photorealistic results, OAISYS uses Physically Based Rendering (PBR) textures [29], see Fig. 2. These textures are randomly merged via a noise shader to create mixtures of different terrains, see Fig. 4. The created materials are then applied to the stage. Since each texture has its own semantic label IDs, the semantic information is not lost when the textures are merged. For adding other objects on the stage, e.g. rocks, grass, or trees, OAISYS can rapidly scatter objects on the terrain surface. For the scattering process, different kinds of noise maps are available as asset density maps, like uniform noise or Worley noise [30]. For realistic light, the simulator can either use a list of High dynamic range (HDR) images, which are then randomly picked, or Blender's internal sky simulation, which creates skies based on the Nishita model [31]. Random camera angles are chosen to render the scene from different perspectives. Outputs of this simulation pipeline are RGB images, corresponding semantic segmentation images, instance images, and depth images. One can choose how many different terrain batches are rendered, and how many samples are taken from one batch. For every batch iteration, the values for the stage deformation and material, as well as the asset distribution on them, are adapted. Light settings and camera position can be changed after each sample. Fig. 3 gives an overview of the simulator. All of the parameters of the simulator are read in a single configuration (cfg) file. Listing 1 shows a section of an example cfg. We provide several example cfgs files and detailed explanation for each parameter with our simulation. The simulator can, in principle, be extended with custom modules. In the following sections, we present the different core modules of the simulation.

Listing 1: Example section of main simulation setup cfg. *numBatches* defines how many terrains are generated. The value of *numSamplesPerBatch* defines how many image samples are taken from one terrain. *numMixTerrains* defines how many terrain textures are maximally merged together. The flag *renderLabelsActive* defines if semantic labels are rendered and *renderLabelDepth* defines how any semantic levels. Activation flags are also available for depth *renderDepthActive*, and instance pass *renderInstanceLabelsActive*.

```
"simulationSetup": {
  "numBatches":100,
  "numSamplesPerBatch":30,
  "numMixTerrains":5,
  "renderLabelsActive":true,
  "renderLabelDepth":2,
  "renderInstanceLabelsActive":true,
  "renderDepthActive": true}
```

A. Stage Simulation Module

The stage is the fundamental object mesh of the simulation. It can be deformed, objects can be placed on it and its material can be changed. The mesh can be loaded from any Blender file. Although the stage can be any kind of mesh, the simulator was designed for planar meshes, which resemble most common base terrains. We provide a selection of such meshes with our simulator. When using the stage as the ground of an outdoor scene, one would like to apply randomizations to the structure of the flat plane.

Therefore, OAISYS gives the ability to deform the chosen mesh. For that, our simulator uses a noise texture, which values determine how the mesh is deformed. For the noise texture, we choose a multifractal noise, which comes with Blender and uses Perlin noise [32] as the base function and is used often for deformation tasks. However, in principle, any noise map could be used. One can choose the settings for the deformation procedure and its values. The parameters are adapted after every batch iteration. Since the noise can lead to harsh surface deformation, we apply a smoothing afterwards. Additionally, we apply an adaptive subsurface modifier, which subdivides the mesh further based on camera proximity to realize fine detail structures.

B. Asset Simulation Module

Once the basic stage is set up, assets can be applied to it. Assets are the objects and terrain textures in our simulation. The terrain textures are combined to a single terrain material, which is then applied to the stage. For each batch iteration, the material, with its textures, is swapped with a new one. Object meshes can be loaded from any other Blender file and placed on the stage or in a specific location anywhere in simulation space. When placing an object, one can place single object instances or apply a function to automatically add multiple copies of it in different locations, sizes, and orientations. The user can arbitrarily place many objects. In contrast to other simulators, this method enables us to simulate thousands of objects randomly placed on the surface. Each asset, texture and object can have multiple semantic labels. Hence, we can render multiple different semantic labels depending on the user's task. Additionally, each object can have its own instance label, which will be kept within one batch. Since it is kept over all frames of a batch, the simulator can also produce ground-truth data for object tracking tasks. In the following section, the two asset types, as well as how semantic information is set up with them, are explained in more depth.

1) Terrain Material: Terrain materials are configured via the cfg file from pure terrain textures, i.e. textures, which are attributed by a single semantic class for each semantic depth level. As an example of terrain classification, a texture might be dried mud, and another one gravel, see Fig. 4. The textures are then merged into one texture, based on a noise texture, and the parameters are randomly chosen and adapted in every batch iteration. For the merging process, OAISYS uses Perlin noise of value $\alpha \in [0, 1]$ for each pixel, as illustrated by ② in the illustration. The noise value determines the opacity value for both terrains, i.e. α is the opacity for texture t_1, and $1-\alpha$ for texture t_2. As a result, the newly created texture t_3, which consists of a set of the two pure textures, is given by

$$t_3 = \sigma \cdot t_1 + (1 - \sigma) \cdot t_2 \quad (1)$$

This new texture can then be merged with another pure texture, which results once again in a new texture. The maximum number of merging steps is a configurable parameter. The merging is done smoothly to create a realistic appearance and to avoid boundary artifacts, which, for instance, would appear when merging two textures with considerably different displacement maps. However, such a smoothing effect would corrupt the semantic labels. Therefore, a threshold is applied for the semantic channel to get clear borders. An example section of a terrain asset cfg is shown in Listing 2.

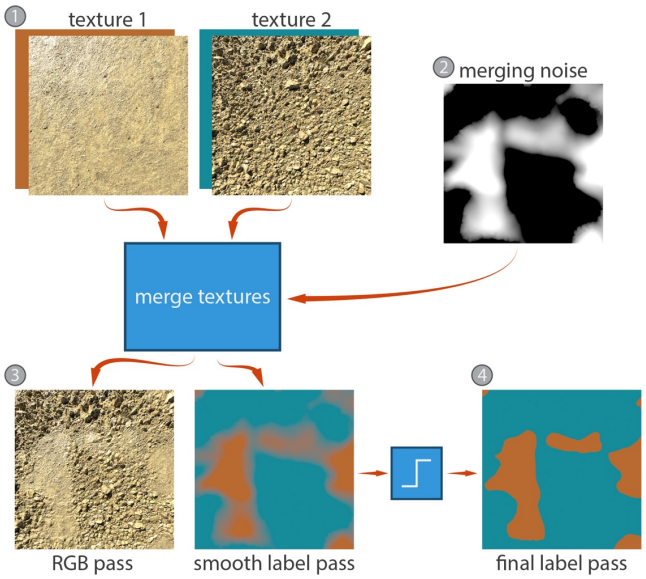

Fig. 4: Merging of two terrain textures. The basic textures with semantic label ① are merged according to the values of the merging noise ②. ③ depicts the result of the merging. In order to get sharp labels, the smoothed labels are processed with a threshold ④.

Listing 2: Example cfg for two terrain textures and some of their parameters. With *labelIDVec*, the semantic class for each semantic image level is defined. With *size*, one can adjust the texture size. The parameter *normalStrength* defines the strength of the displacment.

```
"terrains": [
{
  "path":"/path/to/texture1/folder/",
  "normalStrength":1.0,
  "labelIDVec": [10,20,5,2],
  "size": 25.0
},{
  "path":"/path/to/texture2/folder/",
  "normalStrength":0.5,
  "labelIDVec": [5,20,7,42],
  "size": 60.0}]
```

2) Individual Meshes: Object assets are meshes, which are defined in the form of Blender files. One can place single or multiple objects scattered on the stage. Single objects can be further defined by providing a csv-file. In case of multiple objects, a specified number of objects are placed on the surface of the stage. In order to increase variation of the objects, each copy varies randomly in size and orientation. To scatter a large amount of objects, we use the particle system supplied by Blender. Particle systems emit many copies of an object from an emitter object. Each emitted object can vary in a set of parameters, like size and orientation. In our case, the emitter is the stage mesh. We are using hair particles, which place the created copies of objects on the surface. With the hair particle system, it is also possible to enforce certain orientations, such as along the normal orientation of the emitter mesh, and to scatter objects with specific orientations, such as trees. Other methods use physics calculations to place the objects on a main plane. In that approach, the

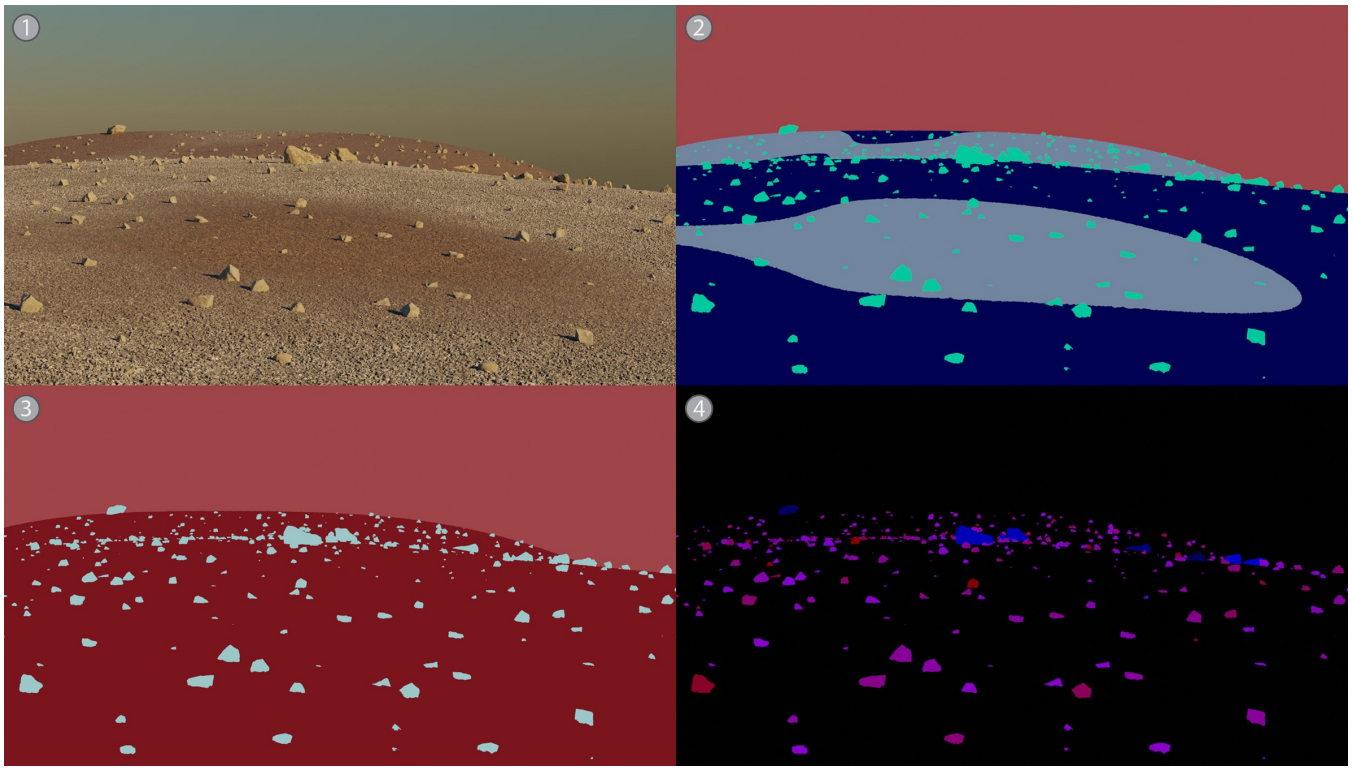

Fig. 5: Example of scenes with four passes. 1 is the RGB pass. 2 is the first semantic level pass. 3 is the second semantic level pass. Note the differences in the two semantic passes. 4 is the semantic instance pass.

falling of the objects on the plane is physically simulated, which is computationally heavy. While this yields a more accurate result, the particle method is by far faster since no physical calculations are necessary. Therefore, it is also possible to place large amount of objects on the stage in the order of thousands. For our training images, we use environments with 30000+ stones on the terrain. As a result, the particle method is more applicable for outdoor scenes. To get more realistic object distributions, one can adjust the distribution pattern for the objects. This is done by defining a noise pattern as density map for the stage. For example, in a planetary case, the rocks are usually more clustered instead of spread uniformly over the ground. We simulate this with the Worley noise [30]. An example section of an object asset cfg can be seen in Listing 3.

Listing 3: Example cfg for object asset. For some parameters, min and max ranges can be defined, like for *numberInstances* in this case. Depending on the value of *instanceLabelActive*, the asset will output an instance label. When the flag *useDensityMap* is enabled, a density distribtion map is used for the particles. The value of *defaultSize* defines the default value.

```
"meshes": [
{
  "path":"/path/to/mesh/",
  "numberInstances": [3000,20000],
  "labelIDVec": [42,200,0,200],
  "instanceLabelActive": true,
  "useDensityMap": true,
  "defaultSize": 3.0}]
```

3) Rendering Labels: Blender cannot directly output semantic and instance labels. Therefore, most other works use the ObjectID render pass as semantic output, which most render engines provide. The output is an image, where each object in the scene has its own ID. While this can be directly used for instance labeling, it has its drawbacks when it comes to semantic labels. Although objects may have an individual object ID, they can share the same semantic label. Furthermore, one object might have several different parts, which have their own semantic label, e.g. the terrain stage can have several different terrains. Lastly, objects may have multiple semantic labels. To address these shortcomings of just rendering the object ID pass, we render the diffuse channel, which outputs the pure material color applied to an object without any physical based rendering applied. As a result, the object can have different labels for different parts of the object, depending on the material. Therefore, this approach can also be used to give the terrains applied on the same object different semantic labels. We go even a step further and allow not just a single semantic label material, but as many as the user specifies, which can be easily done within an array in the cfg-file using parameter *labelIDVec* as shown in Listing 3. However, the problem with this approach is that the rendered image is in the RGB-value space. Therefore, it is necessary to encode the semantic label into RGB values. This can be easily done for the red, green, blue channel R, G, B as follows:

$$\begin{aligned} R &= \left\lfloor \frac{\mathrm{ID}_{\mathrm{label}}}{\#c^2} \right\rfloor \cdot \Delta s \\ G &= \left\lfloor \frac{\mathrm{ID}_{\mathrm{label}} (\mathrm{mod} \#c^2)}{\#c} \right\rfloor \cdot \Delta s \\ B &= \left\lfloor \left(\mathrm{ID}_{\mathrm{label}} (\mathrm{mod} \#c^2)\right) (\mathrm{mod} \#c) \right\rfloor \cdot \Delta s \end{aligned} \tag{2}$$

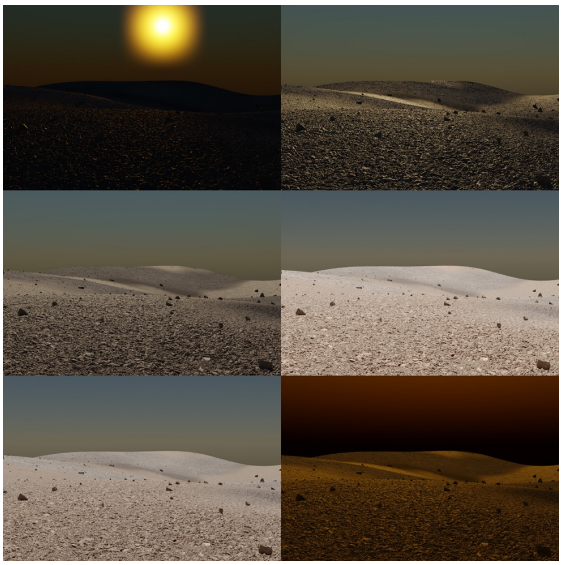

Fig. 6: Same scene with different light settings and atmosphere properties.

where $\#c$ is the number of step values assigned for each individual channel and Δs is the label step size. ID_{label} is the semantic class label. The same method is used for the instance class labels. Here, ID_{label} is replaced by $\text{ID}_{\text{object}}$. In the case of particles, this is the particle ID.

C. Light Simulation Module

OAISYS can simulate different light setups to produce varied conditions. The user can either choose HDR images as illumination background or Blender's internal "Sky Texture". The Sky Texture comes with a variety of sun parameters, like its size, elevation, intensity, and altitude. Furthermore, it is possible to change the atmosphere value with the three values: air, dust, and ozone. All of these parameters can be randomly picked by the OAISYS by defining their minimum and maximum values. Fig. 6 illustrates the same scene rendered with different light and atmosphere settings.

D. Camera Simulation Module

Our simulator provides two options to simulate the camera. The first option is to specify precise camera motions as sequence poses with a csv file. The second option is to sample random values for the camera poses. For the second option, we use a target object faced by the camera to guarantee that the camera is focusing on a particular section of the simulation space. The user can choose separate constraining values for the random movement of the target position. For instance, the target object might be in the same area, where the stage is present. As a result, the camera is always facing the terrain. One can also specify the target object position relative to the camera position, e.g. to focus the camera to very close or far areas. The target object is invisible to the rendering engine. Although our simulator ships by default with one camera sensor type, it can be easily extended by custom sensors and sensor noise models. It is also possible to arbitrarily simulate many different sensors at the same time (e.g. simulating a stereo camera rig).

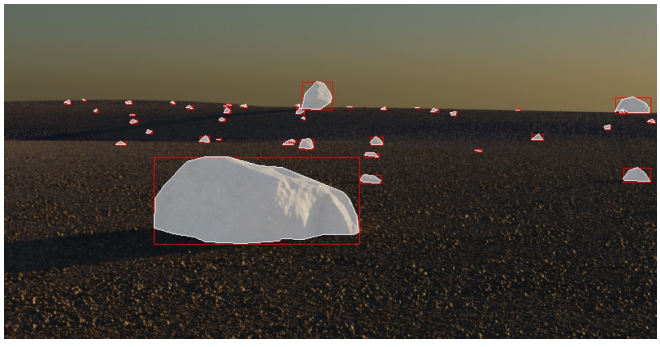

Fig. 7: Example of COCO annotation for rocks.

TABLE I: RMS ATE of two simulated datasets.

	sequence 1	sequence 2	sequence 3
DSO	0.208729	0.217106	0.136558
ORB-SLAM3	17.615447 (abort)	0.376687	0.140577

E. Post-Processing Module

We also provide a post-processing script for additional label filtering (e.g. instance labels within a certain distance or size). The script decodes the semantic labels, gives threshold options for instance labels, and provides two widely used data formats, COCO and hdf5. An example of a generated COCO file is shown in Fig. 7.

IV. Experiments

As testbed for the presented simulator, we evaluate it on three common use-cases found in planetary robotic missions. Firstly, we test the feasibility of the data to evaluate ego-motion estimation, as required for autonomous mobile agents on planetary missions [33], [34]. Secondly, we synthesize a varied dataset of planetary scenes and train a semantic segmentation network with it, to test its ability to correctly segment previously unseen real world data from a planetary analogue site. Thirdly, we study the applicability of the synthetically generated data to perform instance segmentation of rocks that are of particular interest during planetary missions for sample collection or scene understanding purposes.

A. Ego-Motion Estimation Task

For evaluation of ego-motion estimation on the synthetically generated data, we generate 3 trajectories on a simulated planetary surface, which we read into our simulator with the csv camera option. The trajectories simulate a flight of an MAV over a planetary surface. We run two state-of-the-art visual odometries on the synthesized dataset: a direct method, DSO [35], and an indirect method, ORB-Slam3 [36]. Fig. 8 shows the three trajectories and Fig. 9 an example image of DSO in action when run on our dataset. The resulting absolute RMS errors are presented in Tab. I. DSO performs well throughout all sequences. ORB-Slam performs well on the sequences 2 and 3, while producing a critical tracking failure on sequence 1. This failure is attributed to fast rotational motions. With these results, we show that we can successfully use the simulator to test and tune robot motions in planetary environments towards their performance in visual odometry. This step is relevant to avoid critical operation failures and investigate corner cases of operation when planning robotic missions where human intervention is limited or impossible.

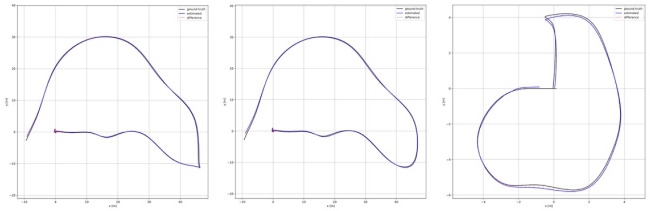

Fig. 8: The three trajectories from top view (black curve: ground-truth; blue curve: estimated by DSO).

B. Semantic Segmentation

Semantic information about the environment can be a crucial input for future planetary robotic missions. It can support the traversability analysis of terrain in order to plan safe robotic paths, or can help to discover scientifically interesting spots. Currently, there exists no publicly available planetary dataset with sufficient semantic annotation of the environment. Therefore, our simulator can serve as a dataset generator for training machine learning methods. In this experiment, we simulate environments consisting of 5 common planetary landscape classes, i.e. rock, sand, dried mud, gravel, and sky. In total, we rendered 8.345 image samples in this environment, using only 7 textures in total, i.e. 3 sand textures, 1 dried mud texture, and 3 gravel textures. Furthermore, we scattered 16 different rock assets over the auto generated terrains. For each rock, we used a sample size between 0 and 13 000 instances. Example dataset images are illustrated in Fig. 10. We fine-tuned the popular semantic segmentation network DeepLabv3 [37] on this dataset, which was pre-trained on the COCO dataset. To evaluate the trained network, we manually annotated 18 images of the MADMAX dataset [20]. Example predictions are depicted in Fig. 11. We achieve the following IoU for the following classes: 81,39 (sand), 81.38 (gravel), 41.17 (dried mud), 90.66 (sky), 73.35 (rock). We note that the performance is generally high, which confirms previous findings of the suitability of training semantic segmentation tasks on synthetic data [23]. Note that the dried mud class has a relatively low IoU. In most failure cases, the network predicted gravel instead of dried mud. We attribute this to the nature of the used data. Test data labelled as dried mud typically contained scattered gravel and was hence misclassified as such. A more fine-grained texture for the dried mud texture would most likely increase performance in this case.

Incorrect estimations also happened for distant areas, where gravel often was mistakenly predicted as sand. Since far away gravel is difficult to visually distinguish from nearby sand, inclusion of a depth channel in training and inference could potentially alleviate this failure case.

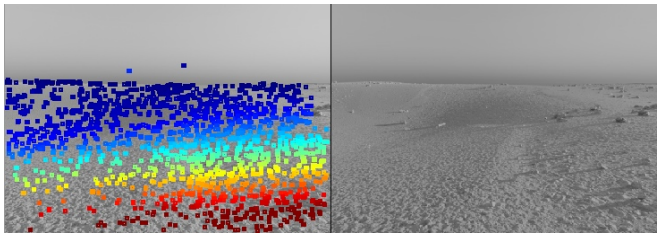

Fig. 9: DSO running on a dataset generated by OAISYS.

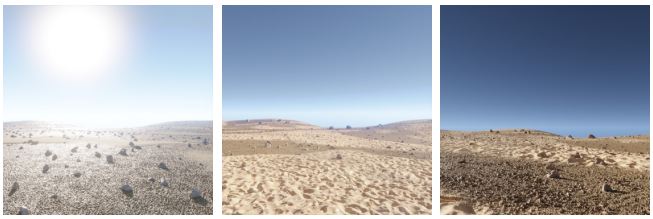

Fig. 10: Training data examples for the semantic segmentation task.

C. Instance Segmentation

Besides semantic segmentation, segmenting individual objects (e.g. rocks) is an important task in order to manipulate and navigate a robot's surrounding. To demonstrate the feasibility of OAISYS for the generation of instance-labelled data, we train the instance segmentation network Mask R-CNN [38] on synthetic data. Specifically, we use the implementation of Wu et al. [39] and fine-tune the network for 15 epochs with 11 205 rendered training images, which only employed 7 rock assets. The evaluation is done on real data recorded during a scouting mission on Mount Etna. On the test data consisting of three manually annotated scenes resulting in a total of 47 samples, we achieved a mean IoU of 53.68. Although, the value indicates a low performance, qualitative results as shown in Fig. 12 show an opposite impression. It can be noticed, that so-called floating rocks are often well segmented, while smaller rock formations result in no detection. We believe that a larger variety of rock samples

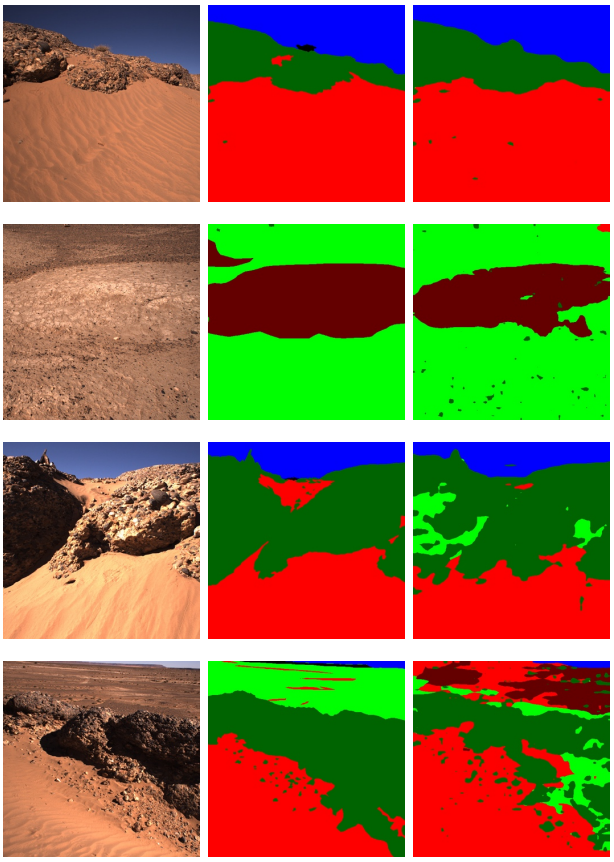

Fig. 11: Left: RGB image from the MADMAX dataset. Middle: Corresponding manually labeled ground truth. Right: Prediction of fine trained network on synthetic data. Classes are: sand (red), gravel (green), dried mud (dark red), sky (blue), and dark green (rock)

Fig. 12: Examples of the instance segmentation predictions.

in the simulation can potentially achieve higher performance. Multi-view can also help the network to further improve on this challenging test data.

V. Conclusion

In this paper, we presented a novel simulator for outdoor environments with the focus on planetary landscapes. We introduced a method for generating arbitrarily many different kinds of terrains, which can be used to create large datasets. Furthermore, we demonstrated a novel method on how to create multiple layers of semantics, which is important for a variety of robotic and scientific tasks. To demonstrate our simulator, we evaluated it with three common robotic scenarios. We were able to show that the simulator can synthesize datasets, which can be used to train neural networks and apply real world data to them. In this paper, we also showed that navigation algorithms can be tested with our simulation. In order to give anyone the possibility to create their own datasets and to extend the simulator by custom modules, we release the code of the Outdoor Artificial Intelligent Systems Simulator.

ACKNOWLEDGMENT

We thank Martin Wudenka and Wout Boerdijk for their help with the experiments. This work was supported by the Helmholtz Association, project ARCHES (`www.arches-projekt.de/en/`, contract number ZT-0033).

References

[1] M. Cordts, M. Omran *et al.*, "The Cityscapes dataset for semantic urban scene understanding," in *Conf. on Computer Vision and Pattern Recog.*, 2016.
[2] J. Deng, W. Dong *et al.*, "Imagenet: A large-scale hierarchical image database," in *Conf. on Computer Vision and Pattern Recog.*, 2009.
[3] B. Zhou, A. Lapedriza *et al.*, "Places: A 10 million image database for scene recognition," *Trans. on Pattern Analysis and Machine Intelligence*, 2017.
[4] N. Silberman, D. Hoiem *et al.*, "Indoor segmentation and support inference from RGBD images," in *Europ. Conf. on Computer Vision*, 2012.
[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Conf. on Computer Vision and Pattern Recog.*, 2012.
[6] J. Sturm, N. Engelhard *et al.*, "A benchmark for the evaluation of RGB-D SLAM systems," in *Int. Conf. on Intelligent Robot Systems*, 2012.
[7] K. Wagstaff, Y. Lu *et al.*, "Deep Mars: CNN classification of mars imagery for the PDS imaging atlas," in *Conf. on Artificial Intelligence*, 2018.
[8] Epic Games, "Unreal Engine," https://www.unrealengine.com.
[9] Unity, "Unity Engine," https://unity.com.
[10] S. Shah, D. Dey *et al.*, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2018.
[11] A. Dosovitskiy, G. Ros *et al.*, "CARLA: An open urban driving simulator," in *1st Annual Conference on Robot Learning*, 2017.
[12] Y. Song, S. Naji *et al.*, "Flightmare: A flexible quadrotor simulator," in *Conf. on Robot Learning*, 2020.
[13] B. O. Community, *Blender - a 3D modelling and rendering package*, 2018.
[14] H. Alhaija, S. Mustikovela *et al.*, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *Int. Journal of Computer Vision*, 2018.
[15] T.-Y. Lin, M. Maire *et al.*, "Microsoft COCO: Common objects in context," in *Europ. Conf. on Computer Vision*, 2014.
[16] J. Xue, H. Zhang *et al.*, "Differential angular imaging for material recognition," *Conf. on Computer Vision and Pattern Recog.*, 2017.
[17] A. Valada, G. Oliveira *et al.*, "Deep multispectral semantic scene understanding of forested environments using multimodal fusion," in *Int. Symp. on Experimental Robotics*, 2016.
[18] M. Wigness, S. Eum *et al.*, "A RUGD dataset for autonomous navigation and visual perception in unstructured outdoor environments," in *Int. Conf. on Intelligent Robots and Systems*, 2019.
[19] G. Neuhold, T. Ollmann *et al.*, "The mapillary vistas dataset for semantic understanding of street scenes," in *Int. Conf. on Computer Vision*, 2017.
[20] L. Meyer, M. Smíšek *et al.*, "The MADMAX dataset for visual-inertial rover navigation on Mars," *Journal of Field Robotics*, 2021, in press.
[21] H. Blum, P.-E. Sarlin *et al.*, "Fishyscapes: A benchmark for safe semantic segmentation in autonomous driving," in *ICCV Workshops*, 2019.
[22] NVIDIA, "Isaac Sim Software," https://developer.nvidia.com/isaac-sim.
[23] M. Johnson-Roberson, C. Barto *et al.*, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" in *Int. Conf. on Robotics and Automation*, 2017.
[24] G. Ros, L. Sellart *et al.*, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Conf. on Computer Vision and Pattern Recognition*, 2016.
[25] M. Denninger, M. Sundermeyer *et al.*, "BlenderProc," *arXiv:1911.01911*, 2019.
[26] S. Parkes, I. Martin *et al.*, "Planet surface simulation with PANGU," in *Int. Conf. on Space Operations*, 2004.
[27] S. Parkes, M. Dunstan *et al.*, "Planet surface simulation for testing vision-based, autonomous planetary landers," in *Int. Astronautical Congress*, 2006.
[28] R. Brochard, J. Lebreton *et al.*, "Scientific image rendering for space scenes with the SurRender software," *arXiv:1810.01423*, 2018.
[29] B. Burley, "Physically-based shading at Disney," in *ACM SIGGRAPH*, 2012.
[30] S. Worley, "A cellular texture basis function," in *ACM SIGGRAPH*, 1996.
[31] T. Nishita, T. Sirai *et al.*, "Display of the earth taking into account atmospheric scattering," *Computer Graphics*, 1996.
[32] K. Perlin, "Improving noise," in *ACM SIGGRAPH*, 2002.
[33] D. S. Bayard, D. T. Conway *et al.*, "Vision-based navigation for the nasa mars helicopter," in *AIAA SciTech Forum*, 2019.
[34] P. Lutz, M. G. Müller *et al.*, "ARDEA - an MAV with skills for future planetary missions," *Journal of Field Robotics*, 2020.
[35] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *Trans. on Pattern Analysis and Machine Intelligence*, 2018.
[36] C. Campos, R. Elvira *et al.*, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *arXiv:2007.11898*, 2020.
[37] L. Chen, G. Papandreou *et al.*, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, 2017.
[38] K. He, G. Gkioxari *et al.*, "Mask R-CNN," in *Int. Conf. on Computer Vision*, 2017.
[39] Y. Wu, A. Kirillov *et al.*, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.