Using Game Engine to Generate Synthetic Datasets for Machine Learning

Tomáš Bubeníček*
Supervised by: Jiri Bittner†

Department of Computer Graphics and Interaction
Czech Technical University in Prague
Prague / Czech Republic

Abstract

Datasets for use in computer vision machine learning are often challenging to acquire. Often, datasets are created either using hand-labeling or via expensive measurements. In this paper, we characterize different augmented image data used in computer vision machine learning tasks and propose a method of generating such data synthetically using a game engine. We implement a Unity plugin for creating such augmented image data outputs, usable in existing Unity projects. The implementation allows for RGB lit output and several ground-truth outputs, such as depth and normal information, object or category segmentation, motion segmentation, forward and backward optical flow and occlusions, 2D and 3D bounding boxes, and camera parameters. We also explore the possibilities of added realism by using an external path-tracing renderer instead of the rasterization pipeline, which is currently the standard in most game engines. We demonstrate our tool by creating configurable example scenes, which are specifically designed for training machine learning algorithms.

Keywords: Computer vision, Dataset generation, Game engines, Machine learning

1 Introduction

Augmented image data such as semantic segmentation help machines separate parts in factory lines, using optical flow data for video compression reduces redundancy, and depth and normals data are useful for approximating a 3D scene topology. Acquiring these image data is often very difficult, cost-prohibitive, or sometimes even impossible. The state-of-the-art research uses machine learning and neural networks to process images shot by a camera and generate augmented data. Training such algorithms requires a large number of ground-truth data, which is not always available.

For some cases, such as object categorization, broad, often human-labeled, datasets are already publicly available. However, for some augmented image data (such as optical flow data), the real measured data is often sparse or not measurable by conventional sensors in general. For this very reason, synthetic datasets, which are acquired purely using a simulated scene, are often used.

Several such synthetic datasets based on virtual scenes already exist and were proven to be useful for machine learning tasks, such as one presented by Mayer et al. [6]. They used a modified version of Blender 3D creation suite, which was able to output additional augmented image data such as optical flow information and segmentation. What is missing is a way for others to create their data in an easy-to-use environment, as the modified Blender suite is not available, and it is not straightforward to add different types of output to its rendering. This paper presents a method of generating such datasets using a game engine in a way where a game scene can be easily modified to provide such outputs. We also present an example configurable scene specifically created for optical flow training, BouncingThings.

In Section 2, we describe a selection of currently used datasets and tools enabling synthetic dataset generation. In Section 3, we discuss the needed features a tool used to generate synthetic datasets should have. Finally, in Section 4, we present our own implementation of such a tool.

2 Related Work

Data used for object segmentation are probably the biggest and most common datasets currently available. For example, the COCO (Common Objects in Context) [14] dataset contains over 200 thousand human-labeled real-life images and is useful for training networks to recognize objects located in photos.

Less common, but still readily obtainable, are real-life datasets containing depth, which can be useful for scene reconstruction. A combination of LIDAR and camera mounted on a car is usually the source of these datasets. This type of measurement is the case for the KITTI datasets [11][15] and the Waymo dataset [7], targeted for autonomous driving car development. ScanNet [5], a different dataset with depth information, uses a different

*tombuben@gmail.com
†bittner@fel.cvut.cz

method of sourcing such data, using off-the-shelf components such as a tablet and a 3D scanning sensor.

One segment where there are issues in obtaining real-life datasets is optical flow information. Optical flow data describes for each pixel in two successive frames the change of position of the surface the pixel represents. Only a very few datasets are containing real measured data, such as the Middlebury dataset [8] released in 2007. The camera shows small scenes covered with a fluorescent paint pattern captured under both visible and UV light. Since the fluorescent paint is evident under the UV lighting, the ground truth data was recoverable. As this method is complicated to reproduce, only eight short sequences using this method exist.

KITTI [11][15], also containing real-life optical flow data, calculated the data with the help of a LIDAR and egomotion of the car. Due to the way the calculation works, the framerate of the flow data is tenth the framerate of the camera itself and is only available for static parts of the scene.

Capturing the optical flow in real-life scenes is a difficult task, so most other datasets build on synthetic generation. The first synthetic datasets used for evaluating optical flow estimation algorithms date back as early as 1994, where Barron, J. et al. [1] used a Yosemite Flow Sequences dataset showing a 3D visualization of the Yosemite mountain range. In Middlebury [8], the eight remaining short scenes available are synthetic scenes rendered using the realistic MantaRay renderer. FlyingChairs [4] is another noteworthy synthetic dataset, later extended into FlyingThings3D [6]; Simple objects (e.g., chairs) floating along random trajectories are rendered using a modified version of the open-source Blender renderer, which allows the reading of optical flow data. Surprisingly, this abstract movement, which has no relation to the real behavior of moving objects (the objects pass through each other), has been shown as an effective way of training neural networks. The use of a modified Blender renderer also allowed for datasets based on scenes from open-source animated shorts, Sintel [2] and Monkaa [6]. Although the use of the preexisting projects is excellent for more diverse outputs, it can also cause issues – for some use cases, camera behavior such as a change in focus may not be desirable.

A recent CrowdFlow dataset [16] shows aerial views of large crowds of people rendered in Unreal Engine. This dataset shows that for some uses, datasets specialized for a single task could be beneficial. In this case, the dataset targets tracking behavior in large crowds.

2.1 Synthetic dataset generators

The previously mentioned synthetic datasets were all published only as final renders wihtout any tool for generating or modifying them, but several utilities for simplified creation of computer vision datasets already exist. Some of them are a part of more massive simulators, such as *CARLA* [3], an autonomous car simulator, or *AirSim* [17], a simulator for autonomous drones and cars. Both of these utilities build on Unreal Engine and provide both C++ and Python APIs to control vehicles in the scene, including retrieving of synthetic image data from virtual sensors attached to the vehicles. Their primary purpose is not the generation of new datasets but simulating entire road or sky scenes for virtual vehicles to move in, so the types of augmented image data are limited mostly to basic types such as depth or segmentation maps.

There are some preexisting plugins for game engines that enable the acquisition of augmented image data. One of which is *NVIDIA Deep learning Dataset Synthesizer (NDDS)* [9], which, built on Unreal Engine, provides blueprints to access depth and segmentation data, along with bounding box metadata and additional components for creation of randomized scenes. Another option built on top of Unreal Engine is *UnrealCV* [10], which, compared to NDDS, exposes Python API to capture the images programmatically and directly feed them to a neural network. The API allows interacting with objects in the scene, setting labeling colors for segmentation and retrieving of depth, normal, or segmentation image data. The system is virtually plug-and-play, where the plugin can be added to an existing Unreal Engine project or game and start generating augmented image data.

By default, the Unreal Engine does not provide access to motion vector data, which represents backward optical flow from the current to the previous frame. Nevertheless, since the source code is available, such functionality can be enabled by modifying the source code and recompiling the engine. *Unreal Optical Flow Demo* [12] presents a patch enabling the functionality used in Unreal based robot simulator *Pavilion* [13].

The last generator analyzed is a simple plugin for the Unity game engine. *ML-ImageSynthesis* [19] is a script providing augmented image data for object segmentation and categorization, depth and normal maps. In contrast to other previously mentioned plugins, it also provides backward optical flow data, which is obtained from Unity motion vectors.

3 Data creation framework

When designing systems to generate synthetic datasets, the designers need to take special care. Jonas Wulf et al. [18] present several issues that can arise when creating a dataset containing optical flow output. As such, when designing the utility, a platform on which we build on is important. We have selected Unity game engine as the platform to develop the application on, mainly because of a straightforward access to motion vectors, ease of use, and the possibility of an integration with third party path tracing renderer OctaneRender.

The utility consists of two parts: FlowGen, a reusable component for generating augmented images and other

metadata, and BouncingThings, a scene useful for training optical flow detection. Optionally, we can use the OctaneRender for Unity plugin to get path traced images with realistic lighting and motion blur. The Figure 1 shows a schematic overview of our tool.

We propose the following set of different outputs generated by FlowGen:

- RGB output. Camera data as generated by the Unity rasterizer.
- Segmentation outputs. For each pixel, an ID of the represented object is encoded.
- Motion segmentation mask. For each pixel, an information whether the represented surface is moving in world space is encoded.
- Optical flow (backward and forward). For each pixel, an information how the represented surface moves in the previous or next frame is encoded.
- Occlusion masks. For each pixel, an information whether the represented surface is visible in the previous or next frame is encoded.
- Depth and normal map outputs. Per pixel information provided by Unity z-buffer and geometry transform.
- Camera parameters. Metadata information including the camera matrix, position, rotation and configuration.
- Bounding box information. Metadata containing info about select tracked objects.

When using a game engine, we can implement most of the image outputs as relatively simple shader programs run on the GPU. The metadata outputs are produced on the CPU and saved as json files for easy parsing.

BouncingThings, the scene provided in the utility is conceptually based on FlyingThings3D [6]. It contains a set of flying objects which float in the scene randomly. In contrast to previous datasets, it uses a rigid-body physics simulation, so the objects do not intersect each other, and allows user configuration. The user can change which objects are used, how many objects are used and what is their minimum and maximum speed.

4 Results

This paper presents example outputs using the BouncingThings scene. Figure 2 shows different types of RGB output generated using our system. We allow for Unity to generate output with or without shadow maps, and the OctaneRender for Unity plugin allows for the creation of path-traced outputs.

In Figure 3, we show different possible segmentation outputs. We provide outputs for motion segmentation, segmenting all GameObjects separately and segmenting objects sharing common meshes.

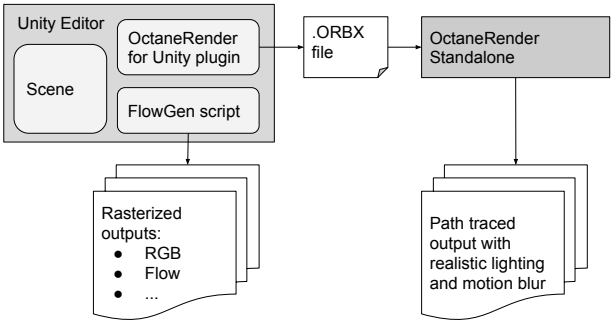

Figure 1: Diagram of data output sources.

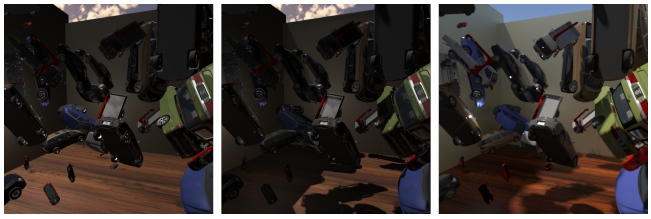

Figure 2: Image outputs with increasing levels of lighting realism. Left to right: Unity rasterizer without shadows, Unity rasterizer with shadow mapping, OctaneRender path-tracer.

The Unity engine provides easy access to depth z-buffer, per-pixel normals, and motion vectors (which represent backward optical flow). These outputs are visible in Figure 4.

Figure 5 shows our implementation of optical flow and occlusions computation. We provide our method of computing forward and backward optical flow based on remembering the transformation matrices of all objects in the previous frame and reprojecting them.

We configured the scene with a set of car models, both static and physically simulated. It contains transparent and specular materials. These properties can cause issues for machine learning, and therefore we plan to extend the outputs with transparency and specularity masks.

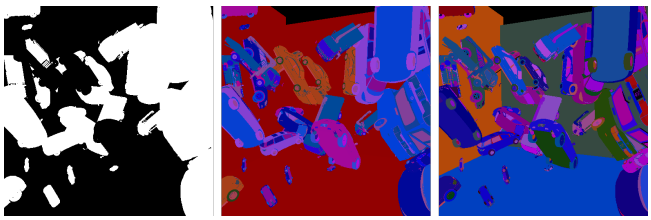

Figure 3: A selection of segmentation masks outputs. Left to right: motion segmentation, mesh segmentation, object segmentation.

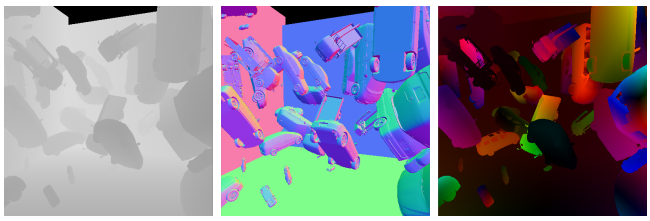

Figure 4: A selection of Unity provided outputs. Left to right: depth map, normal map, motion Vectors.

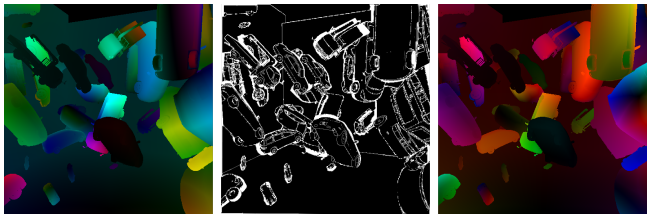

Figure 5: A selection of outputs calculated using reprojection from previous frames. Left to right: forward optical flow, backward occlusions, backward optical flow.

5 Conclusions

We have introduced a utility simplifying the creation of synthetic datasets and a scene suitable to train machine learning algorithms to detect optical flow. Our motivation is to create datasets used to train machine learning algorithms and evaluate already existing algorithms. Although we target the scene at optical flow estimation, the dataset is useful to estimate other quantities, such as object segmentation, depth information, or object tracking. We work on verification of our data for motion estimation, segmentation, and 3D object detection and tracking. We plan to publish our tool and the dataset in the near future and believe that our utility will boost machine learning research for challenging tasks in computer vision, as it allows other researchers to create additional datasets based on their own needs.

Acknowledgments

This research was supported by Toyota Motor Europe and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS19/179/OHK3/3T/13.

References

[1] John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

[2] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, 2012.

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[4] A. Dosovitskiy et al. Flownet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2015. http://lmb.informatik.uni-freiburg.de/Publications/2015/DFIB15.

[5] Angela Dai et al. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[6] N. Mayer et al. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134. http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16.

[7] Pei Sun et al. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[8] Simon Baker et. al. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[9] Thang To et al. NDDS: NVIDIA deep learning dataset synthesizer, 2018. https://github.com/NVIDIA/Dataset_Synthesizer .

[10] Weichao Qiu et al. Unrealcv: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, 2017.

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[12] Fan Jiang. Unreal optical flow demo, August 2018.

[13] Fan Jiang and Qi Hao. Pavilion: Bridging photorealism and robotics. In *Robotics and Automation (ICRA), 2019 IEEE International Conference on*, May 2019.

[14] Tsung-Yi et al. Lin. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[15] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] Gregory Schröder, Tobias Senst, Erik Bochinski, and Thomas Sikora. Optical flow dataset and benchmark for visual crowd analysis. In *IEEE International Conference on Advanced Video and Signals-based Surveillance*, 2018.

[17] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[18] Jonas Wulff, Daniel J Butler, Garrett B Stanley, and Michael J Black. Lessons and insights from creating a synthetic optical flow benchmark. In *European Conference on Computer Vision*, pages 168–177. Springer, 2012.

[19] Renaldas Zioma. Image synthesis for machine learning, December 2016. https://bitbucket.org/Unity-Technologies/ml-imagesynthesis/.