ZPY: Open Source Synthetic Data for Computer Vision

Technical Report

Hugo Ponte* Zumo Labs hugo@zumolabs.ai

Norman Ponte Zumo Labs norman@zumolabs.ai

Sammie Crowder Zumo Labs sammie@zumolabs.ai

Kory Stiger Zumo Labs kory@zumolabs.ai

Steven Pecht Zumo Labs steven@zumolabs.ai

Michael Stewart Zumo Labs michael@zumolabs.ai

Elena Ponte Zumo Labs elena@zumolabs.ai

Abstract

Synthetic data presents a unique solution to the huge data requirements of computer vision with deep learning. In this work, we present zpy [2], an open source framework for creating synthetic data. Built on top of Blender, and designed with modularity and readability in mind.

Keywords Computer Vision · Synthetic Data · Machine Learning · Open Source · Python · Blender

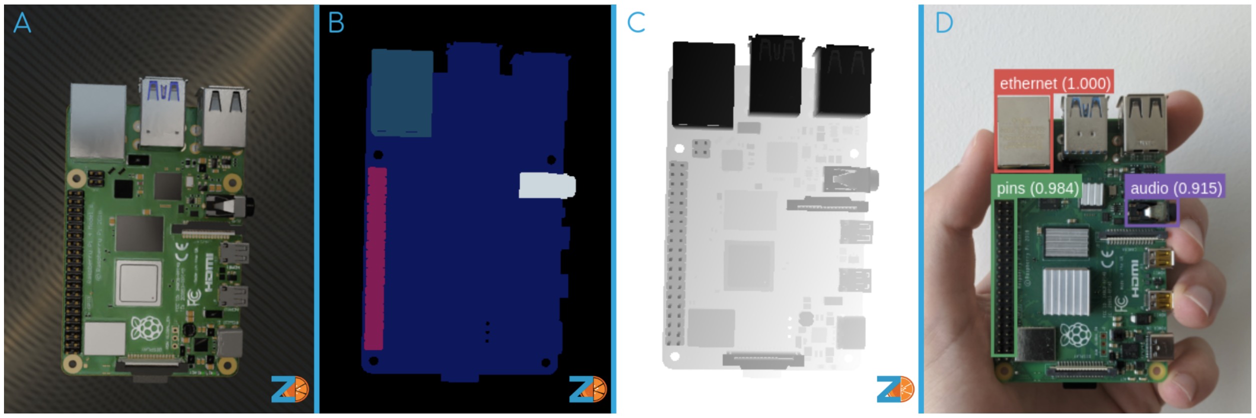

Figure 1: Synthetic images of a raspberri pi created with zpy: (A) color image, (B) segmentation image, (C) depth image. These images are used to train a deep learning model, which predicts bounding boxes on key components as seen in the (D) prediction image

1 Introduction

Open source machine learning frameworks (references)

Deep learning has exploded in popularity due to large open source frameworks such as tensorflow and pytorch What is synthetic data (references)

Challenges with synthetic data

Sim2Real Gap and domain randomization

*correspondence author

[2]All code is available on GitHub at `http://github.com/ZumoLabs/zpy`

Black box nature of ML

2 Background

2.1 3D

3D, short for the three dimensions of space we live in, is a catch-all term used to describe the varied technologies used to create virtual worlds. 3D's technology stack can be roughly split into two broad categories: *asset creation* and *asset scripting*. Asset creation **??** is the process of creating assets: virtual objects, scenes, and materials. Asset scripting **??** is the process of manipulating those assets and their interactions over the fourth dimension of time. Decades of progress have resulted in sophisticated software tools that make 3D workflows more automated and straightforward, but a significant amount of human expertise and artistic talent is still required.

2.1.1 Asset Creation

Assets are digital representations of a 3D object. One type of asset is a mesh: a connected graph of 3D points also called vertices, which define the surface of an object. Edges interconnect vertices, and a closed loop of vertices creates a polygon known as a face. The engineering and manufacturing world creates meshes using computer-aided design (CAD) software such as AutoCAD [**?**], Solidworks [**?**], Onshape [**?**], and Rhino [**?**]. The entertainment industry creates meshes using modeling software such as Maya [**?**], 3DSMax [**?**], and Cinema4D [**?**].

Whereas a mesh describes the shape and form of an object, a material asset describes the texture and appearance of a virtual object. A material may define rules for the reflectivity, specularity, and metallic-ness of the object as a function of lighting conditions. Shader programs use materials to calculate the exact pixel values to render for each face of a mesh polygon. Modeling software usually comes packaged with tools for the creation and configuration of materials.

Finally, asset creation encompasses the process of scene composition. Assets can be organized into scenes, which may contain other unique virtual objects such as simulated lights and cameras. Deciding where to place assets, especially lights, is still almost entirely done by hand. Automatic scene composition remains a tremendous challenge in the 3D technology stack.

2.1.2 Asset Scripting

The fourth perceivable dimension of our reality is time. Asset scripting is the process of defining the behaviors of assets within scenes over time. One type of asset scripting is called animation, which consists of creating sequential mesh deformations that create the illusion of natural movement. Animation is a tedious manual task because an artist must define every frame; expert animators spend decades honing their digital puppeteering skills. Specialized software is often used to automate this task as much as possible, and technologies such as Motion Capture (MoCap) can be used to record the movement of real objects and play those movements back on virtual assets.

Game Engines are software tools that allow for more structured and systematic asset scripting, mostly by providing software interfaces (e.g., code) to control the virtual world. Used extensively in the video game industry after which they were named, examples include Unity [**?**], Unreal Engine [**?**], GoDot [**?**], and Roblox [**?**]. These game engines support rule-based spawning, animation, and complex interactions between assets in the virtual world. Programming within game engines is a separate skillset to modeling and animating and is usually done by separate engineers within an organization.

2.1.3 Blender

Blender is an open-source 3D software tool initially released in 1994 [**?**]. It has grown steadily over the decades and has become one of the most popular 3D tools available, with a massive online community of users. Blender's strength is in its breadth: it provides simple tools for every part of the 3D workflow, rather than specializing in a narrow slice. Organizations such as game studios have traditionally preferred specialization, having separate engineers using separate tools (such as Maya for modeling and Unreal Engine for scripting). However, the convenience of using a single tool, and the myriad advantages of a single engineer being able to see a project start to finish, make a strong case for Blender as the ultimate winner in the 3D software tools race.

Many of the world's new 3D developers opt to get started and build their expertise in Blender for its open-source and community-emphasizing offering. This is an example of a common product flywheel: using a growing community of users to improve a product over time. With big industry support from Google, Amazon, and even Unreal, Blender also has the funding required to improve its tools with this user feedback.

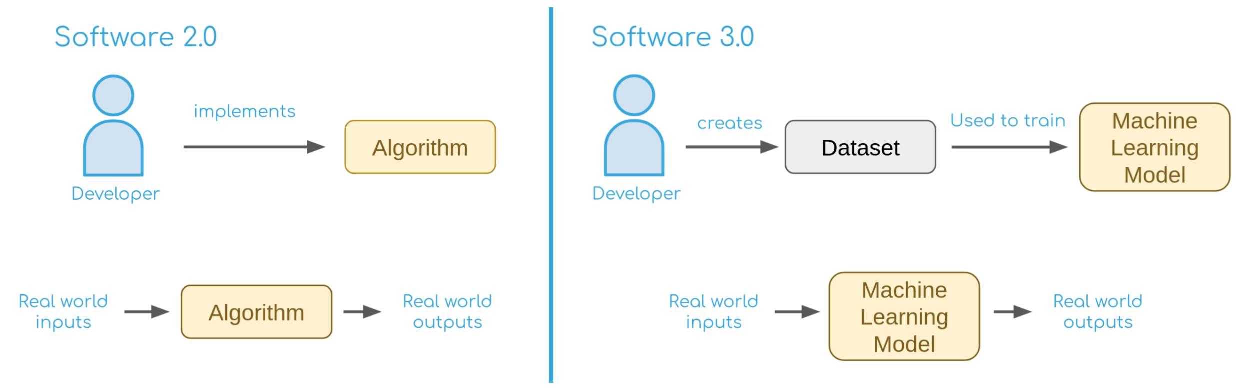

Figure 2: Software 3.0: the developer transitions from writing explicit algorithms to creating and curating datasets which are used to train machine learning models.

In addition to supporting the full breadth of the 3D workflow, Blender has the unique strength of using Python as the programming language of choice for asset scripting. Python has emerged as the lingua franca for modern deep learning, in part due to the popularity of open-source frameworks such as TensorFlow [**?**], PyTorch [**?**], and Scikit-Learn [**?**]. Successful adoption of synthetic data will require Machine Learning Engineers to perform asset scripting, and these engineers will be much more comfortable in Blender's Python environment than Unity's C or Unreal Engine's C++ tools.

2.2 Synthetic Data

The synthetic data workflow can be broadly divided into three categories: simulation design, generation, and tuning.

BlenderProc, Synthesis, LexSet

3 Motivation

3.1 Democratization of Data

The datasets used by companies today are almost exclusively collected. Images and annotations are gradually saved as they are generated by users of a product. Collecting a dataset large enough to train a robust machine learning model can take years.

Only companies that have the scale and have set up the infrastructure to collect and store large datasets will have the datasets required to train models.

In order to compete, small or newly formed companies often resort to purchasing data from a third party supplier. This market for the selling and reselling of collected data presents an existential threat to privacy. Though some regulations have emerged, famously GDPR in Europe, these have yet to change the landscape of the market for collected data.

The high cost of data labeling makes it unavailable to those without huge resources

3.2 Fairness and Bias

The real world is biased and unfair.

ML systems trained on this data will reflect the same bias during inference.

3.3 Software 3.0

Data creation as a new paradigm for "programming".

In the software world today, developers write explicit sets of rules (known as algorithms). These algorithms are then deployed into production systems, which consume input data and output actions.

In the software of tomorrow, developers will curate a dataset which will be used to train a deep learning model. This model will then be deployed into a production system, which will consume input data and output actions. This changes the workflow of developers from explicitly writing rules to instead creating the datasets which are then parsed to create algorithms.

4 Project Features

Given

4.1 Blender Addon

As explained in section X, deep learning is a python-first discipline. If we wish to include deep learning practitioners in the 3D stack it is thus of critical importance that we provide a python interface for the 3D workflow.

4.2 Cloud Backend

Computing has traditionally relied on Moore's Law to increase the power of individual computers. In the past decade the individual compute power of a single computer has not increased significantly, and instead the ability to coordinate a large number of individual computers on a single task has become the method for increasing computation. This type of parallel computing has been democratized through the availability of cloud computing platforms such as AWS, GCP, and Azure. However, these platforms remain difficult for the average developer to use effectively, and domain experts are usually required to take software running on a single computer and scale it across many computers in parallel.

Abstracting away the difficulties of the cloud workflow and providing an intuitive and convenient interface for parallelizing computation is thus important for the democratization of deep learning and synthetic data workflows. Much like platforms have emerged for the training and tuning workflows of deep learning, an opportunity exists to create platforms for the generation and tuning workflows of synthetic data.

4.3 User Interfaces

We provide three different interfaces to interact with our product: a Python API **??**, a CLI **??**, and a graphical WebApp **??**. Power users want an API (application programming interface) and CLI (command line interface). Building a ramp for the bulk of the developer community requires a GUI

```python
import zpy
zpy.generate()
```

Listing 1: Generating a dataset using the zpy python API.

```bash
# First you will need to log in and set the project (image generation gets billed
    according to project)
zpy login $USERNAME
zpy project set $PROJECT_UUID

# Generate the dataset
zpy create dataset "redbull cans and packs" can_v6 num_images 1000
```

Listing 2: Generating a dataset using the zpy CLI

5 Design

Hidden complexity

When designing software systems, there is usually a tradeoff between flexibility and simplicity. Simplicity is the ability to perform a task with minimal amount of work and a limited understanding of the software package. Flexibility is the ability to support many different tasks and allow for customization.

Random hdri and textures use default random textures unless a specific path is given

Pythonic syntax

One of the core features of Python is the language's human readable syntax. Python does not enforce function and variable type annotations, which makes it quicker to prototype code.

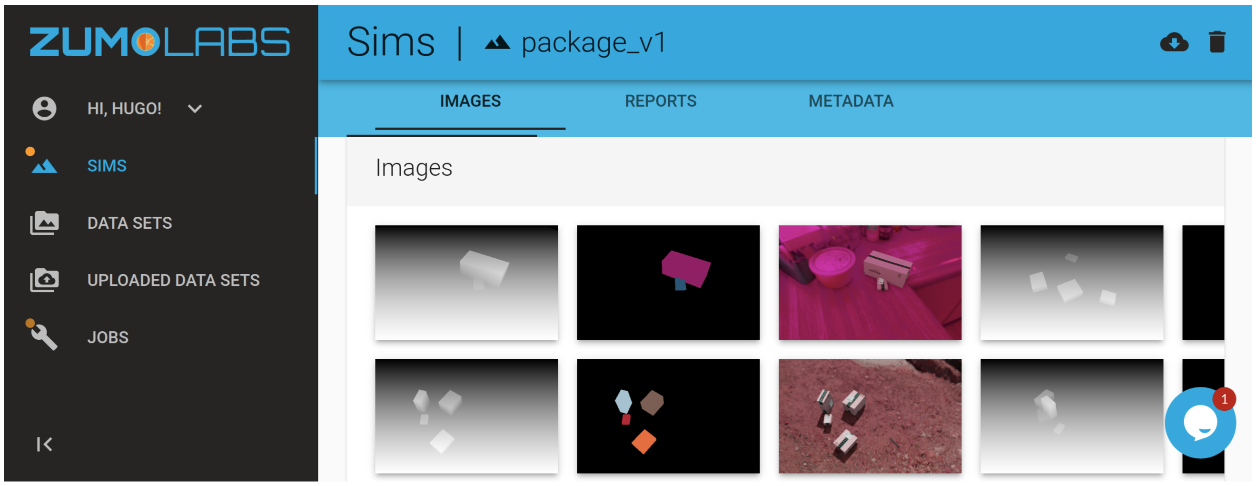

Figure 3: A visual interface for synthetic data creation via a WebApp.

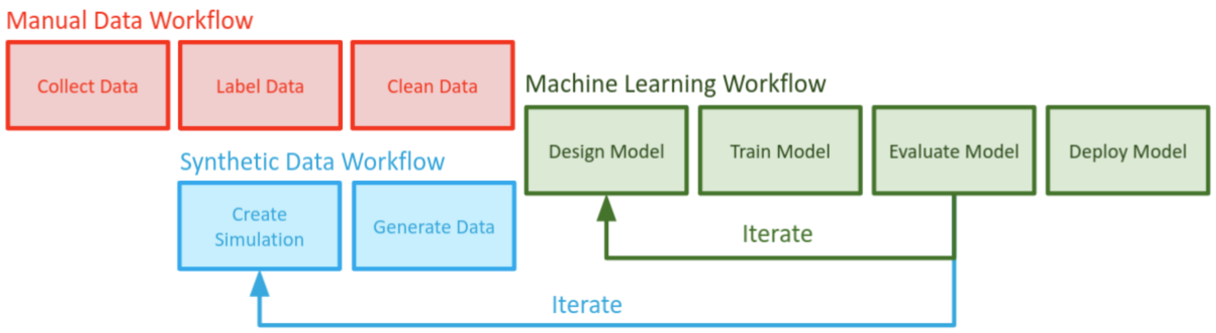

Figure 4: The Synthetic Data Workflow allows for iteration of the dataset.

Functions in zpy are flexible in the arguments that they accept. The ‘zpy.opject.segment()‘ function call can accept an object directly of type ‘bpy.types.Object‘, but it will also accept the unique string name of that object.

Extensibility

Zpy modules are separated by dependencies

Zpy modules are independent of each other, a monolithic system is much harder to update and maintain

6 Workflow

The full workflow for synthetic data can be reduced into four key steps: *Design* **??**, *Generation* **??**, *Evaluation* **??**, *Iteration* **??**. The synthetic data workflow is similar to the workflow when using collected data, with the key exception that it allows for iteration on the dataset itself **??**.

6.1 Design

The first step in the syntehtic data workflow is to design and create the sim, short for simulation. A sim is a collection of assets controlled through a single script. The run() function in the script acts as the point of entry for any generation process. Relevant parameters are put as kwargs in the run() function, which allows configuration through gin-config, a python package for configuration of python libraries.

6.2 Generation

Datasets are rendered frame by frame. Separate rendering passes are required for segmentation and color images.

Datasets can be rendered locally directly inside the Blender GUI. This allows for easy debugging when developing a sim. Once it is working locally, a sim can be exported into a zip file which will contain all the asset dependencies required. This zip file can be uploaded to the cloud with any of the user interfaces described in **??**.

Dataset are then generated in the cloud **??**.

We provide workflows for packaging several smaller datasets into a single larger dataset, as well as sorting individual datapoints into test, train, and validation buckets.

6.3 Evaluation

Sweeping over dataset hyperparameters (much like sweeping over model architecture parameters)

Fine-tuning pre-trained models as a proxy for dataset performance

6.4 Iteration

Least understood part of developing with synthetic data

The promise of AutoML for datasets

7 Example

Package detection

- Background material and texture.
- Lighting intensity and positioning.
- Object material and texture.
- Number of objects in view.

7.1 Domain Randomization

Effect of domain randomized lighting

Effect of domain randomized backgrounds

7.2 Training Curicculum

Pre-training w/ real and fine-tuning on synthetic

Training only on synthetic

Training on mixed synthetic and real

8 Conclusion

TODO