Delft University of Technology

Synthetic Data Generation

Realistic Images of Crushed Waste

Final Report group 2E

CSE2000

Authors:

Liudas Mikalauskas

Simon Mariën

Abel Van Steenweghen

Rahul Kochar

Augustas Vilcinskas

Project committee:

Recycleye - Client

Marco Loog - Coach

Max van Deursen - Teaching Assistant

Regina Tange-Hoffmann - Technical Writing Professor

Jan Bergen - Ethics Professor

January 24, 2021

Preface

This report was written by a group of 5 Computer Science and Engineering students at Delft University of Technology. For our Software Project course in the second year we developed an application that is able to automatically generate synthetic data to be used for neural network training.

We assume that the reader has some general technological knowledge and understands the core concepts and terms used in computer science.

We would like to thank our client Recycleye for this opportunity, giving their honest opinion and helping us overcome hurdles. We also like to mention our T.A. Max Van Deursen for the weekly feedback and help with problems. Also thanks to our coach Marco Loog for attending the presentations. We are glad that we got this opportunity to experience a real project for a client and improve our software engineering skills doing this.

Delft, June 21 2020

Liudas Mikalauskas
Simon Mariën
Abel Van Steenweghen
Rahul Kochar
Augustas Vilcinskas

Summary

The development of a synthetic data generator for training a neural network exists out of many parts. This summary gives a quick overlook of the main parts of our final product.

The synthetic data generator allows for accepting long-term parameters such as lighting and background, and short-term hyper-parameters such as the number of objects and types of objects. Also the material proportions are configurable, this was a request of our client such that they can simulate the real garbage compositions by being able to select on material rather than on a specific item.

Realism of the models depend mainly on the quality of the models used and the material configurations. To keep the compositions realistic the generator also allows for adding skins such as labels or logos, which can get randomized to get more diverse data.

Using a scene setup stage the camera angle, light type and angle and the background can be easily adapted with the hyper-parameters.

For simulating the crushing and object interactions the powerful physics functionality of Blender was used. It makes use of modifiers which can give certain properties to models, which is useful for simulation or deformation of an object. With the Rigid Body modifier objects are dropped over each other and with the Soft Body modifier objects are deformed.

When the images are rendered a JSON file with labels and bounding boxes is generated such that the data can be easily trained on by neural network it is produced for. We followed the COCO Dataset format since this was the preference of our client.

To make the final product easily deployable it is presented as a user-friendly flask server. It runs the specified image composition requests in a docker container such to avoid platform-specific problems.

In the final sections we talk about future improvements, recommendations and ethical implications.

Contents

1 Introduction

Currently the recycling industry is very labor intensive. Our client Recycleye wants to solve this problem, by replacing the human labor with a combination of robotics and computer vision. For recognizing and classifying the garbage they will be making use of a neural network. However, for a good performance a large enough dataset is required. To manually collect and label appropriate images would be too labor intensive and too slow to react to changes in requirements. Therefore, we have developed a synthetic data generator. This solution allows to produce unlimited amounts of synthetic images which can be altered to the users' needs.

The goal of our system is to synthesize good quality data that can be generated based on the need of the user. This report shows how this solution was implemented, what choices were made and what research was used for making these choices. We also explain how we dealt with limiting conditions such as the use of third-party models.

The report is structured as follows. Firstly, in section 2 the problem is analyzed. It contains a problem statement as well as previous related work that has been valuable in the design process. In section 3 the requirements are set out. It displays a list of all requirements, both functional and non-functional, noted with the MoSCoW method. In section 4 the design of the product is explained. It further clarifies the research and decisions that lead to the final design. In section 5 the implementation is clarified. It takes a look at the most important parts of the algorithm and the hurdles that were encountered. In section 6 the final product is discussed as well as possible extensions and recommendations for future development. In section 7 the ethical implications of the project are reviewed. Finally, in section 8 the report ends with a conclusion.

2 Problem analysis

2.1 Problem Statement

The problem stated by Recycleye can be divided into 3 parts:

1. **Synthetic Data Generation** : Given a parameter input from the user, synthetically create numerous crushed permutations of objects on a conveyor belt based on these.
2. **Verify** : Verify that the detail within the generated images is on par with object detail found in recent video games and a baseline model trained on them performs well detecting real data
3. **Deploy** : Create an interactive interface that is easily accessible to the rest of Recycleye's pipeline

Recycleye has also provided us with possible future extensions to the project. These are not part of the project but the work done by us in this project should be extendable in the directions mentioned below:

- Alter generated synthetic data to improve detection model's ability to detect real data.
- Teacher-learner model alter synthetic data mid training to best suit detection model learning.

2.2 Related work

A few previous works that offer an approach for the problem at hand are discussed below. Peng et al.[1] describe the pipeline for the generation as follows - a set of factors are chosen that manipulate the object's appearance using computer graphics techniques, namely: "object texture, color, 3D pose and 3D shape, as well as background scene texture and color." Then a dataset of CAD models is obtained having $5 - 25$ models per one abstract object. After that for each CAD model the viewpoint is adjusted to generate $3 - 4$ different "views". Next from each generated model "view" a set of perturbations is generated by adding several small rotations. Finally, for each perturbation, the texture, color and background image is selected and a virtual image is rendered to include in the virtual training data-set. Figure 1 illustrates the pipeline discussed in this paper. Note that even though in the paper the pipeline is extended with model training, this does not concern us since it is outside the scope of our project.

Another report by Sarkar et al. [2] discusses a similar approach. Firstly the CAD models are selected. They are then rendered with different camera angles. A tessellated icosahedron is used for finding the camera angles that are distributed uniformly. Different tessellation levels provide an increase in angles. It is also stated that using real backgrounds improved the performance of the system.

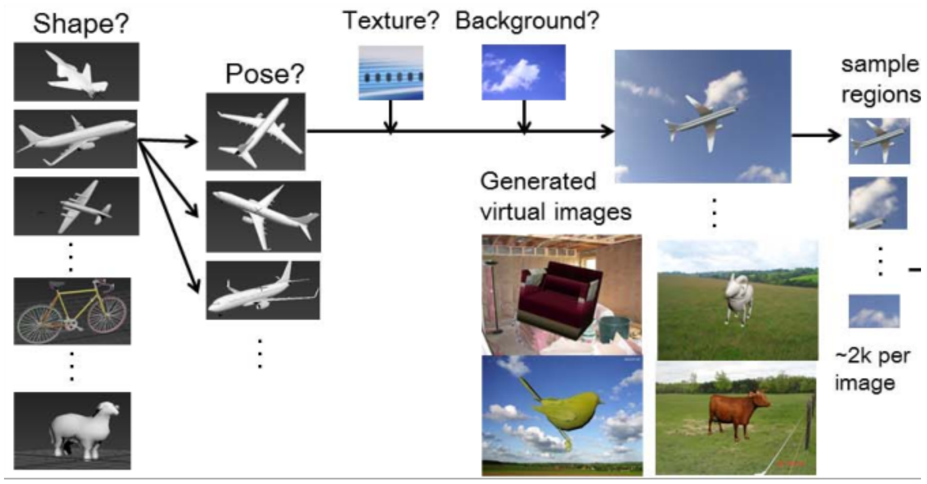


Figure 1: The core pipeline described by Peng et al. [1]

Another article [3] proposes solving the synthetic data generation problem using the software library implemented by the authors. The article suggests downloading desired 3D models, setting parameters (i.e.lighting, pose, texture, background, shape deformation, etc.) in a script and running the software in Autodesk 3ds Max. Doing this renders a large data-set of labeled images which can be applied to train the Recycleye model.

Tremblay's paper [4] has an interesting observation about Domain Randomization. In a few words, it's suggests, placing random number 3D models of objects of interest in a 3D scene at random positions and orientations. Add a random number of geometric shapes called Flying Distractors to teach the algorithm to ignore objects in the scene that are not of interest. Random textures are then applied to both the objects of interest and the Flying Distractors. A random number of lights of different types are inserted at random locations, scene is rendered from a random camera viewpoint and the result is composed over a random background image. The resulting images, with automatically generated ground truth labels (e.g., bounding boxes), are used for training the neural network. In conclusion, the Domain Randomization approach has several benefits: the images are faster to create and they offer a large amount of variety which forces the neural network to focus on the important aspects of the problem at hand.

In the Blender Training Data Generation article [5] a synthetic data generator is developed in Blender with a very similar goal: the data is meant to resemble waste in nature which should be detected. Thier project makes a great guideline for us by using the Blender API with Python to develop an end product similar to ours. This is an excellent indicator for the feasibility of our project. Since the article is used for a similar purpose (waste detection) we will be trying to replicate their methods in our project. However, methods from other works can also be used later on to increase performance.

3 Requirements of Synthetic Data Generator

To define appropriate requirements for the project, we conducted meetings with some of the key stakeholders. The transcripts of these interviews form the foundation for many of the use cases for the system and some key requirements. Recycleye provided us with a very useful document which described the most important requirements with required criteria.
In 3.1 we describe all use cases. Subsection 3.2 lists all non-functional and 3.3 functional requirements. Functional requirements are prioritized using the MoSCoW method.

3.1 Use Cases

- The user can select at least one base model for each type of material.
- The user can select from multiple base models if a material has multiple shapes/appearances.
- The user can select background/environment to place object in (background should represent real life backgrounds).
- The user can inspect physical attributes of objects
- When a user inspects an object, the object will remain consistent in it's physical attributes with respect to the environment.
- The user should have an aerial view of objects (from top).
- The user can alter parameters to change the generated image (output).
- The user can easily expand model and material libraries.
- The user can quickly change lighting/illumination conditions given input parameters and create multiple variations.
- The user can change colours of models and create multiple variations.
- The user can change graphics/shapes of model.
- The user can create crushed objects with infinite permutations.
- The user can replicate crushing by repeating input.
- The user can place multiple objects in an image .
- The user can alter composition of object within generated image.

3.2 Non-functional Requirements

- Quality of CAD models will be similar or better than that of GTA V.
- Materials with multiple shapes or appearances will have multiple base models.
- Dataset output can be used for training with standard Mask-RCNN model.
- Image generation should take less than or equal to 5 seconds on GeForce GTX 1080ti.
- Test coverage will be greater than or equal to 75% meaningful branch coverage.
- Tests, environments, objects and features are well documented.
- The code has comments where and as needed.
- Every function will have at most 15 lines of code.
- Code will have low Sonarcloud Cognitive Complexity[1].
- Create a user guide to help the client.

3.3 Functional Requirements

The MoSCoW method[6] will be used to prioritize requirements. The method involves dividing requirements into 'Must have', 'Should have', 'Could have' and 'Won't have'. Must haves are essential requirements (primary). Should haves are secondary and will be done after primary requirements are achieved. Could haves are nice to have (tertiary/bonus requirements).

3.3.1 Must Have

- The user can select material of object.
- The user can select background/environment of object.
- The user can expand the library of materials and models (code should not be hard-coded).
- The code will be deployed on cloud as a server on VM or make a webapp.

[1]https://www.sonarsource.com/resources/white-papers/cognitive-complexity.html

3.3.2 Should Have

- The user can change illumination/lighting conditions for generated objects.
- The user can vary background image for generated data.
- The user can simulate crushing of object.
- The user can vary population in an image.
- Bounding boxes will be used to optimize data generation.

3.3.3 Could Have

- The user can vary multiple material types in an image.
- The user can mask items.

3.3.4 Won't Have

- The user can add CAD models from the interface.

Now that we specified the user stories, non-functional and functional requirements we have a proper understanding of what is expected from the product. In the next section we will discuss the design choices and their reasons.

4 Design of Synthetic Data Generator

In this section we explain the design choices made to meet the clients requirements and quality needs. Starting with the pipeline, we talk about the software structure and code quality.

4.1 Synthetic Data Generator Pipeline

The product exists of a front-end flask server which can take the request for a set of generated images. Here Recycleye can specify the parameters for the image composition before the rendering pipeline. The back-end is set-up as one pipeline consisting of 3 main stages.

In the first stage the models of the requested materials get imported and a random selection of models per material is made. Directly after importation a model goes to the crushing pipeline, which simulates the crushing of the objects such that there is variance between all imported objects, even between the ones based on the same model.

In the second stage the scene, in which all the different compositions will be constructed, is prepared. Border planes are attached to a background plane to form a box in which items are dropped with random orientation and location. To simulate the drop all objects and planes are transformed into rigid bodies.

The third and last stage is the rendering stage. For each image the objects get randomly placed above the box. Next the drop is simulated. Finally the image is rendered and exported together with the labeled bounding box coordinates.

4.2 Synthetic Data Generator Software Structure

We have implemented a Client-Server architecture in our software by using a collection of Python classes, a configuration file and a directory for Blender Wavefront[7] models. The project is distributed to the server, source and util components to best separate the responsibilities of different classes. The server component is responsible for the graphical user interface and validating the user input, the util component is responsible for parsing the hyper-parameters and the source component is responsible for the data generation. Figure 2 illustrates how the Python classes interact with each other. The server package contains a main class which imports the generate class. The generate class accesses the main class in the source package which then, parses the configuration file with parser class in the util package, imports object and scene classes to store information about the images to be generated and finally uses the Blender class which acts as an interface to pass information to a Blender instance running in the background.

4.3 Synthetic Data Generator Code Quality

We think that every high-quality software project has to have good code quality as well since good code quality assures that the code works properly, is properly formatted, is convenient to read. We as developers had certain requirements to adhere to in order to ensure the quality of our code. The client gave us a requirement that our code had to achieve 80% coverage. they also asked for us to use static code analysis tools, the company also wanted the code to be properly documented so their developer team can build on the codebase easier.

In the project specification by the client, it is mentioned that a cloud-based testing-suite that provides various code metrics should be used. As the client suggested we used SonarCloud. SonarCloud uses powerful static code analyzers to provide information about the codebase such as bugs, security flaws and other signs of poorly written code. While using SonarCloud we made sure that the final version of our code had a high rating in reliability, security, maintainability, and coverage since all of these ratings were *A* we passed the Quality Gate[8] which means that our project passes the good code quality requirements either set by the client or the default ones.

First and the most important way to ensure code quality is to test it. The client's project specification says that 70% code coverage is needed to know that it is functioning correctly. We chose branch coverage as our coverage metric since branch coverage ensures as many branches are hit as possible and since we are trying to find bugs, branches that the developer overlooks could be where bugs lie. We intended to write the unit tests for the code you wrote, but sometimes other developers can take over to help each other. Our test suite runs after each push to the repository and the pipeline should fail if the coverage is below the guideline. We achieved 87% coverage which made us pass the SonarCloud quality gate.

It is required that the code is properly documented. Proper documentation helps the user and future developers understand the code better and aid in future development since it easy to understand why is something done how it is or what does this particular method do. We have documented the code using doc-strings which describe what a function does, what it requires as arguments and what it returns. We also generated `.HTML` files in which you can view the methods and what they require. We also provided guides on how to use the software.

Another essential and needed code quality feature is the code style. A detailed code style standard named `PEP8`[9] was published by the creators of Python that fits the need of most Python projects including ours. To make sure the code we write follows the `PEP8` code standard we use the `flake8` module for Python. `Flake8` allows you to see errors such as a line being too long or a line not being indented correctly as an error and allows for an easy fix by the developer. We include the flake8 in the pipeline to make sure that each version is adhering to the standard. We added additional rules so the function length isn't too large, we changed from default value 100 to 16, we also increased the maximum line length from 79 to 100 since 79 is not enough if you have long variable names.

.

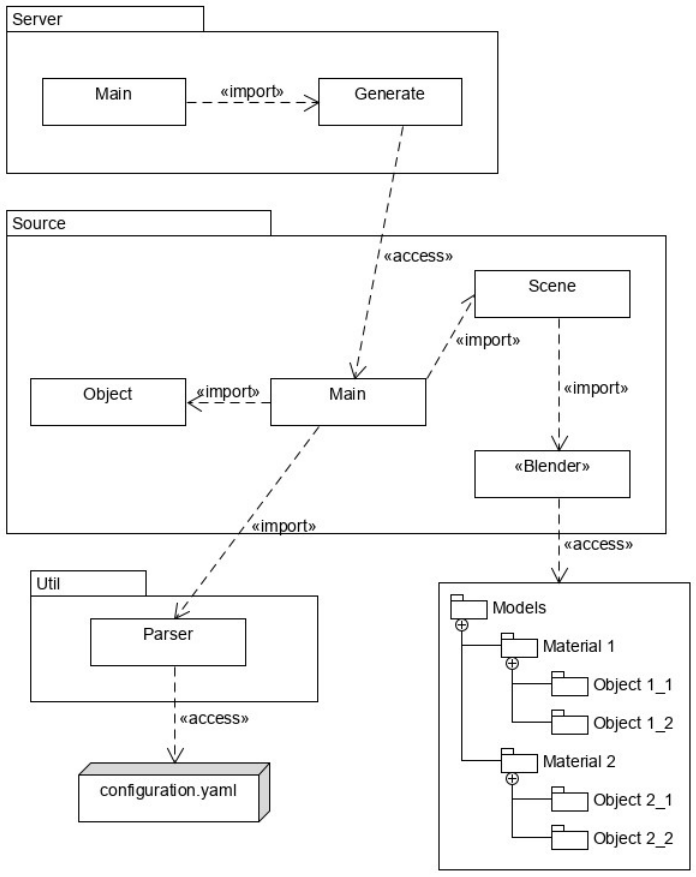


Figure 2: The software structure of our project. June 20, 2020.

The next section, Implementation will discuss the code we wrote and work we did in greater detail

5 Implementation in Code

Here we go in depth and talk about our implementations of various concepts in Python.

5.1 Hyper-parameters

Since our client intends to use our synthetic data generator on a machine learning model, we implemented our program to take a wide array of parameters. Implementing parameters that allow the user to control the generated dataset easily and exhaustively is crucial since the training data must be adapted to the existing model to achieve the best performance. We implemented two sets of hyper-parameters. Short-term parameters, which change very often and long-term parameters that rarely change, perhaps when switching to a different recycling plant. Both sets of parameters are parsed at the beginning of execution and then used in the code. We talk about them in more detail in the following subsections.

5.1.1 Short term hyper-parameters

Frequently changing parameters such as objects in an image or images to generate or chaning the ratio of objects in an image are some examples of short term hyper-parameters. These are inputted from the flask server and can also be done from the run.sh script.

5.1.2 Long term hyper-parameters

Rarely changing parameters such as camera co-ordinates and angle and light position, energy and type. To parse these inputs first a path to a `.yaml` file is specified (this allows the user to have different .yaml files for different scene setups) and then a Python dictionary is created from the file using the PyYAML module. Then that dictionary is used in the main method when creating a scene object.

5.2 Models

All about models, how they were made, and more about using and managing the model library.

5.2.1 Creation

Building 3-d realistic CAD models often have a learning curve attached to them, due to time constraints, we did not make the models ourselves. All models have been downloaded and credit has been given to the respective online free resources.

5.2.2 Photorealism

Recycleye will use the output of our Data Generator as input for their various Learning and Domain Adaptation algorithms. Hence, absence of realistic models will severely negatively effect later stages of Recycleye's pipeline.

To make our models behave realistically, attributes such as smoothness of surface, reflection of incident light rays, opaqueness, colour and other such physical attributes have been set using the Principled BSDF shader[2].
To make the objects look realistic, Image Textures[3] have been used. This allows applying an image to a selected group of triangles.

5.2.3 Randomized Skins

To improve the quality of data generated, skins on models are randomized. For instance, a plastic bottle has multiple skins i.e. brands such as Coca Cola, Thumbs Up, Fanta and so on. The client did not want to have the ability to specify the skin and therefore, it is randomly selected from the library of skins stored in the config file.

5.2.4 Expanding the Library

Recycleye specifically requested that the library of models should be easily expandable preferably using a configuration file. This has been achieved, the user needs to create a new directory in the `Models` directory, add the `obj`, `mtl` files and skins and finally update the config file which stores all skins.
If the client wishes to simply add or remove skins from the existing library, the respective action needs to be performed only in the config file. This will stop the particular skin from being used until re-added to the config file. To permanently remove the skin, delete it from the `Models` directory in addition to the config file.

5.3 Scene Setup

Aside from the non-crushed and crushed objects there are things that need to be set up in the Blender scene before rendering. Blender itself always loads in a default scene after startup which contains a cube object, a point light and a camera. From these we remove always the cube and use the camera and point in our scene. In the following subsections you will read how we setup the different scene objects to make our rendering realistic.

5.3.1 Light

In Blender there are certain types of lights. The options are 'POINT', 'SUN', 'SPOT', 'HEMI'and 'AREA'. The light object can be specified a location, energy

[2]Principled BSDF Shader
[3]Image Texture

level and type with the .yaml file as explained in 5.1.2. By using different parameters it is possible to create a lot of different images in different lighting conditions for training.

5.3.2 Camera

The camera in our scene specifies from where and which angle the image is rendered. As with the lighting in 5.3.1, the parameters are specified in the .yaml file. The possibilities to change are the camera location and rotation.

5.3.3 Background

The background is a very important factor for the realism of our image. We use a Blender add-on which makes us able to bind a JPG image on a plane as a texture. We then calculate the location and size of the plane using the camera coordinates, rotation and view angle. By doing this, we ensure that the background is positioned well and the size is proportional to maximize the realism of our future output images.

5.4 Physics

To make the images realistic a lot of things need to be taken into account, such as for example no intersecting models, random but realistic positioning of the elements and random but realistic crushing of the models. For these things the physics engine of Blender was used.

5.4.1 Positioning

To position multiple objects in a randomly yet realistically, we made use of the Rigid Body functionality of the physics engine. Real positions means no objects floating above the surface or intersecting each other. The rigid body functionality offers a way to simulate the objects being dropped on the conveyor belt. The model that are used in this project posed some problems however with the collision mechanisms of this functionality. Since the collisions were based on intersecting edges vertex-dense areas of the model caused the model s to slip through the conveyor belt or bounced from it. We solved these respectively by increasing the calculation steps per frame and by adding border planes which would prevent the models from falling off.

5.4.2 Crushing

The second part we used the physics engine for is the crushing of the objects. We had to come up with a system to deform the objects realistically and efficiently. In this section the crushing sequence and used modifiers are explained.

Blender offers a Soft Body modifier. When applied this allows for deforming an object by letting it collide with other objects on which a Collision modifier is applied. This could be automated but since there were two main obstacles. First and foremost the Soft Body modifier is a very expensive modifier when applied on more detailed models, which have many vertices and faces take too long to simulate efficiently. Second, our crushing sequence had to be reliable, the crushing needs to work on all models and not deform them to a level where they are unrecognizable. This would sometimes be the case if the Soft Body modifier got applied directly to the models.

An article on Caging [10] that uses the Soft Body modifier to imitate metal being crushed gave us an idea. This solution shows the concept of caging. Caging is the act of making a low-poly model which resembles the original model in a general way. Next this model is bound to the original model, meaning each vertex from the model is assigned to a vertex of the cage. This many-to-one binding is the part that takes the most time of the crushing sequence, but it allows for a faster simulation since the number of vertices becomes orders of magnitude smaller.

Another challenge was the automation of this caging sequence. Since all models differed in size and form we had to come up with a way to make an object of a similar form with a low vertex count. The solution we implemented was based on the Shrinkwrap modifier. This modifier can move the vertices of a model close to that of another model thereby 'wrapping' itself around the model. We placed the target model inside a big enough sphere that fully contained the model, next we applied the Shrinkwrap modifier and adjusted some properites such as offset so that we ended up with a simple cage where the vertices were evenly distributed over the whole surface thanks to the sphere.

After making a cage and binding it to the model using a Mesh Deform modifier we applied a Soft Body modifier to the cage. By changing the plastic property to its maximum of 100, the cage would keep its deformed state after colliding with certain objects. Increasing the bending and push property lead to a less extreme deformation allowing for more realistic crushes.

Finally the scene gets simulated for a certain number of frames. Then it is exported to another file of which a reference is returned to the main method which will use it during rendering.

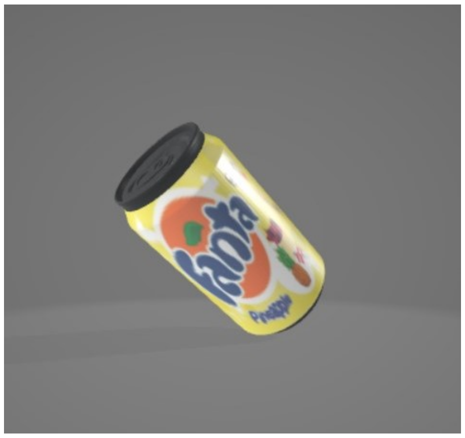


Figure 3: Image of an uncrushed can. June 20, 2020.

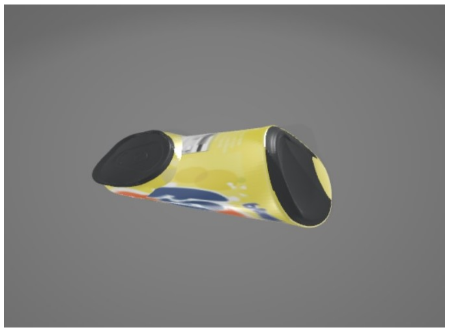


Figure 4: Image of a crushed can. June 20, 2020.

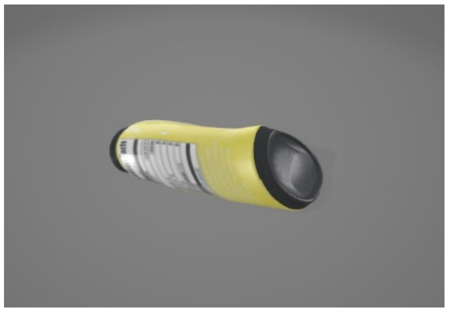


Figure 5: Image of a differently crushed can. June 20, 2020.

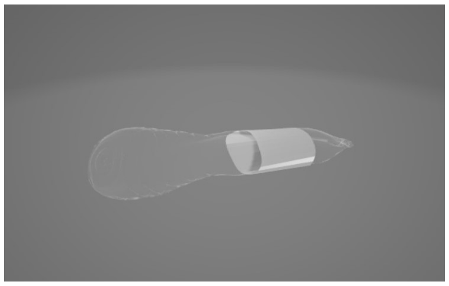


Figure 6: Image of a crushed water bottle. June 20, 2020.

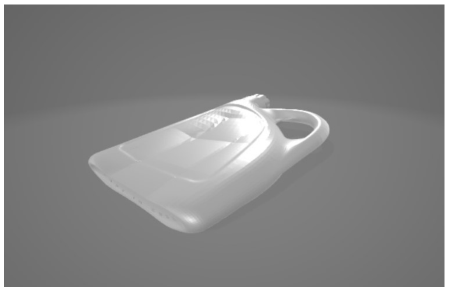


Figure 7: Image of a crushed wash bottle. June 20, 2020.

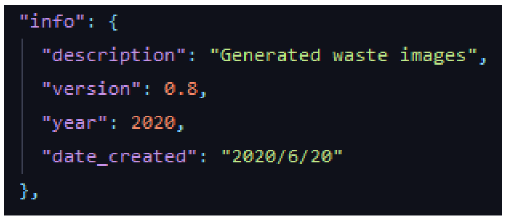
5.5 Data labeling

5.5.1 COCO Dataset format

The client provided a bonus activity that specifies that the data is labelled according to the COCO dataset format which describes pictures for object segmentation in a `.json` format. The `.json` file should include information about the dataset such as information in figure 8. Similar to general information and additionally date captured, resolution and the name needs to be specified about the images. The categories need to be provided if annotations are, the categories include the categories with their identifiers, name, and supercategory. The name is the material and the supercategory is always material in the current implementation.

5.5.2 Annotations

The annotations require you to provide the category id of the object, we find these from the configuration file, providing material for the object name, and the category id for the material. For the segmentation, we need to specify 4 vertices, and since we have the top corner from the bounding box, we find the other ones while adding the width and the height as needed. The function that calculates the bounding boxes was found online since the one that we developed didn't achieve the necessary results. Sadly it still doesn't quite align the bounding boxes so the function that generates them needs to be worked on, to improve the bounding boxes. Other than that the information about the dataset is generated correctly according to the COCO format.



```
"info": {
  "description": "Generated waste images",
  "version": 0.8,
  "year": 2020,
  "date_created": "2020/6/20"
},
```

Figure 8: Image of the information field in the `.json` file. June 20, 2020.

5.6 Deployment

One of our goals for this project is that we develop an application that is easily deployable and maintainable. One of the major issues here was that we need to run our python scripts inside an actual installed blender. We managed to achieve this by using a docker container which contains an actual installed blender. Another issue was regarding how the projects was used. Calling a bash script while inputting the parameters is not what the user wants to do so we deployed our software on a flask server that uses Docker.

5.6.1 Docker

As Wikipedia defines: 'Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers' [11]. Simply put, this means that Docker is a platform that can be used to run a specific container. A container is a process that runs on top of your host OS but is isolated from the rest of the system. Docker supplies all necessary files needed. This makes us able to create, deploy and run our applications using a container. The developer can add all his dependencies and libraries here to ensure easy deployment and maintainability.

We had the problem with Blender that it needed to be locally installed and the OS dependent path specified to be used in our python scripts. Docker solved this issue by using an image with Blender pre-installed. Our image is based on the Blender Docker image developed by The New York Times [12]. The Docker image runs a simplified Linux Ubuntu 18.04 as operating system specifically made for efficient use with Docker. The image installs all the necessary dependencies and packages as well as Blender itself. The working directory from the container is mounted inside the container so Blender has access to all the necessary files. We can thus run our scripts using this Docker container without a local build of Blender on host machine. The Docker image is specified in the Dockerfile which can be built and run anywhere at any time.

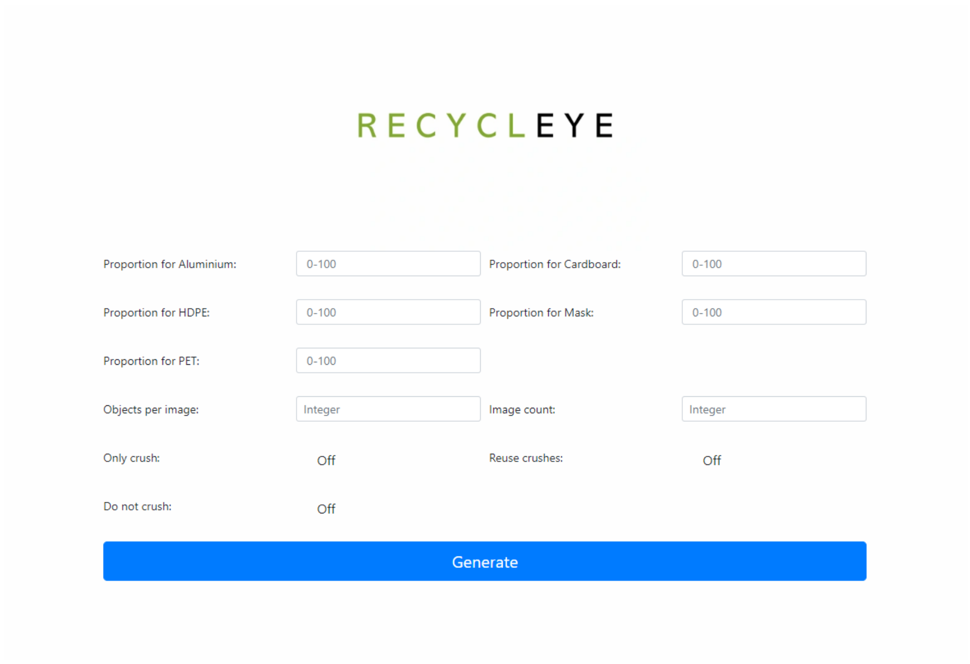
5.6.2 Flask server

To use our software the user would have to get used to all the parameters, be careful while inputting them and it could be a hurdle for some. To tackle this issue we decided to put all that functionality under the hood and provide a user with an interface. In which he can select the needed parameters and start generation of the data. With the client's suggestions we deployed it using a python module named flask which provides an easy way to start up your own server. As defined by the developers: "Flask is a lightweight WSGI web application framework." [13]

When first implementing the flask server flask_bootstrap module seemed like the right appraoch to take since Bootstrap provides an easy way to style your HTML documents, unfortunately that did not go as planned as it seemed broken and did not work completely. What was done is just including the Bootstrap `.css` and `.js` files in the header of the HTML and it worked fine and allowed for the use of Bootstrap classes which in turn made our HTML pages look much better and allows for easy improvements since the HTML files don't get too confusing.

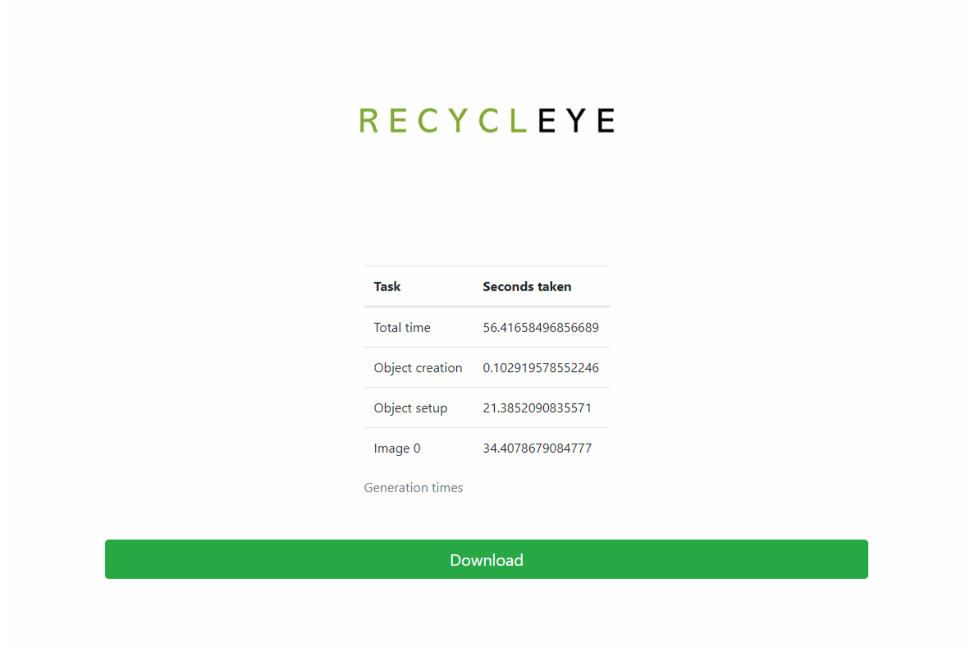
There are three views in our server: index, generate as seen in figure 9, and download as seen in figure 10. The index view shows options of what you can do, generate data, and download the last generated data. When you select the generate data option you are then transferred to the generate view and you can then put in the parameters for the materials and the proportions and for other options as well. Then the server calls the Docker bash script with the input parameters and you get to the download view when the generation is finished where you can download your generated data.

This was a really long and very important section of our report. It is very hard to conclude in a few sentences as this section is designed to describe the many features and aspects of our Synthetic Data Generator in great detail.


RECYCLEYE

Proportion for Aluminium:	0-100	Proportion for Cardboard:	0-100
Proportion for HDPE:	0-100	Proportion for Mask:	0-100
Proportion for PET:	0-100		
Objects per image:	Integer	Image count:	Integer
Only crush:	Off	Reuse crushes:	Off
Do not crush:	Off		

Figure 9: Image of the generate view of the Flask server. June 20, 2020



Task	Seconds taken
Total time	56.41658496856689
Object creation	0.102919578552246
Object setup	21.3852090835571
Image 0	34.4078679084777

Generation times

Figure 10: Image of the download view of the Flask server. June 20, 2020

6 Product Discussion and Recommendations

A critical analysis of our product, suggestions and future plans.

6.1 Initial Plan

From the get go, the goal was lucid, a simulator that mimics waste in a recycling plant. The client specifically talked about the importance of automation of our work for Recycleye's pipeline, emphasized on the ”easy to use” factor and high code quality in terms of architecture, design decisions, implementations and frameworks used. A key metric was our documentation, code smells (Sonarcloud) and tests.

After working on the project for 10 weeks, we have created a simulator in Blender that outputs realistic data with a high degree of automation, is very easy to use, has excellent documentation and most importantly, is flexible such that the client should not face any major trouble in absorbing our Synthetic Data Generator into their pipeline, or deploying it where and how they choose to.

6.2 Critical Product Analysis

Here we look at our Synthetic Data Generator from a different perspective as used before, we will consider the long term implications and look at the larger picture.

A user can input parameters that change frequently through the interface set up using Flask and Django. Other parameters that don't change as often, can be configured through a yaml file. This setup allows the client to easily use our work and integrate with their own pipeline, automate but at the same time, allow the user to change as many parameters as possible in a user friendly way. It may be nice to manage the model library from the interface, however it was part of our Won't haves in MoSCoW method of prioritization in section 3.3.4.

An effort has been made such that our code can be used by the average person. The user can input materials, ratio of objects, number of objects per image and receive an output with a certain number of objects per image with the specified material ratio constraints. Individual objects will have different skins such as a Coke can, Diet Coke, Red Bull, etc. This gives our client variation in terms of skins and number of objects, this is very good as the client will use our output as input for Machine Learning and Domain Adaptation algorithms. Some models such as plastic boxes come in many different colours, they are assigned random colours. A larger library of skins will improve the quality of data, however, it would mean the training size of data will increase and by extension, training time. Fortunately, the former is not a problem as data is synthetically generated.

To help the client with future learning algorithms, our data has been labelled. Models are arranged in sub-directories corresponding to materials, inside a directory dedicated to models. This structure is consistently used as it provides a simple yet efficient way to organize models while providing the option to modify (expanding the library that includes adding and removing skins of an existing model, adding new models and skins and new materials) the current structure to suit future unseen requirements the client may have. The skins are stored as png images, the paths of the skins are stored in the config file.

We currently use one crushing algorithm with a spherical cage. A sphere is used as it's the best "One Size Fits All" for our models. In future implementations, changing the cage to better match the geometry of the object would improve the crushing to be more realistic.

The only Must Have we failed to meet is rendering an image in 5 seconds. Crushing in real time is computationally expensive, realistic crushing is harder to achieve. We achieved realistic crushing and significantly reduced the render time by re-using crushes. As the number of objects in each image increases, the render time increases logarithmically instead of linearly. While requiring more time when there are fewer objects, our code is very scalable because the render time for a large number of objects is not as large as it should be, meaning, the cost of all important scalability is the 5 second render per image. As a very well documented requirement, we have failed to achieve it, but delivered something equally if not more valuable in return.

6.3 Recommendations

We believe we will be giving our client a reasonably high quality product but there is always room for improvement.

- Library of models can always be improved
- Crushing can be done in many different ways with many different algorithms resulting in different output
- Specifying the degree of crushing or alternatively, choosing a random value so that a greater variation in data generated is achieved
- Using a cage which takes advantage of object's geometry for crushing will improve crushing and make it more realistic
- Managing the model library from the interface is an option but not in the scope of the current project

7 Discussion on ethical implications

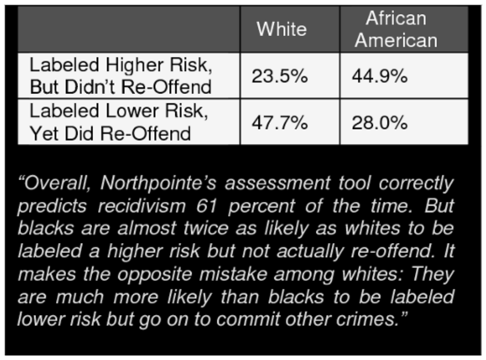
In our project we have numerous ethical subjects to talk about. In this section we will discuss those issues in the current industry and remarks that we think are relevant in our case. We will talk about how our design might exacerbate these and present our ideas to take them into account in our project. A discussion of how we might invoke them and recommendations for how those issues may be avoided is also sufficiently included. First we will include a short summary about our project.

In this software project we will help Recycleye, a company that develops computer vision algorithms and is operating within the recycling industry. We do this by developing a program that automatically generates synthetic data images based on certain parameters that can be used for training classification algorithms. With this data they can further improve their recognition software and make the world a cleaner place.

One of the major ethical problems in the computer industry is privacy. The privacy issue focuses on the computer's most basic functions, "its capacity to store, organized, and exchange records"[14]. The concern here is that computers made it possible to gathers amounts of information and this places people's personal information in a valuable position [15]. If someone with bad intentions can access this sensitive information (e.g. by hacking the system and infiltrating the database) this will have major issues on the privacy of the user. The malicious user can use this vital information for bad use.

Our application itself will only be used as an internal tool inside the Recycleye organization. Because of this fact we won't have major issues with user data and privacy of external people. This doesn't mean that we won't have attention for the privacy of the people of Recycleye. We take this into consideration in our current implementation by not asking for personal data of the user or even storing a single bit of identification data in a database. With this solution we can state that we avoid privacy issues and guarantee safe use in terms of user data. If for some case, there is an extension of our product in the future where the user needs to be authenticated or recognized, we can't avoid storing user data. The solution to avoid the violation of the ethics of privacy is the use of a secure database and possibly encryption. We expect the company to not abuse their own employee's data so this won't be a problem in normal circumstances.

Data is used in numerous places to train machine learning algorithms to automate different tasks. The ethical implication here is that data is not chosen accordingly we could create bias in the machine learning algorithms. Yapo and Weiss explain the different problems regarding the bias in their paper [16]. There are numerous examples of these, e.g. a machine learning model that predicts the reoffending of inmates in prison. This algorithm had a clear racially-based bias as can be seen in figure 11. This example indicates the importance of this issue and thus we need to take that into account in our project



	White	African American
Labeled Higher Risk, But Didn't Re-Offend	23.5%	44.9%
Labeled Lower Risk, Yet Did Re-Offend	47.7%	28.0%

"Overall, Northpointe's assessment tool correctly predicts recidivism 61 percent of the time. But blacks are almost twice as likely as whites to be labeled a higher risk but not actually re-offend. It makes the opposite mistake among whites: They are much more likely than blacks to be labeled lower risk but go on to commit other crimes."

Figure 11: Biased reoffending chance prison inmates [16]

Our data itself needs to be appropriate and display the recycling industry evenly. We do this by making sure to add a lot of relevant CAD models that are used in all environments Recycleye operates. So e.g. if we only use CAD models from high-quality products as they are used in rich areas, we will have trouble using this algorithm in poorer areas. We thus need to make data to train their model according to the use and environment of deployment. A design choice that is helpful to mitigate this problem here is that we made it relatively easy to add new models to influence and potentially remove bias towards certain items. It is as simply as just adding it to the materials directory. Removing objects that cause a bad balance and bias are as straight forward as just removing them from the directory.
Another solution could be to train different algorithms that can be used in different places and circumstances. This means that there needs to be done research every time Recycleye wants to install their product in new territory. With this research, the data can be tweaked and adapted based on the situation. We recommend using the first solution if this is feasible but otherwise the second with proper research and adaptation.
Certain researchers who wrote a paper on the training of a waste classification and recycling system [17] did some testing with an unbalanced test set. They came to the conclusion that the minority class is harder to learn and the model tends to over forecast the majority class with highly skew data. This logical consequence is what we need to avoid in our algorithm by the good balance of data as stated above.

As lastly, we would like to point out that our project as a whole has ethical implications. We are helping a company which is trying to reduce the waste quantity to a minimum by improving the recycling process using computer vision. This noble cause of indirectly saving the environment is by itself ethically responsible and we are happily taking part in their mission.

8 Conclusion

For our Software Project, Recycleye requested a data generator to synthetically generate data in the form of images to be used in the recycling industry. The objective of this report is to describe how we tackled this problem by introducing the reader to all the steps we took to complete the project and arguments to motivate our decisions.

The first two weeks were invested in brainstorming, researching on our topic, previous work, choosing software (Blender vs Houdini), choosing programming language (Python), conducting a feasibility study of the project, formally writing and prioritizing requirements of the client and other such activities aimed at long term success of the project. There were multiple interactions with the client to seek clarity as described in section 2 and 3. Armed with our plan, enthusiasm and meticulous research, we starting writing code. The features and implementations can be found in4 and 5. In section 6 we critically analyse our work and talk about the future. Section 7 concludes with our view on the ethical implications that our product creates and how they are suitably dealt with. Every step we took is documented with our arguments to support our decisions, individually and collectively in the depths of this report.

Overall we firmly believe that we will close the curtains on a successful project, we are proud of our work but most importantly, just by the sheer amount of knowledge and working experiance gained (being a first time for many of us), the course objectives have certainly been met.

An effective feedback loop consisting of our client and TA, we never lost sight of the larger picture or let the quality of our code fall. We, as a group worked very closely, together, and brought out the best in each other to develop a product that fulfils the requirements of the client in excellent proportions. As a matter of fact, we fulfilled every fundamental requirement and almost all bonus features. The user has all more options than requested on our Flask server in a user friendly interface. We implemented realistic crushing and our output is industry standard for image classification. The only requirement we did not achieve was 5 seconds per image render because of the bonus features we implemented on a GTX 1080 ti with real time crushing and has been explained in greater detail in 6. An example image can be seen in figure 12.

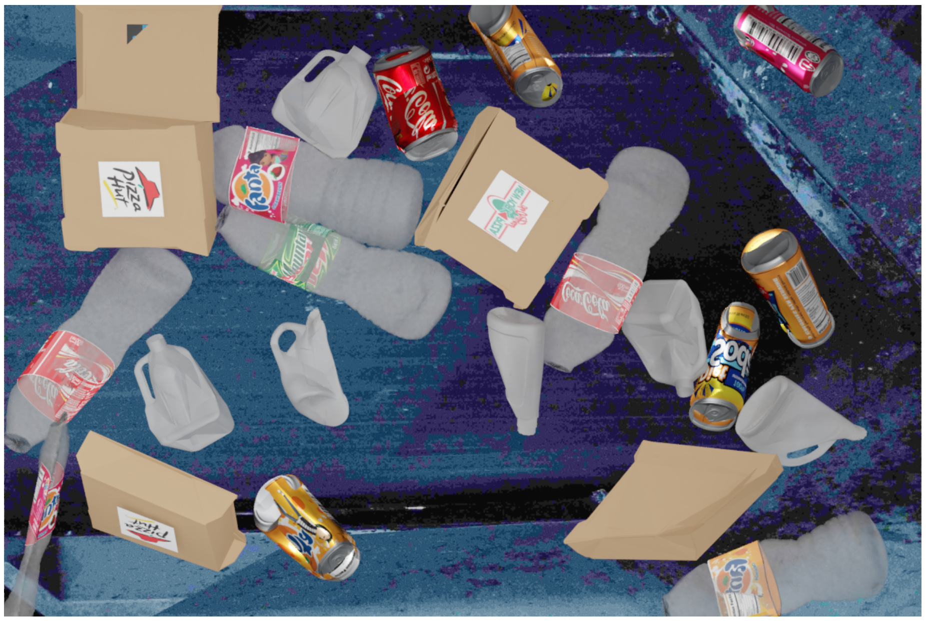


Figure 12: An example of a generated image. June 21, 2020.

We would like to end this report with recommendations to improve the product in the future.
The first improvement is definitely expanding the models, materials and backgrounds directory. Our product is designed to adapt and include these automatically.
The second improvement could be to explore different crushing algorithms to improve the performance and meet the 5 second mark in real time crushing.

References

[1] X. Peng, B. Sun, K. Ali, and K. Saenko, “Learning Deep Object Detectors from 3D Models.” https://arxiv.org/abs/1412.7122.

[2] K. Sarkar, K. Varanasi, and D. Stricker, “Trained 3D models for CNN based object recognition.” https://www.researchgate.net/publication/315857394_Trained_3D_Models_for_CNN_based_Object_Recognition.

[3] B. Sun, X. Peng, and K. Saenko, “Generating large scale image datasets from 3d cad models,” in *CVPR Workshop*, 2015.

[4] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, “Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization.” https://arxiv.org/pdf/1810.10093.pdf.

[5] “Blender training data generation.” https://olestourko.github.io/2018/02/03/generating-convnet-training-data-with-blender-1.html, journal=blenderCovNet.

[6] https://en.wikipedia.org/wiki/MoSCoW_method.

[7] https://en.wikipedia.org/wiki/Wavefront_.obj_file.

[8] “Quality Gates I SonarQube Docs.” https://docs.sonarqube.org/latest/user-guide/quality-gates/.

[9] “PEP 8 – Style Guide for Python Code.” https://www.python.org/dev/peps/pep-0008/.

[10] D. Zerba, “Crushing Metal Using Soft Bodies In Blender.” https://davidzerba.wordpress.com/2012/09/30/crushing-metal-tutorial/, September 2012.

[11] “Docker (software).” https://en.wikipedia.org/wiki/Docker_(software), Jun 2020.

[12] juniorxsound, “nytimes/rd-blender-docker github repository.” https://github.com/nytimes/rd-blender-docker, Jun 2020.

[13] “Flask (software).” https://palletsprojects.com/p/flask/.

[14] Johnson and D. G. E. Cliffs, “Computer Ethics,” 1985.

[15] B. Lou, “Ethical Problems in Computing.” www.comptia.org/blog/ethical-problems-in-computing, May 2020.

[16] A. Yapo and J. Weiss, “Ethical implications of bias in machine learning,” in *HICSS*, 2018.

[17] Y. Chu, C. Huang, X. Xie, B. Tan, S. Kamal, and X. Xiong, “Multilayer hybrid deep-learning method for waste classification and recycling,” *Computational Intelligence and Neuroscience*, vol. 2018, p. 1–9, Jan 2018.

Appendix A: Original project description

Recycleye is a computer vision company in the recycling/waste industry. The industry at present uses a large amount of manual labour to sort waste, often in poor conditions, high staff turnover and expensive operationally for a waste plant. Recycleye is aiming to replicate human vision through algorithms and replace human hands with robotics. The main problem is data, not only must the model detect different types of waste but also in numerous crushed permutations. The project specification aims to alleviate this by generating unlimited synthetic data for training a detection model to classify waste.

Project Specification: Given a photo of a collection of products, synthetically create numerous crushed permutations of these objects on a conveyor. Train a detection model on the synthetic data and optimise for the best detection model performance for detection.

Appendix B: Individual reflection on the project

Abel Van Steenweghen

Brief introduction

In this individual reflection I go over how the project went, which conflicts we encountered, how we resolved them and what my role in these situations was.

The goal of the project was to develop a synthetic data generator for Recycleye, a company that aims to automate the recycling process by applying machine vision and robotics to sort waste. With this synthetic data generator they can train a neural network for recognizing and classifying the waste.

For this development we mainly used Blender's Python API called 'bpy'. The final product is hosted on a flask server and run within a Docker Container for easy deployment.

Our team consisted of five 2nd year bachelor students CSE at Delft University of Technology. This project is a part of the CSE2000 'Software Project' course.

Task-related conflict

Because of the Covid-pandemic the circumstances in which we worked on this project were different than prior projects. This presented us with some extra obstacles. The biggest obstacle in my opinion was the sub-optimal communication that comes from not being able to discuss in real life. Communication is vital in a project with many people and solely using the medium of video-calls, chat-groups and issue-boards proved a difficult challenge. This had as a consequence that the most task-related conflicts were caused by communication errors.

For example the development of some features was done twice. The solution for this was to make better use of the assign functionality and the Kanban Board in GitLab. This allows for keeping a clear overview of who is doing what and what the current stage of that feature is. Also an important lesson was to finish every sprint meeting with a quick recap of what each member of the team would do before the next sprint meeting.

Another problem was that some code got refactored too often because of different interpretations, this is a good example of a task-related conflict. To solve these difference in opinion a video-call showed to be the best method to get everyone on the same page.

Intragroup conflict

There were no severe intragroup conflicts that lead to undesirable situations. The group dynamic went smoothly over the whole course. There was some social loafing when features took longer than expected to develop, which lead to frustration. This was solved by having clear communication about the expectations per member of the group and coming to a compromise on the development milestones.

Intervention techniques

This process has taught me a lot about how to anticipate possible conflicts and how to intervene efficiently when they do happen. An example of these interventions techniques is requesting clear statements of all team members about what they are expected to deliver. Another is that when the opinions on a feature differ every member receives the chance to speak out his thoughts. Next, the advantages of different methods can be compared and a general compromise can be made.

In a nutshell, the communication challenges posed by having to rely solely on non-physical presence made this project the most challenging for me. I personally tried to keep an eye on the progress and tried to manage the project in a way that would prematurely solve conflicts and still lead to a great final product. The project provided me with valuable experiences that can be taken into a next project with similar limitations.

Simon Mariën

In this section I will discuss my personal experience in this projec tregarding confilcts and intervention techniques. I will present a reflection on what went well and what caused problems in this group environment. The ways we avoided problems and solved them are also sufficiently dicussed. First there is a brief introduction on our project subject and team.

Brief introduction

For Recycleye, a computer vision company in the recycling industry based in London, we developed a synthetic data generation algorithm to train their classification models. Using a free and open-source graphics engine called Blender we can generate rendered images based on altering several parameters. The output data will meet the industry standard and contain bounding boxes to ensure high-quality data to train sufficiently. We developed using the python programming language, several libraries, Docker and Flask to make our application easily deployable and maintainable.

For this software project we are working as a group of 5. The expertise we individually bring is more specifically our software development skills and programming experience. The task we will do as a group is the specific software project I described above. Our project has a timespan of 10 weeks to be completed. This brings us to the conclusion that we are a so-called project team, which means that we are a work team, in which people with certain expertise are brought together to perform a task, with a defined period of time[4].

Task-related conflict

In a project team it is extremely important to not get in conflict with eachother. One type of conflict is task-related conflict. We will discuss in this section what task-related conflict is and how we got it to a minimum and solve it.

In a business or organization, task conflict occurs when two parties are unable to move forward on a task due to differing needs, behaviors or attitudes[5]. This means that due certain circumstances, the group dynamic and flow slows down and causes project delay. In a project with a strict deadline and timeframe this is a major issue so we should do as much as needed to avoid it.

I personally didn't really experience a lot of task-related conflict in our project. One time in particular I experienced something small regarding this which I will explain in detail now. When we were working on setting up the test coverage I did some work that was already done by someone else. This implementation was a major issue and took a lot of time because of the dependency that all

[4]Sundstrom, 1999

[5]What Is Task Conflict? - Definition Explanation, 2016

our Python scripts need to be ran through the graphics engines (Blender) own Python environment to get access to the necessary library for scene manipulation. We had a lot of trouble setting this up and after some time we asked our TA for help. Then my group members managed to get it to work but I wasn't really aware of this because this task wasn't assigned to me. That evening I tried to get it to work on my own to help them and spent some hours on it. Finally I got it to work but then I noticed that it was already done. This was a learning experience for me which could be prevented by inspecting our GitLab repository or asking my assigned team members.

To prevent task-related conflict we kept in close contact through our Whatsapp group and regular Jitsi meetings. This way we kept eachother uptodate with our progress, potential problems and further planning. We also made use of the GitLab issue board for assigning tasks, avoiding double work and not forgetting important requirements and tasks. I personally think that the 'estimate' feature from GitLab issues is very handy, it helps a lot with dividing tasks evenly and planning.

Intragroup conflict

Another type of conflict in group is intragroup conflict which is in my opinion worse then task-related conflict. It has a negative influence and is more personal related instead of task related. There are several roots of intragroup conflicts: conflict over power, resources, control, competition and personal issues. E.g. there are two group members that both want to be the leader, this is a power conflict.

Luckily I personally didn't experience any kind of intragroup conflict in this project. We worked with each other professionally and respected everyone's value's and opinions. Everyone was treated equally and no conflict emerged for leadership or something of this sort. When it would have emerged we should first try to solve this in our group with a good conversation to express our frustrations. When this approach doesn't resolve the conflict we could contact our TA or a course staff to help to find a solution.

I think conflicts can't be fully avoided at all time but we should try to. In the best scenario this should start at the beginning of the project with a conversation to try to learn everyone's expectations and value's. Then there can be agreed on certain plans and rules to do the project. E.g. already choose a team leader or a responsible person for each part. This will avoid future conflicts around these things.

Intervention techniques

There are intervention techniques for certain conflicts. Because we didn't really experience intragroup conflicts we hadn't really had a chance to put them into

action and discuss it in this section. Instead I will give some examples with potential conflicts and a possible resolution for these.
When there is a case of social loafing (perform less then expected on a task in group) we could make every task identifiable and keep track of everyone's work effort to identify problems.
Another way to avoid social loafing is identifying the people we actually do the expected amount and possibly rewarding them. With this technique, people who perform less hopefully also want this and try their best.

Rahul Kochar

Here I will pen my opinions on the project in the form of an Individual Reflection on the Project.

Brief Introduction

Our client, Recycleye, is based in London, UK. They aim to make the world a better place by increasing efficiency and reducing costs at waste recycling facilities by using Computer Vision. Recycleye was represented by Peter, and sometimes Oliver (rarely).
Our team of five, Abel Van Steenweghen, Augustas Vilčinskas, Liudas Mikalauskas, Simon Mariën and I are 2nd year students at Delft University of Technology. Our TA is Max.
We will help Recycleye by building a Synthetic Data Generator in 10 weeks that will create input for later stages of Recycleye's pipeline which includes various Learning and Domain Adaptation algorithms.

We used Python, important frameworks are libraries used are bpy (Blender Python), Flask, Django and Docker. Our models are made in Blender, a free and open source application. The user will be able to specify parameters based on which realistic images will be generated.

Task-related Conflict

In every project and plan ever made, optimal usage of resources is vital. It is important to assign everybody well defined roles and responsibilities to prevent task related conflicts due to overlapping work. Sometimes, it is necessary to work in smaller teams of two or three, this can create problems when there is a divergence of ideas, thoughts and opinions on implementation of a feature, usage of a feature and so on.

We used a Scrum Board/Issue Board and Weekly Sprints to work together. Issues were made and open for selection, somebody who has finished his/her work will choose another open issue. The submitted work will be reviewed by at least another two members of the group before being accepted. This prevented any task related conflict while working individually as the person doing the work had complete independance in making whatever decision he/she needed to. During the review stage, the two reviewers expressed their opinion and the person could choose to accept the suggestions or provide a valid counter-arguement. Personally, I have received some very good suggestions which improved the quality of my work and I hope by suggestions were equally helpful for others.

At the start of the project, we were required to setup Blender (initially used locally installed build), a key component of our project. It was however very platform dependent, I used Linux, my peers used Windows and MacOS, our

client may have some other constraints or might deploy in an environment under different conditions, each of us had a different setup and we needed a solution to rise above all this. We did not fight about it, we could have, we just accepted the minor inconvenience of changing a line of code every time. At that point I knew that I was working with professionals. The solution was easy, Docker (suggested by Max, our TA). A project of this scale (15 ECTS) and for an actual client instead of the mock projects we have done earlier is a first time for many of us. We did not have any significant experiance with Docker or other such frameworks and solutions in general. Fortunately, we were all willing to learn and accept suggestions/advice in a positive manner to improve our code and knowledge, and not as criticism.
Once while working with a team mate to get some work done (I will not give more details for obvious reasons), I felt my opinions were not being given due consideration. After re-phrasing my ideas, we managed to come to an agreement. It turned out that my colleague did not correctly understand what I was trying to say. Apart from this, I did not personally face any major task related conflict.

Apart from using a Blender image on Docker, there were multiple other such situations where a framework existed such as using a `.yaml` file to store configurations or to use the ArgParse library for input, they were all resolved in a similar manner as mentioned above, with Max or Peter suggesting an approach. However, it is not fair to call it a task-related conflict since nobody really knew an alternate solution, whatever StackOverFlow recommended, we blindly did. We mainly faced "Choosing a Solution" conflict from McGrath's model[**?**] but without the options, we had a problem, and we were fine with any working solution. This is not a good solution from many perspectives such as security, scalability, maintainibility, etc. Peter and Max correctly understood the situation and mentored us accordingly. We were given freedom to make our own decisions, but at the same time there were criteria we had to meet to maintain a certain level of quality in our work.

Intragroup Conflict

Intragroup conflicts are disagreements within the team, between two or more colleagues. These can potentially be disastrous for the team. Such conflicts are caused when there is a difference in opinion that is usually not related to the project or work. It may be a smaller issue in the larger issue of dislike and contempt for the other person(s). Eventually, this may leads to negative competition where somebody may try to undermine and/or outdo the other person and in the process adversely effect the other team members and project as a whole.

While working in smaller teams of 2-3 on a particular task, I did not experiance any trouble as my colleagues were always very rational and we discussed all conflicting ideas with rest of the team to take a collective decision. In conclusion, due to our use of Scrum Board, regular contact with team mates, TA and client through Jitsi meetings, Whatsapp and Skype, we did not lose sight of the larger picture and always respected the other person's opinion.

Fortunately, I did not face this issue in any way. I have always respected my team mates and their opinions, and been respected the same way by my colleagues. While it is not possible to not face any conflict at all, many times a difference in ideas helps us to improve our code. All suggestions must be objectively evaluated and discussed in a professional manner. It is very important to communicate effectively, and make simple clear logical statements to back your ideas/claims to convince other people that your idea/suggestion is indeed good.

Intervention Techniques

Different types of problems require different solutions. Since we didn't really experiance any such difficulties, we could not implement any strategy and then observe the impact.

Efficient planning and prioritizing with a healthy feedback loop mitigates most problems before they start. Assigning different members well defined responsibilities and roles is very helpful.
When there is an issue in deciding who the leader of the group should be, rotating leadership is a good idea. Even better, we used a decentralized structure with no clear leader. We trusted everybody else to be professionals, divided the work very nicely and did not have any issues.
The Scrum Board, weekly Sprints and meetings with the TA and client clearly bring out the work done since the last meeting. If somebody does not have any work to show, it would mean they are "Social Loafing". Fortunately, we did not have this problem, everybody worked hard. It is true that sometimes the work to show was incomplete because I was having some trouble finishing it, in that case in the meeting I was given feedback and alternate approaches to complete the work. It is important to point out that the work was never done for me, I was only given advice on finishing it.

Augustas Vilcinskas

Brief introduction

For the software project, I and my peers assembled a team and decided to work on Recycleye's project. Recycleye is a technology company that wants to tackle the issue of sorting recyclable waste in waste disposal facilities, which is a huge issue at the moment since this task is human labour intensive. The company utilizes computer vision, according to Wikipedia"Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do".[6] Using computer vision Recycleye aims to train a model to distinguish between recyclable waste, to train such a model a large amount of data is needed which can prove to be difficult to obtain.
That is where our project comes in. The company wanted a team of students to develop a synthetic data generation tool to improve the accuracy of their model. Synthetic data is exactly what it sounds like, it is images of waste that resembles real-life waste as in, it can vary based on colour, form, material. To generate such data our project team used a physics engine to simulate the real-life physics of the objects, a graphics engine to render such images and a front-end façade that the user can use to generate them with ease. Our team of 5 students worked on this project for a period of 10 weeks and managed to develop software that does what is needed but can still be built on further to improve the quality of generated data.

Task-related conflict

During the development it is normal for conflicts to arise, so did they in our group. First task-related conflict that arose is the way we use the Blender engine. We had mixed opinions if using it the way that we were was feasible since during the deployment this could render our software heavily platform dependent. Which was a negative thing, but on the other hand a simple solution that works is sometimes the best one. This conflict first arose in the early stages of development as for the Tuckman's model it would be during the transition from the first stage of group development, forming. During this transition as a group, we were still not active enough in voicing our thoughts and somewhat impartial about the project. Which correlates with the Tuckman's stage. Since a conflict arose, we moved to the storming phase of the model. A characteristic described in this stage are polarization and coalition forming which indeed did happen, a few members were more eager to change it and others not.

Since we disagreed on how to do something, it was a decision-making task and it had no clear or right answer so it was consequential that such a conflict arose. The task-circumplex model by McGrath aligns with our experience, it

[6]https://en.wikipedia.org/wiki/Computer_vision

was indeed a conceptual task that created a conflict so the description of the conflict of type 4 is the conflict we experienced. It is normal for such a dispute to arise in such circumstances since all of our team members are novices in software developing, and no one had enough knowledge to back up his viewpoint and there was no clear answer to the task. What made it even harder to resolve this conflict is the fact that the group was just recently formed and did not know or trust each other that well. No other conflicts occurred in the team since throughout the development we used an issue board to divide tasks and trusted that the person will deliver.

Intragroup conflict

Many project teams experience intragroup conflicts regularly, our team was no exception. Before the project even begun an intragroup conflict arose in which one member of the team wanted to start working earlier since his acquaintances who were also doing the software project had already started and other team members disagreed since they felt comfortable starting on the designated time. It is natural for such conflicts to arise since some people are more motivated and dedicated to the project than others. Fortunately, this conflict could be considered a healthy one. Since healthy conflicts arise from task-related circumstances and can help the project in the long run, so did this conflict. It helped team members think more about what is an appropriate time to start working since maybe in-fact it is beneficial to start early due to the complexity of our project. It would have been an unhealthy conflict if, for example, the member was concerned about leading and controlling the team instead of extending the timeline which gives us more time to do the designated task.

Another popular intragroup conflict that arises in many groups is social loafing, which means that certain members do not invest as much time into the project than others which decreases the group performance as it has a snowballing effect – certain member that invests more time will invest less if he notices that social loafing is present in the group. Sometime during the development period, I felt like I was the one not providing enough hence I took up more tasks and completed them, this increased my motivation and mental attitude towards the team. Other than that, not much social loafing was present in our team and later on in the project members had designated tasks in their preferred areas which helped us build a more complete system while working as a team.

Intervention techniques

Conflicts occur in every team but not every team deals with them correctly. Dealing with conflict correctly means that the conflict was deescalated as soon as possible and a common consensus was reached between the team members. A bad way to deal with a conflict would be to escalate it further while not providing any valuable resolution for it. This erodes the team morale and increases the process losses that decrease the overall performance of the team. For the task-related conflict I mentioned earlier we wanted to be transparent with our communication and decided to look for help so we sought guidance from our teaching assistant Max, and Peter from Recycleye. During a meeting, we brought the issue of how to use Blender in our project up and Peter advised us on using Docker to make our product easily deployable and not hardware dependent, we were not familiar with Docker and when we did look into it more, it seemed like a perfect solution to our problem and hence the conflict was resolved. This helped us grow in the future since we realized that we ought to use the resources we have at hand and communicate with our higher-ups on what the right course of action is. As for the intragroup conflict, the way I handled this conflict made me feel good since I am usually the quiet one and the way I resolve conflicts is through avoidance, which is not helpful to the team. This conflict was resolved by making a compromise with among team, all members looked more into when is an adequate time to start and after a discussion, the conflict was resolved and we decided to start right after the kick-off which was the first day of the project. This helped our team in a way that when the starting day came, we were ready and did start working the very same day and got on to a good start. Other conflicts within the group were resolved due to proper communication and trust, we trusted each other with the tasks that were taken up by ourselves based on our preferences or the necessity of the task and people came through which helped our project be better than it could have been

Liudas Mikalauskas

Brief introduction

Our project team did not have a name. When naming our private group chat, the group chat with our teaching assistant or the group chat with the client we either used the name “Recycleye”, which is the name of our client company or “synthetic data generation”, which is what our product does. The team consisted of five members, who are computer science students. The team was also supported by a coach, teaching assistant, technical writing expert, ethics professor. The goal of the project was to create a synthetic data generator designed to train a neural network. That neural network is an already existing model, that classifies different materials in images of waste on a conveyor belt.

Task-related conflict

According to Tuckman task-related conflicts in a group are unavoidable. One of the five stages of group development outlined by him is storming. This stage happens after the group is formed and it is characterised by disagreement over procedures and tension among team members. During this stage task-related conflict happens. Our group was not exception to the rule and we experienced the storming stage as well. Task-related conflicts in our group happened to the extent of doing the same issue twice, not knowing which person is doing an issue or not knowing what an issue exactly is. An example of not knowing which person is doing an issue would be the documentation issue. One time Simon asked me whether I was doing the documentation. I looked the issue board and saw that Rahul has assigned the issue to himself and added the doing label to it. That’s what I said to Simon. Then he said that Rahul was not doing it and he said that Augustas was doing it and Augustas said that I was doing it. In the end Simon did it. Not knowing who was doing this issue was caused by poor use of the issue board and not clear communication. Because of this we completed the issue much later than we could have completed it and lost some time. This caused tension among the group. Another example would be doing the same issue twice. This happened when both me and Simon were implementing the test coverage. Same as the last problem this was too caused by not using the issue board properly and poor communication. This too made us lose time and in turn created some tension.

Intragroup conflict

One intragroup conflict that happened in our group was social loafing. Social loafing is natural for any larger group. For example, I made my biggest spurts of work at the beginning of the project; However, as the project approached the end I started avoiding big changes and just nitpicking at code quality and testing using the philosophy ”don’t fix what’s not broken”. Other people acted differently. While they didn’t give a hundred per cent of their effort, in the beginning, the approaching deadline provided them with more motivation and

they started doing big changes towards the end. So the bottom line is no matter the reason whether it's motivation, state of the project or personal arrangements all members had weeks when they did social loafing.
Since this was a software development project and any task has multiple implementation ways our group encountered some approach conflicts too. They mostly happened at planning meetings in forms of discussion when one person had different ideas about implementing a task than the other.

Intervention techniques

The task-related issues were intervened by formulating our goals more clearly, improving our division of tasks and making our communication more transparent. These techniques were not applied outright but sort of came along step by step as we spent more time on the project and became more familiar with each other. These techniques worked out and made our group more cohesive.
The main technique we used for intragroup issues namely social loafing was making each team member identifiable. This was done in our weekly meetings by each member having to present his progress for that week. And in the rare case when a person had not merged any code on our repository that week the teaching assistant asked the person to explain the reason for that. This technique helped a lot to avoid social loafing and each person did some tasks each week.
The intragroup issue of conflicts regarding implementing approaches was solved by giving people autonomy. This way each person had a different aspect of the project he was responsible for and thus had the freedom to implement his ideas uninterrupted by others. This reduced the number of conflicts regarding the approaches for implementation.

Appendix C: Info sheet

Synthetic Data Generation

Provided by Recycleye

The company Recycleye utilizes computer vision to tackle the issue of human labour in waste recycling. Their recognition model as any is heavily dependent on data, that's where our team comes in, our project aims to synthesize data for their model.

Challenge

Our project at its core aims to provide realistic images of waste. To improve the realism the waste items should have the ability to be crushed, have realistically looking models. There should be different materials present in the images such as aluminium, plastic and so on. The project should also be deployed to allow easy use.

Process

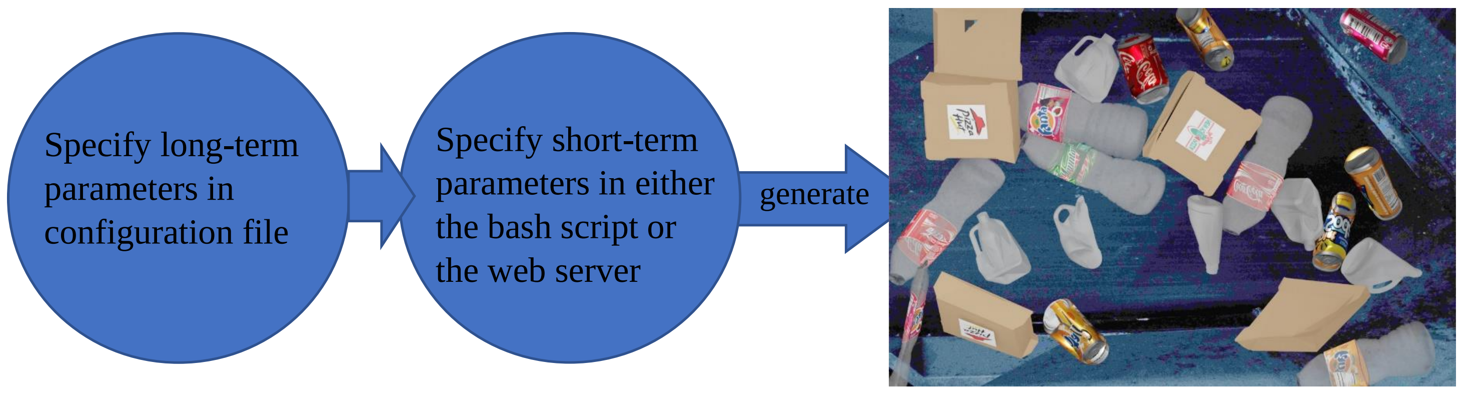
The process of our project was clearly defined since the client, Recycleye, provided us with a nice project specification with granular tasks that build on each other. While working in a SCRUM environment we worked in weekly sprints in which we tackled those issues. Each team member took up his task or formed small groups to work on different issues together at the start of the sprint and delivered either at the end of the sprint or continued working during the next one. We had weekly meetings with the Teaching Assistant and the client on Mondays and one with only the client on Fridays. During these, we discussed our progress, asked for guidance, and received feedback on the work done.

Product

The final product achieved all of the must-have requirements. It has a configuration file in which you can vary different parameters such as server configuration, lightning and so on. Other parameters such as object count, the background can be specified in the bash script and are taken as command-line arguments. Objects have been made to look more realistic. The product can be run from a bash script, but a web server where you can specify the script parameters and generate the images through the bash script and download them was developed. We tested the software using unit-testing and achieved the Quality Gate on Sonarcloud.

Outlook

The company gave us various feedback but is overall happy with our progress. We provided documentation on how to use it and how to build on it further. Developers working at the company plan to build on it if when the product is taken a deeper look into, it is deemed sufficiently good.


Specify long-term parameters in configuration file
Specify short-term parameters in either the bash script or the web server
generate

Appendix D: Division of labour

Task	Team members that contributed
🟥 Ability to crush an object	Abel, Simon
🟥 Ability to render a background	Simon
🟥 Images can be added to objects	Rahul
🟥 Create or implement the use of a Blender Docker container	Simon, Rahul
🟥 Make CAD models realistic	Rahul
🟥 Code base refactorizations	Augustas, Liudas, Simon, Abel, Rahul
🟥 Flask server	Augustas
🟥 Ability to randomize object location, orientation	Augustas, Liudas, Abel
🟥 Bug fixes	Augustas, Liudas, Simon, Abel, Rahul
🟥 Testing	Augustas, Liudas, Simon, Abel, Rahul
🟥 General documentation	Augustas, Liudas, Simon, Abel, Rahul
🟨 Ability to render an object	Abel
🟨 Objects of different materials can be rendered	Rahul
🟨 Ability to change lightning parameters	Liudas
🟨 Finding CAD models	Liudas, Rahul
🟨 Ability to change camera parameters	Liudas
🟨 Object color can be changed	Simon, Liudas
🟨 Fix Sonarcloud issues	Liudas, Augustas
🟨 Implement the use of the configuration file in different methods and altogether	Augustas, Liudas, Simon, Rahul
🟨 Annotations according to COCO dataset format	Augustas
🟨 Background improvements	Abel, Simon
🟨 Command line arguments	Liudas, Abel, Augustas
🟨 Time information	Abel, Augustas
🟩 Generate documentation HTML files	Simon
🟩 Initial Blender setup	Abel
🟥 Task required immense effort	
🟨 Task required effort	
🟩 Task required little effort	