

TECHNICAL UNIVERSITY OF MUNICH
Department of Materials Engineering

**SOFTWARE DEVELOPMENT INTERNSHIP
ADDITIVE MANUFACTURING OPTIMIZATION
ALGORITHMS USING PYTHON**

INTERNSHIP PROGRAM REPORT

Kaan Karataş

FALL / 2023

Table of Contents

1	INTRODUCTION	1
2	DESCRIPTION AND ANALYSIS OF THE INTERNSHIP PROJECT	1
2.1	Usage	1
2.1.1	PyCharm IDE Alternative Usage	2
2.2	Requirements	2
2.3	Main Function	3
2.4	Operations Function (Optimization Logic)	4
2.5	Plotting the Results	7
2.6	Example Cases	8
2.6.1	Bullet Object	9
2.6.2	Wheel Object	10
2.6.3	Truck Object	12
3	CONCLUSIONS	13
4	REFERENCES	14

1 INTRODUCTION

During my internship in the field of additive manufacturing, I embarked on an insightful journey into the world of 3D printing and materials science. This experience provided me with a unique opportunity to mix my prior knowledge of programming with the intricacies of 3D-printing for enhanced structural integrity and performance. My primary goal during this internship was to contribute to the field of additive manufacturing by developing practical programming solutions for optimization challenges.

Throughout the internship, I explored various facets of additive manufacturing, with a particular focus on the optimization of geometry orientation. I worked on developing algorithms and methodologies aimed at enhancing the structural integrity of printed objects. This involved an in-depth analysis of 3D-printing parameters, material properties, design considerations and algorithm concepts.

Furthermore, I had the privilege of working on a real-life project, where the outcomes of the optimization efforts will have a direct impact on the efficiency of additive manufacturing processes. The successful optimization of 3D printing parameters and object orientation will not only reduce support material waste but will also enhance the overall quality of printed parts.

2 DESCRIPTION AND ANALYSIS OF THE INTERNSHIP PROJECT

The key highlight of this internship was the development of an optimization algorithm tailored to improve the orientation of 3D-printed objects. This algorithm aimed to minimize overhangs, reduce support material usage, and enhance the overall quality of printed parts. By optimizing the orientation of objects throughout the printing process, significant progress was achieved in code testing, even in the absence of physical 3D printing trials.

As part of the project, I actively engaged in the analysis of 3D models and the utilization of mathematical and computational techniques to optimize the printing process. This hands-on experience equipped me with valuable skills in algorithm development, data analysis and additive manufacturing concepts.

2.1 Usage

To use the program, after changing the directory to the project, simply run it from the command line using the following format:

```
python main.py stl_file
```

Replace `stl_file` with the path to desired STL file. After executing the program using `python main.py stl_file`, such as `'python main.py examples/bullet.stl'`, it will determine the optimal rotation angles for maximizing 3D printing efficiency. This process yields an output in the following format:

Optimal rotation for `stl_file` is [radians] with value [value] in [time] seconds.

2.1.1 PyCharm IDE Alternative Usage

If you prefer using PyCharm as your integrated development environment (IDE), you have the convenience of running the program directly from the IDE itself, offering an alternative to executing it in the terminal. This approach proves particularly valuable if you intend to debug the algorithm, as it enables straightforward program execution and debugging within the IDE.

To set up this configuration, follow these steps:

- Open PyCharm and navigate to the "Run" menu.
- Select "Edit Configurations."
- In the Configurations dialog, click the "+" button and choose "Python."
- A new Python configuration will be added. In this configuration, you can specify various parameters and settings for your program.

The parameters can be seen in Figure 1. The value entered to "Parameters" is the name of the desired STL file input to the program.

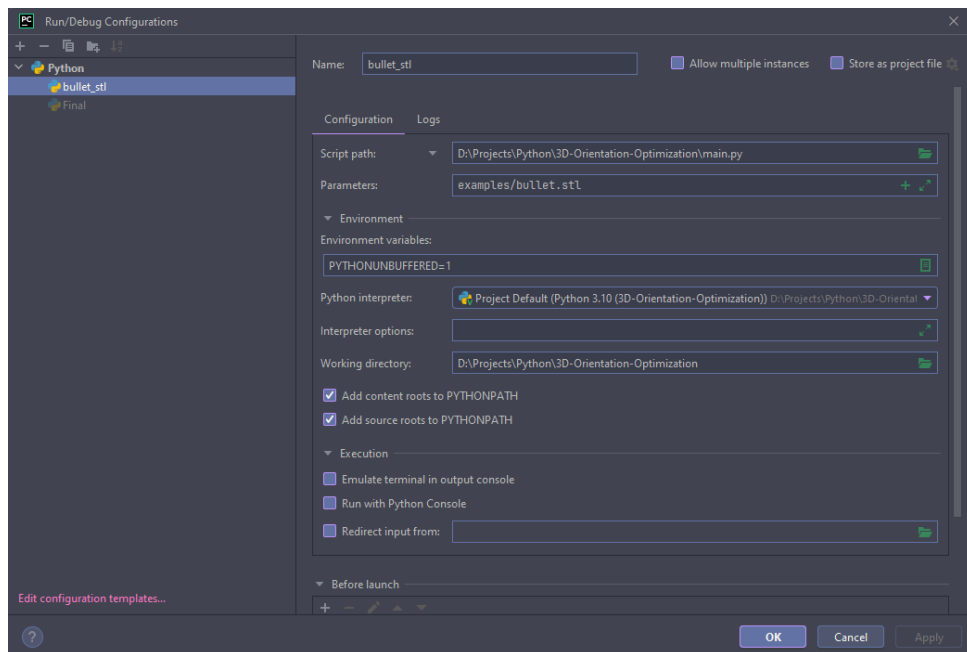


Figure 1: Custom Parameters for the Configuration

2.2 Requirements

When your project includes a 'requirements.txt' file within its directory, your IDE will typically provide an automatic pop-up to facilitate the installation of the necessary packages. While the core dependencies are imported throughout the code, a comprehensive list of all the requirements, are automatically crafted with the command below.

```
pip freeze > requirements.txt
```

The complete list can be found in Figure 2 as well as within the 'requirements.txt' file located in the project's directory. If the requirements can not be installed automatically, the code below can be used from the terminal to install them.

```
pip install -r requirements.txt
```

```
ansi2html==1.8.0
anyio==4.0.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.2.3
asttokens==2.4.0
asynclru==2.0.4
attrs==23.1.0
Babel==2.12.1
backcall==0.2.0
beautifulsoup4==4.12.2
bleach==6.0.0
certifi==2023.7.22
cffi==1.15.1
charset-normalizer==3.2.0
click==8.1.7
colorama==0.4.6
comm==0.1.4
ConfigArgParse==1.7
contourpy==1.0.7
cycler==0.11.0
dash==2.13.0
dash-core-components==2.0.0
dash-html-components==2.0.0
dash-table==5.0.0
debugpy==1.6.7.post1
decorator==5.1.1
defusedxml==0.7.1
exceptiongroup==1.1.3
executing==1.2.0
```

(a) Requirement 1

```
fastjsonschema==2.18.0
Flask==2.2.5
fonttools==4.39.3
fqdn==1.5.1
idna==3.4
ipykernel==6.25.2
ipython==8.15.0
ipywidgets==8.1.0
isoduration==20.11.0
itsdangerous==2.1.2
jedi==0.19.0
Jinja2==3.1.2
json5==0.9.14
jsonpointer==2.4
jsonschema==4.19.0
jsonschema-specifications==2023.7.1
jupyter-events==0.7.0
jupyter-lsp==2.2.0
jupyter_client==8.3.1
jupyter_core==5.3.1
jupyter_server==2.7.3
jupyter_server_terminals==0.4.4
jupyterlab==4.0.5
jupyterlab_pygments==0.2.2
jupyterlab_widgets==3.0.8
jupyterlab_server==2.24.0
kikisolver==1.4.4
MarkupSafe==2.1.3
matplotlib==3.7.1
matplotlib-inline==0.1.6
```

(b) Requirement 2

```
mistune==3.0.1
nbclient==0.8.0
nbconvert==7.8.0
nbformat==5.7.0
nest-asyncio==1.5.7
notebook==7.0.3
notebook_shim==0.2.3
numpy==1.24.2
numpy-stl==3.0.1
open3d==0.17.0
overrides==7.4.0
packaging==23.1
pandocfilters==1.5.0
parso==0.8.3
pickleshare==0.7.5
Pillow==9.5.0
platformdirs==3.10.0
plotly==5.16.1
prometheus-client==0.17.1
prompt-toolkit==3.0.39
psutil==5.9.5
pure-eval==0.2.2
pycparser==2.21
Pygments==2.16.1
pyparsing==3.0.9
python-dateutil==2.8.2
python-json-logger==2.0.7
python-utils==3.7.0
pywin32==306
pywinpty==2.0.11
```

(c) Requirement 3

```
PyYAML==0.0.1
pyzmq==25.1.1
referencing==0.30.2
requests==2.31.0
retrying==1.3.4
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rpds-py==0.10.2
scipy==1.11.2
Send2Trash==1.8.2
six==1.16.0
sniffio==1.3.0
soupsieve==2.5
stack-data==0.6.2
tenacity==8.2.3
terminado==0.17.1
tinycss2==1.2.1
tomli==2.0.1
tornado==6.3.3
traitlets==5.9.0
typing_extensions==4.7.1
uri-template==1.3.0
urllib3==2.0.4
wcwidth==0.2.6
webcolors==1.13
webencodings==0.5.1
websocket-client==1.6.2
Werkzeug==2.2.3
widgetsnextextension==4.0.8
```

(d) Requirement 4

Figure 2: All Requirements of the Project Directly Taken from requirements.txt

2.3 Main Function

The main function, central to the program's execution, offers a straightforward way to interact with the 3D orientation optimization process. It guides users through the essential usage, from loading an STL file to visualizing the results.

The program starts by verifying the user's input. A single command-line argument is expected, specifying the path to the STL file of the 3D model to be optimized. If the user's input is incorrect or missing, the program provides a usage message, helping users correctly initiate the optimization process by printing "Usage: python main.py <stl file>"

Once the command-line argument is validated, the program then loads the STL file. The STL file represents the 3D model that will be analyzed and optimized for 3D printing. With the STL file loaded, the program constructs an objective function, named 'obj_fun.' This objective function plays a crucial role in evaluating the quality of the 3D printing orientation. It unites various parameters, including angles and geometric properties, to determine the optimal orientation.

The heart of the program is the optimization process. The core objective is to find the optimal orientation for 3D printing. This optimization is executed through the 'rotate_and_calculate_time' function, which calculates the most optimized rotation angles that will yield the best printing results. Consequently, the optimization generates three essential outputs: the optimal rotation angles ('theta'), the value of the objective function ('obj_fun'), and the time taken for the optimization process ('rotate_time').

The optimal rotation angles and the objective function value are reported, both rounded to three decimal places for readability. Furthermore, the time taken for the optimization process is displayed, allowing users to gauge the efficiency of the program.

As an additional feature, the program offers a visual representation of the 3D model in its optimized orientation. This visualization is created using the 'plot_stl' function, providing users with a tangible view of the model's final orientation.

```
import numpy as np
import sys
from operations import build, rotate_and_calculate_time, load_stl
from plot import plot_stl

if __name__ == '__main__':
    if (len(sys.argv) != 2):
        print("Usage: python main.py <stl file>")
        sys.exit(1)
    stl_name = sys.argv[1]
    mesh = load_stl(stl_name)
    obj_fun = build(mesh)

    theta, rotate_time, obj_fun = rotate_and_calculate_time(obj_fun)

    print(
        f'Optimal rotation for {stl_name} is {np.round(theta, decimals=3)} '
        f'with value {np.round(obj_fun, decimals=3)} in '
        f'{np.round(rotate_time, decimals=3)} seconds'
    )

    plot_stl(mesh, theta)
    sys.exit(0)
```

2.4 Operations Function (Optimization Logic)

The 'operations' module serves as the main calculation logic of the project, encapsulating critical functions and algorithms that drive the optimization of 3D printing orientations for STL models.

The module begins by offering a utility function called 'load_stl.' Its purpose is to import STL files, converting them into a structured format. The 'stl.mesh.Mesh' object, a part of the 'numpy-stl' [1] library, is used to represent the 3D model. This function bridges the gap between external STL files and the program's internal data structures.

```
from typing import Dict, List, Callable, Tuple
import numpy as np
import math
import time
import stl
from stl import mesh
from scipy import optimize
from direct import direct

def load_stl(stl_name: str) -> stl.mesh.Mesh:
    """Load an STL file into a numpy array"""
    return stl.mesh.Mesh.from_file(stl_name)
```

The 'rotate' function is the core optimization algorithm in the 'operations' module. It receives an objective function as input, represented as 'fun,' and aims to find the optimal rotation angles for the 3D printing process. The optimization is executed using the 'optimize.minimize' function from the 'scipy' [2] library. It explores a solution space defined by angular bounds, searching for the most efficient orientation. The result of this optimization is a tuple containing the optimal rotation angles ('theta') and the value of the objective function ('fun') in this optimized state.

The 'rotate_and_calculate_time' function receives the objective function ('fun') and returns a tuple that includes the optimal rotation angles ('theta'), the time taken for optimization ('rotate_time'), and the value of the objective function ('fun'). This time measurement allows users to observe the efficiency of the optimization process.

```
def rotate(fun: Callable[[List[float]], float]) -> Tuple[List[float], float]:
    """Rotate the function f to find the optimal orientation"""
    res = optimize.minimize(fun, [0, 0], method=direct,
        bounds=np.array([-math.pi, math.pi], [-math.pi, math.pi]),
        options=dict(maxfev=1000))
    return res.x, res.fun

def rotate_and_calculate_time(fun: float) -> Tuple[List[float], float, float]:
    """Rotate the function and calculate the time it takes to do so"""
    start_time: float = time.time()
    theta, fun = rotate(fun)
    finish_time: float = time.time()
    return (theta, finish_time - start_time, fun)
```

The 'compute_products' function takes the 'stl.mesh.Mesh' object as input and computes product values for pairs of mesh vectors. These products are labeled and organized, offering valuable insights into the geometric properties of the 3D model. The computed values are stored in a dictionary ('sp') for future reference.

```
def compute_products(mesh: stl.mesh.Mesh) -> Dict[str, List[float]]:
    sp: Dict[str, List[float]] = {}
    for i in range(3):
        for j in range(3):
            for k in range(3):
                for l in range(3):
                    product = f'{i + 1}{chr(ord("x") + j)}' \
                               f'{k + 1}{chr(ord("x") + l)}'
                    product_rev = f'{k + 1}{chr(ord("x") + l)}' \
                                   f'{i + 1}{chr(ord("x") + j)}'
                    if i == k or j == l:
                        continue
                    if product_rev in sp:
                        sp[product] = sp[product_rev]
                        continue
                    sp[product] = mesh.vectors[:, i, j] * mesh.vectors[:, k, l]

    sp.update(sa=sp['1y2x'] - sp['1x2y'] + sp['1x3y']
              - sp['1y3x'] + sp['2y3x'] - sp['2x3y'],
             sb=sp['1z2x'] - sp['1x2z'] - sp['1z3x']
              + sp['2z3x'] + sp['1x3z'] - sp['2x3z'],
             sc=sp['1z2y'] - sp['1y2z'] - sp['1z3y']
              + sp['2z3y'] + sp['1y3z'] - sp['2y3z'])

    return sp
```

The 'build' function assembles the objective function that guides the orientation optimization. It combines the computed product values, the rotation angles ('theta'), and various trigonometric functions to construct a comprehensive metric. This metric evaluates the quality of the 3D printing orientation, considering factors such as overhangs and geometric balance. The result is a function ('fun') that encapsulates the essence of the optimization process.

```
def build(mesh: stl.mesh.Mesh):
    """Build the objective function based on the mesh and computed products"""
    sp: Dict[str, List[float]] = compute_products(mesh)

    def fun(theta: List[float]) -> float:
        """Compute the objective function value for the given rotation"""
        sinx, cosx = math.sin(theta[0]), math.cos(theta[0])
        siny, cosy = math.sin(theta[1]), math.cos(theta[1])
        cosxcosy, cosysinx = cosx * cosy, cosy * sinx

        S: List[float] = sp['sa'] * cosxcosy + sp['sb'] \
                          * cosysinx + sp['sc'] * siny

        height: List[float] = mesh.vectors[:, :, 2] \
                               * cosxcosy - mesh.vectors[:, :, 1] \
                               * cosysinx - mesh.vectors[:, :, 0] * siny
        height_min: float = np.min(height)
```



```

overhang_bias: List[float] = np.tanh(S) + 1.0
bottom_bias: List[float] = np.tanh((np.sum(height, axis=1)
                                     - 3 * height_min) / 10.0)

return np.sum(np.abs(S) * overhang_bias * bottom_bias) / 4.0

return fun

```

In essence, the 'operations' module brings together the core functionalities required for 3D orientation optimization. It spans from the fundamental task of loading STL files to executing the optimization algorithm, measuring the optimization time, computing essential product values, and constructing a holistic objective function. Together, these operations drive the main logic of the program, offering a comprehensive solution for optimizing 3D printing orientations and facilitating additive manufacturing processes.

2.5 Plotting the Results

The 'plot' module is instrumental in visualizing the results of the 3D orientation optimization. It leverages the 'matplotlib' [3] and 'mpl_toolkits' [4] libraries to provide a clear representation of the optimal orientation for 3D printing.

Upon receiving the 3D model ('mesh') and the optimal rotation angles ('theta') as input, the module initiates the plotting process. It creates a 3D plot with the 'matplotlib' library [3], configuring it to display the optimized orientation for the given 3D model.

The module makes a copy of the original mesh data to preserve the model's integrity. Then, it applies the calculated rotation angles to the copy of the mesh, effectively rotating it to the optimal orientation. This rotation process aligns the model with the recommended angles for efficient 3D printing.

The 'if' condition allows for the consideration of multiple rotation angles, although in most cases, only two angles (x and y axes) are used. The module can accommodate the possibility of a third rotation angle if required.

To enhance the visual representation, the module employs the 'mplot3d' [5] module from 'mpl_toolkits' [4]. It creates a 'Poly3DCollection' [6] from the mesh vectors, allowing for the visualization of the model's geometry. The model is rendered with a gold-colored surface and black edges for clear visibility.

The final touch is adjusting the scale of the plot to ensure that the 3D model fits within the display area. This dynamic scaling accounts for variations in the model's size and ensures that it is presented in the most suitable manner.

Ultimately, the 'plot' module offers a visual representation of the optimized 3D printing orientation, making it easier for users to understand and implement the recommended orientation in their additive manufacturing processes.

```

from typing import List
import numpy as np
import stl

from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d

def plot_stl(mesh: stl.mesh.Mesh, theta: List[float]):

```

```

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_title(f'Optimal Orientation for Figure')

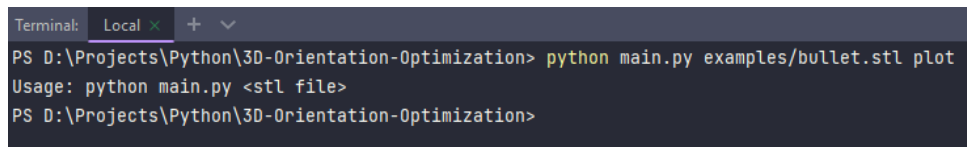
mesh = stl.mesh.Mesh(mesh.data.copy())
mesh.rotate([1, 0, 0], theta[0])
mesh.rotate([0, 1, 0], theta[1])
if len(theta) > 2:
    mesh.rotate([0, 0, 1], theta[2])

stl_polys = mplot3d.art3d.Poly3DCollection(mesh.vectors)
stl_polys.set_facecolor('gold')
stl_polys.set_edgecolor('black')
ax.add_collection3d(stl_polys)
scale = mesh.points.ravel()
ax.auto_scale_xyz(scale, scale, scale)
plt.show()

```

2.6 Example Cases

In this section, illustrative examples of how the program is used are provided. Program usage, anticipated outcomes, potential errors and 3D plots of the objects are included.



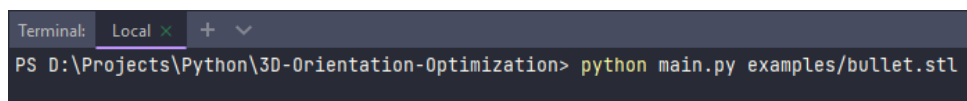
```

Terminal: Local x + v
PS D:\Projects\Python\3D-Orientation-Optimization> python main.py examples/bullet.stl plot
Usage: python main.py <stl file>
PS D:\Projects\Python\3D-Orientation-Optimization>

```

Figure 3: Wrong Usage of the Program

Terminal usage for the 'bullet.stl' file located in the 'examples' folder inside the project files.



```

Terminal: Local x + v
PS D:\Projects\Python\3D-Orientation-Optimization> python main.py examples/bullet.stl

```

Figure 4: Terminal Usage

2.6.1 Bullet Object

The original orientation of the 3D object.

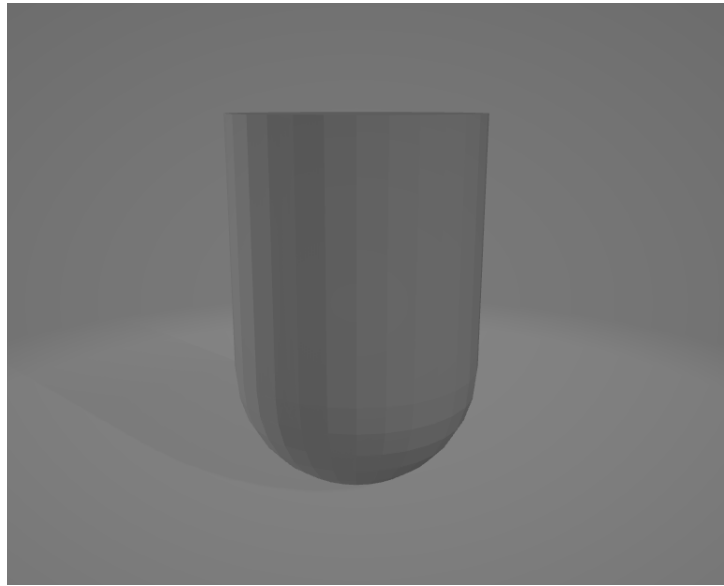


Figure 5: Former Orientation of the STL file

The expected Output for the 'bullet.stl' file.

```
Terminal: Local x + v
Iteration 109 f([0.5      0.9999999])=16.40097427368164 with fev=981
Iteration 110 f([0.5      0.9999999])=16.40097427368164 with fev=985
Iteration 111 f([0.5      0.9999999])=16.40097427368164 with fev=989
Iteration 112 f([0.5      0.9999999])=16.40097427368164 with fev=993
Iteration 113 f([0.5      0.9999999])=16.40097427368164 with fev=997
Iteration 114 f([0.5      0.9999999])=16.40097427368164 with fev=1001
Optimal rotation for examples/bullet.stl is [0.    3.142] with value 16.401 in 0.465 seconds
PS D:\Projects\Python\3D-Orientation-Optimization>
```

Figure 6: 'bullet.stl' Output

The plot of the 'bullet.stl' file created with Matplotlib [3] library.

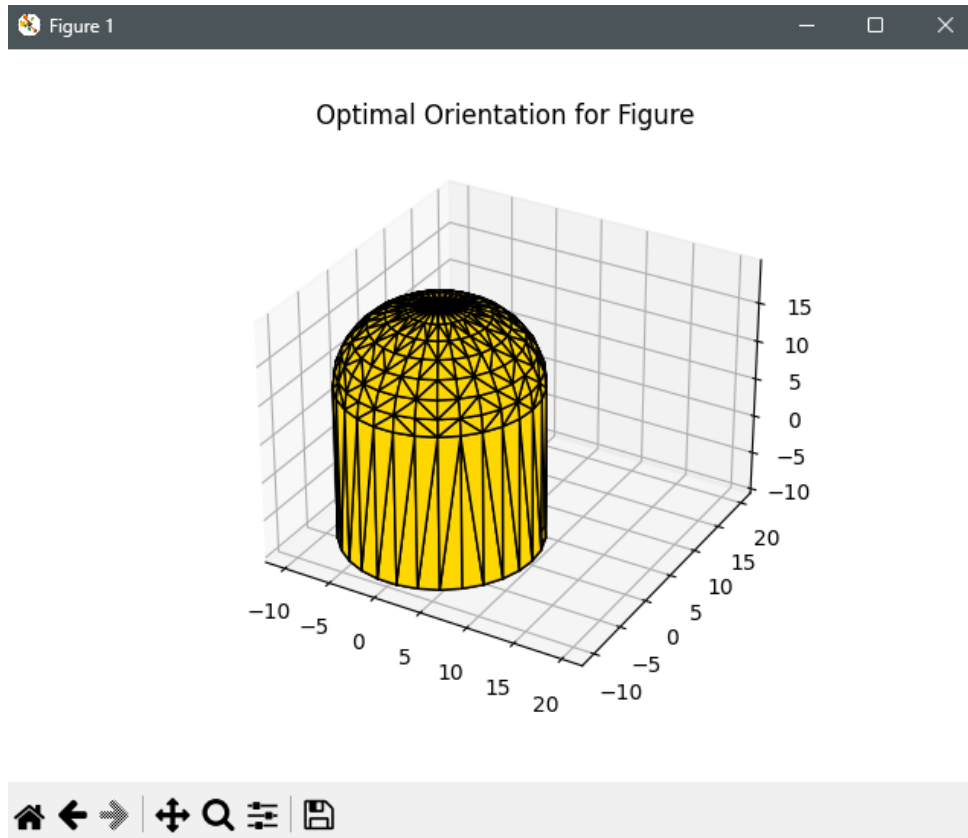


Figure 7: 3D Plotting of the Geometry

2.6.2 Wheel Object

The original orientation of the 3D object.

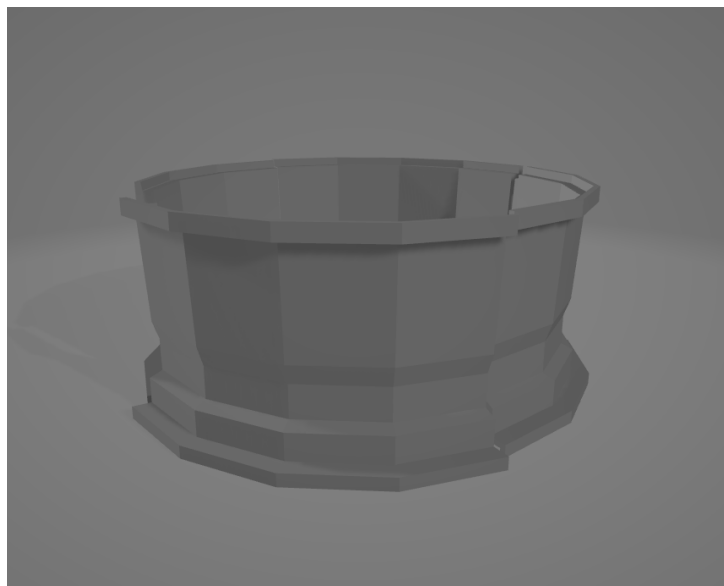


Figure 8: Former Orientation of the STL file

The expected Output for the 'wheel.stl' file.

```
Terminal: Local x + v
Iteration 175 f([0.5 0.5])=2875.22021484375 with fev=909
Iteration 176 f([0.5 0.5])=2875.22021484375 with fev=929
Iteration 177 f([0.5 0.5])=2875.22021484375 with fev=947
Iteration 178 f([0.5 0.5])=2875.22021484375 with fev=963
Iteration 179 f([0.5 0.5])=2875.22021484375 with fev=979
Iteration 180 f([0.5 0.5])=2875.22021484375 with fev=1003
Optimal rotation for examples/wheel.stl is [0. 0.] with value 2875.22 in 0.98 seconds
PS D:\Projects\Python\3D-Orientation-Optimization>
```

Figure 9: 'wheel.stl' Output

The plot of the 'wheel.stl' file created with Matplotlib [3] library.

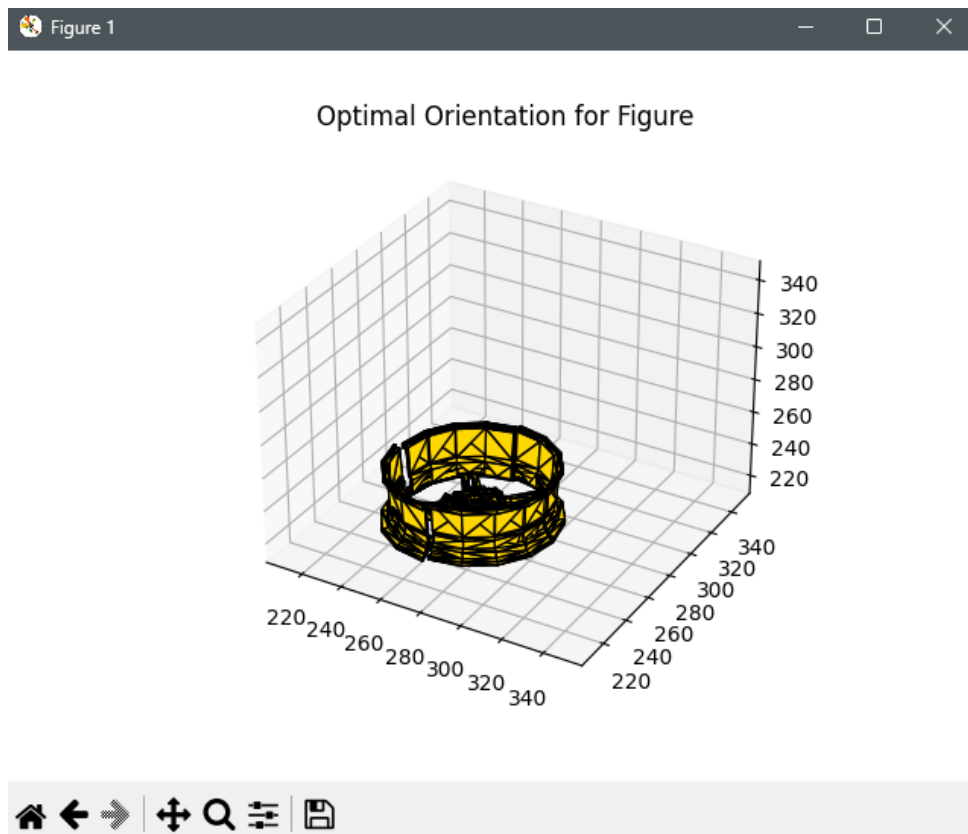


Figure 10: 3D Plotting of the Geometry

2.6.3 Truck Object

The original orientation of the 3D object.

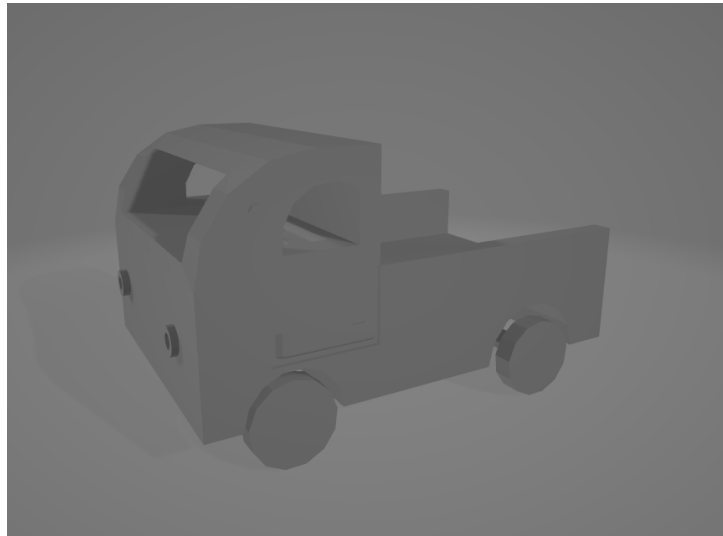


Figure 11: Former Orientation of the STL file

The expected Output for the 'truck.stl' file.

```
Terminal: Local x + v
Iteration 86 f([0.25055745 0.50128707])=953.568359375 with fev=971
Iteration 87 f([0.25055745 0.50128707])=953.568359375 with fev=975
Iteration 88 f([0.25055745 0.50128707])=953.568359375 with fev=981
Iteration 89 f([0.25055745 0.50128707])=953.568359375 with fev=987
Iteration 90 f([0.25055745 0.50128707])=953.568359375 with fev=997
Iteration 91 f([0.25055745 0.50128707])=953.568359375 with fev=1007
Optimal rotation for examples/truck.stl is [-1.567 0.008] with value 953.568 in 0.567 seconds
PS D:\Projects\Python\3D-Orientation-Optimization>
```

Figure 12: 'truck.stl' Output

The plot of the 'truck.stl' file created with Matplotlib [3] library.

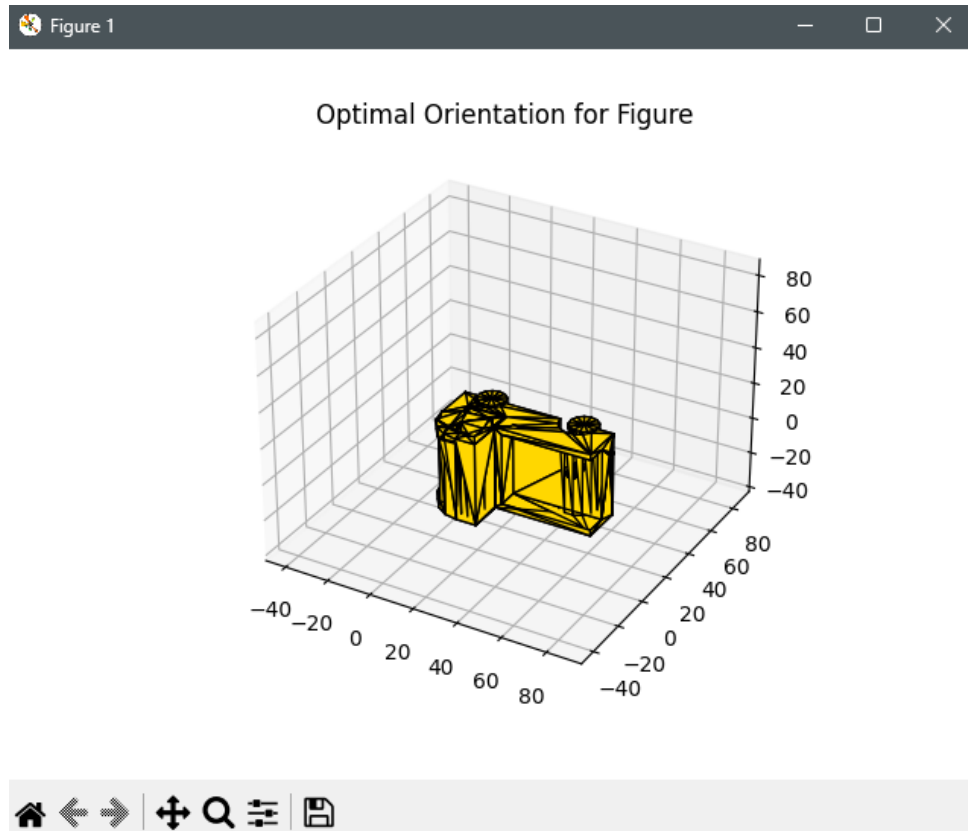


Figure 13: 3D Plotting of the Geometry

3 CONCLUSIONS

In this project, an innovative additive manufacturing optimization tool that leverages software engineering techniques to enhance the 3D printing process have been developed. By strategically optimizing the orientation of objects, the program provides valuable insights into the printing angles, contributing to cost-effectiveness and improved manufacturing quality.

This aim of the project was building a bridge between software engineering and additive manufacturing, offering a powerful tool that can significantly impact the efficiency and quality of 3D printing processes.

Throughout the project, I have demonstrated the program's capabilities through practical examples, highlighting its potential to streamline additive manufacturing workflows. Expected outcomes and potential errors have been discussed, underscoring the program's adaptability and robustness.

4 REFERENCES

- [1] <https://numpy.org>
- [2] <https://scipy.org>
- [3] <https://matplotlib.org>
- [4] <https://pypi.org/project/mpl-toolkits.clifford/>
- [5] <https://matplotlib.org/stable/users/explain/toolkits/mplot3d.html>
- [6] https://matplotlib.org/3.1.1/api/_as_gen/mpl_toolkits.mplot3d.art3d.Poly3DCollection.html