An intoduction to the **Auto-scrutineer**

Contents

- New submission folder structure
- Basic overview
  - Rule types
  - Explanation and examples

Part 1
Relevant to everyone

- In-depth overview (wip)
- Rule definitions (wip)
- VTK-pipelines (wip)

Part 2
Mostly relevant to staff and maintainers
(but everyone is welcome)

Disclaimer:
I am **NOT** a software engineer, so set your expectations accordingly.
Please let me know if there are critical bug/errors in my code, or other areas that might be improved.

Part 1

This part is relevant to everyone, competitors, staff etc.

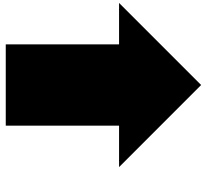
please familiarize yourself with the content.

New submission folder structure

Competitors must specify which .stl's are considered floor, frontwing etc.

Old structure

```
MyRacingTeam_Round1_v2
├── high_res_surfaces
│   ├── FW_endplate.stl
│   ├── FW_flap.stl
│   ├── rearwing.stl
│   └── etc
├── vehicle_body
│   ├── body.stl
│   ├── nose.stl
│   ├── mirror_strut.stl
│   └── etc
└── etc
    └── etc
```



New submission folder

```
MyRacingTeam_Round1_v2
├── frontwing_flaps
│   ├── FW_flap_1.stl
│   └── FW_flap_2.stl
├── frontwing_mainplane
│   └── fw_mp.stl
├── main_floor
│   └── floor_v13_revK_final2.stl
└── etc
    └── etc
```

In the new structure there is a folder for each ”part” referenced in the rules

Each folder can contain multiple stls.

So, more folders but parts can easily be evaluated individually.

Full list of parts should be specified in the rules.

After scrutineering: stls are copied to a new “input_file” folder, following the “old structure”, thus making the submission “MFlow ready”.

Rule types

Currently supported “rule types”:

- Obscure (hide, conceal etc.)
- Number of sections

Things like “10 mm rule” and “no floating parts” cannot be checked with this tool. If someone has ideas on how check these, please share them.

Obscure rule

The obscure rule is quite versatile. It can be used to test a range of different rules. These include:

- Part-X must be fully contained within RV-Y

- From *angle* Part-X must obscure RS-Y

- Part-Y must be lowest part of the car within *area*

Common for all these rules is that if X is visible from a certain angle, then that indicates a breach of the rule.

Two different geometries must be specified:

- “given_geometry” (X in the examples above)
- “ref_geometry” (Y in the example above)

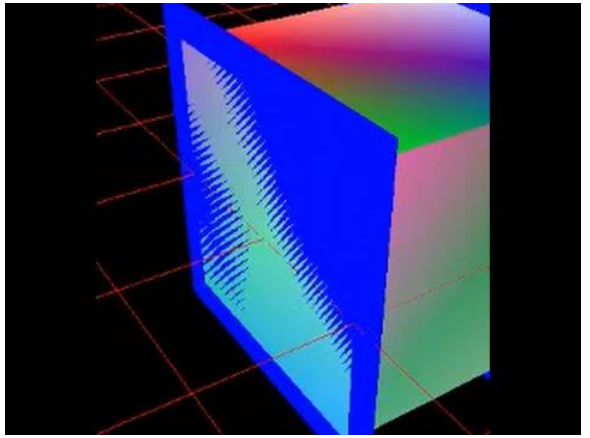
An additional ”mask” geometry can also be specified. The mask can help to specify the *area* in the example above

When checking this rule, a scene is created where the given geometry is colored red, and the reference and mask are colored black. Images of this scene are rendered. You can now check the rendered images for red pixels. Any red pixel = illegal geometry.

If an infringement is found, then the color palette is altered slightly, and an image is saved that more clearly shows the infringement (see example next slide)

NB: This rule is sensitive to Z-fighting, so competitors should avoid making geometry that touches the reference volumes!

Z-fighting example →



Obscure rule

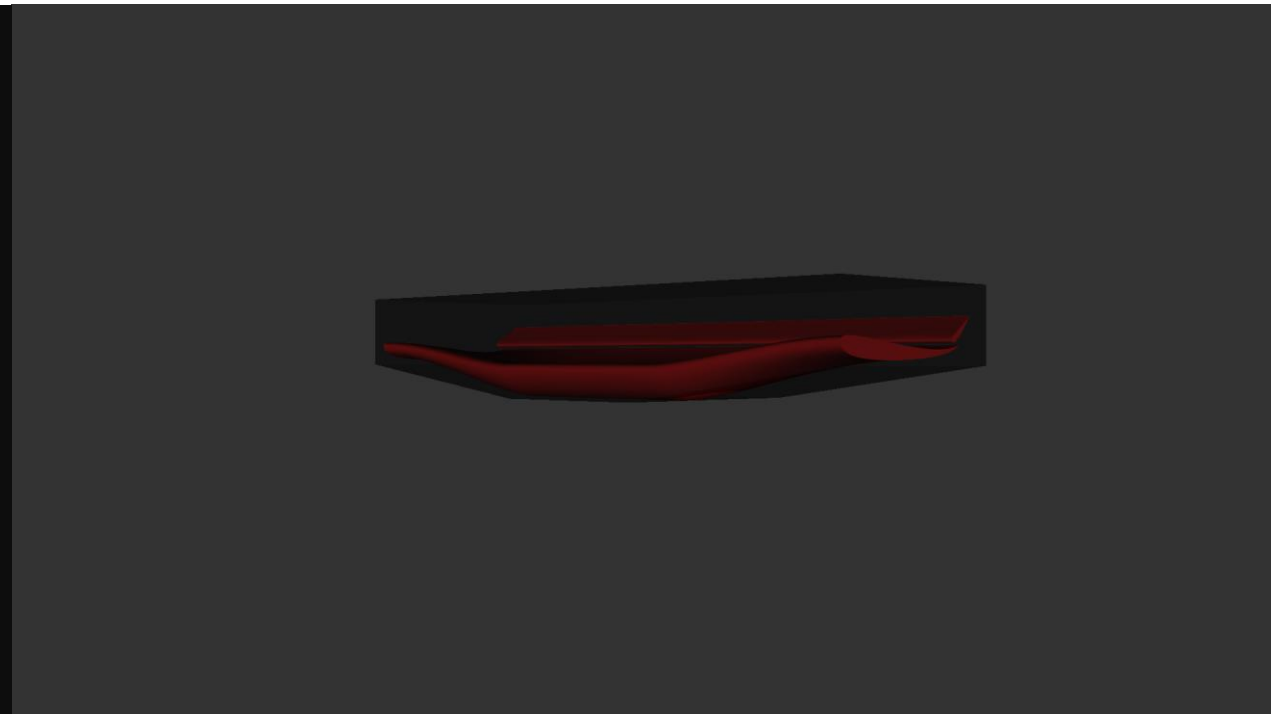
The rule begin checked:

```
{
    'rule_section_name': '§1.0',
    'rule_type': 'obscure',
    'angles': 'all',
    'ref_geometry': 'RV-RW-PROFILES_v',
    'given_geometry': 'rw_mainplane',
    'mask': 'none',
    'focus': 'given'
},
```

What the rule-checker sees:
(flat-coloring)

Red = illegal

Rendered for more clarity:
(rendered with shadows)



The bottom surface of the mainplane is not fully contained within the RV

Number of sections

”Number of sections” –rule can be used to count number of sections in frontwings, rearwings etc.

It works by sectioning the geometry with a number of planes. The planes are defined with a normal vector, and an origin point.

The normal vector is specified in the rules.

for example (0, 1, 0) for y-sections.

The origin points can be defined either as a strict list of points, or as a ”semi-random” list of points in an interval. For semi-random sampling a ”maximum unprobed interval” must be defined, this is the maximum distance between planes.

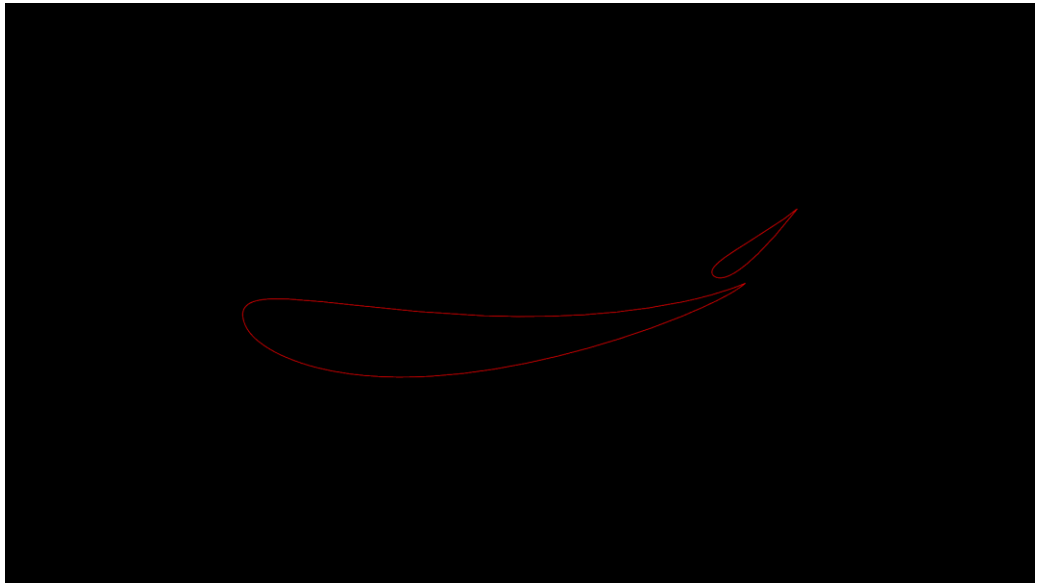
The 3d coordinates of the origin points are found by multiplying each scalar in the list with the normal vector.

An image of the cut section is rendered. Then it counts the number of sections visible in the image. (see next slide)

NB! Since it only checks a limited number of sections, small infringements may or may not be caught!

```
{
    'rule_section_name': '§2.0, mainplane sections',
    'rule_type': 'n sections',
    'angle': 'left side',
    'number_of_sections': "= 1",
    'given_geometry': 'rw_mainplane',
    'cutting_plane_normal' : [0, 1, 0],
    # 'sampling_method': 'specific',
    'sampling_method': 'semi_random',
    'sampling_interval': (0.0, 0.535),
    'max_unprobed_interval': 25e-3,
    # 'probing_locations': np.array([ 0.25 ]),
},
```

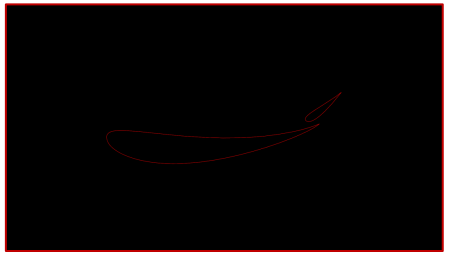
Rendered image of a section:



Number of sections

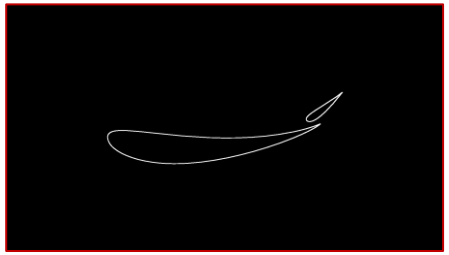
The function counts the number of sections in the rendered image, with the following method:

Rendered cut section
(red frame added for clarity)



(8bit RGBa array)

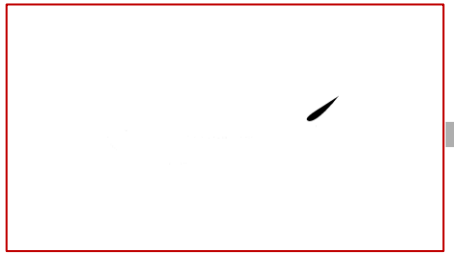
Convert to B/W



(Boolean array)

”Bucket fill” with white
from a corner

Find a black pixel,
Bucket fill from this pixel
Add one to the counter
Repeat until image is all white



Result: 2 sections

Why this way?
Another method could be to simply count the number of red sections. I decided against this approach because you often have overlapping geometries (because it is good for meshing). But you don't want to count any “internal” section as a section. This way you only count the number of sections in contact with the external flow. Imo this is closer to ideal.

Part 2

Mostly relevant to staff/contributors

This part should be irrelevant to competitors that don’t wish to contribute to the script.

Rule definition

All rules are gathered in a single list. This list is defined in rules.py

Each rule is defined as a dictionary

The rule-checker iterates through the list of rules. The keyword: **”rule_type”** is important, this specifies which function is called.

All rules must return a string that is added to the report.

New rules can be added by simply defining a new function and adding this function to the if-statement in the scrutineer function.

Two different rules in the rules list

```
{
    'rule_section_name': '§1.3',
    'rule_type': 'obscure',
    'angles': 'all',
    'ref_geometry': 'RV-RW-PYLON_v',
    'given_geometry': 'rw_pylon',
    'mask': 'none',
    'focus': 'given'
},

{
    'rule_section_name': '§2.0, mainplane sections',
    'rule_type': 'n sections',
    'angle': 'left side',
    'number_of_sections': "= 1",
    'given_geometry': 'rw_mainplane',
    'cutting_plane_normal' : [0, 1, 0],
    # 'sampling_method': 'specific',
    'sampling_method': 'semi_random',
    'sampling_interval': (0.0, 0.535),
    'max_unprobed_interval': 25e-3,
    # 'probing_locations': np.array([ 0.25 ]),
},
```

From scrutineer function:

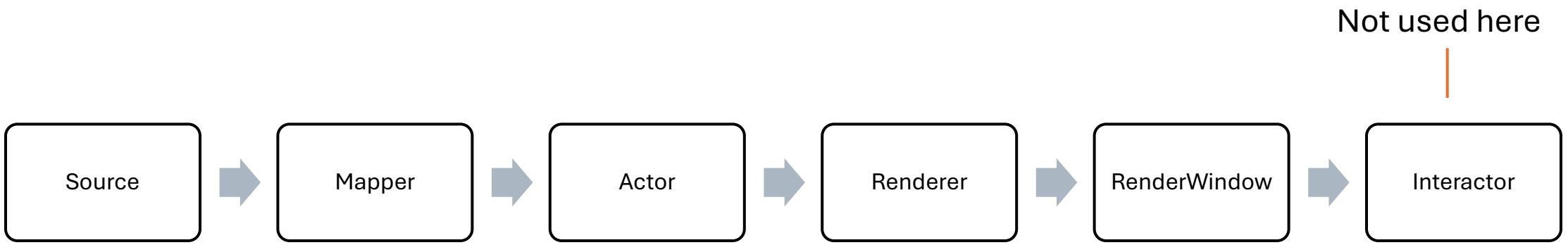
```
for rule in allRules:
    print('Checking rule: ' + rule['rule_section_name'] )

    if rule['rule_type'] == 'obscure':
        report += ruleObscure(rule, settings)

    elif rule['rule_type'] == 'n sections':
        report += ruleNSections(rule, settings)
```

Obscure rule definition

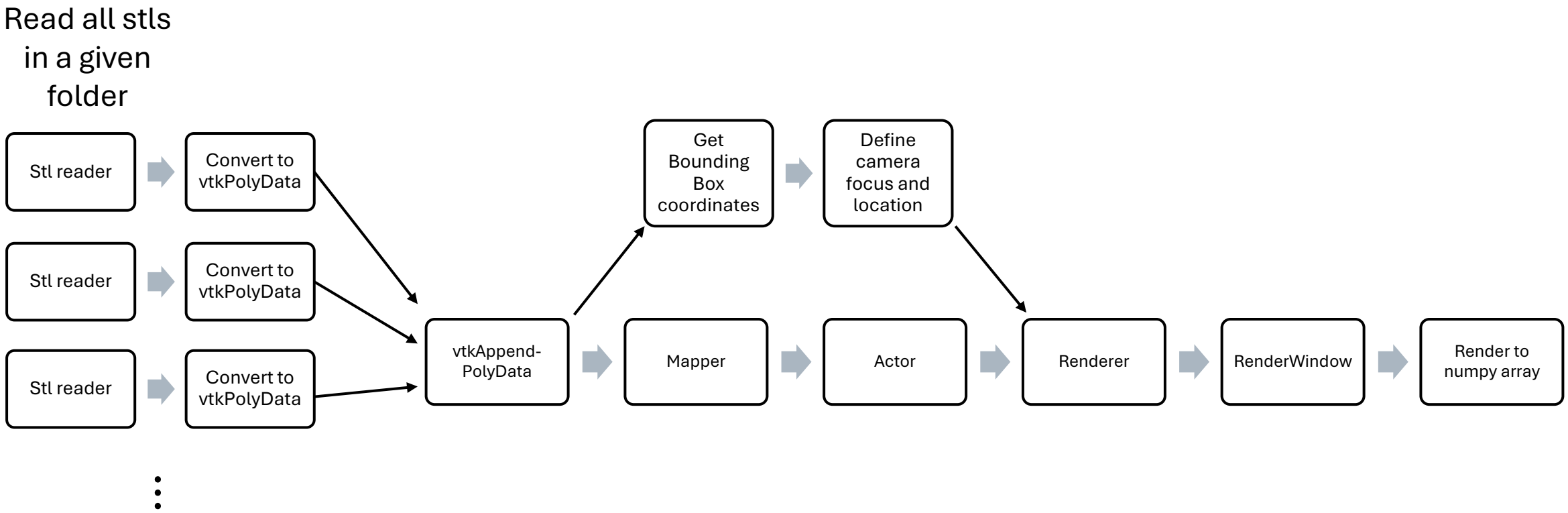
- **Rule section name** is used for the report
- **Angles**, specifies the camera angles. Options include ”all”, ”above”, ”below” etc.
- **Ref_geometry** colored black
- **Given geometry** colored red
- **Mask** colored black
- Focus specifies the part to focus the camera on, here the given geometry. ½

```
rule11_1 = {
    'rule_section_name': '11.1',
    'rule_type': 'obscure',
    'angles': 'all',
    'ref_geometry': 'RV_REARWING_V',
    'given_geometry': 'rear_wing',
    'mask': 'none',
    'focus': 'given'
    }
```

Overview of generic VTK-pipeline



Overview of VTK-pipeline for Obscure rule



Overview of VTK-pipeline for N sections rule

Most likely room for improvement.