GSP Technical Debt

Table of Contents

Technical Debt ?

Part 1

When Does it Happen ?

> *"When past choices are no more suitable for current needs"*

- When it is early in the project and requirements are still unclear
- When motivation/skill is lacking to do it properly
- When a quick solution is chosen to meet a deadline

What is the Cost ?

- It slows down progress
- It increases the risk of bugs
- It makes the code harder to understand and maintain

For Libraries Specific Case

It repels potential contributors

- "*No to messy code*" and "*No to lack of tests*"

For those who decide to contribute ?

- Makes their onboarding *slower* and *more painful*

Current Technical Debt

Part 2

- TODO here put all the bugs you remember
- be nice and always do it softly
- do not blame anyone, it is not personal
- it is just a fact, nobody is perfect, we all do mistakes

List of Known Issues

- no notion of camera
    - so from which point of view are we rendering ?
    - how to handle multiple viewports ?
- even the most basic, aka generating command doesnt work. it never worked. it is not buggy as in doesnt work as expected, the code is just not there. no ./examples
- add the len(on transform) in most constructor
    - TODO count it

Commands Serialisation based on `eval()`

- Portability issue
  - this works only for python. What if we want a JS/C++ client ?
  - They will need to emulate python `eval` which is not realistic
- Maintainability issue
  - hard to track which symbols are used where

Commands Serialisation based on `eval()` (Contd.)

- Security issue: arbitrary code execution
- can be mitigated by link

```
safe_globals = {"__builtins__": None}
safe_locals = {"abs": abs, "pow": pow, "max": max}
result = eval(user_input, safe_globals, safe_locals)
```

- It would require to rewrite all the commands to avoid

  `__import__` and other tricks

Many Undefined Symbols

- undefined symbol for vmin, vmax - typo here
- undefined symbol for uint32 - likely forgot `np.` here
- Undefined function sRGBA_to_RGBA doesn't exist here
- Undefined local symbol data doesnt exist here
- here, here and here
- For a list go here

PS: this code ever got executed ?

Confusing Naming

- `vec3` is a vector of 3 elements for many. Well known in GLSL
  - here it is an **array** of `vec3`, and it is impossible to have a single `vec3`
- `._viewports` is a list/dict of viewports ?
  - here it is a list of matplotlib artists link
- `List` is typing.List ?
  - no it is a list of object link

Inconsistency in vector layer

- there is a notion of `vec2`, `vec3`, `vec4` but no `vec1`
- so how to encode indices or size etc... ?

Just use an array of float ?

- Some would say "use an array of float" but it wont go thru the same code path
- All the benefits from the `vec` layer will be lost
  - all the `vec` conversion, all the tracked features, all the

Type Hinting

> *"far too few, and when it is there, it is often flacky"*

Why static type checking is important ?

- catch errors early
- good for libraries users and for team developpers

Type Hinting: An Example

```python
def foo(data : memoryview | bytes = None):
    ...
```

- Either the default value is not of the right type
- Or the type hinting is just wrong

> *Wrong type hinting better than no type hinting ?*

How to Address It ?

Part 3

Diagnose the issues

- is the code widely used ?
- is it behaving as expected ? (aka is it well tested ?)
- how large is the code ?
- is there still people who understand it ?
- how complex is the code ?
- how well is it documented ?
- which parts is affected and how large is it compared to the rest

Notes

- We lost controls, nobody understand this code, if the code was large or if a lot of people were depending on it, reimplementation would not be possible
- what are the possible engineering strategy when you face such a problem ? reimplementation is reasonable. it is a small project for now, the base we got is fragile and doesn't do much. keeping it as a base will slow down progress.

Conclusion

- better catch it early