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 ?
- the visual got no `reference` (e.g. position, rotation, scale). You want to move that, you move ALL the points/pixels/etc... manually
- commands are too black magic, just creating a temporary array will duplicate the data over the network - the commands generated by decorator will always have this issue - viewport.add(visual) explicit + scene graph API would fix this
    - issue with the temporary array of vec3 and how it duplicate things
        - here

Subclassing Numpy ndarray

- numpy advices against subtyping it and say it is bad for maintenability here
    - unpredictable behaviour
    - subclass may loose information

GSP is subclassing ndarray 4 times:

- vec3 -> swizzle -> tracked -> tracked -> np.ndarray

Tracked: ndarray subclass to optimise network usage

- aka we send only the data which changed
- but tracked doesnt even contains serialisation - as in delta encoding
- do not believe it is easy, i had to reimplement it 3 times before getting it right.

Buffer : an Array of Bytes ?

- Buffer is not a "array of byte". it has a shape and a dtype. this is ndarray/tensor thing
    - aka multi dimensional array of a given type. aka a shape and a dtype
    - here

Confusing Naming

- `._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
- `io` is the stdlib io ?
    - no it is a module for serialisation link
- 'transform' is 3d transformations matrix ?
    - or 'transform' is a data transformation chain ?
    - up to you to guess, but 'transform' is everywhere in the source

Transform Assumptions

- transform are assuming object to be sorted, so objects are z-sorted at every render.
- So it sorted "billions of objects" at every frame even if no transform is here
- here we are sorting points here
- or pixels here

Confusing Naming (Contd.)

- `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`
- Inconsistency in vec2, vec3, vec4 conversions
    - in *to_vec3()*, `vec2 -> vec3 : x,y -> x,y,0` link
    - in. *to_vec4()*, `vec2 -> vec4 : x,y -> x,y,0,1` link
    - so here `to_vec4` is not consistent with `to_vec3`, and should be considered `to_homogeneous` or similar

Coding Style Issues

- single letter variable
- no comment
- confusing names
- lack of type hinting
- globals / eval

black magic: Args converted by parsing `__annotations__`

- see code here
    - decorator based black magic
    - so complex, hard to debug. Nobody understand how it works. see bug #14

PS: why not using functions call to convert things explicitly ?

Commands: Data Duplication Issue

- commands are too black magic,
- just creating a temporary array will duplicate the data over the network
    - issue with the temporary array of vec3 and how it duplicate things here

```
V = glm.to_vec3(np.random.uniform(-1, +1, (n,2)))
P = glm.to_vec3(np.random.uniform(-1, +1, (n,2)))
pixels = visual.Pixels(P)
```

It will sent V over the network, even if unused.

Code commited without being run: (the commands)

- commands generation/parsing never have been executed
- see bug #14
- it is not buggy as in doesnt work as expected, in a special case.
- the code is just relying on unexisting API, it throws exception all the time.
- no ./examples using it

Code commited without being run: (the transforms)

- most visual relies on `len(positions)` but positions can be a transform
- transform doesnt know the length of their output, and no `__len__` is defined
- so calling `len(transform)` throws exception

Code commited without being run: (Contd.)

- Light is never used, Data is never used -> commited but never ran
    - data.py
    - light.py

Code commited without being run: 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 ?

Race condition in Tracked

- tracked is disabled globally in each visual constructor
    - here
- race condition if the code is executed in thread/async context
- it will fail silently

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 the commands to avoid `__import__` and other tricks

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 `raggle` features
    - none of that will happen to a `array of float`, while it would to a `vec1`

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

```
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 ?

Possible strategies

- refactor it incrementally

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