Thesis rebuttal for P5

Jordi van Liempt

June 2020

1 Done improvements

- Written the abstract.
- Written a reproducibility reflection.
- I have changed the CBOR section in the related work to binary JSON in general. I now mention the other viable binary JSON formats as well and quickly explain why I chose to use CBOR in the thesis.
- Added a section on glTF in the related work because it is used by B3DM.
- Gave information on the tools that I have used.
- Added average amount of vertices and average amount of attribute characters per city object to the dataset characteristics of Section 5.5.
- Added dataset snippets in the appendix.
- Noted that quantisation is actually used in the thesis (within CityJSON files themselves with "transform".
- Changed the baseline of the benchmarking that I had done for P4. Original CityJSON datasets are now processed in the client as well rather than on the server. This makes for a fairer comparison, especially when performing an operation on only one City Object.
- I have done additional benchmarking with datasets in which the attributes are removed. Now you can see the workings of compression types on "full" datasets and 3D city models that only have geometries.
- Also I have performed benchmarks with another server-client implementation. Instead of compressing datasets in advance and processing them in the client, the data is prepared on the fly on the server when a request is made by a client, and compressed when that is finished. Only then is it submitted to the client. This is more likely to be done in practice and it means that data can be queried before transmitting it.

- Improved the benchmark statistics. Before, the graphs were showing the median results while the tables showed averages. Now both of them show medians.
- Finalised the conclusion.
- Added preview of results to the introduction.
- Adapted the research question a little bit (Ken's critique). From

  To what extent can the implementation of different compression techniques improve CityJSON's web use experience, considering file size, visualisation, querying, spatial analysis, and editing performance?

  to:

  To what extent can the implementation of different compression techniques improve CityJSON's performance on the web, considering file size, visualisation, querying, spatial analysis, and editing performance?
- In Section 2.1, I have indicated that the optimal compression trade-off (compactness versus processing speed) in the case of this thesis is about the net speed gain (time gained in network transfer time versus additional (de)compression time, which I had mentioned in Chapter 1) (Ken's critique).
- Moved OGC API + streaming of CityJSON to Chapter 8. It is not a main part of the thesis and distracted from it.
- Properly assessed the loss of the replace method.
- Updated conclusion.
- Small changes that you could see in the diff file.

2 Critique that has not been implemented

- Improving the introduction. Despite that it could be written more attractively, I preferred to focus on the technical improvements as I felt it would take me quite some time.
- Improving the theory chapter. Indeed there is a lot more to explain about the compression techniques. But again it would be time consuming and I think the other improvements add more to the quality of the thesis. Now you would have to read the cited sources to get a deeper understanding, as Ken mentioned, which is not so nice but at least it is possible to do that.

- Altered the datasets (or created new ones) to isolate specific characteristics. This would enable a more thorough analysis and give more insight of the performance of compression types with different dataset characteristics. There are however too many characteristics that you would have to take into account. Multiple LODs, languages, amount of attributes, City Object types, sizes. And you could create combinations of all of these. Both creating these datasets, creating benchmark results, and analyzing them would have taken way too much time. That is if doing it well. Therefore I had chosen to only do additional tests with the attributes removed.

- From Ken's critique:
  Section 2.1.1 is a bit vague. Not sure that you're hitting the main points here. Often there's no clear original form of the data, just different ways to encode it. I think this would be better described as a signal processing problem (how many bits do you use) or as a tradeoff between size and possible distortions. Note that text can be compressed in a lossy way (eg transcoded from unicode to ascii) and that JPEG files can be lossless too (although they usually aren't).
  I believe I understand what he means by this. I think any kind of digital data is "compressed" in some way as it is an encoding of a real-world thing, be it a letter (ink to pixels) or sound (continuous signal of a sound from a natural source to a discretised digital signal). CityGML and CityJSON are both "lossless" compressions of real-world data. Did not have time to properly think this through and write it down.

- These three of Ken are useful additions, but didn't get around doing it.
  It Would be nice to give some examples of where the compression techniques you describe are used (like typical file formats).

  Maybe you need to describe what is actually a binary file vs text (in terms of actual bits and bytes)? Explain how they're more compact because they're able to use more combinations of bits.

  Maybe explain how Huffman encoding actually works? Just explain how a tree is generated and the link between the bits and the tree.

- *Some more detail in the methodology would help, including some snippets of the different methods. These 20 formats are rather important, and right now we're left with guessing exactly how they look like.*
  I was considering adding pseudocode for them, but did not have enough time in the end.

- Creating my own Huffman tree and table (Section 2.2.4).

- Explaining CityJSON in more detail (3.1). I did add some extra information but I think it is not enough.

- Testing of performance of other binary JSON formats. I did write a piece of benchmarking code to do it, but the results did not make much sense to me (for example CBOR having pretty long processing times). Could have been because of Python library implementations. I can discuss about this with you @Hugo if you want.