Replication Notes for the paper "ShapeBench: a new approach to benchmarking local 3D shape descriptors"

Bart Iver van Blokland

1 Overview

This document is intended as a manual for anyone looking to run the benchmark to replicate its results.

The benchmark starts out with 3D object files from the Objaverse dataset. Based on these, a sequence of intermediate files is computed. All of these files are subsequently used to compute the results of the benchmark. The dependency chain of these intermediate files is shown in Figure 1.

The purpose of these files:

- Compressed mesh files: a local cache of Objaverse mesh files, but compressed to save on space. The benchmark automatically downloads any files it needs, and stores them here.
- dataset.json: A list of all files contained in the dataset. A minimum bounding sphere is computed for each object, which is stored in this file.
- Support radius: A single number computed for each method. This also produces a separate output file with a few statistics. These are used to produce the charts in Figure 4.
- Reference and Sample descriptors: two files with cached descriptors for each method.

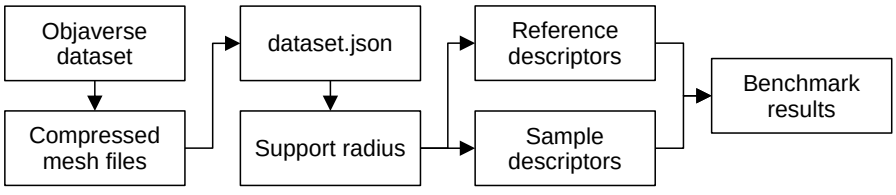

Figure 1: Overview over various files that are written to disk, and where the contents of each (and all previous files in the sequence) are used.

- Benchmark results: JSON files containing computed benchmark results. One JSON file with results produces one chart in the paper. You can find these charts in Figures 7 to 13, subfigures (a) to (f).

To verify the results of the benchmark, each of these intermediate files must be replicated.

The replication script is keyboard based. Navigate using the arrow keys, and use enter to select an option.

2 System Requirements

What you will need to run this benchmark:

- At least 100GB of available disk space
- At least 32GB of RAM
- A CUDA capable GPU (only for Figure 1. The remainder does not require the GPU)

The project has been tested with CUDA 12.1 and GCC 12.3.0. However, it is also known to compile when using GCC 11.4.0.

While it is *possible* to run the benchmark suite with 32GB of RAM, it is in a number of filters not sufficient. 64GB of RAM should allow you to run everything.

The 100GB of storage space is needed for the precomputed cache files (dataset.json, and reference and sample descriptors for each method), and the precomputed results. This also includes an approximately 10GB allocation for a local dataset cache.

3 Compilation

The script should handle all compilation. The repository includes most of the libraries it uses in source form, which may need some time to compile. The same is true for the linking step, which is serialised on all systems I've tested on.

If you cloned the repository with the `--recursive` flag (if not, try `git submodule update --init`), and installed the required packages using the script, everything should compile out of the box.

You may, however, encounter an error that looks like this:

```
/usr/include/c++/11/bits/std_function.h:435:145:
    error: parameter packs not expanded with '...':
435 |         function(_Functor&& __f)
/usr/include/c++/11/bits/std_function.h:435:145: note: '_ArgTypes'

/usr/include/c++/11/bits/std_function.h:530:146:
```

```
    error: parameter packs not expanded with '...':
530 |         operator=(_Functor&& __f)
/usr/include/c++/11/bits/std_function.h:530:146: note: '_ArgTypes'
```

This error is caused by a known incompatibility between GCC and CUDA. You can try to use the versions listed in the previous section.

Another thing to note is that the Ceres library likes to eat up a lot of memory on many core systems. If you open a terminal in the `bin` directory, you can try to compile with fewer threads (e.g. `ninja -j 10`).

4 Replication of results

What follows here are notes for each of the steps in the chain of intermediate files.

4.1 Compressed dataset and dataset.json

The replication script automatically enables verification of the minimum bounding sphere that is stored in the dataset.json file for each file being used in the benchmark (see `src/benchmarkCore/common-procedures/meshLoader.h` line 47). That effectively covers the compressed mesh files, and the dataset.json cache file.

Should you wish to replicate dataset.json from scratch, you must download the Objaverse dataset (approximately 8.9TB) into the `input/objaverse-uncompressed` folder (specifically the numbered folders that are of the form `000-123`). If this folder is empty, the benchmark will download files as needed, compress them, and store them in the local cache. The size and location of this cache can be altered in the settings menu in the replication script. The total size of the entire compressed dataset is approximately 1.6TB.

4.2 Support Radius

The support radius is computed using two sets of 100,000 descriptors calculated for 100 different support radii. Each descriptor is computed for a random vertex sampled from a random object in the dataset. Some descriptor methods require quite a bit of memory. The benchmark therefore is therefore able to compute the statistics for only a single radius instead. The difference in execution time is not very large.

Whether the support radius should be computed entirely, or only a single line from the statistics upon which the decision is based can be selected in the top menu entry after selecting the option to replicate Figure 4 in the replication script.

Note that the two sets of 100,000 descriptors requires a sizeable portion of the dataset to be available. It may be more efficient to download the entire dataset, if possible. The built in downloading code does allow multiple files to be downloaded simultaneously for better bandwidth utilisation.

4.3 Reference and Sample descriptors

These are two sets of 1,000,000 descriptors. The script allows all of these to be recomputed, or a randomly selected subset of a specified size. In order to replicate these, you must replicate any of the charts in Figures 7 to 16, and request these files to be replicated in the replication settings. By default the replication of these caches is disabled.

The descriptors *should* replicate identically. If one or more are different, the benchmark will tell you how many could not be replicated, and you can choose whether to proceed anyway, or abort. If all descriptors were successfully validated, the program will print a message and move on without pausing.

One thing worth noting is that if you request a subset of values of be replicated, then all values for the scene each chosen sample point belongs to is replicated. For example, 100 descriptors are computed for each sample object. If the replication of only one sample point is requested, 100 data points will be replicated when the experiment completes. You may therefore notice that the number of replicated samples is much larger than the number you entered. In the case of the reference descriptor set, only one descriptor is computed per object.

4.4 Experimental results

Now that all the intermediate files have been replicated, we can move on to replicating the results themselves. The script is by default set to replicate 100 sample points. You can change the replication behaviour in the replication settings if you want to replicate all results, or replicate a different number of sample points.

When the replication is complete, you will be shown a table with statistics for the deviations between various replicated values, and their counterparts in the published results. This for instance includes the occlusion and clutter surrounding each point, which is computed for each scene point regardless of whether the clutter or occlusion filter is included in the filter sequence. If a generated scene is different in some way, it will show up as deviations in the computed clutter and occlusion values.

The table includes columns for:

- Total deviation: sum of absolute differences between all replicated values
- Average deviation: average absolute difference across all replicated values
- Maximum deviation: highest observed deviation
- Identical values: how many replicated data points were identical matches. This fraction should be high, though will not always be entirely identical due to rounding errors caused by differences between compilers.

The most important values with respect to the charts are the computed Descriptor Distance Index, and the PRC distance between the nearest and second nearest neighbours.

Along with the summary table, a CSV file with a complete comparison between all replicated values is written to disk. The path to where this file has been saved is shown near the bottom of the table.

The bottom entry in the menu for replicating Figures 7 to 16 can be used to take all precomputed results, and produce one PDF chart for each experiment, along with the support radius charts, and overview chart.

5 Troubleshooting

If your system is running out of memory when running a specific filter, you can try to limit the number of threads used for that filter in the replication settings. Memory usage will on average scale linearly with the number of threads that can work on a particular filter at the same time. Unfortunately, the exact memory requirements of each filter is rather difficult to predict, so you may need to play around a bit with the number of threads to find a good balance between speed and memory requirements.

6 Miscellaneous

Some final notes:

- A random seed is used for choosing each random subset of values to replicate. You can change this random seed in the replication settings.
- 4 out of the 10 charts use rendering to a framebuffer to determine the visibility of triangles. This rendering is done using OpenGL. Unfortunately, there are differences between how this rendering is implemented across different vendors. The benchmark therefore renders to an X virtual framebuffer by default. If this is giving you issues, you can enable visualisations in the replication settings. For the results presented in the paper, Mesa 23.1.4 was used for rendering. You may also want to enable visualisations and replicate a few of the clutter results to see the physics simulation in action.