Replication Notes for the paper "SHREC 2025: Partial Retrieval Benchmark"

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:

- 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. The replication script downloads a precomputed version of this file, and facilitates replicating the bounding sphere in the replication settings menu. You can find this menu under the "Replicate results and experiments" entry in the main menu.
- Reference descriptors: cached descriptors (usually 1 million of these) for each method. Precomputed versions of these are also downloaded by the script. The replication settings menu has an option to replicate these. Run any experiment, and the benchmark will perform a replication check before moving on to the selected experiment. Note that the benchmark will only halt execution when replication fails. If results are identical, the benchmark will show a terminal message and move on immediately.
- Benchmark results: each tested method will produce a JSON results file for each experiment, including those for measuring execution times (Figures 9 to 20, and 21+22 in the paper).

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 120GB of available disk space
- At least 32GB of RAM
- A CUDA capable GPU (only for the methods usung machine learning)

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. **Important: CUDA 13 does not work with the GEDI descriptor.** The new version introduces a number of breaking changes and deprecations that cause the GEDI descriptor to become unusable. We have tried to the best of our abilities to make this work, but unfortunately ended up running out of time.

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 120GB of storage space is needed for the precomputed cache files (dataset.json, and reference and sample descriptors for each method), a prepackaged conda environment for the GEDI descriptor, and the precomputed results. This also includes an approximately 10GB allocation for a local dataset cache.

Running the replication script only requires an installation of Python 3, which should run out of the box.

3 Downloading of data

The top entry in the main menu allows you to download the aforementioned files that are needed by the benchmark. The precomputed results and cache files are mandatory for all experiments, and the prebuild conda environment is only mandatory for the GEDI descriptor.

4 Compilation

To compile the project, you must use all entries in the *Install Dependencies* menu (except for CUDA, see the note above). After this, use the *Compile Project* entry in the main menu. When this process completes, you should be able to run all experiments of the benchmark. Note that 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 and can take several minutes.

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-conda-gedi` and `bin-python-cops` directories, you can try to compile with fewer threads (e.g. `ninja -j 10`).

5 Replication of results

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

5.1 Compressed dataset and dataset.json

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.

5.2 Reference descriptors

A set of 1,000,000 reference descriptors is computed for each method. 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 9 to 22. By default the replication of these descriptors 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.

5.3 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.

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.

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 JSON files used to generate the published charts. 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 is the computed Descriptor Distance Index. Also keep an eye out for occlusion and clutter values in experiments that list these in their descriptions.

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 9 to 20, and 21+22 can be used to take all precomputed results, and produce one PDF chart for each experiment, along with the execution time charts. Only a single chart is computed for experiment 9.

Separate replication entries are provided for figures 5 and 6, as well as the automated selection of the MICI density threshold parameter.

5.4 Execution times

Execution times are inherently non-deterministic, so the replication script only provides a means to run the experiments that were done as part of the paper, and to compute the execution time charts based on the precomputed results files. If you want to see the results from your own measurements, you can replace the `execution_times` directory in `precomputed_results` with the one from `output_paper`, and run the script that generates all charts. Also take note of the script's message asking you to disable CPU boosting in the BIOS. You should also try to minimise interference from other programs while doing performance testing.

6 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.

7 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.
- 7 out of the 9 experiments 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 25.0.7 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.