Guide for reproducing the results presented in the paper "Partial 3D Object Retrieval using Local Binary QUICCI Descriptors and Dissimilarity Tree Indexing"

Bart Iver van Blokland

July 2021

1 Overview

Greetings!

The repository in front of you contains a (hopefully batteries included) package for the reproduction of all results presented in the paper. This guide is intended to be a series of step by step instructions. You should follow them in the order they are listed.

Here are the system specifications you should have. Minimum specifications allow most results to be reproduced, while a system with the recommended ones should be able to do all.

Specification	**Minimum**	**Recommended**
CPU	Any x86 processor	
RAM	32 GB	64 GB
GPU	NVidia GPU with 8GB of VRAM	NVidia GPU with 16GB of VRAM
Hard drive	180GB available	
OS	Ubuntu 20.10	

You'll also find time indicators listed for various replication strategies. These are meant to be a rough lower bound for how long different replication steps take to complete. They were measured on a system with a Ryzen 3900X (12 core CPU) and an RTX 3090 GPU.

Before we start the replication process itself, we should first take a bird's eye view over all experiments in the paper's evaluation. It hopefully gives you an idea what to expect of the different experiments we've done. For any details you can of course refer to the paper.

The paper overall presents a pipeline for taking a portion of the surface of a 3D object, and locating which 3D object it is part of in a database of complete ones.

We use binary images called QUICCI descriptors for this. These images have either a resolution of 32x32, 64x64, or 96x96 bits (depending on the experiment).

The input we use to test the pipeline is a dataset from 2016, called the SHREC Partial Object Retrieval dataset. It unfortunately doesn't have many query objects, so we've developed another dataset which is derived from it. This derived dataset has two parts; a "Best Case" one which leaves triangles the same, and a "Remeshed" one, where the Best Case objects have been sent through a remeshing algorithm.

Figures 3, 4, and 6 are illustrations showing some design decisions that were made based on data.

Figures 10-17 are are experiments testing individual pipeline components, and finally the performance of the pipeline front to back.

1.1 Caveats with respect to replication

As far as we to the best of our abilities can tell, all data computed as part of this paper should be perfectly replicable.

The only issue of note is Figure 16, where we have used data from other papers directly. Since no source code is available for them, we cannot verify their accuracy.

2 Preparation

Time to get going. Before we can move on to the different figures, we need to install dependencies, compile the codebase, download datasets, and precompute some files that are used later on.

2.1 Install Dependencies

Start the 'replicate.py' script found in the root of the repository:

```
python3 replicate.py
```

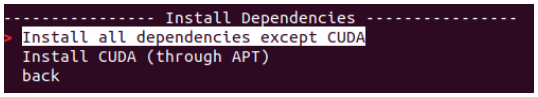
Please make sure the script is executed from the root of the repository, due to the many relative paths used in the script.

If the script fails to start, you can try installing the project's dependencies using one of the scripts provided in the scripts/ directory. These are the as those which can be installed through the script's menu system.

You can navigate these menus using the arrow keys, and press 'enter' to select.

The first step is to install all dependencies and download the auxiliary datasets (which were too large to host on github directly).

At this end, select the top entry called "Install dependencies":



We have separated the dependencies into 'the CUDA SDK' and 'everything that is not the CUDA SDK', as when CUDA has been installed using the run-file method from NVidia's website, installing it through APT can render both installations unusable.

If you're installing the project on a fresh installation of Ubuntu, everything should work out of the box when you use the APT method.

Make sure all dependencies are met before proceeding.

2.2 Download Datasets

Return to the Main Menu, and select the option "2. Download datasets".

The only strictly mandatory download is the SHREC 2016 partial shape dataset. All other results can be derived from this one.

However, the augmented dataset requires images to be rendered through OpenGL, which may not be possible if you try to run this script on a remote server. Generating it does not take much time (10m).

We recommend downloading the precomputed dissimilarity tree for at least the 96x96 descriptors. Computing all three trees took around 3.5 - 4 hours on my machine, 2.5 of which are taken up by the 96x96 one. The dissimilarity trees are needed for practically all figures.

The precomputed descriptors take about one hour to generate on my machine. It may be faster to download the precomputed ones. They are needed when replicating the index, and for replicating Table 1 and Figure 13.

2.3 Compile Project

Not much to this one, select the "Compile project" entry in the Main Menu, and all project files should be compiled automatically. Emphasis on should.

2.4 (Optional) Select GPU to use

Some systems have more than one GPU. Use this to select which one should be used.

2.5 Compute Descriptors

Descriptors are used for replicating the dissimilarity tree indexes, Figure 13, and Table 1 in the paper.

Using the "Generate all descriptors" option takes about an hour on my machine. If you downloaded the precomputed descriptors instead, make sure you use the "Copy all descriptors precomputed by authors" option.

2.6 Compute Dissimilarity Trees

Make sure you either generate each of these trees, or copy over whichever you downloaded previously.

3 Reproduction of Figures Shown in the Paper

3.1 Figure 3: Vote Counting Experiment

This experiment takes about 8 minutes on my machine.

Once it's finished, open the file:

```
output/Figure_3_voteCountProgression/query_progression.csv
```

In Libreoffice Calc, select all cells (Ctrl+A), then create an "XY Line" chart. Delete the legend to see the results.

3.2 Figure 4: Average Search Result Distance

The 1,000 queries which were executed as part of this figure take a long time to run. Therefore, the "Compute random search result" option allows you to generate one of them, then compare the results against those generated by us. Running one of these took less than a minute on my machine.

There's subsequently an option to compute the figure from our results, as well as an option to compute the entire chart from scratch.

3.3 Figure 6: Occurrence Count Heatmap

Not much to this one. Run the main menu entry, wait for a couple of minutes, observe the heatmap that pops up in a matplotlib window.

Unfortunately this does not work when you are on a remote machine. If so, download the following files:

```
output/Figure_6_OccurrenceCountHeatmap/shrec16_occurrence_counts.txt
scripts/heatmap.py
```

Into a directory on your local machine, and run the following command to render it:

```
python3 heatmap.py shrec16_occurrence_counts.txt
```

The script uses matplotlib and numpy, which you can install using pip3.

3.4 Figure 10 and 17: Dissimilarity Tree Query Times

There are two figures computed over the same data, so we have grouped them together.

If you look at Figure 10, there are two curves. One for indexed query execution times, and one for sequential searches.

Replicating all 100,000 indexed and 1,000 sequential queries will take a long time. Therefore, as before, we've added options to compute random subsections of them. For the indexed queries, you can compute 50 at a time, and the sequential ones go with 10 per batch. Both options require about 3 minutes on my machine.

Once each finishes executing, a table is shown which compares the computed results against those generated by us.

The third option takes the results computed by us, and produces a CSV file:

```
output/Figure_10_and_17_indexQueryTimes/authors_indexed_search_times.csv
```

The left four columns are for replicating Figure 10, and the first and last three are for replicating Figure 17. If you open the file in libreoffice, use the XY Line chart type.

3.5 Figure 11 and 12: QUICCI Modification Evaluation

When selecting this option from the main menu, there's a single experiment that runs, which takes about 6 minutes on my machine.

After the experiment is complete, the data is automatically processed into CSV files:

```
output/Figure_11_and_12_unwantedBitEvaluation/Figure_11_unwanted_bit_reductions.csv
output/Figure_11_and_12_unwantedBitEvaluation/Figure_12_overlap_with_reference.csv
```

In each case, open the file in libreoffice (or equivalent), and create XY Line charts of them.

3.6 Table 1 and Figure 13: All to all retrieval

This is the longest experiment of the paper to replicate in its entirety. Each row in the 383x383 confusion matrices in Figure 13 takes approximately 6-8 minutes to compute on my machine.

There are also two parameters here, whose permutations correspond to cells in Table 1. We use either the Best Case or Remeshed query objects from the augmented dataset we created, or enable/disable the modification to the QUICCI descriptor we proposed in the paper. Each parameter affects the quality of the search results in some way.

The top four menu entries allow you to replicate what is effectively one row in each of the confusion matrices in Figure 13 (although the paper does not show matrices for results where the original QUICCI descriptor is used). When selected, a random query object is selected, and upon completion of the experiment, the results are compared against those computed by us in a table.

When you're satisfied, you can compute both the table and confusion matrices using fifth menu entry.

The sixth entry allows you to compute the entire chart from scratch, which will probably take quite some time.

3.7 Figure 14 and 15: Evaluation of the Retrieval Pipeline

If you look at Figure 14, you can see that there are three parameters for the evaluation of the retrieval pipeline. There's the Best Case and Remeshed sets of query objects, there are three different descriptor resolutions being tested (32x32, 64x64, and 96x96 bits), and three different thresholds (10, 25, and 50).

As before, testing all of these will take a long time, although there's an option in the script for that should you wish to do so (menu entry 3). Alternatively, there's the option of trying a random sequence of 10 (menu entry 1) or 50 (menu entry 2) query objects.

Due to the number of parameters, rather than give a menu option for each permutation, I made a small menu system you can use to swap out parameters (menu entries 4, 5, and 6).

The remeshed queries take much longer than the best case ones, and so do higher threshold values. Here's how long a batch of 10 queries took to run on my machine when using a threshold of 10:

Resolution	Best Case Queries (seconds)	Remeshed Queries (seconds)
32x32	2	2
64x64	3	6
96x96	12	18

3.8 Figure 16: Artificial part of the SHREC 2016 benchmark

We test our pipeline on the two sets of 21 artificial queries from the SHREC'16 partial object retrieval benchmark. One set has 25% partiality, the other 40%.

Since the 21 object queries don't take all that long to execute, each option in the menu allows you to replicate one of the bars in Figure 16.

Note that since the benchmark is standardised, we were able to use the results presented in other papers directly. As no source code is available for them, we are not able to verify their correctness.