

Readme

Building on linux

In order to build the project the following dependencies should be fulfilled (all are in the ubuntu repositories):

- `cmake` (`build-essential`)
- `GL`, `GLU`, `GLEW` and `GLUT` (`mesa-common-dev` `libglew-dev` `freeglut3-dev`)
- `QT5` (`qtbase5-dev`)
- `eigen3` (`libeigen3-dev`)
- `GLM` (`libglm-dev`)
- `libann` (`libann-dev`)
- `boost` (`libboost-all-dev` or else at least `thread`, `system` and `chrono`)
- (optional) `doxygen` and `graphviz` for documentation
- (optional) `libglu1-mesa-dev` for testing the old branch with the old UI

To install all of the required ubuntu packages in one command, run

```
sudo apt-get install build-essential libglm-dev mesa-common-dev libglu1-mesa-dev
libglew-dev freeglut3-dev qtbase5-dev libeigen3-dev libann-dev libboost-all-dev
```

This codebase uses `C++14` and requires a modern `C++` compiler. When using GNU GCC at-least `g++ 4.9` is required. Make sure this is installed; use `g++ -v` to verify the version.

Build by executing from the project root:

```
mkdir build
cd build
cmake ..
make
#(optional) build doxygen documentation in build/doc
make doc
#put the files in the right places in a fancy redistributable directory
#needed as cmake has no `make dist`, the files will by default be 'installed'
#in '${CMAKE_BINARY_DIR}/VoxelSkeleton/'
make install
```

Other operating systems

Since the project uses CMake as the build system, it can be build on many platforms, including OS X and Windows. However, the installation of the dependencies may vary depending on the platform.

Formatting

Automatic formatting was preformed using

```
astyle --style=allman -s2 -p -W3 -c *.h *.cc
```

Try and keep to allman style, 2 spaces for indentation.

Usage

```
./VoxelSkeleton <path_to_model>
```

Running this command launches the QT application with the specified input volume. The application contains an options menu, and a rendering view. For a technical manual of using the application, please read this [link](#).

Directory structure

Folder structure options and naming conventions for this project.

Top-level directory layout

```
.
├── build                # Compiled files (not distributed)
├── cmake-modules        # Cmake modules which are not included by default
├── doc                  # Documentation files
├── SkeletonLib          # Skeleton library, contains core functionality
├── SkeletonTest         # Test scripts (manual)
├── include              # Include files (For the App)
├── src                  # Source files (For the App)
├── VoxelSkeleton        # Contains shader and table files needed by runtime
├── CMakeLists.txt       # CMake list file
└── README.md
```

The important algorithms are contained in the `SkeletonLib` directory.

Important code

In this section the important code of Herman's thesis is listed. The global streamline filtering, and the derivative filtering algorithms are contained in `SkeletonLib/src/SaliencyMetricEngine.cc`. Here, the Global streamline method (the final approach in the thesis) is defined by the function:

```
Volume<float> SaliencyMetricEngine::ComputeGlobalMonotonic(Volume<Vector>& velocity,
surface::Graph& graph, float impThreshold);
```

Which requires a velocity field `velocity` (for derivative computation), the graph `graph` for the

feature points, and the importance value `impThreshold`, which is the pre-pruning threshold for removing small-scale noise using the geodesic measure, which can be used for eliminating smaller ligature parts. The function returns a scalar volume which can be used to recover filtered connected skeletons skeletons by thresholding it, which should contain core skeleton parts intact (even the parts with low geodesic importance).

The reconstruction algorithm is given in `SkeletonLib/src/SkeletonReconstructor.cc`, which also includes the minification filter. The reconstruction function is defined by:

```
Volume<byte> Reconstruct(ReconstructionSmoothingType smoothingType, int radius,
    const Volume<float>* imp = nullptr, float threshold = 0, surface::Graph* graph
= nullptr,
    FeaturePoints* featurePoints = nullptr, bool constrainSearchToSkeleton =
false);
```

where `smoothingType` is the type of smoothing as defined in the thesis (minification, opening, mean, etc), `radius` is the radius of the smoothing kernel, `imp` is the importance volume of the skeleton, `threshold` the importance threshold used to extract the skeleton, `graph` the graph of the surface of the skeleton, `featurepoints` the featurepoints of the skeleton and `constrainSearchToSkeleton`, whether the kernel should be constrained to the skeleton (explicit mode or implicit mode). The function returns a (binary) byte volume containing the reconstructed model.