University of Crete

Computer Science Department

Thesis

Architecture Visualizer

(Archv)

Soultatos Stefanos

A.M: 3845

Advisor: Anthony Savidis, Computer Science Professor

Acknowledgments

I am extremely thankful to my supervisor Prof. Anthony Savidis for his invaluable advice during my thesis. I would also like to thank Krystallia Savvaki, since a large part of this work is based on her master thesis: “Architecture Miner”.

Table of Contents

1. Introduction

It’s imperative to methodically think through the software architecture for effective development of large scale systems, in order to define attributes such as performance, quality, scalability, maintainability, manageability and usability. However, many times what ends up happening, is that this conceptual model that formally describes the structure of a system, doesn’t accurately express the actual underlying physical architecture.

Thus, the purpose of this thesis is to provide a tool to visualize, in a user-friendly way, the program class relationships of a system’s software architecture, that has been reconstructed from the actual source code (C++), by the: “Architecture Miner” project. This is achieved by rendering a graph in three dimensions, where each vertex represents a class, and each edge represents a physical dependency between its source and target vertices; while providing a Graphical User Interface for interacting with various graph properties, like its layout, clusters, scale and more.

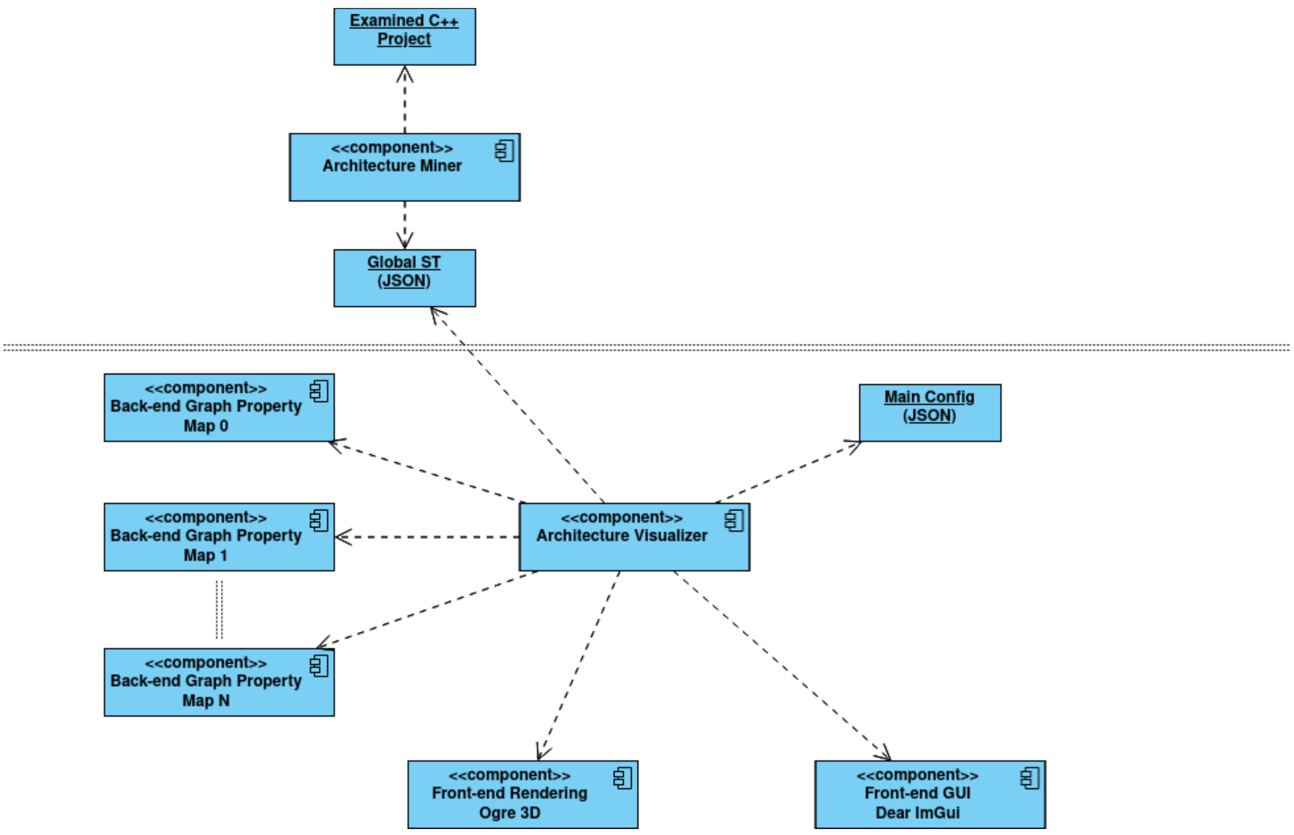
That way, the overall design quality of a system can be evaluated, by assessing the cohesion and coupling of its modules, in an intuitive and interactive way.

2. System overview

The overall system that was implemented for this thesis consists of two main layers: the symbol table export, (via the “Architecture Miner”) and the architecture visualization, (via the “Architecture Visualizer”). The former is responsible for reconstructing a system’s software architecture from C++ source code based on program class relationships. It achieves this by using the Clang compiler fontend to generate a dependency graph and by storing it in JSON format. The latter uses this JSON file to render a visualization of the dependency graph in 3D space, via the OGRE 3D rendering engine, while providing a Graphical User Interface, via Dear ImGui, to interact with various extrinsic and intrinsic graph properties. Each graph property, (like the position of each vertex), is interfaced through a back-end, property map based, subsystem, to compose a flexible graph interface.

(To read more about property maps, see: https://www.boost.org/doc/libs/1_48_0/libs/property_map/doc/property_map.html).

The design is presented on the figure below.


Examined C++ Project
<<component>> Architecture Miner
Global ST (JSON)
<<component>> Back-end Graph Property Map 0
<<component>> Back-end Graph Property Map 1
<<component>> Back-end Graph Property Map N
Main Config (JSON)
<<component>> Architecture Visualizer
<<component>> Front-end Rendering Ogre 3D
<<component>> Front-end GUI Dear ImGui

3. (Extra) Set-Up / Build

The following guide will explain to you how to build a fork of the “Architecture Miner” from source.

The “Architecture Miner” project produces the output .json file, from which the “Architecture Visualizer” will render the architecture graph.

CMake will be used as the build system, (tested on Ubuntu Debian Pop!_OS version 22.04 LTS).

3.1. Getting dependencies

- **Clang**
  Follow the steps described at: https://github.com/llvm/llvm-project.
  Make sure to configure the build process with: `-DLLVM_ENABLE_PROJECTS="clang"`.
  (Tested with the 15.0.2 clang version).

- **Jsoncpp**
  $ *`sudo apt-get install libjsoncpp-dev`*
  This will install the compatible JsonCpp library on your system.

- **wxWidgets**
  For building this library, please see platform-specific documentation under `docs/<port>` directory at: https://github.com/wxWidgets/wxWidgets.
  (Tested with the 3.1 w.x version).

3.2. CMake optional variables

Before building the application, some final CMake variables can optionally be set. Namely, the: CLANG_INCLUDE_DIR, INCREMENTAL_GENERATION, GUI.

CLANG_INCLUDE_DIR (`string`)
The directory path to the .h files for clang.
E.g: /usr/lib/llvm-15/lib/clang/15.0.2/.
If this variable is not configured appropriately, many mining attempts may result to compilation failures.

INCREMENTAL_GENERATION (`bool`)
Controls wether to mine incrementally after cancel and re-run, (true by default).
When mining incrementally, the miner can pickup and continue previous mining sessions on launch.

GUI (`bool`)
Controls wether to display a GUI progress bar, in order to monitor the mining session and cancel it if needed, (true by default).

3.3. Building

Clone the repo:

```
$ git clone https://github.com/SoultatosStefanos/Code-Smell-Detector
```

You should create a build directory outside the project’s sources/

Then:

```
$ cd build && cmake ..
$ make
```

In order to build the application.

4. Set-Up / Build

The Architecture Visualizer (Archv) uses CMake as its build system on all supported platforms (tested on Ubuntu Debian Pop!_OS version 22.04 LTS).
This guide will explain to you how to use CMake to build the project from source, under this OS.

4.1. Getting dependencies

- **GoogleTest**
  (This dependency can be skipped if you do not wish to build the Archv tests.)
  Download the v1.12.1 release from: https://github.com/google/googletest. You should now create a build directory for GTest somewhere outside GTest's sources.
  Then:
  ```
  $ cd build && cmake ..
  $ make
  $ sudo make install
  ```
  Archv should now be able to locate a CMake configuration file for GTest. If not, the path to the directory containing the CMake configuration file must be given to CMake with the CMake variable: Gtest_DIR

- **Boost**
  ```
  $ sudo apt-get install libboost-all-dev
  ```
  This will install all of the required Boost modules (Graph, Signals2, Log, Exception). If a compile/linking error occurs when building Archv from source, make sure that the CMake variables: Boost_INCLUDE_DIR, Boost_LOG_LIBRARY_RELEASE , Boost_LOG_SETUP_LIBRARY_RELEASE and Boost_PROGRAM_OPTIONS_LIBRARY_RELEASE are configured appropriately

- **Jsoncpp**
  ```
  $ sudo apt-get install libjsoncpp-dev
  ```
  This will install the compatible JsonCpp library on your system. Archv should now be able to link against the libjsoncpp target.

- **Infomap**
  This dependency is being used as a git submodule (see https://git-scm.com/book/en/v2/Git-Tools-Submodules) and will be fetched and built from source when building the Archv application. (see Set-Up).

- **IconFontCppHeaders**
  This dependency is being used as a git submodule (see https://git-scm.com/book/en/v2/Git-Tools-Submodules) and will be fetched and built from source when building the Archv application. (see Set-Up).

- **SDL**
  ```
  $ sudo apt-get install libsdl2-dev
  ```
  This is also an Ogre dependency. (see https://ogrecave.github.io/ogre/api/1.12/building-ogre.html).

This will install the compatible SDL library on your system. Archv should now be able to link against the libSDL2 target.

- **OpenGL**
  ```
  $ sudo apt-get install libgles2-mesa-dev
  ```
  This is an Ogre dependency. (see https://ogrecave.github.io/ogre/api/1.12/building-ogre.html). Archv should now be able to locate a CMake configuration file for OpenGL.

- **Freetype**
  See: https://github.com/freetype/freetype/blob/master/docs/INSTALL.UNIX
  Clone the repo from: https://github.com/freetype/freetype.
  ```
  $ git clone https://github.com/freetype/freetype
  ```
  You should now create a build directory for Freetype somewhere outside Freetype's sources.
  Then:
  ```
  $ cd build && cmake ..
  $ make
  $ sudo make install
  ```
  Archv should now be able to locate a CMake configuration file for Freetype. If not, the path to the directory containing the CMake configuration file must be given to CMake with the CMake variable: FreeType_DIR

- **Stb**
  ```
  $ sudo apt install libstb-dev
  ```
  This will install the compatible stb library on your system. Archv should now be able to link against the libstb target.

- **Dear ImGui**
  Just clone the repo from: https://github.com/ocornut/imgui somewhere.
  ```
  $ git clone https://github.com/ocornut/imgui
  ```
  The build process of the Ogre Overlay component will take care of the rest.

- **OGRE**
  See: https://ogrecave.github.io/ogre/api/1.12/building-ogre.html.
  Download a v13 release from: https://github.com/OGRECave/ogre.
  You should now create a build directory for Ogre somewhere outside Ogre's sources.
  Then:
  ```
  $ cd build
  ```
  Archv makes use of the Bites & Overlay components, so be sure to build them as well. Thus the CMake variables: OGRE_BUILD_COMPONENT_BITES, OGRE_BUILD_COMPONENT_OVERLAY, OGRE_BUILD_COMPONENT_OVERLAY_IMGUI, OGRE_BUILD_DEPENDENCIES must be set to: TRUE. Also, make sure the CMake variable IMGUI_DIR is set to the directory path of the imgui headers.
  The rest of Ogre's settings can be configured as you wish.
  (Tip: use the cmake-gui tool here, see: https://cmake.org/cmake/help/latest/manual/cmake-gui.1.html).
  ```
  $ make
  $ sudo make install
  ```

Archv should now be able to locate a CMake configuration file for Ogre. If not, the path to the directory containing the CMake configuration file must be given to CMake with the CMake variable: OGRE_DIR

- **Ogre Procedural**
  This is a dependency that is not packaged with Ogre from the previous step.
  Clone the repo from: https://github.com/OGRECave/ogre-procedural.
  ```
  $ git clone https://github.com/OGRECave/ogre-procedural
  ```
  You should now create a build directory for Ogre Procedural somewhere outside Ogre Procedural's sources.
  Then:
  ```
  $ cd build && cmake ..
  ```
  (But make sure that the CMake variable: OGRE_DIR is set to the directory path containing a CMake configuration file for Ogre).
  ```
  $ make
  $ sudo make install
  ```
  Archv should now be able to locate a CMake configuration file for Ogre Procedural. If not, the path to the directory containing the CMake configuration file must be given to CMake with the CMake variable: OGRE_PROCEDURAL_DIR

4.2. CMake optional variables

Before building the application, some final CMake variables can optionally be set. Namely, the: ARCHV_RESOURCE_GROUP.
The ARCHV_RESOURCE_GROUP CMake variable is the resource group name used by Archv in order for Ogre to locate its assets. (see Configuration Files section at: https://ogrecave.github.io/ogre/api/1.12/setup.html).

NOTE: If the Archv resource group name has not been manually specified, it will be defaulted to: 'Archv'.

4.3. Notes

Archv makes use of several C++20 library features (input range adapters), thus certain compilers that do not support these features will fail to build this project, (like Clang libc++), (as of March 2023).
(See https://en.cppreference.com/w/cpp/compiler_support).
Archv has been tested with GCC libstdc++.

4.4. Building

Clone the repo:
```
$ git clone https://github.com/SoultatosStefanos/archv
```
Init the git submodules:
```
$ git submodule update --init --recursive
```
You should create a build directory for Archv somewhere outside Archv's sources.
Then:

```
$ cd build && cmake ..
$ make
$ make archv_app
$ make archv_tests
```

In order to build all the targets, the application, or the tests respectively.

4.5. Set-Up

Lastly, before attempting to run the application, the directory paths of all of the used assets must be specified for Ogre, under the Archv resource group name.
Add the following lines to your resources.cfg file: (probably located under: /usr/local/share/OGRE)

```
[<Archv resource group name>]
FileSystem=/path/to/archv/data/textures
FileSystem=/path/to/archv/data/materials
FileSystem=/path/to/archv/data/models
FileSystem=/path/to/archv/data/fonts
FileSystem=/path/to/archv/data/particles
```

(See Configuration Files section at: https://ogrecave.github.io/ogre/api/1.12/setup.html).

NOTE: If the Archv resource group name has not been manually specified, it will be defaulted to: 'Archv'. (see CMake optional variables).

NOTE: Any additional resources added to the above directory paths, (or any other paths specified at the resources.cfg), will be automatically located at runtime.

5. Usage

The visualizer uses the output .json file from a fork of the “Architecture Miner” project (https://github.com/SoultatosStefanos/Code-Smell-Detector), as command line input.

Additionally, a .json configuration file must be specified.

```
$ ./archv_app foo/bar/buzz/graph.json zoo/buzz/myconfig.json
```

Or, in order to run the tests:

```
$ ./tests/archv_tests
```

6. Configuration

Archv supports both interactive configuration, via GUI, and startup configuration with a .json configuration file.
This section will explain to you how to best configure the application's style, and/or graph visualization properties.

6.1. Graph layout

The graph's visualization properties regarding its layout & topology in 3D space can be configured with plugged in algorithms and values.

Example .json configuration:

```
"layout" :
{
    "layouts" :
    [
        "Gursoy Atun"
    ],
    "topologies" :
    [
        "Cube",
        "Sphere"
    ],
    "layout" : "Gursoy Atun",
    "topology" : "Sphere",
    "scale" : 1300
}
```

layouts (`string list`)

The available layout algorithms that can be selected at runtime.
A layout algorithm generates the position of each vertex inside the 3D space.

Possible values: <**Gursoy Atun** | **Random**>

topologies (`string list`)

The available topologies that can be selected at runtime.
A topology is a description of space inside which the layout is performed.

Possible values: <**Cube** | **Sphere**>

layout (`string`)

The default layout algorithm used.

Possible values: **one listed under layouts**.

topology (`string`)

The default topology used.

Possible values: **one listed under topologies**.

scale (`double`)

The default scale used to generate the graph layout.
This variable controls the "magnitude" of the layout, that is, how far the graph will take up space.

Possible values: **any positive floating point number**.

6.2. Edge weight properties

Physical dependencies across C++ components are expressed with edges in the visualized dependencies graph. The "weight" value of each type of dependency can be specified here.

The weight values of each dependency affect the output of both clustering and layout algorithms.

Example .json configuration:

```
"weights" :
{
    "dependencies" :
    [
        {
            "Inherit" : 1
        },
        {
            "Friend" : 1
        }
    ]
}
```

Where each **dependency** (`string`) is paired with a **weight** (`int`).

NOTE: Each dependency found in the graph .json input file must be included here.

6.3. Vertex scaling properties

Archv can be configured in order to apply dynamic scaling to each graph vertex, depending on the vertex underlying class metadata.
That way, quick visual comparisons can be made between vertices.
Here, scaling factors are defined. These factors can be taken into account, and/or combined, in order to scale the graph's vertices in relation to each other.

Example .json configuration:

```
"scaling" :
{
    "factors" :
    [
        {
            "Fields" :
            {
                "enabled" : true,
                "dimensions" : [ true, true, true ],
                "baseline" : 3,
                "ratio" : { "min" : 0.5, "max" : 2.5 }
            }
        }
    ]
}
```

factors (`objects list`)

The available scaling factors that can be selected at runtime.

Possible values: <**Fields** | **Methods** | **Nested**>.
In order to scale according to the number of the vertex underlying class: fields , methods, nested classes, respectively.

For each scaling factor the following variables can be specified:

enabled (`bool`)

Wether this scaling factor / class metadata property is taken into account when computing the final vertex scale.

dimensions (`bool array`)

The axes on which the scaling is applied on the vertex. For each axis, (x, y, z) ,a boolean value is specified, in order to indicate that the scaling is applied.
E.g. `[true, false, true]` means that the scaling will be applied on the x, z axes only.

baseline (`double`)

This is the assumed, system-wide, average value of each class metadata property. E.g. A baseline of: `3` for a `Methods` scaling factor means that it is assumed that on average a class of the visualized software contains 3 methods. Thus, vertices whose underlying classes contain more than 3 methods will appear larger, and vertices whose underlying classes contain less than 3 methods will appear smaller.

Possible values: **any positive floating point number**.

ratio

min (`double`)
max (`double`)

The min/max ratio values of each scaling factor underlying class metadata property, in comparison to the baseline, can be specified here.
That way, with an e.g. min ratio value of `1`, we can be sure that vertices, whose underlying class metadata property fall behind the baseline, will never appear smaller than the average one.
Useful in order to prevent vertices from going invisible or appearing too big.

Possible values: **any positive floating point number**.

6.4. Graph clustering

Archv features real-time graph clustering, with plugged in graph clustering algorithms.
In addition, for many clustering algorithms, specific parameters can be configured.

Example .json configuration:

```
"clustering" :
{
    "clusterers" :
    [
        "Louvain Method",
        "Layered Label Propagation",
        "Infomap"
    ],
    "min-spanning-tree-finders" :
    [
        "Prim MST"
    ],
    "clusterer": "Infomap",
    "intensity" : 2000,
    "min-spanning-tree-finder" : "Prim MST",
    "k" : 3,
    "snn-threshold" : 5,
    "min-modularity" : 0.2,
    "llp-gamma" : 0.2,
    "llp-steps" : 2
}
```

clusterers (`string list`)

The available clustering algorithms that can be selected at runtime.

Possible values: <**k-Spanning Tree | Shared Nearest Neighbour | Strong Components | Maximal Clique Enumeration | Louvain Method | Layered Label Propagation | Infomap**>

min-spanning-tree-finders (`string list`)

The available minimum spanning tree finding algorithms used by the **k-Spanning Tree** clustering algorithm.

Possible values: <**Prim MST | Kruskal MST**>

clusterer (`string`)

The default clustering algorithm selected.

Possible values: **one listed under clusterers**.

intensity (`double`)

The scale of "clustered" layout, which attempts to showcase the adjacency of the clusters.

Possible values: **any floating point number**.

min-spanning-tree-finder (`string`)

The default minimum spanning tree finding algorithm selected.

Possible values: **one listed under min-spanning-tree-finders**.

k (`int`)

The "k" value of the **k-Spanning Tree** clustering algorithm.

Possible values: **any positive integral number larger than one**.

snn-threshold (`int`)

The "t" value of the **Shared Nearest Neighbour** clustering algorithm.

Possible values: **any integral number**.

min-modularity (`double`)

The minimum required modularity increase at each modularity optimization step, of the **Louvain Method** clustering algorithm.
The resolution of the algorithm.

Possible values: **any floating point number**.

llp-gamma (`double`)

The **Layered Label Propagation** clustering algorithm "gamma" variable.

Possible values: **any floating point number**.

llp-steps (`double`)

The maximum number of iterations of the **Layered Label Propagation** clustering algorithm.

Possible values: **any integral positive number**.

6.5. Edge color coding

Physical dependencies across C++ components are expressed with edges in the visualized dependencies graph. The "color" value of each type of dependency can be specified here.

Example .json configuration:

```
{
  "color-coding": {
    "dependencies": [
      {
        "dependency": "Inherit",
        "color": [255, 0, 0, 1],
        "active": true
      }
    ]
  }
}
```

Where each **dependency** (`string`) is paired with a **color** (`double array`), and an **active** (`boolean`) value, which indicates wether the color coding will be rendered.

NOTE: Each dependency found in the graph .json input file must be included here.

6.6. Vertex degrees particles

Additionally, Archv features particle system effects rendering at each vertex, depending on the vertex in/out degree. Some configuration options for these effects can be configured as well.

Example .json configuration:

```
"degrees" :
{
    "in-degree" :
    {
        "thresholds" :
        {
            "light" : 1,
            "medium" : 4,
            "heavy": 8
        },
        "particle-systems" :
        {
            "light" : "Degrees/SwarmLight",
            "medium" : "Degrees/SwarmMedium",
            "heavy" : "Degrees/SwarmHeavy"
        },
        "applied" : true
    },
    "out-degree" :
    {
        "thresholds" :
        {
            "light" : 1,
            "medium" : 4,
            "heavy": 8
        },
        "particle-systems" :
        {
            "light" : "Degrees/SwarmLight",
            "medium" : "Degrees/SwarmMedium",
            "heavy" : "Degrees/SwarmHeavy"
        },
        "applied" : false
    },
},
```

The particle system effects rendering options can be specified for both in and out vertex degrees.

For each degree the following can be specified:

thresholds

light (`int`)
medium (`int`)
heavy (`int`)

Here, the thresholds for when to render each effect for each degree value are specified.

Possible values: **any integral positive number**.

particle-systems

light (`string`)
medium (`string`)
heavy (`string`)

Here, the name of the particle system to be rendered for each level of degree is specified. When a vertex in/out degree is between the light and medium thresholds, then the light particle system is rendered, when it's between the medium and heavy thresholds, then the medium particle system is rendered, and when it's above, or equal to the heavy threshold, then the heavy particle system is rendered.

Possible particle system name values: **any particle system name defined in an imported .particle file**.

applied (`bool`)

Wether to render the particle system for each type of vertex degree.

6.7. Graph misc. rendering options

Archv can be configured regarding the architecture graph's extrinsic rendering properties, like the materials and meshes used to render the scene.

Example .json configuration:

```
"rendering" :
{
    "background" :
    {
        "skybox-material" : "Gradient/Dark",
        "skybox-distance" : 5000,
        "ambient-color" : [ 0, 0, 0 ],
        "diffuse-color" : [ 0.4, 0.43, 0.42 ],
        "specular-color" : [ 0.337, 0.325, 0.325 ],
        "cam-far-clip-distance" : 0,
        "cam-near-clip-distance" : 5
    },
    "graph" :
    {
        "vertex-mesh" : "Cube.001.mesh",
        "vertex-material" : "Shaded/StudioGold",
        "vertex-scale" :  [     5, 5, 5 ],
        "vertex-id" :
        {
            "font-name" : "Gecko-Personal",
            "char-height" : 6,
            "color" : [ 1, 1, 1 ],
            "space-width" : 0
        },
        "edge-material" : "Shaded/LightBlue",
        "edge-tip-mesh" : "triangle.mesh",
        "edge-tip-material" : "Shaded/LightBlue",
        "edge-tip-scale" : [ 1.25, 1.25, 1.25 ],
        "edge-type" :
        {
            "font-name" : "Gecko-Personal",
            "char-height" : 2.5,
            "color" : [ 1, 1, 1 ],
            "space-width" : 0
        }
   },
    "minimap" :
    {
        "left" : 0.5,
        "top" : -0.5,
        "right" : 0.95,
        "bottom" : -0.95,
        "background-color" : [ 0.0, 0.0, 0.0 ],
        "zoom-out" : 300,
        "render-shadows" : false,
        "render-sky" : true,
        "render-vertices" : true,
        "render-vertex-ids" : false,
        "render-edges" : true,
        "render-edge-types" : false,
        "render-edge-tips" : false,
        "render-particles" : false
    }
}
```

Possible mesh values: **any .mesh file name defined under an imported directory**.

Possible materials values: **any material name defined in any imported .material script**.

Possible color values: **any floating point value between 0 and 1**.

Possible font name values: **any font name defined in an imported .fontdef file**.

6.8. GUI misc. options

Archv's gui style can be configured as well.

Example .json configuration:

```
"gui" :
{
    "color-theme" : "Dark",
    "frame-rounding" : 0,
    "window-bordered" : true,
    "frame-bordered" : true,
    "popup-bordered" : true
}
```

color-theme (`string`)

The color theme of the gui.

Possible values: <**Dark** | **Light** | **Classic**>

frame-rounding (`double`)

Specifies how rounded the frames of the gui appear.

Possible values: **any floating point number**.

window-bordered (`bool`)

Wether the windows will appear bordered.

frame-bordered (`bool`)

Wether the frames will appear bordered.

popup-bordered (`bool`)

Wether the popups will appear bordered.

7. Demo

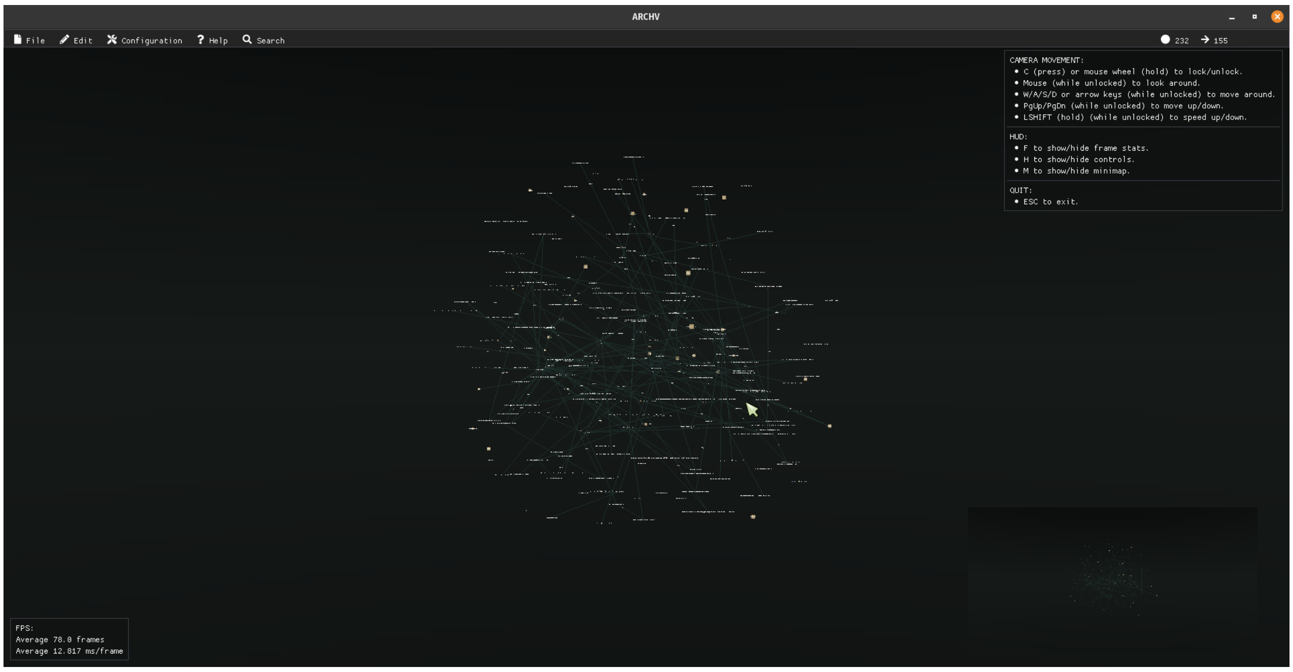
The following section is a demonstration of the Architecture Visualizer’s features.

It showcases a visualization of an architecture graph, that was constructed from mining a section of the standard C++ library, (Ubuntu Debian Pop!_OS version 22.04 LTS).

NOTE*:* All the screenshots have been captured from a debug build, thus the application’s performance, as measured from the fps overlays, should not be evaluated from this section.

7.1. Overview

After launching the application, a zoomed-out view of the architecture graph, in 3 dimensions, will be rendered. The graph that is being currently visualized consists of 232 vertices (classes), and 155 edges (dependencies), (upper-right corner, on the navigation bar).



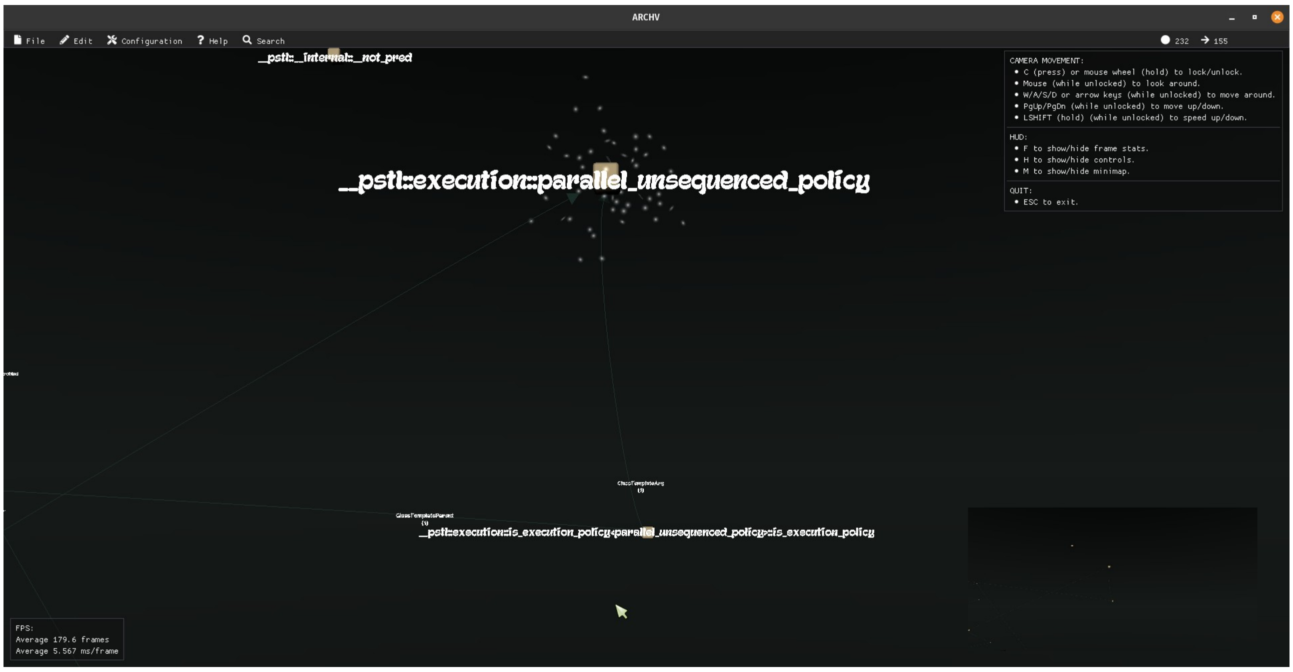
CAMERA MOVEMENT:
- C (press) or mouse wheel (hold) to lock/unlock.
- Mouse (while unlocked) to look around.
- W/A/S/D or arrow keys (while unlocked) to move around.
- PgUp/PgDn (while unlocked) to move up/down.
- LSHIFT (hold) (while unlocked) to speed up/down.

HUD:
- F to show/hide frame stats.
- H to show/hide controls.
- M to show/hide minimap.

QUIT:
- ESC to exit.

FPS:
Average 78.0 frames
Average 12.817 ms/frame

During this visualization session, each vertex is illustrated as a cube, while each edge is illustrated as a curved arrow, pointing from the dependent to the depended.


ARCHV
File
Edit
Configuration
Help
Search
232
155
__pstl::__internal::__not_pred
__pstl::execution::parallel_unsequenced_policy
ClassTemplateArg
ClassTemplateParent
__pstl::execution::is_execution_policy<parallel_unsequenced_policy>::is_execution_policy

CAMERA MOVEMENT:
- C (press) or mouse wheel (hold) to lock/unlock.
- Mouse (while unlocked) to look around.
- W/A/S/D or arrow keys (while unlocked) to move around.
- PgUp/PgDn (while unlocked) to move up/down.
- LSHIFT (hold) (while unlocked) to speed up/down.

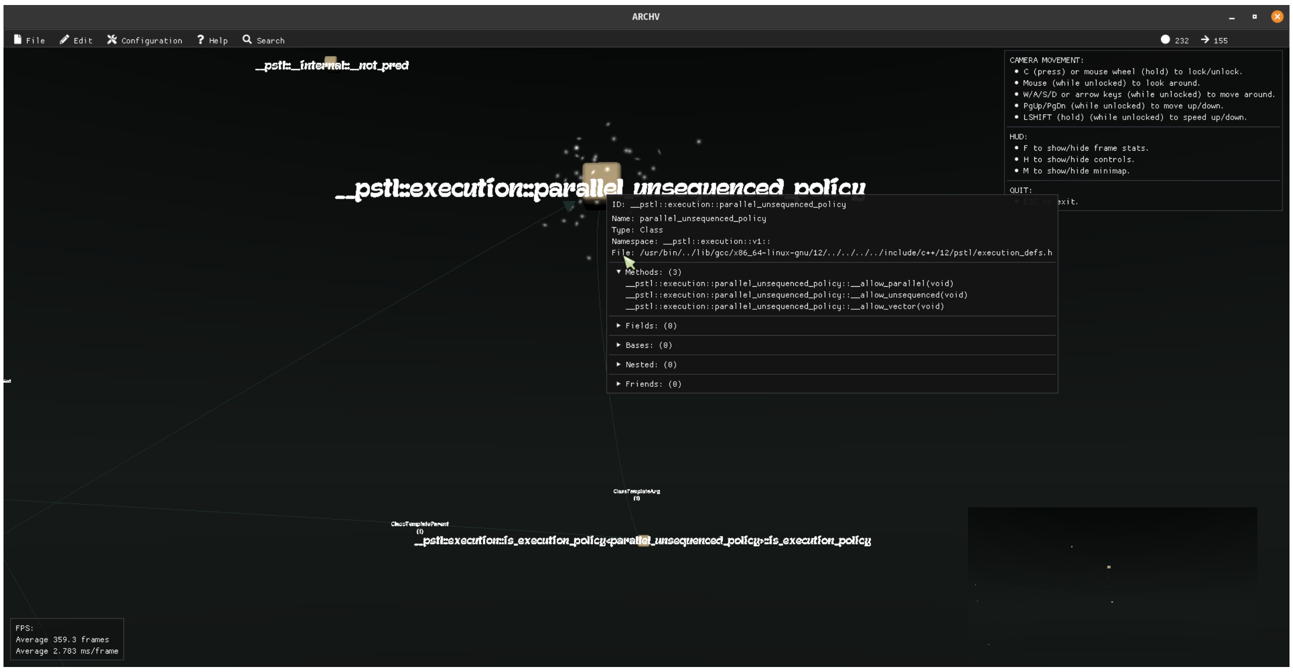
HUD:
- F to show/hide frame stats.
- H to show/hide controls.
- M to show/hide minimap.

QUIT:
- ESC to exit.

FPS:
Average 179.6 frames
Average 5.567 ms/frame

7.2. Vertex meta-data

The visualizer features class meta-data inspection. When clicking at a vertex, a popup modal window is shown, which offers listings of various class properties, like its fields, methods, base classes, friends, etc.


ARCHV
File
Edit
Configuration
Help
Search
232
155
__pstl::__internal::__not_pred
__pstl::execution::parallel_unsequenced_policy
ClassTemplateArg
ClassTemplateParent
__pstl::execution::is_execution_policy<parallel_unsequenced_policy>::is_execution_policy

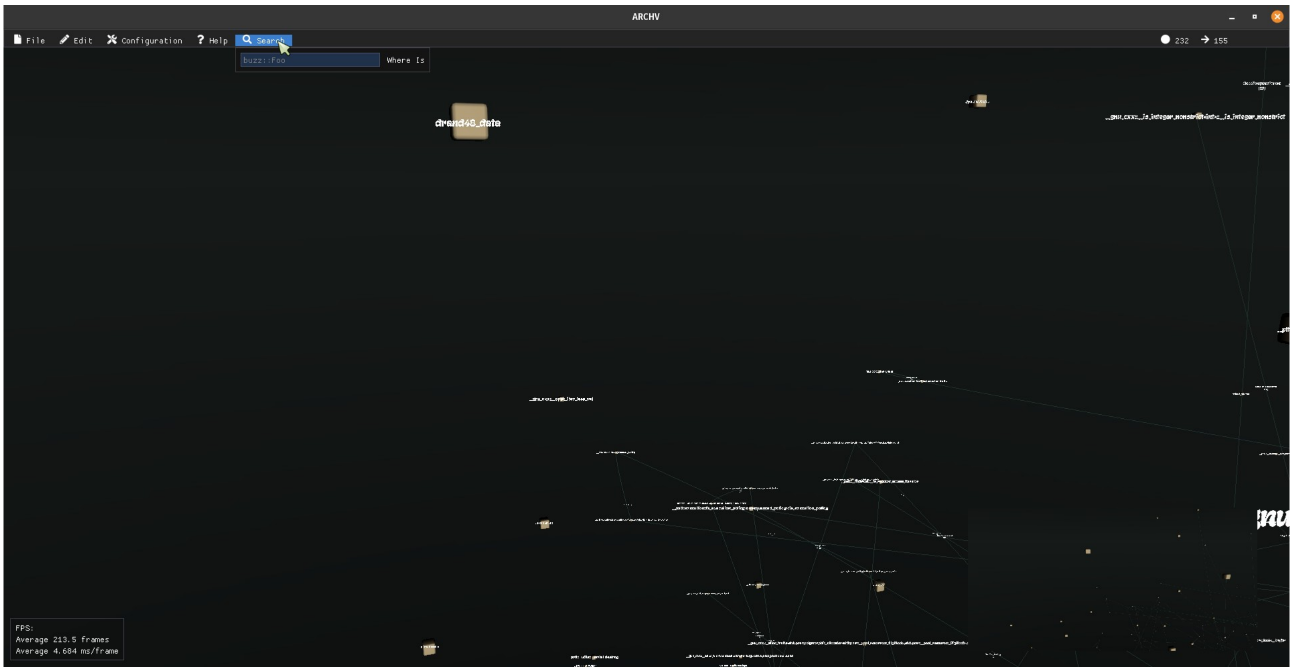
```
ID: __pstl::execution::parallel_unsequenced_policy
Name: parallel_unsequenced_policy
Type: Class
Namespace: __pstl::execution::v1::
File: /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/pstl/execution_defs.h
▼ Methods: (3)
  __pstl::execution::parallel_unsequenced_policy::__allow_parallel(void)
  __pstl::execution::parallel_unsequenced_policy::__allow_unsequenced(void)
  __pstl::execution::parallel_unsequenced_policy::__allow_vector(void)
▶ Fields: (0)
▶ Bases: (0)
▶ Nested: (0)
▶ Friends: (0)
```

```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

```
FPS:
Average 359.3 frames
Average 2.783 ms/frame
```

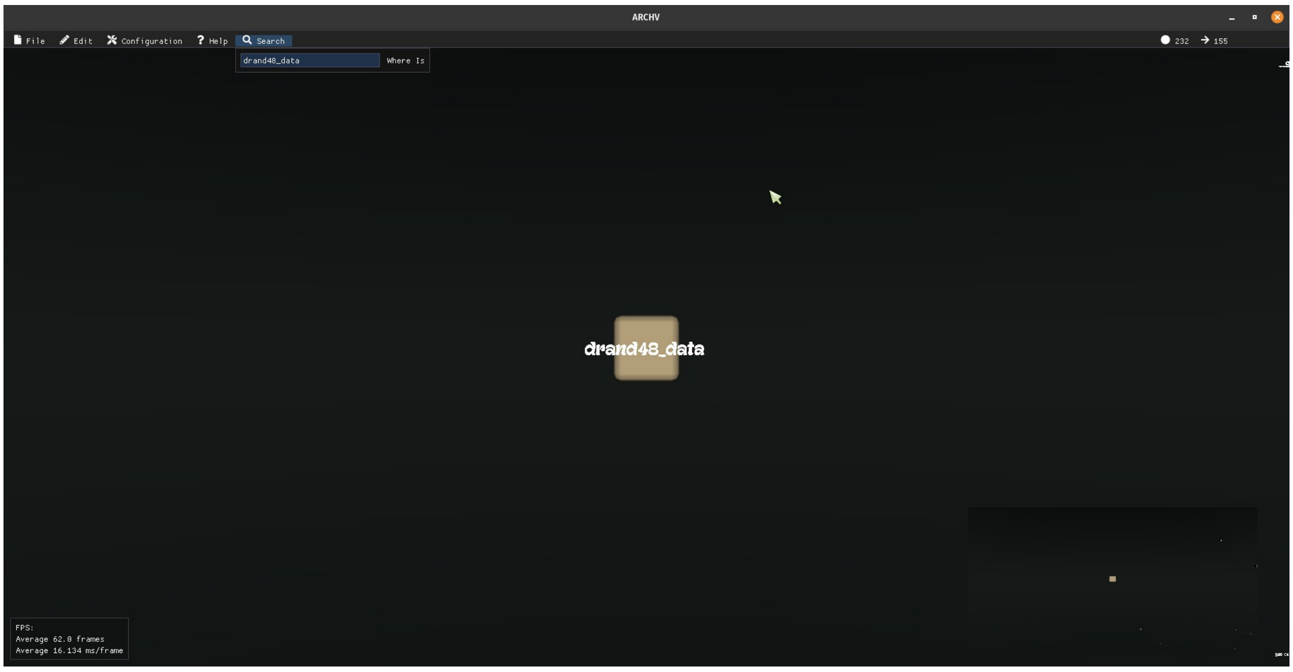
7.3. Vertex search

The following images showcase another useful feature: looking-up vertices, by their class names, from the search bar, on the navigation bar.



```
FPS:
Average 213.5 frames
Average 4.684 ms/frame
```

The camera will pan to the found vertex, once found.

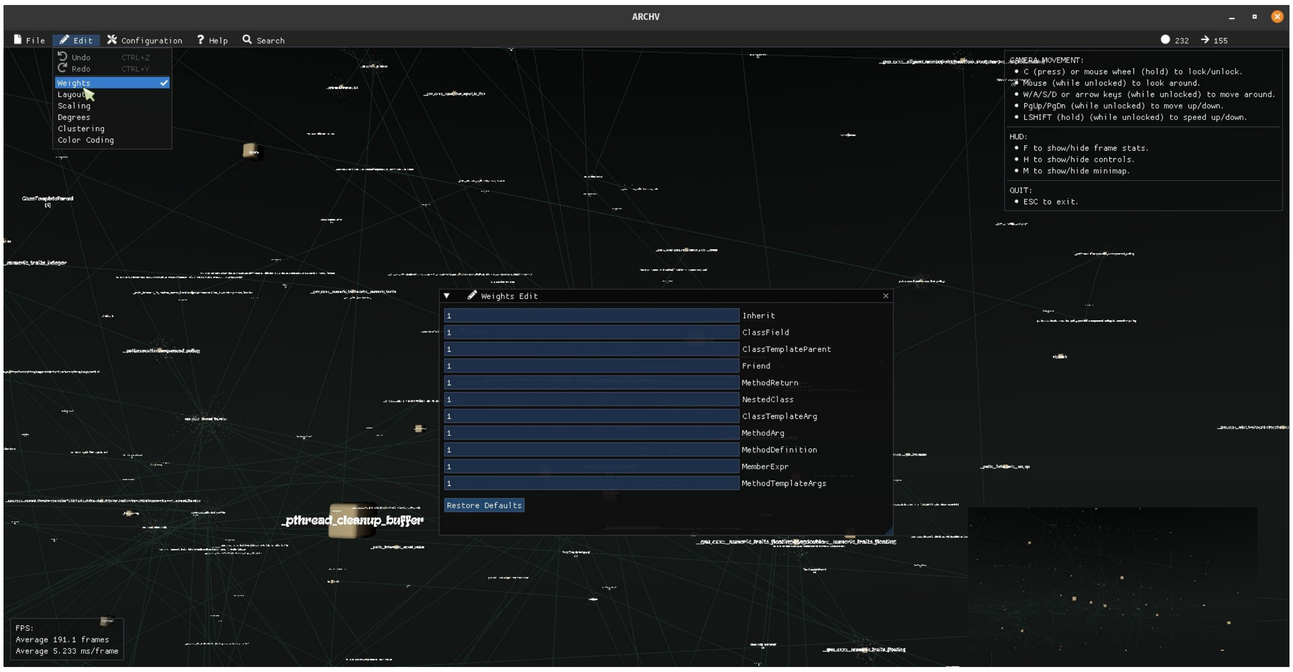


```
FPS:
Average 62.0 frames
Average 16.134 ms/frame
```

7.4. Editing

As mentioned, multiple intrinsic graph properties can be edited in real time, in order to evaluate the overall design of the visualized software system.

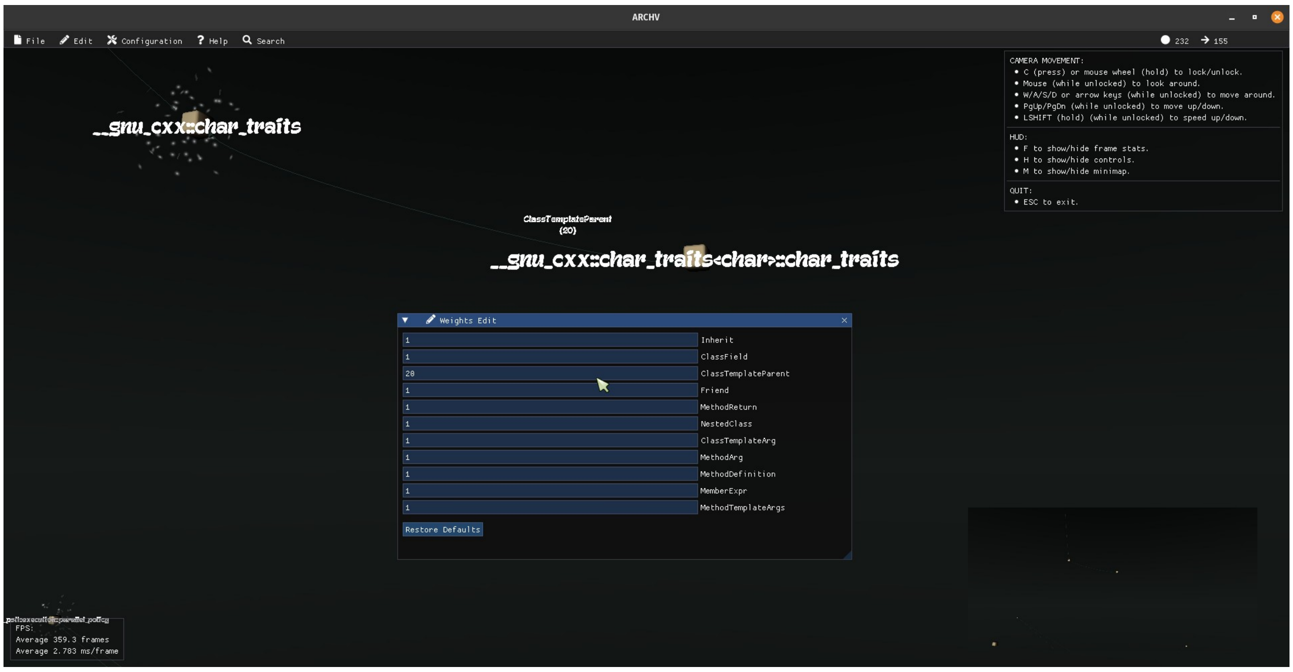
Thus, the edge weights can be changed at any time:



```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

1	Inherit
1	ClassField
1	ClassTemplateParent
1	Friend
1	MethodReturn
1	NestedClass
1	ClassTemplateArg
1	MethodArg
1	MethodDefinition
1	MemberExpr
1	MethodTemplateArgs

```
FPS:
Average 191.1 frames
Average 5.233 ms/frame
```



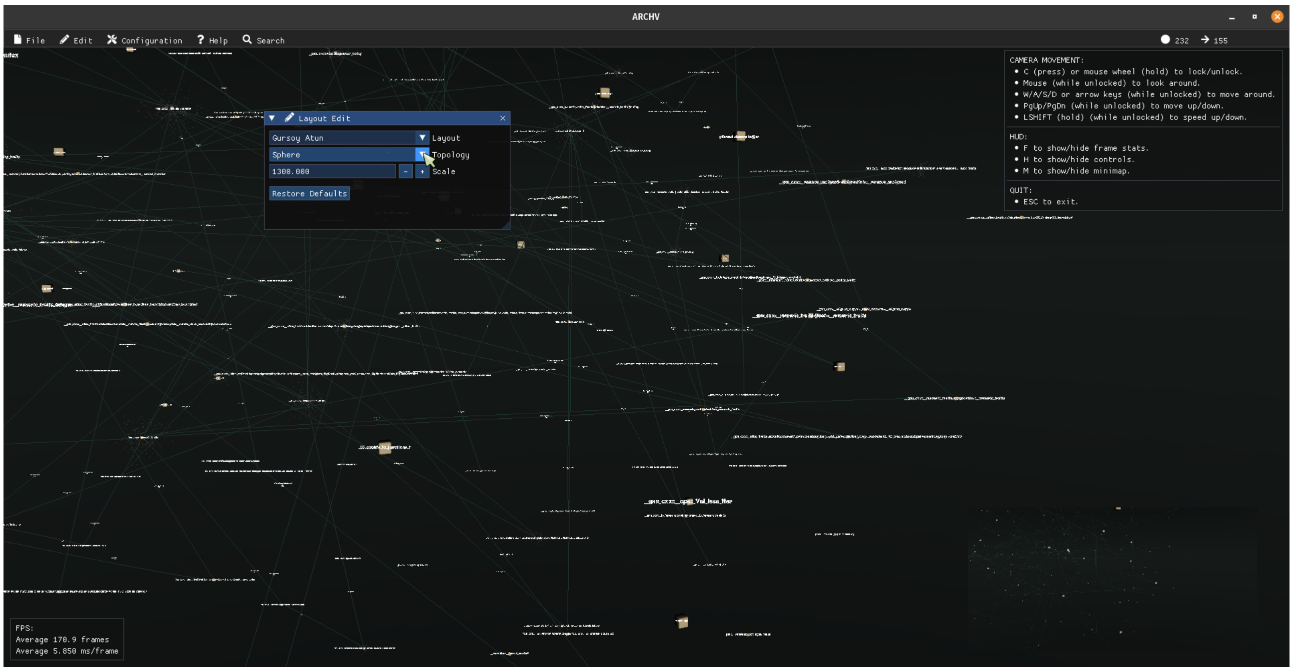
```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

1	Inherit
1	ClassField
20	ClassTemplateParent
1	Friend
1	MethodReturn
1	NestedClass
1	ClassTemplateArg
1	MethodArg
1	MethodDefinition
1	MemberExpr
1	MethodTemplateArgs

```
FPS:
Average 359.3 frames
Average 2.783 ms/frame
```

These affect the output of both clustering and layout algorithms.

The graph layout can be edited:

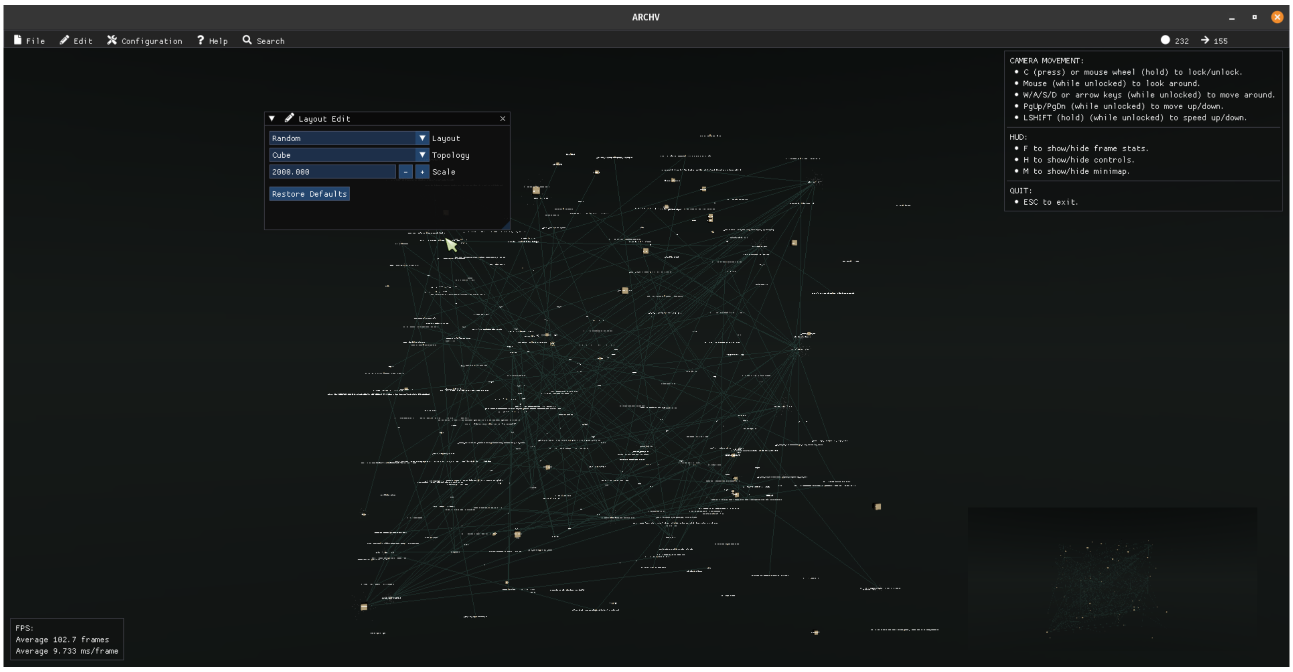


```
Gursoy Atun    Layout
Sphere         Topology
1300.000       Scale
```

```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

```
FPS:
Average 170.9 frames
Average 5.850 ms/frame
```

And be e.g computed randomly, within a cube topology:

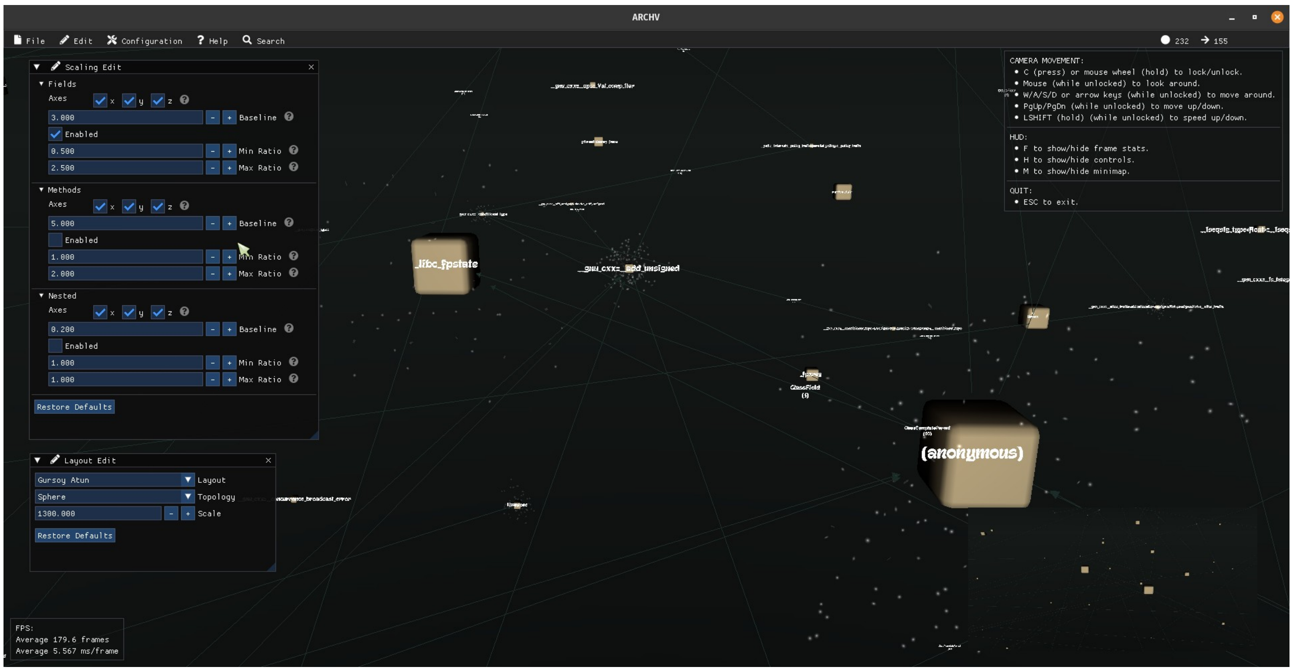


```
Random         Layout
Cube           Topology
2000.000       Scale
```

```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

```
FPS:
Average 102.7 frames
Average 9.733 ms/frame
```

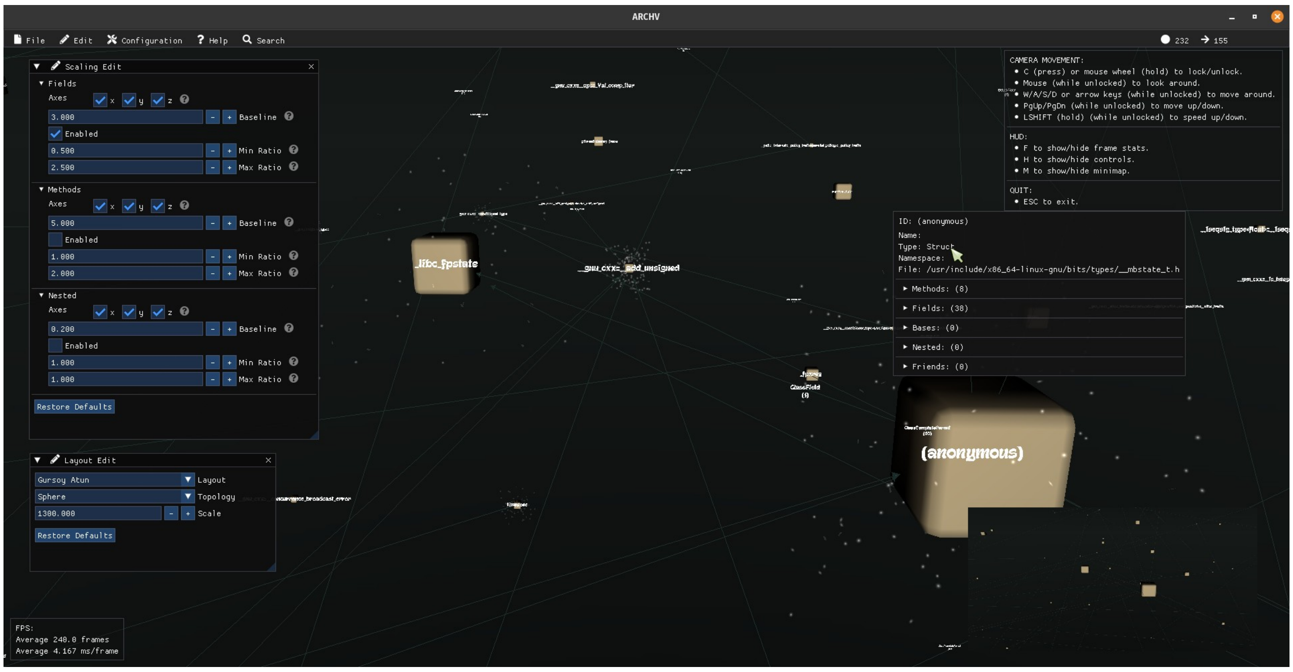
Dynamic scaling to each graph vertex, depending on the vertex underlying class metadata, can be applied:



```
Scaling Edit
 Fields
   Axes      [✓] x [✓] y [✓] z
   3.000     - + Baseline
   [✓] Enabled
   0.500     - + Min Ratio
   2.500     - + Max Ratio
 Methods
   Axes      [✓] x [✓] y [✓] z
   5.000     - + Baseline
   [ ] Enabled
   1.000     - + Min Ratio
   2.000     - + Max Ratio
 Nested
   Axes      [✓] x [✓] y [✓] z
   0.200     - + Baseline
   [ ] Enabled
   1.000     - + Min Ratio
   1.000     - + Max Ratio
 Restore Defaults

Layout Edit
 Gursoy Atun   Layout
 Sphere        Topology
 1300.000  - + Scale
 Restore Defaults

FPS:
Average 179.6 frames
Average 5.567 ms/frame
```

```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

Here, each vertex is scaled in all 3 dimensions, depending on the number of its fields:

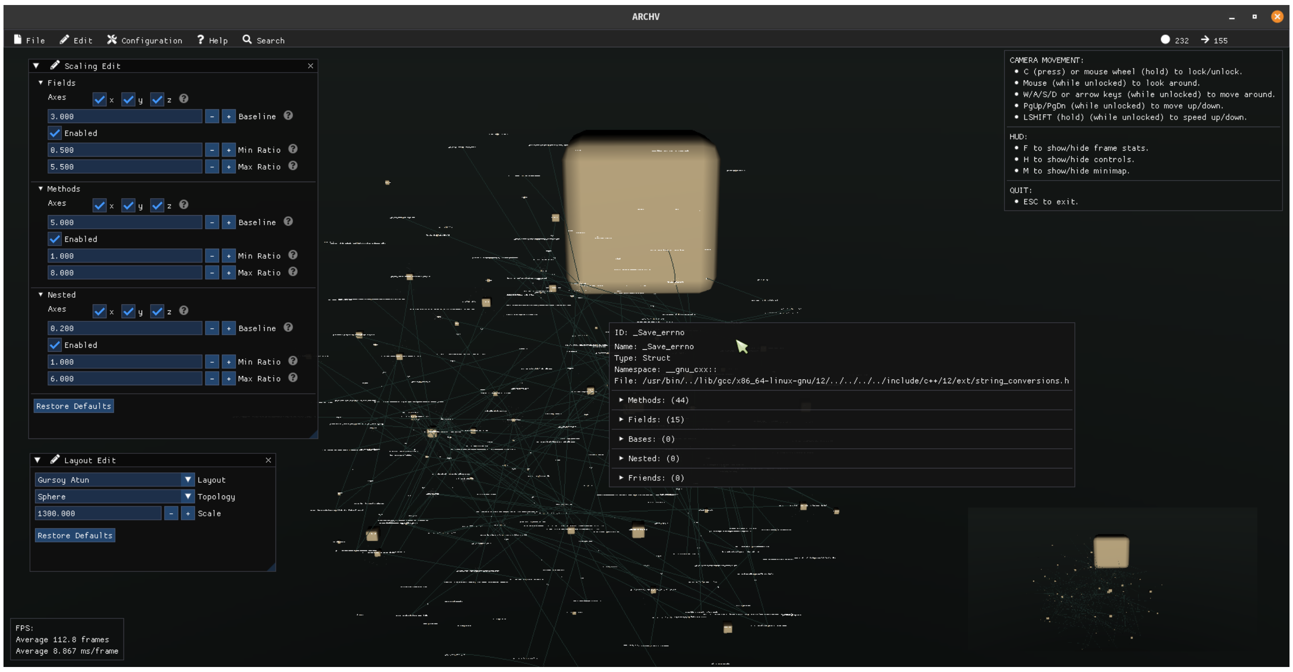


```
Scaling Edit
 Fields
   Axes      [✓] x [✓] y [✓] z
   3.000     - + Baseline
   [✓] Enabled
   0.500     - + Min Ratio
   2.500     - + Max Ratio
 Methods
   Axes      [✓] x [✓] y [✓] z
   5.000     - + Baseline
   [ ] Enabled
   1.000     - + Min Ratio
   2.000     - + Max Ratio
 Nested
   Axes      [✓] x [✓] y [✓] z
   0.200     - + Baseline
   [ ] Enabled
   1.000     - + Min Ratio
   1.000     - + Max Ratio
 Restore Defaults

Layout Edit
 Gursoy Atun   Layout
 Sphere        Topology
 1300.000  - + Scale
 Restore Defaults

FPS:
Average 240.0 frames
Average 4.167 ms/frame
```

```
CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.
```

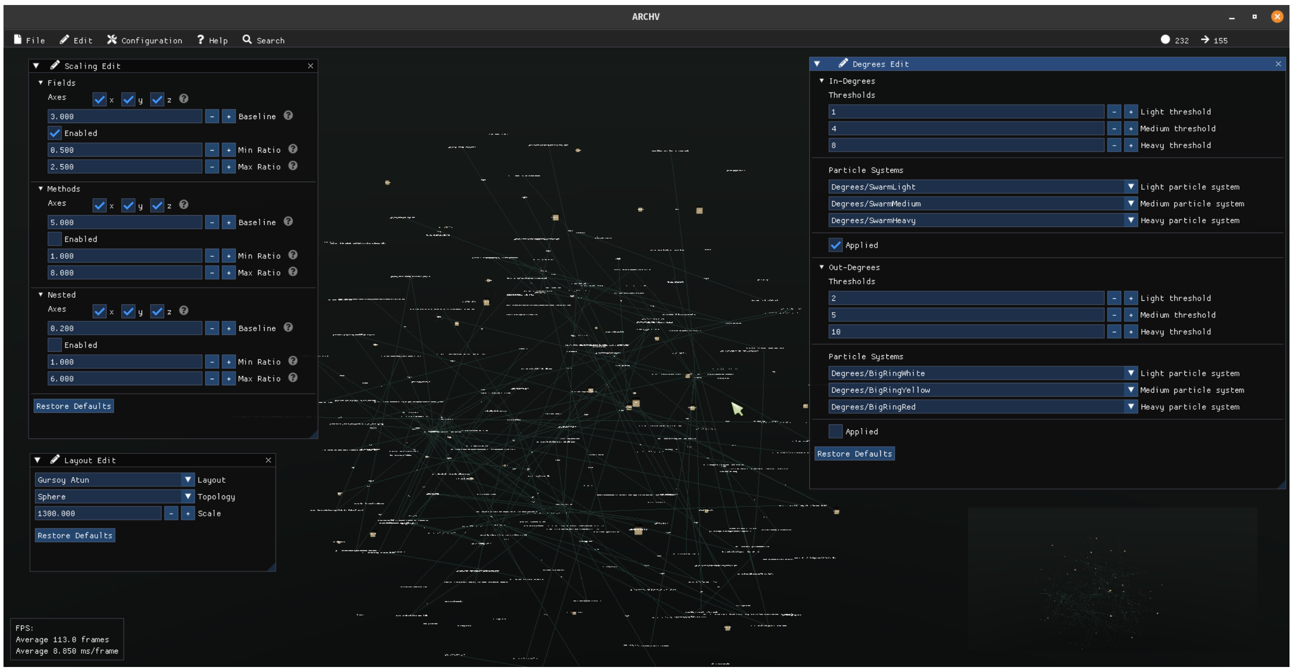
```
ID: (anonymous)
Name:
Type: Struct
Namespace:
File: /usr/include/x86_64-linux-gnu/bits/types/__mbstate_t.h
 ▸ Methods: (8)
 ▸ Fields: (38)
 ▸ Bases: (0)
 ▸ Nested: (0)
 ▸ Friends: (0)
```

Multiple scaling factors can be combined, as shown below:

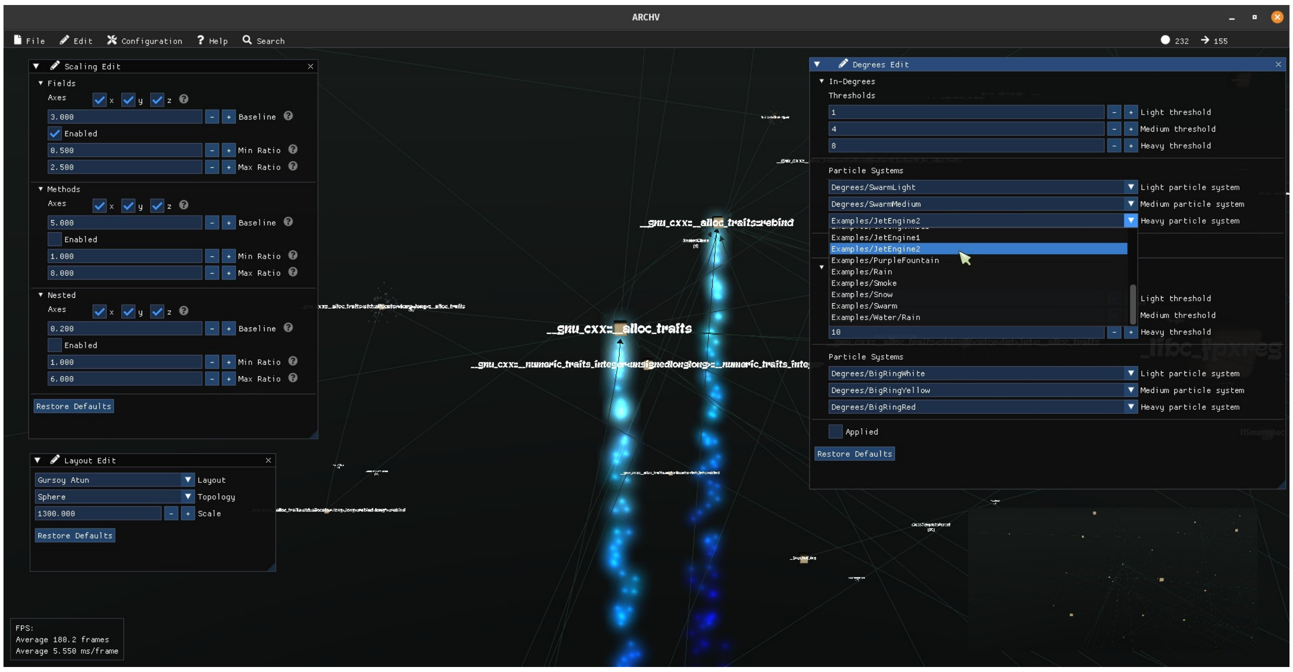


```
Scaling Edit
▼ Fields
  Axes  [✓] x  [✓] y  [✓] z
  3.000   Baseline
  [✓] Enabled
  0.500   Min Ratio
  5.500   Max Ratio
▼ Methods
  Axes  [✓] x  [✓] y  [✓] z
  5.000   Baseline
  [✓] Enabled
  1.000   Min Ratio
  8.000   Max Ratio
▼ Nested
  Axes  [✓] x  [✓] y  [✓] z
  0.200   Baseline
  [✓] Enabled
  1.000   Min Ratio
  6.000   Max Ratio
Restore Defaults

Layout Edit
Gursoy Atun   Layout
Sphere        Topology
1300.000      Scale
Restore Defaults

CAMERA MOVEMENT:
 • C (press) or mouse wheel (hold) to lock/unlock.
 • Mouse (while unlocked) to look around.
 • W/A/S/D or arrow keys (while unlocked) to move around.
 • PgUp/PgDn (while unlocked) to move up/down.
 • LSHIFT (hold) (while unlocked) to speed up/down.
HUD:
 • F to show/hide frame stats.
 • H to show/hide controls.
 • M to show/hide minimap.
QUIT:
 • ESC to exit.

ID: _Save_errno
Name: _Save_errno
Type: Struct
Namespace: __gnu_cxx::
File: /usr/bin/../lib/gcc/x86_64-linux-gnu/12/../../../../include/c++/12/ext/string_conversions.h
▸ Methods: (44)
▸ Fields: (15)
▸ Bases: (0)
▸ Nested: (0)
▸ Friends: (0)

FPS:
Average 112.8 frames
Average 8.867 ms/frame
```

The particle system effects, depending on the number of in/out edges of each vertex, are setup as shown:



```
Scaling Edit
▼ Fields
  Axes  [✓] x  [✓] y  [✓] z
  3.000   Baseline
  [✓] Enabled
  0.500   Min Ratio
  2.500   Max Ratio
▼ Methods
  Axes  [✓] x  [✓] y  [✓] z
  5.000   Baseline
  [ ] Enabled
  1.000   Min Ratio
  8.000   Max Ratio
▼ Nested
  Axes  [✓] x  [✓] y  [✓] z
  0.200   Baseline
  [ ] Enabled
  1.000   Min Ratio
  6.000   Max Ratio
Restore Defaults

Layout Edit
Gursoy Atun   Layout
Sphere        Topology
1300.000      Scale
Restore Defaults

Degrees Edit
▼ In-Degrees
  Thresholds
  1    Light threshold
  4    Medium threshold
  8    Heavy threshold
  Particle Systems
  Degrees/SwarmLight    Light particle system
  Degrees/SwarmMedium   Medium particle system
  Degrees/SwarmHeavy    Heavy particle system
  [✓] Applied
▼ Out-Degrees
  Thresholds
  2    Light threshold
  5    Medium threshold
  10   Heavy threshold
  Particle Systems
  Degrees/BigRingWhite    Light particle system
  Degrees/BigRingYellow   Medium particle system
  Degrees/BigRingRed      Heavy particle system
  [ ] Applied
Restore Defaults

FPS:
Average 113.0 frames
Average 8.850 ms/frame
```



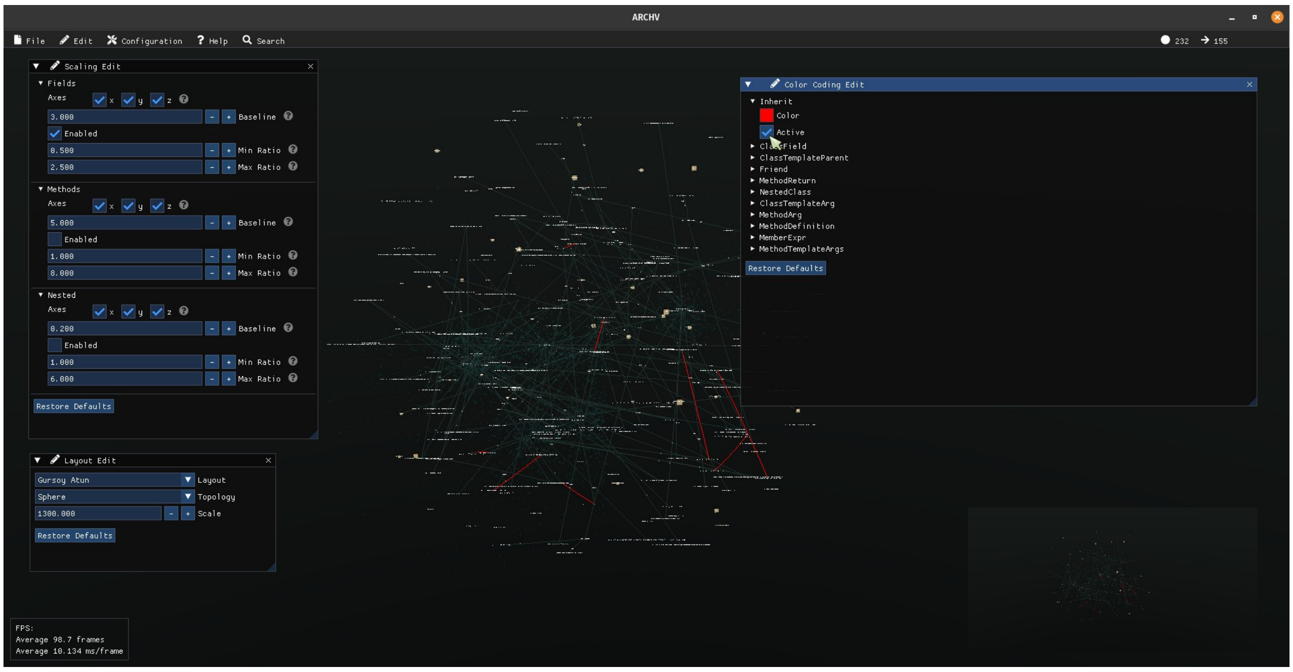
Scaling Edit
- Fields
  - Axes: ✓ x ✓ y ✓ z
  - 3.000 Baseline
  - ✓ Enabled
  - 0.500 Min Ratio
  - 2.500 Max Ratio
- Methods
  - Axes: ✓ x ✓ y ✓ z
  - 5.000 Baseline
  - Enabled
  - 1.000 Min Ratio
  - 8.000 Max Ratio
- Nested
  - Axes: ✓ x ✓ y ✓ z
  - 0.200 Baseline
  - Enabled
  - 1.000 Min Ratio
  - 6.000 Max Ratio
- Restore Defaults

Layout Edit
- Gursoy Atun: Layout
- Sphere: Topology
- 1300.000: Scale
- Restore Defaults

```
FPS:
Average 180.2 frames
Average 5.550 ms/frame
```

Degrees Edit
- In-Degrees
  - Thresholds
    - 1: Light threshold
    - 4: Medium threshold
    - 8: Heavy threshold
  - Particle Systems
    - Degrees/SwarmLight: Light particle system
    - Degrees/SwarmMedium: Medium particle system
    - Examples/JetEngine2: Heavy particle system
      - Examples/JetEngine1
      - Examples/JetEngine2
      - Examples/PurpleFountain
      - Examples/Rain
      - Examples/Smoke
      - Examples/Snow
      - Examples/Swarm
      - Examples/Water/Rain
- Light threshold
- Medium threshold
- 10: Heavy threshold
- Particle Systems
  - Degrees/BigRingWhite: Light particle system
  - Degrees/BigRingYellow: Medium particle system
  - Degrees/BigRingRed: Heavy particle system
- Applied
- Restore Defaults

Each type of dependency can be color coded, (e.g red can be assigned to inheritance and yellow to nested classes):

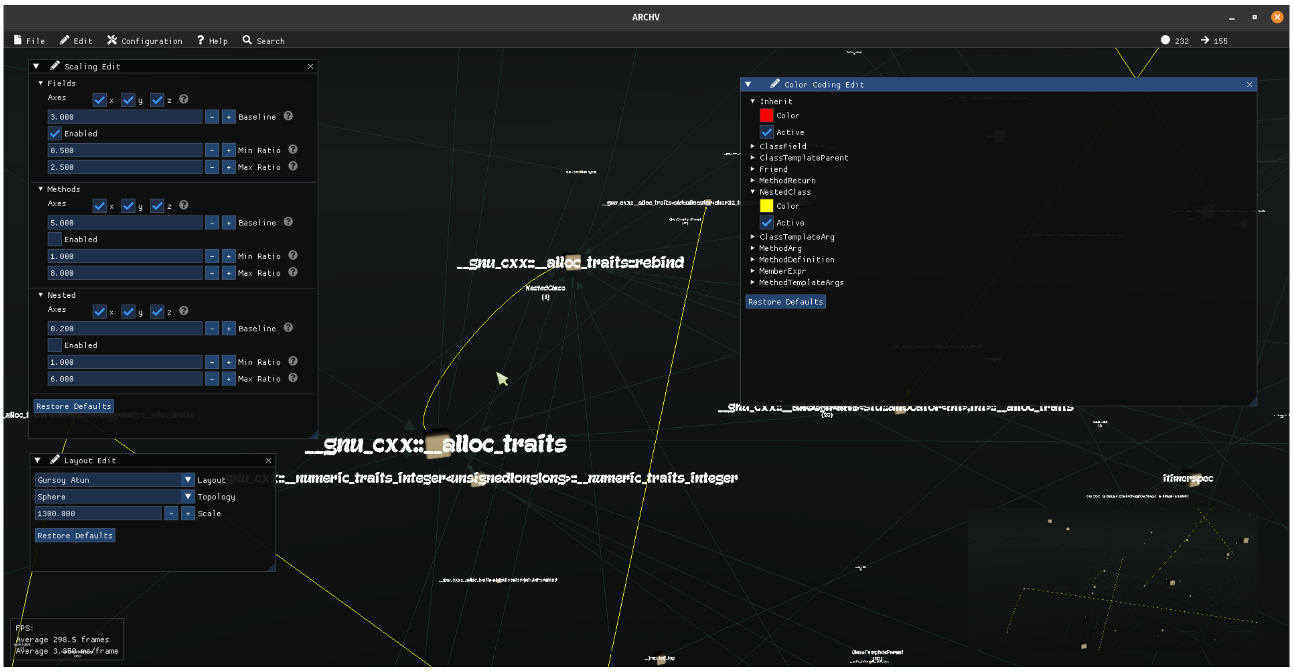


Scaling Edit
- Fields
  - Axes: ✓ x ✓ y ✓ z
  - 3.000 Baseline
  - ✓ Enabled
  - 0.500 Min Ratio
  - 2.500 Max Ratio
- Methods
  - Axes: ✓ x ✓ y ✓ z
  - 5.000 Baseline
  - Enabled
  - 1.000 Min Ratio
  - 8.000 Max Ratio
- Nested
  - Axes: ✓ x ✓ y ✓ z
  - 0.200 Baseline
  - Enabled
  - 1.000 Min Ratio
  - 6.000 Max Ratio
- Restore Defaults

Layout Edit
- Gursoy Atun: Layout
- Sphere: Topology
- 1300.000: Scale
- Restore Defaults

```
FPS:
Average 98.7 frames
Average 10.134 ms/frame
```

Color Coding Edit
- Inherit
  - 🟥 Color
  - ✓ Active
- ClassField
- ClassTemplateParent
- Friend
- MethodReturn
- NestedClass
- ClassTemplateArg
- MethodArg
- MethodDefinition
- MemberExpr
- MethodTemplateArgs
- Restore Defaults


ARCHV
File
Edit
Configuration
Help
Search
232
155
__gnu_cxx::__alloc_traits::rebind
__gnu_cxx::__alloc_traits
__gnu_cxx::__numeric_traits_integer<unsignedlonglong>::__numeric_traits_integer
itimerspec

Scaling Edit

Fields
Axes ✓ x ✓ y ✓ z
3.000 Baseline
✓ Enabled
0.500 Min Ratio
2.500 Max Ratio

Methods
Axes ✓ x ✓ y ✓ z
5.000 Baseline
Enabled
1.000 Min Ratio
8.000 Max Ratio

Nested
Axes ✓ x ✓ y ✓ z
0.200 Baseline
Enabled
1.000 Min Ratio
6.000 Max Ratio

Restore Defaults

Layout Edit

Gursoy Atun Layout
Sphere Topology
1300.000 Scale

Restore Defaults

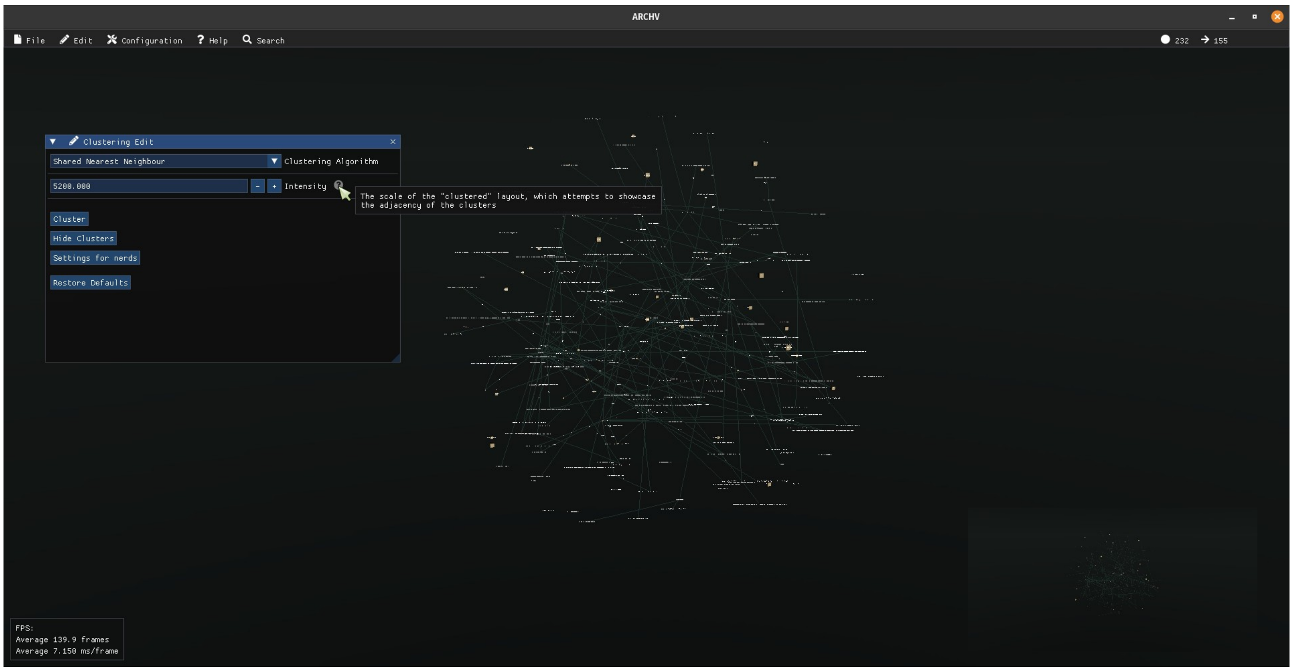
Color Coding Edit

- Inherit
  - 🟥 Color
  - ✓ Active
- ClassField
- ClassTemplateParent
- Friend
- MethodReturn
- NestedClass
  - 🟨 Color
  - ✓ Active
- ClassTemplateArg
- MethodArg
- MethodDefinition
- MemberExpr
- MethodTemplateArgs

Restore Defaults

FPS:
Average 298.5 frames
Average 3.350 ms/frame

Finally, the architecture graph can be clustered via a plethora of clustering algorithms:



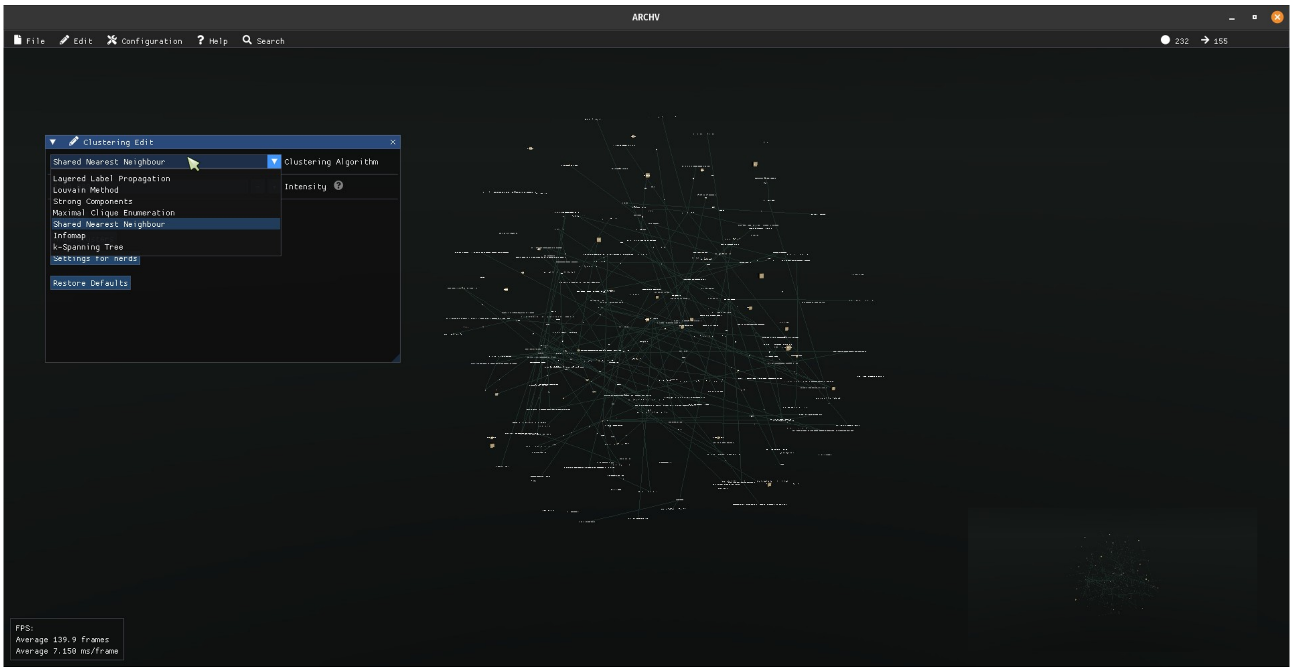
Clustering Edit

Shared Nearest Neighbour Clustering Algorithm
5200.000 Intensity

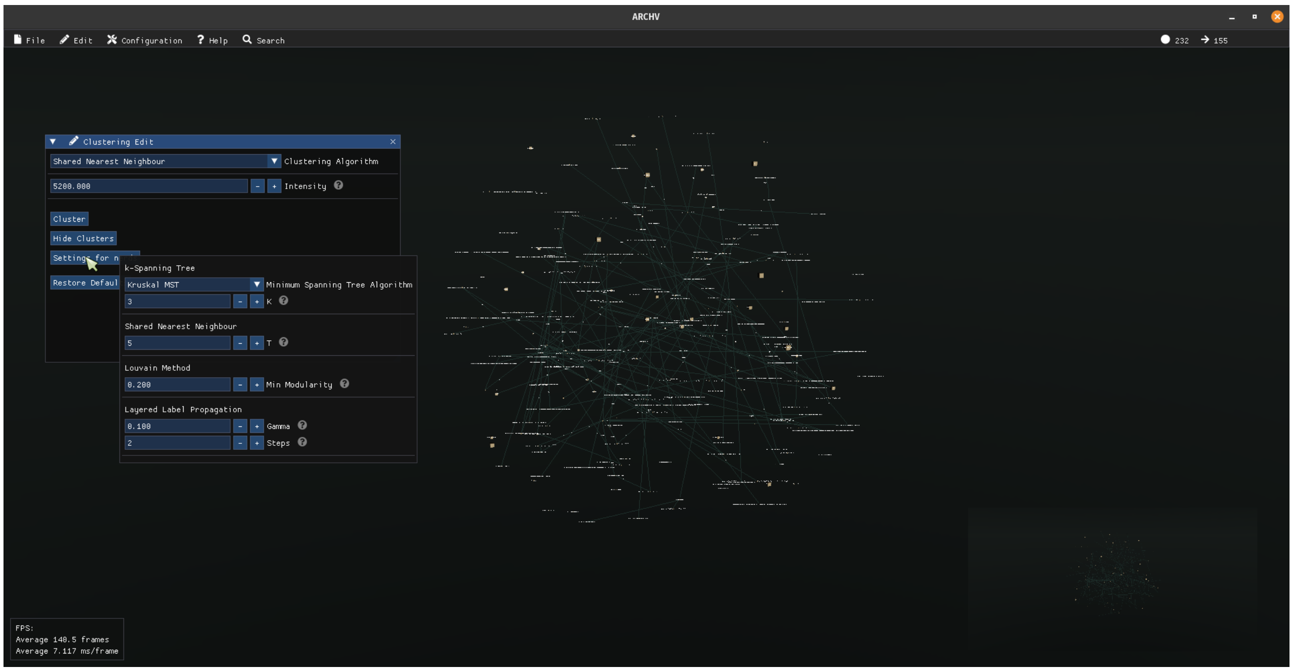
The scale of the "clustered" layout, which attempts to showcase the adjacency of the clusters

Cluster
Hide Clusters
Settings for nerds
Restore Defaults

FPS:
Average 139.9 frames
Average 7.150 ms/frame

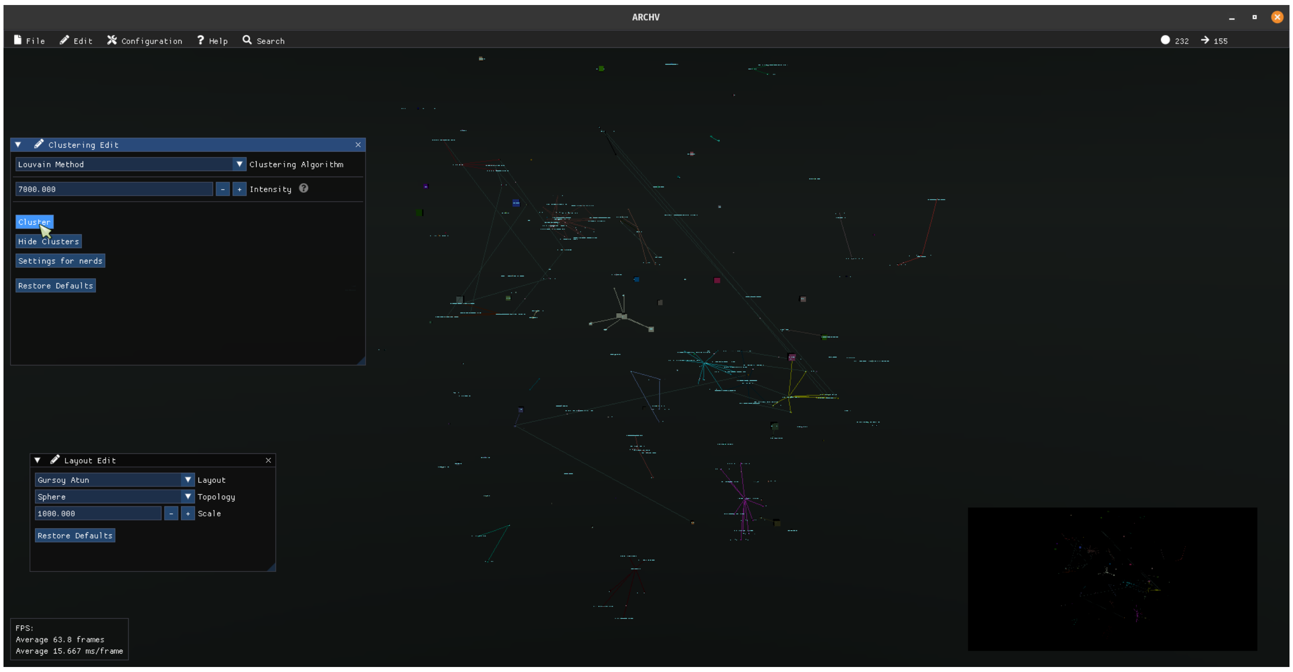


A number of advanced settings for some clustering algorithms can be configured:



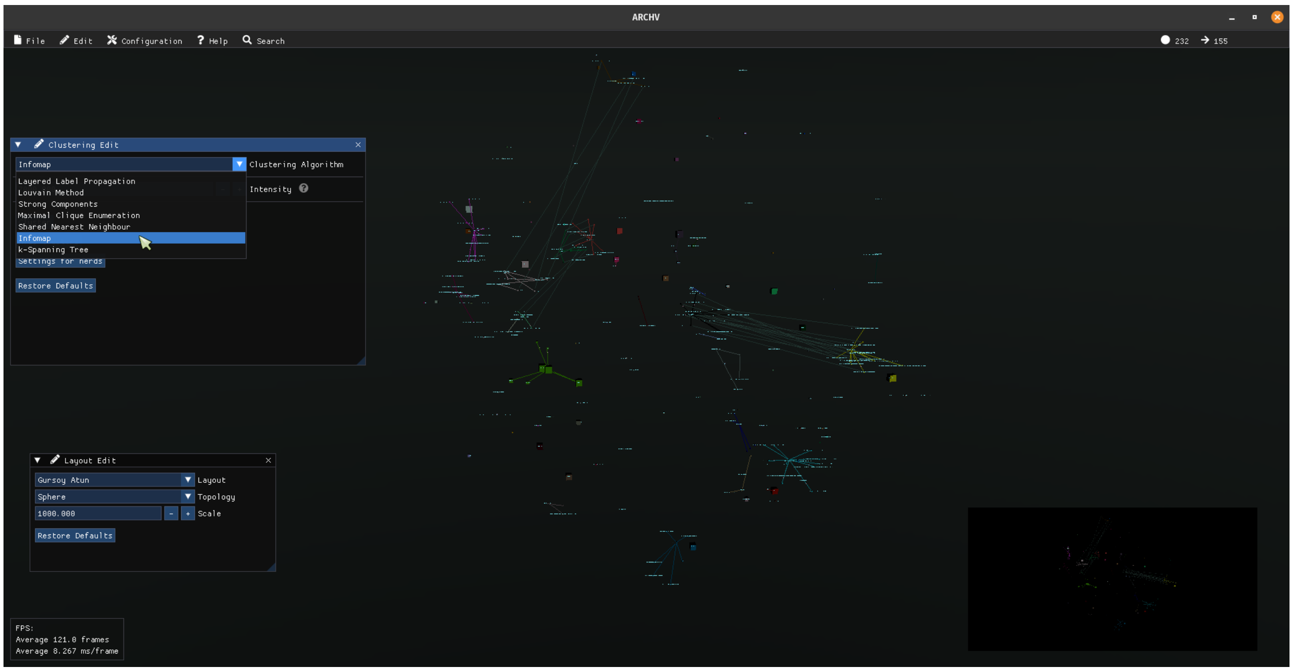
```
Shared Nearest Neighbour        Clustering Algorithm
5200.000                        Intensity

k-Spanning Tree
Kruskal MST                     Minimum Spanning Tree Algorithm
3                               K

Shared Nearest Neighbour
5                               T

Louvain Method
0.200                           Min Modularity

Layered Label Propagation
0.100                           Gamma
2                               Steps
```

Once clustered, given the Louvain Method and Infomap clustering algorithms respectively, the architecture graph may be rendered as follows:



```
Louvain Method    Clustering Algorithm
7000.000          Intensity
Gursoy Atun       Layout
Sphere            Topology
1000.000          Scale
```

```
FPS:
Average 63.8 frames
Average 15.667 ms/frame
```



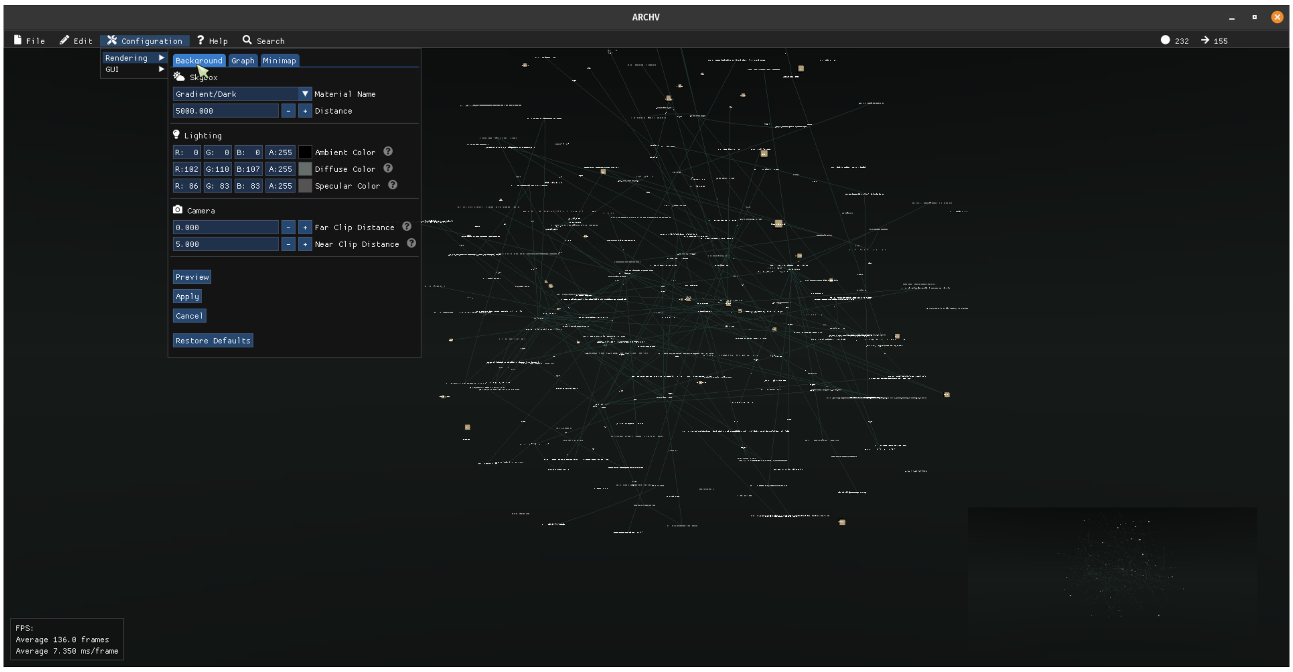
```
Infomap           Clustering Algorithm
                  Intensity
Gursoy Atun       Layout
Sphere            Topology
1000.000          Scale
```

```
FPS:
Average 121.0 frames
Average 8.267 ms/frame
```

7.5. Configuration

As mentioned, this subsection demonstrates how various rendering options can be set.

The background, some extrinsic properties of the graph itself, the mini-map view and the GUI can be configured:

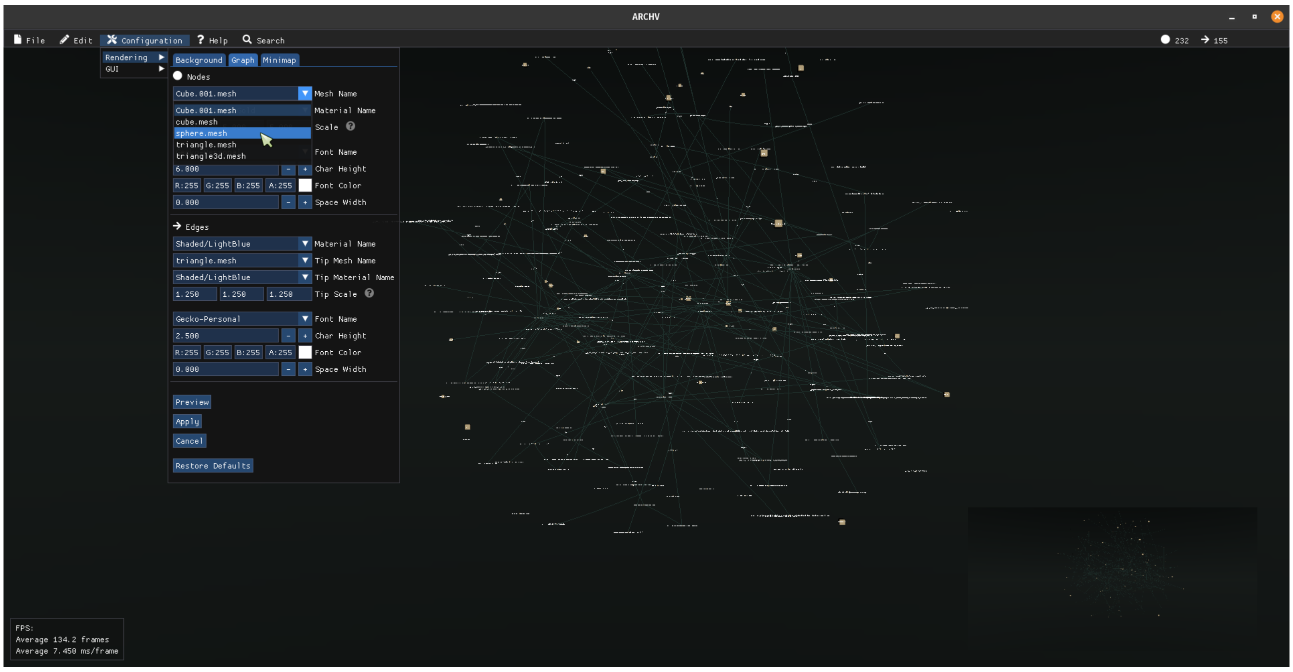


Skybox
Gradient/Dark Material Name
5000.000 Distance

Lighting
R: 0 G: 0 B: 0 A:255 Ambient Color
R:102 G:110 B:107 A:255 Diffuse Color
R: 86 G: 83 B: 83 A:255 Specular Color

Camera
0.000 Far Clip Distance
5.000 Near Clip Distance

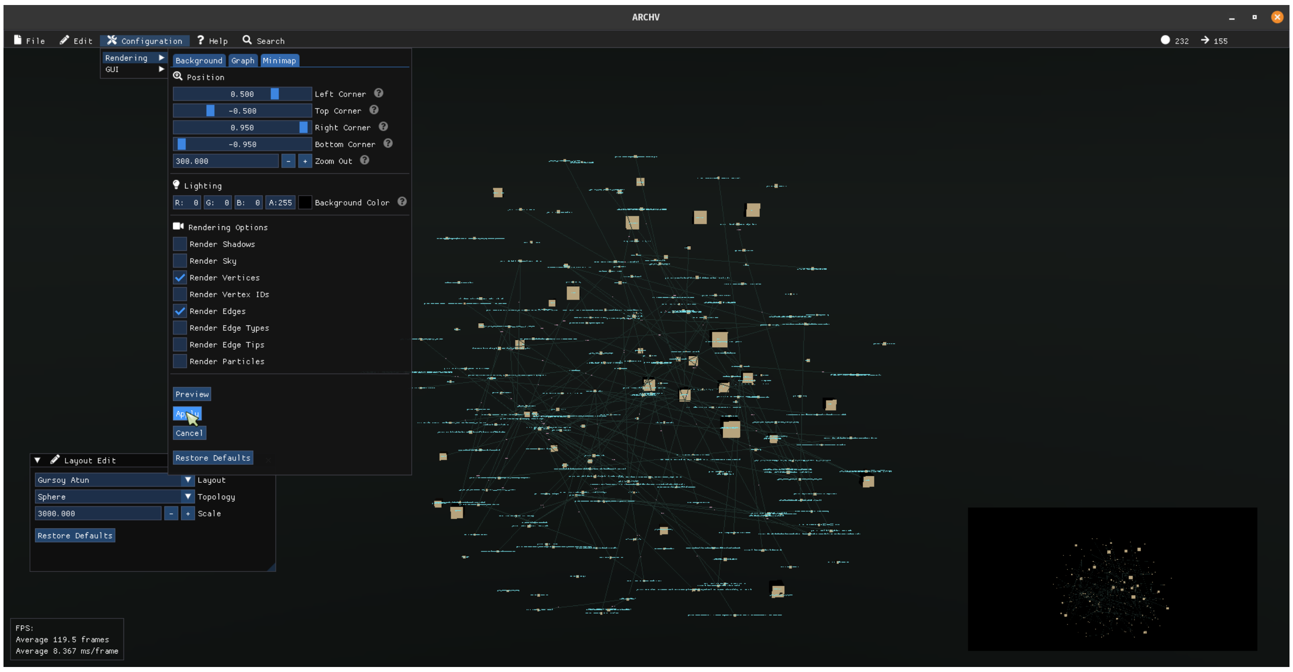
```
FPS:
Average 136.0 frames
Average 7.350 ms/frame
```



Nodes
Cube.001.mesh Mesh Name
Cube.001.mesh
cube.mesh
sphere.mesh
triangle.mesh
triangle3d.mesh
Material Name
Scale
Font Name
6.000 Char Height
R:255 G:255 B:255 A:255 Font Color
0.000 Space Width

Edges
Shaded/LightBlue Material Name
triangle.mesh Tip Mesh Name
Shaded/LightBlue Tip Material Name
1.250 1.250 1.250 Tip Scale
Gecko-Personal Font Name
2.500 Char Height
R:255 G:255 B:255 A:255 Font Color
0.000 Space Width

```
FPS:
Average 134.2 frames
Average 7.450 ms/frame
```



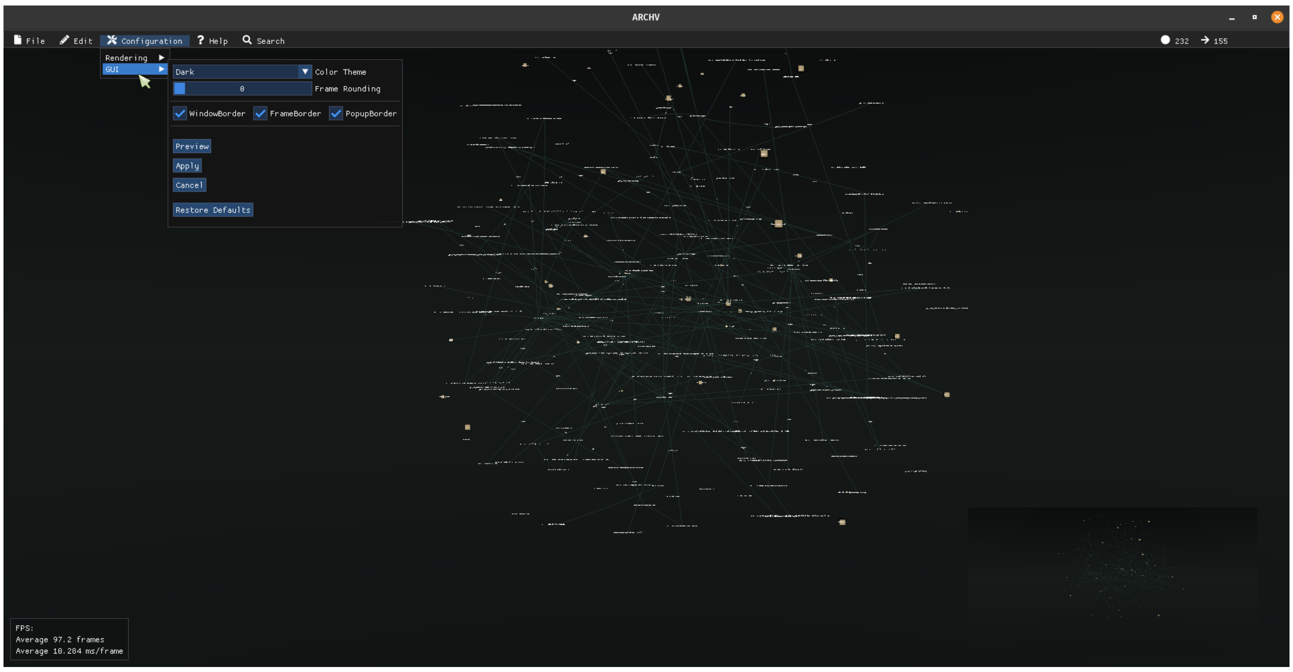
Position
0.500 Left Corner
-0.500 Top Corner
0.950 Right Corner
-0.950 Bottom Corner
300.000 Zoom Out

Lighting
R: 0 G: 0 B: 0 A:255 Background Color

Rendering Options
[ ] Render Shadows
[ ] Render Sky
[✓] Render Vertices
[ ] Render Vertex IDs
[✓] Render Edges
[ ] Render Edge Types
[ ] Render Edge Tips
[ ] Render Particles

Layout Edit
Gursoy Atun Layout
Sphere Topology
3000.000 Scale
Restore Defaults

```
FPS:
Average 119.5 frames
Average 8.367 ms/frame
```

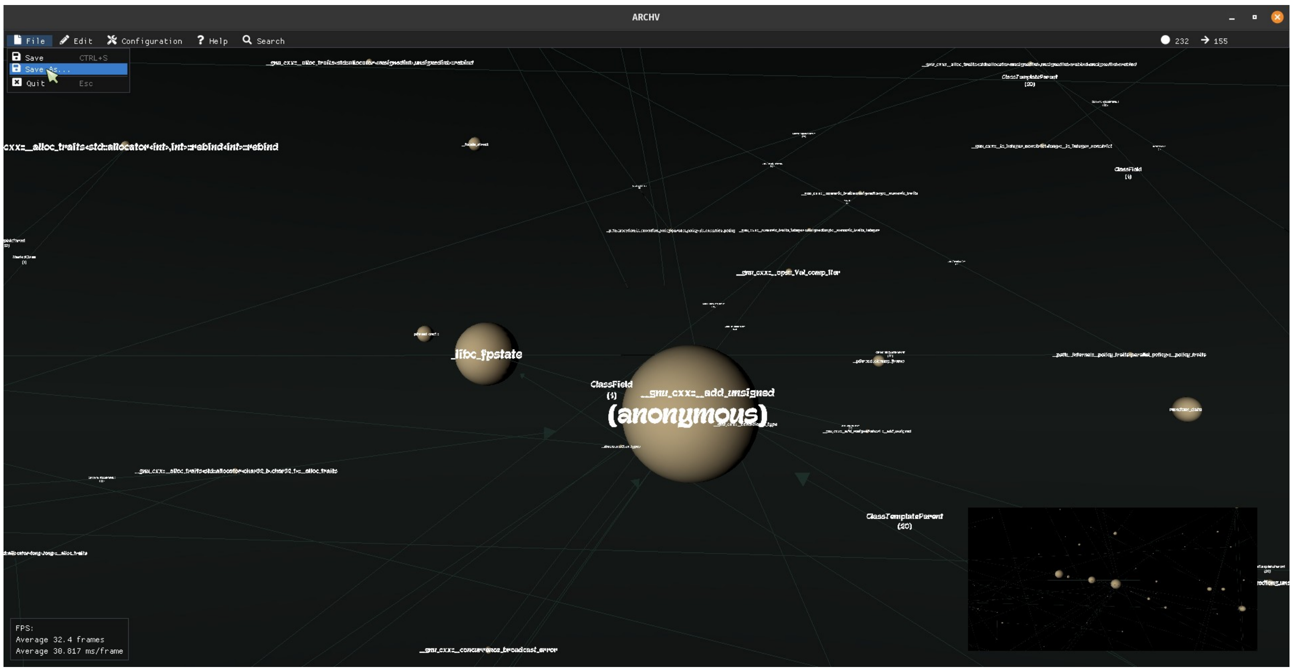


Dark Color Theme
0 Frame Rounding
[✓] WindowBorder [✓] FrameBorder [✓] PopupBorder

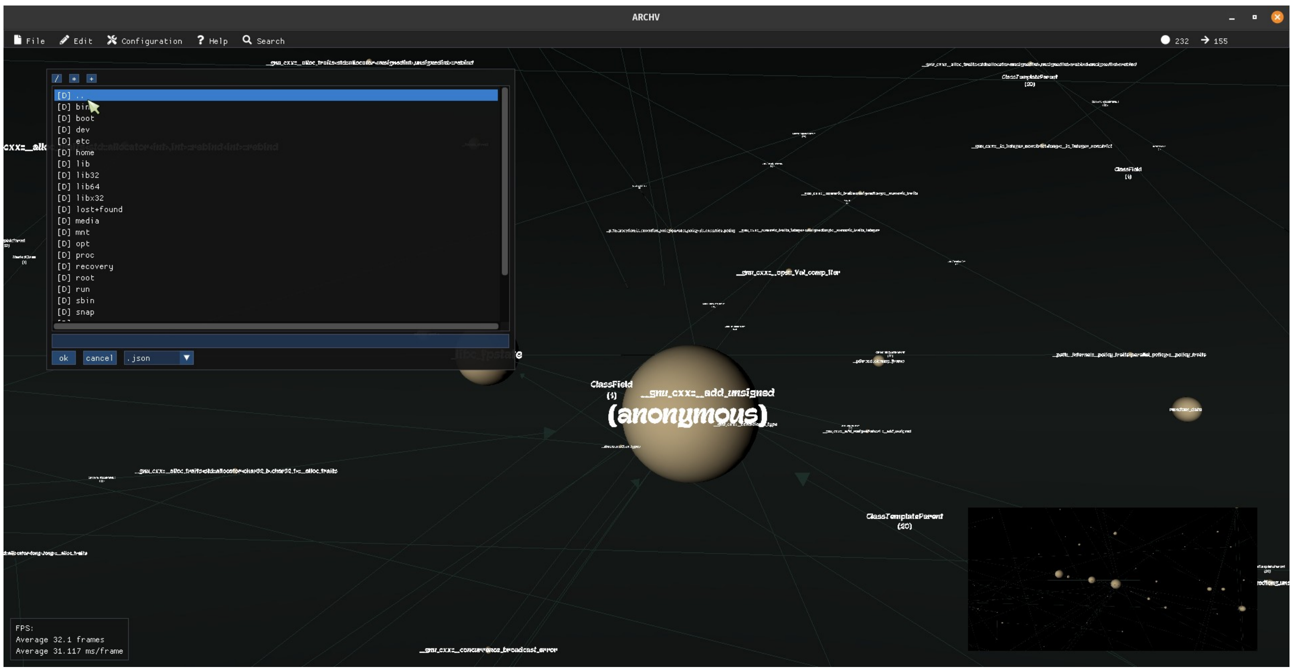
```
FPS:
Average 97.2 frames
Average 10.284 ms/frame
```

7.6. Save / Load

Finally, the visualization session's configurations can be saved and loaded at a later time:



```
FPS:
Average 32.4 frames
Average 30.817 ms/frame
```



```
FPS:
Average 32.1 frames
Average 31.117 ms/frame
```

As mentioned, the configurations can be loaded when launching the application with the saved configuration file:

```
$ ./archv_app foo/bar/buzz/graph.json zoo/buzz/myconfig.json
```