CASC

1.0.5

Generated by Doxygen 1.9.1

Chapter 1

Colored Abstract Simplicial Complex (CASC) Library

Master CI: Development CI:

CASC is a modern and header-only C++ library which provides a data structure to represent arbitrary dimension abstract simplicial complexes with user-defined classes stored directly on the simplices at each dimension. This is achieved by taking advantage of the combinatorial nature of simplicial complexes and new C++ code features such as: variadic templates and automatic function return type deduction. Essentially CASC stores the full topology of the complex according to a `Hasse diagram`. The representation of the topology is decoupled from interactions of user data through the use of metatemplate programming.

1.1 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

1.1.1 Prerequisites

CASC does not have any dependencies other than the C++ standard library. If you wish to use CASC, you can use the header files right away. There is no binary library to link to, and no configured header file. CASC is a pure template library defined in the headers.

We use the CMake build system (version 3+), but only to build the documentation and unit-tests, and to automate installation.

Doxygen and Graphviz is used to generate the documentation.

To use CASC in your software all you will need is a working C++ compiler with full C++14 support. This includes:

- GCC Versions 5+
- Clang Versions 3, 5+[†]
- XCode 8+[†]

[†] Note that there is a known issue with Clang 4.x.x versioned compilers (including XCode version 9.[0-2]), where the most specialized unique specialization is not selected leading to a compiler error. The current workaround to this problem is to either use GCC or to obtain Clang version 5+ (XCode version 9.3beta+).

1.1.2 Installing

CASC is header only meaning that there is nothing to compile out of the box. To use CASC, simply copy the desired headers into your project and included as necessary. If you wish to install CASC using CMake to your system, even though the library is header only, you must first create a new folder to prevent in-source "builds".

```
mkdir build
cd build
```

Subsequently run CMake specifying the installation prefix and the path to the root level `CMakeLists.txt` file.

```
cmake -DCMAKE_INSTALL_PREFIX=/usr/local/ ..
make install
```

Unit tests are also packaged along with CASC and are dependent upon `Googles C++ test framework`. If you wish to build and run the tests, set the flag `-DBUILD_CASCTESTS=on` in your CMake command. CMake will then download and build `googletest` and link it with the CASC unit tests.

```
cmake -DBUILD_CASCTESTS=on ..
make
make tests          # Run tests through make
./bin/casctests     # Alternatively run the tests directly (more verbose)
```

Additional examples provided with CASC can be built in a similar fashion by passing the `-DBUILD_CASCEXAMPLES=on` flag to CMake.

1.1.3 Documentation

A current version of the documentation is available online via `github pages`. You can also build the documentation locally if you have Doxygen and Graphviz on your system. CMake will automatically try to find a working Doxygen installation. If Doxygen is found then the documentation can be built using `make casc_doc`. Otherwise CMake will report that it could not find Doxygen.

1.2 Versioning & Contributing

We use `Github` for versioning. For the versions available, please see the `releases`. If you find a bug or with to request additional functionality please file an issue in the `CASC Github project`.

1.3 Authors

John Moody
Department of Mathematics
University of California, San Diego

Christopher Lee
Department of Chemistry & Biochemistry
University of California, San Diego

See also the list of `contributors` who participated in this project.

1.4 License

1.5 Acknowledgments

This project is supported by the National Institutes of Health under grant numbers P41-GM103426 (NBCR), T32-GM008326, and R01-GM31749. It is also supported in part by the National Science Foundation under awards DMS-CM1620366 and DMS-FRG1262982.

Chapter 2

CASC License

The CASC library is free software distributed under the GNU Lesser General Public License Version 2.1. You can redistribute it and/or modify it under the terms of the LGPLv2.1 as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. A copy of the GNU LGPLv2.1 is reproduced below.

2.0.1 GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

```
Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]
```

2.0.2 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

2.0.3 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- **a)** The modified work must itself be a software library.
- **b)** You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- **c)** You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- **d)** If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

  (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- **a)** Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- **b)** Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
- **c)** Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
- **d)** If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- **e)** Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- **a)** Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- **b)** Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and

conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

2.0.4 END OF TERMS AND CONDITIONS

2.0.5 How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the library's name and an idea of what it does.
Copyright (C) year  name of author

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in
the library 'Frob' (a library for tweaking knobs) written
by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

Chapter 3

Building the documentation

The documentation for CASC can be generated locally using `Doxygen`. You must have a working copy of doxygen installed on your machine in order to build the documentation.

If CMake is able to find your doxygen installation then the following sequence of commands will build the basic documentation.

```
cmake ..
make casc_doc
```

3.0.1 Documentation for Developers

If you are contributing to or modifying the CASC library you may wish to document private class members or currently hidden metatemplate helper functions. Whether or not documentation for these items is generated can be controlled by modifying the default doxygen configuration: `doc/Doxyfile.in`.

To document private class functions and members toggle: `EXTRACT_PRIVATE = YES`

To enable metatemplate helper functions enable the conditional: `ENABLED_SECTIONS = detail`

Chapter 4

Frequently Asked Questions

1. Why is my simplex data not storing correctly?

If you are retrieving the data from the `SimplexID` using the dereference operator, you must retrieve the result as a reference in order to modify it. See the following example.

```
MeshType mesh = MeshType();
int key = mesh.insert({1}, 10);
auto data = *mesh.get_simplex_up({key});
data = 5;
std::cout «*mesh.get_simplex_up({key}); « std::endl // Prints 10
auto &dataRef = *mesh.get_simplex_up({key});
dataRef = 5;
std::cout « *mesh.get_simplex_up({key}) « std::endl // Prints 5
```

Chapter 5

Namespace Index

5.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 8

Namespace Documentation

8.1 casc Namespace Reference

Namespace for everything CASC.

Data Structures

- struct Orientable

  Class representing the orientation.
- struct SimplexMap

  A multimap to represent a map of simplex indices to a set of simplices.
- struct SimplexSet

  A multiset to store simplices in a simplicial_complex.
- class simplicial_complex

  The CASC data structure for representing simplicial complexes of arbitrary dimensionality with coloring.

Typedefs

- template<typename KeyType , typename ... Ts>
  using AbstractSimplicialComplex = simplicial_complex< detail::simplicial_complex_traits_default< KeyType, Ts... > >
- template<typename T >
  using NodeSet = std::unordered_set< T, simplex_set_detail::hashSimplexID< T > >

  Helpful alias defining a unordered_set of simplices. See also hashSimplexID.

Functions

- template<typename Complex >
  void getStar (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the star of a SimplexSet.
- template<typename Complex , typename Simplex >
  void getStar (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Gets the star of a simplex.

- template<typename Complex >
  void getClosure (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the closure of a simplex set.
- template<typename Complex , typename Simplex >
  void getClosure (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Compute the closure of a simplex.
- template<typename Complex >
  void getLink (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the link of a SimplexSet.
- template<typename Complex , typename Simplex >
  void getLink (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Gets the link of a simplex.
- template<typename Complex >
  void writeDOT (const std::string &filename, Complex &F)

  Writes out the topology of an ASC into the dot format.
- template<typename Visitor , typename SimplexID >
  void visit_BFS_up (Visitor &&v, typename SimplexID::complex &F, SimplexID s)

  Traverse BFS up the complex and apply a visitor function to each simplex visited.
- template<typename Visitor , typename SimplexID >
  void visit_BFS_down (Visitor &&v, typename SimplexID::complex &F, SimplexID s)

  Traverse BFS down the complex and apply a visitor function to each simplex visited.
- template<typename Visitor , typename EdgeID >
  void edge_up (Visitor &&v, typename EdgeID::complex &F, EdgeID s)

  Traverse across edges BFS.
- template<class Complex , std::size_t level, class InsertIter >
  void neighbors (Complex &F, typename Complex::template SimplexID< level > nid, InsertIter iter)

  Push the immediate face neighbors into the provided iterator.
- template<class Complex , class SimplexID , class InsertIter >
  void neighbors (Complex &F, SimplexID nid, InsertIter iter)

  This is a helper function to assist neighbors to automatically deduce the integral level.
- template<class Complex , std::size_t level, class InsertIter >
  void neighbors_up (Complex &F, typename Complex::template SimplexID< level > nid, InsertIter iter)

  Push the immediate coface neighbors into the provided iterator.
- template<class Complex , class SimplexID , class InsertIter >
  void neighbors_up (Complex &F, SimplexID nid, InsertIter iter)

  This is a helper function to assist neighbors to automatically deduce the integral level.
- template<class Complex , std::size_t level, typename Iterator >
  void kneighbors_up (Complex &F, int ring, std::set< typename Complex::template SimplexID< level > > &nbors, Iterator begin, Iterator end)

  Code for returning a set of k-ring neighbors.
- template<class Complex , class SimplexID >
  void kneighbors_up (Complex &F, SimplexID nid, int ring, std::set< SimplexID > &nbors)

  Helper function to help kneighbors_up to deduce the integral level of SimplexID.
- template<class Complex , std::size_t level, typename Iterator >
  void kneighbors (Complex &F, int ring, std::set< typename Complex::template SimplexID< level > > &nbors, Iterator begin, Iterator end)

  Code for returning a set of k-ring neighbors.
- template<class Complex , class SimplexID >
  void kneighbors (Complex &F, SimplexID nid, int ring, std::set< SimplexID > &nbors)

  Helper function to help kneighbors to deduce the integral level of SimplexID.
- template<typename Complex >
  void perform_removal (Complex &F, casc::SimplexSet< Complex > &S)

  Remove simplex in SimplexSet S from complex F.

- template<typename Complex >
  void perform_insertion (Complex &F, typename decimation_detail::SimplexDataSet< Complex >::type &S)

  Insert all simplices in SimplexSet S into complex F
- template<typename Complex , template< typename > class Callback>
  void run_user_callback (Complex &F, casc::SimplexMap< Complex > &S, Callback< Complex > &&clbk, typename decimation_detail::SimplexDataSet< Complex >::type &rv)

  Run the user specified callback function.
- template<typename Complex , typename Simplex , template< typename > class Callback>
  void decimate (Complex &F, Simplex s, Callback< Complex > &&clbk)

  Decimate a simplex of any dimension while considering any meta-data stores on decimated simplices.
- template<typename Complex , typename Simplex >
  Complex::KeyType decimateFirstHalf (Complex &F, Simplex s, SimplexMap< Complex > &simplexMap)

  Given a simplex to decimate generate a pre-post mapping.
- template<typename Complex >
  void decimateBackHalf (Complex &F, SimplexMap< Complex > &simplexMap, typename decimation_↩ detail::SimplexDataSet< Complex >::type &rv)

  Given a simplexMap and mapped resulting data execute the decimation.
- template<typename Complex >
  void init_orientation (Complex &F)

  Initialize the partial ordering of the simplex edges.
- template<typename Complex >
  void clear_orientation (Complex &F)

  Clear the orientation of the facets.
- template<typename Complex >
  std::tuple< int, bool, bool > compute_orientation (Complex &F)

  Initializes and calculates the orientation of a simplicial_complex.
- template<typename Complex >
  std::tuple< int, bool, bool > check_orientation (Complex &F)

  Checks for self consistent orientation and fill in missing orientations.
- template<std::size_t k, typename Complex >
  static auto & get (SimplexMap< Complex > &S)

  Get the map for a simplex dimension.
- template<std::size_t k, typename Complex >
  static auto & get (const SimplexMap< Complex > &S)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.
- template<std::size_t k, typename Complex >
  static auto & get (SimplexSet< Complex > &S)

  Get the NodeSet for a simplex dimension from a SimplexSet.
- template<std::size_t k, typename Complex >
  static auto & get (const SimplexSet< Complex > &S)
- template<typename Complex >
  bool operator== (const SimplexSet< Complex > &lhs, const SimplexSet< Complex > &rhs)

  Compare if the sets are equivalent.
- template<typename Complex >
  bool operator!= (const SimplexSet< Complex > &lhs, const SimplexSet< Complex > &rhs)

  Compare if the sets are not equivalent.
- template<typename Complex >
  static void set_union (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

  Compute the set union.
- template<typename Complex >
  static void set_intersection (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

Compute the set intersection.

- template<typename Complex >
  static void set_difference (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

  Compute the set difference.
- template<typename T , std::size_t k>
  std::string to_string (const std::array< T, k > &A)

  Returns a string representation of the vertex subsimplicies of a given simplex.

8.1.1 Typedef Documentation

8.1.1.1 AbstractSimplicialComplex

```
template<typename KeyType , typename ...  Ts>
using casc::AbstractSimplicialComplex = typedef simplicial_complex< detail::simplicial_↩
complex_traits_default<KeyType, Ts...> >
```

Alias to generate a CASC from a list of traits. See also simplicial_complex_traits_default. Example – To create a tetrahedral mesh with integer data on all simplices:

```
auto mesh = AbstractSimplicialComplex<
    int, // KEYTYPE
    int, // Root data
    int, // Vertex data
    int, // Edge data
    int, // Face data
    int  // Volume data
>();
```

8.1.2 Function Documentation

8.1.2.1 check_orientation()

```
template<typename Complex >
std::tuple<int, bool, bool> casc::check_orientation (
            Complex & F )
```

Parameters

F	Simplicial_complex

Template Parameters

Complex	Typename of the simplicial_complex.

Returns

A tuple of the number of connected components, where the complex is orientable, and if it is psuedo manifold.

8.1.2.2 clear_orientation()

```
template<typename Complex >
void casc::clear_orientation (
            Complex & F )
```

Parameters

F	Simplicial complex of interest

Template Parameters

Complex	Typename of the simplicial complex

8.1.2.3 compute_orientation()

```
template<typename Complex >
std::tuple<int, bool, bool> casc::compute_orientation (
            Complex & F )
```

Parameters

F	Simplicial_complex

Template Parameters

Complex	Typename of the simplicial_complex.

Returns

A tuple of the number of connected components, where the complex is orientable, and if it is psuedo manifold.

8.1.2.4 decimate()

```
template<typename Complex , typename Simplex , template< typename > class Callback>
void casc::decimate (
            Complex & F,
            Simplex s,
            Callback< Complex > && clbk )
```

Parameters

in	*F*	simplicial_complex to operate on.
in	*s*	Simplex to decimate.
in	*clbk*	Callback function to map meta-data

Template Parameters

Complex	Typename of the simplicial_complex
Simplex	Typename of the simplex
Callback	Typename of the template template callback functor

Alias for SimplexSet

Alias for SimplexMap

8.1.2.5 decimateBackHalf()

```
template<typename Complex >
void casc::decimateBackHalf (
            Complex & F,
            SimplexMap< Complex > & simplexMap,
            typename decimation_detail::SimplexDataSet< Complex >::type & rv )
```

Parameters

F	Simplicial complex to operate on
simplexMap	SimplexMap mapping simplices before and after decimation
rv	Resulting data for each simplex

Template Parameters

Complex	Typename of the complex of interest

8.1.2.6 decimateFirstHalf()

```
template<typename Complex , typename Simplex >
Complex::KeyType casc::decimateFirstHalf (
            Complex & F,
            Simplex s,
            SimplexMap< Complex > & simplexMap )
```

Parameters

in	*F*	simplicial_complex to operate on.
in	*s*	Simplex to decimate.
	simplexMap	The simplex map to populate

Template Parameters

Complex	Typename of the simplicial_complex
Simplex	Typename of the simplex

Alias for SimplexSet

8.1.2.7 edge_up()

```
template<typename Visitor , typename EdgeID >
void casc::edge_up (
            Visitor && v,
            typename EdgeID::complex & F,
            EdgeID s )
```

Parameters

in	*v*	Visitor functor to apply.
	F	The simplicial_complex to traverse.
in	*s*	The edge to start at.

Template Parameters

Visitor	Typename of the functor.
EdgeID	Typename of the edge.

8.1.2.8 get() [1/3]

```
template<std::size_t k, typename Complex >
static auto& casc::get (
            const SimplexSet< Complex > & S )  [inline], [static]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

8.1.2.9 get() [2/3]

```
template<std::size_t k, typename Complex >
static auto& casc::get (
            SimplexMap< Complex > & S )  [inline], [static]
```

Parameters

S	SimplexMap to retrieve from.

Template Parameters

k	Simplex dimension.
Complex	Typename of the complex.

Returns

Returns a map of std::Array<KeyType, k> to SimplexSet.

8.1.2.10 get() [3/3]

```
template<std::size_t k, typename Complex >
static auto& casc::get (
            SimplexSet< Complex > & S )  [inline], [static]
```

Parameters

S	SimplexSet of interest.

Template Parameters

k	Simplex dimension desired.
Complex	Typename of the simplicial_complex.

Returns

A NodeSet which holds simplices of dimension 'k' and a member of SimplexSet 'S'.

8.1.2.11 getClosure() [1/2]

```
template<typename Complex >
void casc::getClosure (
            Complex & F,
            casc::SimplexSet< Complex > & S,
            casc::SimplexSet< Complex > & dest )
```

Parameters

in	*F*	Complex of interest.
in	*S*	SimplexSet to compute the closure of.
out	*dest*	Destination SimplexSet

Template Parameters

Complex	Typename of the complex.

8.1.2.12 getClosure() [2/2]

```
template<typename Complex , typename Simplex >
void casc::getClosure (
            Complex & F,
            Simplex & s,
            casc::SimplexSet< Complex > & dest )
```

Parameters

in	*F*	Complex of interest.
in	*s*	Simplex of interest.
out	*dest*	Destination SimplexSet.

Template Parameters

Complex	Typename of the complex.
Simplex	Typename of the simplex.

8.1.2.13 getLink() [1/2]

```
template<typename Complex >
void casc::getLink (
            Complex & F,
            casc::SimplexSet< Complex > & S,
            casc::SimplexSet< Complex > & dest )
```

Parameters

in	*F*	Complex of interest.
in	*S*	SimplexSet to get the link of.
out	*dest*	Destination SimplexSet.

Template Parameters

Complex	Typename of the complex.

8.1.2.14 getLink() [2/2]

```
template<typename Complex , typename Simplex >
void casc::getLink (
            Complex & F,
            Simplex & s,
            casc::SimplexSet< Complex > & dest )
```

Parameters

F	Complex of interest.
s	Simplex of interest.
dest	Destination SimplexSet.

Template Parameters

Complex	Typename of the complex.
Simplex	Typename of the simplex.

8.1.2.15 getStar() [1/2]

```
template<typename Complex >
void casc::getStar (
            Complex & F,
            casc::SimplexSet< Complex > & S,
            casc::SimplexSet< Complex > & dest )
```

Parameters

in	*F*	Complex of interest.
in	*S*	SimplexSet to compute the star of.
out	*dest*	Destination SimplexSet.

Template Parameters

Complex	Typename of the complex.

8.1.2.16 getStar() [2/2]

```
template<typename Complex , typename Simplex >
void casc::getStar (
            Complex & F,
            Simplex & s,
            casc::SimplexSet< Complex > & dest )
```

Parameters

in	*F*	Complex of interest.
	s	Simplex to get the star of.
out	*dest*	Destination SimplexSet.

Template Parameters

Complex	Typename of the complex.
Simplex	Typename of the simplex.

8.1.2.17 init_orientation()

```
template<typename Complex >
void casc::init_orientation (
            Complex & F )
```

Parameters

F	Simplicial complex of interest

Template Parameters

Complex	Typename of the simplicial complex

8.1.2.18 kneighbors() [1/2]

```
template<class Complex , std::size_t level, typename Iterator >
void casc::kneighbors (
            Complex & F,
            int ring,
            std::set< typename Complex::template SimplexID< level > > & nbors,
            Iterator begin,
            Iterator end )
```

Parameters

in	*F*	The simplicial_complex to traverse.
in	*ring*	The number of rings of neighbors to collect.
out	*nbors*	Set of previously seen simplices.
in	*begin*	The begin
in	*end*	The end

Template Parameters

Complex	Typename of the simplicial_complex.
level	Simplex dimension of the simplex and neighbors.
Iterator	{ description }

8.1.2.19 kneighbors() [2/2]

```
template<class Complex , class SimplexID >
void casc::kneighbors (
            Complex & F,
            SimplexID nid,
            int ring,
            std::set< SimplexID > & nbors )
```

Parameters

in	*F*	The simplicial complex
in	*nid*	Simplex of interest to get the nieghbors of.
in	*ring*	The number of rings to include as a neighbor.
out	*nbors*	Set of neighbors to populate.

Template Parameters

Complex	Typename of the complex.
SimplexID	Typename of the SimplexID.

8.1.2.20 kneighbors_up() [1/2]

```
template<class Complex , std::size_t level, typename Iterator >
void casc::kneighbors_up (
            Complex & F,
            int ring,
            std::set< typename Complex::template SimplexID< level > > & nbors,
            Iterator begin,
            Iterator end )
```

Parameters

in	*F*	The simplicial_complex to traverse.
in	*ring*	The number of rings of neighbors to collect.
out	*nbors*	Set of previously seen simplices.
in	*begin*	The begin
in	*end*	The end

Template Parameters

Complex	Typename of the simplicial_complex.
level	Simplex dimension of the simplex and neighbors.
Iterator	{ description }

8.1.2.21 kneighbors_up() [2/2]

```
template<class Complex , class SimplexID >
void casc::kneighbors_up (
            Complex & F,
            SimplexID nid,
            int ring,
            std::set< SimplexID > & nbors )
```

Parameters

in	*F*	The simplicial complex
in	*nid*	Simplex of interest to get the nieghbors of.
in	*ring*	The number of rings to include as a neighbor.
out	*nbors*	Set of neighbors to populate.

Template Parameters

Complex	Typename of the complex.
SimplexID	Typename of the SimplexID.

8.1.2.22 neighbors() [1/2]

```
template<class Complex , class SimplexID , class InsertIter >
void casc::neighbors (
            Complex & F,
            SimplexID nid,
            InsertIter iter )
```

Parameters

	F	The simplicial complex.
in	*nid*	Simplex to get neighbors of.
in	*iter*	The iterator to push members into.

Template Parameters

Complex	Type of the simplicial complex

Template Parameters

level	The integral level of the node
InsertIter	Typename of the iterator.

8.1.2.23 neighbors() [2/2]

```
template<class Complex , std::size_t level, class InsertIter >
void casc::neighbors (
            Complex & F,
            typename Complex::template SimplexID< level > nid,
            InsertIter iter )
```

This function gets the set of neighbors which share a common face. We compute this by traversing to all faces of the simplex of interest. Then we get all cofaces of this set. Depending on the type of iterator passed, duplicate simplices will be included or excluded. Note that this is the traditional definition of neighbor. For example, faces which share an edge are neighbors.

Parameters

	F	The simplicial complex
in	*nid*	Simplex to get neighbors of.
in	*iter*	The iterator to push members into.

Template Parameters

Complex	Type of the simplicial complex
level	The integral level of the node
InsertIter	Typename of the iterator.

8.1.2.24 neighbors_up() [1/2]

```
template<class Complex , class SimplexID , class InsertIter >
void casc::neighbors_up (
            Complex & F,
            SimplexID nid,
            InsertIter iter )
```

Parameters

	F	The simplicial complex.
in	*nid*	Simplex to get neighbors of.
in	*iter*	The iterator to push members into.

Template Parameters

Complex	Type of the simplicial complex
level	The integral level of the node
InsertIter	Typename of the iterator.

8.1.2.25 neighbors_up() [2/2]

```
template<class Complex , std::size_t level, class InsertIter >
void casc::neighbors_up (
            Complex & F,
            typename Complex::template SimplexID< level > nid,
            InsertIter iter )
```

Parameters

	F	The simplicial complex.
in	*nid*	Simplex to get neighbors of.
in	*iter*	The iterator to push members into.

Template Parameters

Complex	Type of the simplicial complex
level	The integral level of the node
InsertIter	Typename of the iterator.

8.1.2.26 operator"!=()

```
template<typename Complex >
bool casc::operator!= (
            const SimplexSet< Complex > & lhs,
            const SimplexSet< Complex > & rhs )
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

Complex	Typename of the simplicial_complex.

Returns

True if the sets are inequal, false otherwise.

8.1.2.27 operator==()

```
template<typename Complex >
bool casc::operator== (
            const SimplexSet< Complex > & lhs,
            const SimplexSet< Complex > & rhs )
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

Complex	Typename of the simplicial_complex

Returns

True if the sets are equal, false otherwise.

8.1.2.28 perform_insertion()

```
template<typename Complex >
void casc::perform_insertion (
            Complex & F,
            typename decimation_detail::SimplexDataSet< Complex >::type & S )
```

Parameters

F	The simplicial_complex to insert into.
S	SimplexSet of simplices to insert.

Template Parameters

Complex	Typename of complex

8.1.2.29 perform_removal()

```
template<typename Complex >
void casc::perform_removal (
            Complex & F,
            casc::SimplexSet< Complex > & S )
```

Parameters

F	The simplicial_complex to remove from.
S	SimplexSet of simplices to remove.

Template Parameters

Complex	Typename of complex

8.1.2.30 run_user_callback()

```
template<typename Complex , template< typename > class Callback>
void casc::run_user_callback (
            Complex & F,
            casc::SimplexMap< Complex > & S,
            Callback< Complex > && clbk,
            typename decimation_detail::SimplexDataSet< Complex >::type & rv )
```

Parameters

in	*F*	The simplicial_complex
in	*S*	SimplexMap of
in	*clbk*	User specified callback functor
out	*rv*	Multi-vector to place results.

Template Parameters

Complex	Typename of the simplicial_complex
Callback	Typename of the template template callback functor

8.1.2.31 set_difference()

```
template<typename Complex >
static void casc::set_difference (
            const SimplexSet< Complex > & A,
            const SimplexSet< Complex > & B,
            SimplexSet< Complex > & dest )  [static]
```

Parameters

in	*A*	A SimplexSet
in	*B*	Another SimplexSet
out	*dest*	The destination SimplexSet.

Template Parameters

Complex	Typename of the simplicial_complex.

8.1.2.32 set_intersection()

```
template<typename Complex >
static void casc::set_intersection (
            const SimplexSet< Complex > & A,
            const SimplexSet< Complex > & B,
            SimplexSet< Complex > & dest )  [static]
```

Parameters

in	*A*	A SimplexSet
in	*B*	Another SimplexSet
out	*dest*	The destination SimplexSet.

Template Parameters

Complex	Typename of the simplicial_complex.

8.1.2.33 set_union()

```
template<typename Complex >
static void casc::set_union (
            const SimplexSet< Complex > & A,
            const SimplexSet< Complex > & B,
            SimplexSet< Complex > & dest )  [static]
```

Parameters

in	*A*	A SimplexSet
in	*B*	Another SimplexSet
out	*dest*	The destination SimplexSet.

Template Parameters

Complex	Typename of the simplicial_complex.

8.1.2.34 to_string()

```
template<typename T , std::size_t k>
std::string casc::to_string (
             const std::array< T, k > & A )
```

Parameters

in	*A*	Array containing name of a simplex.

Template Parameters

T	Typename KeyType.
k	Dimension of the simplex.

Returns

String representation of the object.

8.1.2.35 visit_BFS_down()

```
template<typename Visitor , typename SimplexID >
void casc::visit_BFS_down (
             Visitor && v,
             typename SimplexID::complex & F,
             SimplexID s )
```

Parameters

in	*v*	Visitor functor to apply.
	F	The simplicial_complex to traverse.
in	*s*	The simplex to start at.

Template Parameters

Visitor	Typename of the functor.
SimplexID	Typename of the simplex.

8.1.2.36 visit_BFS_up()

```
template<typename Visitor , typename SimplexID >
void casc::visit_BFS_up (
            Visitor && v,
            typename SimplexID::complex & F,
            SimplexID s )
```

Parameters

in	*v*	Visitor functor to apply.
	F	The simplicial_complex to traverse.
in	*s*	The simplex to start at.

Template Parameters

Visitor	Typename of the functor.
SimplexID	Typename of the simplex.

8.1.2.37 writeDOT()

```
template<typename Complex >
void casc::writeDOT (
            const std::string & filename,
            Complex & F )
```

The resulting dot file can be rendered into an image using tools such as GraphViz.
`dot -Tpng input.dot > output.png`

Parameters

in	*filename*	Filename to write out to.
in	*F*	Simplicial complex to generate the DOT of.

Template Parameters

Complex	Typename of the simplicial complex.

8.2 index_tracker Namespace Reference

Index tracker namespace.

Namespaces

- index_tracker_detail

  B-tree internal data structures.

Data Structures

- class index_tracker

  Tracker of available indices implemented as a B-tree of intervals.

Functions

- template<typename T , std::size_t d>
  std::ostream & **operator**<< (std::ostream &out, const index_tracker_detail::BTreeNode< T, d > *head)

8.3 index_tracker::index_tracker_detail Namespace Reference

B-tree internal data structures.

Data Structures

- struct Interval

  Interval object represents a range.
- struct BTreeNode

  An array based BTree.

Typedefs

- template<typename Node >
  using **Pointer** = typename Node::Pointer
- template<typename Node >
  using **Data** = typename Node::Data
- template<typename Node >
  using **Scalar** = typename Node::Scalar

Functions

- template<typename T >
  bool **operator**< (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  bool **operator**> (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  bool **operator**< (T x, const Interval< T > &y)
- template<typename T >
  bool **operator**> (const Interval< T > &x, T y)
- template<typename T >
  bool **operator**< (const Interval< T > &x, T y)
- template<typename T >
  bool **operator**> (T x, const Interval< T > &y)
- template<typename T >
  bool **operator==** (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  std::ostream & **operator**<< (std::ostream &out, const Interval< T > &x)

- template<typename T >
  int **merge** (Interval< T > &A, T x)
- template<typename Node >
  void **rebalance** (Pointer< Node > head, std::size_t i)
- template<typename Node >
  void **insert_H** (Pointer< Node > head, const Data< Node > &data)
- template<typename Node >
  Pointer< Node > **insert** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  bool **get** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  void **get_replacement** (Pointer< Node > head, Data< Node > &key)
- template<typename Node >
  void **remove_H** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  Pointer< Node > **remove** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  void **fill_left** (Pointer< Node > head, Data< Node > &x)
- template<typename Node >
  void **fill_right** (Pointer< Node > head, Data< Node > &x)
- template<typename Node >
  void **insert_scalar_H** (Pointer< Node > head, Scalar< Node > data)
- template<typename Node >
  Pointer< Node > **insert_scalar** (Pointer< Node > head, Scalar< Node > data)
- template<typename Node >
  void **insert_left** (Pointer< Node > head, const Data< Node > &x)
- template<typename Node >
  bool **remove_scalar_H** (Pointer< Node > head, Scalar< Node > x)
- template<typename Node >
  bool **remove_scalar** (Pointer< Node > &head, Scalar< Node > data)
- template<typename Node >
  Scalar< Node > **pop_scalar** (Pointer< Node > &head)
- template<typename Node >
  void **destruct** (Pointer< Node > head)
- template<typename Node >
  Data< Node > **check_order** (Pointer< Node > head, Data< Node > curr)

8.4 util Namespace Reference

Metatemplate programming utilities namespace.

Data Structures

- struct range

  A range object to support range based for loops.
- struct type_holder

  Queue based data structure to hold list of types.
- struct type_holder< T, Ts... >

  Partial specialization to allow FIFO access of typenames.
- struct type_get

  Helper to get the kth element from a type_holder.
- struct type_get< 0, type_holder< Ts... > >

Specialization for terminal case.

- struct type_get< k, type_holder< Ts... > >

  Specialization to recursively pop types to get the kth type.

- struct type_map

  Map the types in C into `V<T>`.

- struct int_type_map

  Maps an integer sequence and typename, F, into outholder.

- struct type_swap

  Move a list of types from one container to another.

- struct type_swap< TUPLE, HOLDER< Ts... > >

  Move a list of types from one container to another.

- struct reverse_sequence

  Reverse an Integer Sequence.

- struct remove_first_val

  General template for removing the first value from a type holder.

- struct remove_first_val< Integer, InHolder< Integer, I, Is... > >

  Specialization for removing first integer from a sequence of compile time integers.

Functions

- template<typename T >
  range< T > make_range (T b, T e)

  Make a range object.

- template<typename T >
  range< T > make_range (std::pair< T, T > p)

  Makes a range object.

- template<class Integer , typename IntegerSequence , typename Fn , typename ... Args>
  void int_for_each (Fn &&f, Args &&... args)

  Calls a function `f.apply<k>()` for a sequence of integer k's.

8.4.1 Function Documentation

8.4.1.1 int_for_each()

```
template<class Integer , typename IntegerSequence , typename Fn , typename ...  Args>
void util::int_for_each (
            Fn && f,
            Args &&...  args )
```

Parameters

in	*args*	Arguments to f
in	*f*	Functor with `apply<k>()` method

Template Parameters

Integer	Integer type
IntegerSequence	Sequence of integers to iterate
Fn	Typename of functor f
Args	Typenames of the arguments

8.4.1.2 make_range() [1/2]

```
template<typename T >
range<T> util::make_range (
            std::pair< T, T > p )
```

Parameters

in	*p*	A pair containing begin and end iterators.

Template Parameters

T	Typename of the iterator.

Returns

Returns a range of the iterators.

8.4.1.3 make_range() [2/2]

```
template<typename T >
range<T> util::make_range (
            T b,
            T e )
```

Parameters

in	*b*	Iterator to the beginning.
in	*e*	Iterator to the end.

Template Parameters

T	Typename of the iterator.

Returns

Returns a range of the iterators.

Chapter 9

Data Structure Documentation

9.1 index_tracker::index_tracker_detail::BTreeNode< _T, _d > Struct Template Reference

An array based BTree.

`#include <index_tracker.h>`

Public Types

- using **Scalar** = _T
- using **Data** = Interval< Scalar >
- using **Pointer** = BTreeNode *

Public Member Functions

- **BTreeNode** (const Data &t)
- template<typename Iter >
  BTreeNode (Iter begin, Iter end)

Data Fields

- std::size_t **k**
- std::array< Data, N > **data**
- std::array< Pointer, N+1 > **next**

Static Public Attributes

- static constexpr std::size_t **d** = _d
- static constexpr std::size_t **N** = 2*d+1

9.1.1 Detailed Description

template<typename _T, std::size_t _d>
struct index_tracker::index_tracker_detail::BTreeNode< _T, _d >

Template Parameters

↩ _↩ *T*	{ description }
↩ _↩ *d*	{ description }

The documentation for this struct was generated from the following file:

- include/casc/index_tracker.h

9.2 casc::simplicial_complex< traits >::EdgeID< k > Struct Template Reference

External reference to an edge or a connection within the complex.

`#include <SimplicialComplex.h>`

Public Types

- using complex = simplicial_complex< traits >

  Typename of the complex.

Public Member Functions

- EdgeID ()

  Default contstructor wraps a nullptr and dummy edge.
- EdgeID (NodePtr< k > p, KeyType e)

  Constructor to wrap an Edge.
- EdgeID (const EdgeID &rhs)

  Copy constructor.
- EdgeID & operator= (const EdgeID &rhs)

  Assignment operator.
- auto const & operator∗ () const

  Dereferencing an EdgeID gets the data on the edge.
- auto & operator∗ ()

  Dereferencing an EdgeID gets the data on the edge.
- KeyType key () const

  Get the key of the edge.
- auto const & data () const

  Return the data stored on the edge.
- auto & data ()

  Return the data stored on the edge.
- SimplexID< k > up () const

  Get the coboundary simplex.
- SimplexID< k-1 > down () const

  Get the simplex below.

Data Fields

- friend simplicial_complex< traits >

  EdgeID is a friend of the complex.

Static Public Attributes

- static constexpr std::size_t level = k

  The dimension of the simplex which the edge points to.

Friends

- bool operator== (EdgeID lhs, EdgeID rhs)

  Equality of wrapped pointers and edges.
- bool operator!= (EdgeID lhs, EdgeID rhs)

  Compare wrapped pointers and edges.
- bool operator<= (EdgeID lhs, EdgeID rhs)

  Compare wrapped pointers and edges.
- bool operator>= (EdgeID lhs, EdgeID rhs)

  Compare wrapped pointers and edges.
- bool operator< (EdgeID lhs, EdgeID rhs)

  Less than defines an ordering of key types on the edges.
- bool operator> (EdgeID lhs, EdgeID rhs)

  Greater than comparison.

9.2.1 Detailed Description

template<typename traits>
template<std::size_t k>
struct casc::simplicial_complex< traits >::EdgeID< k >

Template Parameters

k	The edge connects a simplex of size k-1 to a simplex of size k.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 EdgeID() [1/2]

```
template<typename traits >
template<std::size_t k>
casc::simplicial_complex< traits >::EdgeID< k >::EdgeID (
            NodePtr< k > p,
            KeyType e )  [inline]
```

Parameters

in	*p*	Pointer to the next Node.
in	*e*	Key of the edge

9.2.2.2 EdgeID() [2/2]

```
template<typename traits >
template<std::size_t k>
casc::simplicial_complex< traits >::EdgeID< k >::EdgeID (
            const EdgeID< k > & rhs )  [inline]
```

Parameters

in	*rhs*	The right hand side

9.2.3 Member Function Documentation

9.2.3.1 down()

```
template<typename traits >
template<std::size_t k>
SimplexID<k-1> casc::simplicial_complex< traits >::EdgeID< k >::down ( ) const  [inline]
```

Returns

SimplexID of the simplex below the edge.

9.2.3.2 up()

```
template<typename traits >
template<std::size_t k>
SimplexID<k> casc::simplicial_complex< traits >::EdgeID< k >::up ( ) const  [inline]
```

Returns

SimplexID of the simplex above the edge.

The documentation for this struct was generated from the following file:

- include/casc/SimplicialComplex.h

9.3 index_tracker::index_tracker< _T, _d > Class Template Reference

Tracker of available indices implemented as a B-tree of intervals.

`#include <index_tracker.h>`

Public Types

- using Node = index_tracker_detail::BTreeNode< _T, _d >
  Typedef of BTree Node.
- using **T** = _T

Public Member Functions

- index_tracker ()
  Number of bins.
- void **insert** (T x)
- index_tracker_detail::Scalar< Node > **pop** ()
- void **remove** (index_tracker_detail::Scalar< Node > x)
- bool **empty** () const

Static Public Attributes

- constexpr static std::size_t d = _d
  Typename of the type to store.

Friends

- std::ostream & **operator**<< (std::ostream &out, const index_tracker &x)

9.3.1 Detailed Description

template<typename _T, std::size_t _d = 16>
class index_tracker::index_tracker< _T, _d >

Template Parameters

↩ _↩ *T*	Typename of the indices
↩ _↩ *d*	Max number of interval bins = 2∗value+1

9.3.2 Constructor & Destructor Documentation

9.3.2.1 index_tracker()

```
template<typename _T , std::size_t _d = 16>
index_tracker::index_tracker< _T, _d >::index_tracker ( )  [inline]
```

Initialize with interval [0∼max)

The documentation for this class was generated from the following file:

- include/casc/index_tracker.h

9.4 util::int_type_map< IntegerType, OutHolder, IntegerSequence, F > Struct Template Reference

Maps an integer sequence and typename, F, into outholder.

```
#include <util.h>
```

Public Types

- using type = typename detail::int_type_map_helper< IntegerType, OutHolder, IntegerSequence, F >::type
  Tuple of `Out<F<0>, F<1>, F<2>, ...>`.

9.4.1 Detailed Description

template<class IntegerType, template< class ... > class OutHolder, class IntegerSequence, template< IntegerType > class F> struct util::int_type_map< IntegerType, OutHolder, IntegerSequence, F >

Given an Integer Sequence `I<0,1,2,3,...>` and template template type `F<I>`, this function produces `Out<F<0>, F<1>, F<2>, ...>`.

Template Parameters

IntegerType	Typename of an integer type
OutHolder	Typename of a holder for types
IntegerSequence	Integral sequence of types
F	Typename of class to be broadcast with integer

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.5 index_tracker::index_tracker_detail::Interval< T > Struct Template Reference

Interval object represents a range.

`#include <index_tracker.h>`

Public Member Functions

- Interval ()
  Default constructor.
- Interval (T a)
  Construct an interval from a to a+1.
- Interval (T a, T b)
  Construct an interval from a to b.
- Interval (const Interval< T > &rhs)
  Copy constructor.
- Interval & operator= (const Interval &rhs)
  Assigment operator overload.
- bool has (T x)
  Is x in the bounds of the interval.
- T lower () const
  Get the lower inclusive bound of the interval.
- T upper () const
  Get the upper exclusive bound of the interval.
- T & lower ()
  Get the lower inclusive bound of the interval.
- T & upper ()
  Get the upper exclusive bound of the interval.
- std::size_t size ()
  Get the size of the interval.

9.5.1 Detailed Description

template<typename T>
struct index_tracker::index_tracker_detail::Interval< T >

Template Parameters

T	Typename of the interval data

9.5.2 Member Function Documentation

9.5.2.1 operator=()

```
template<typename T >
Interval& index_tracker::index_tracker_detail::Interval< T >::operator= (
            const Interval< T > & rhs )  [inline]
```

Parameters

in	*rhs*	The right hand side

Returns

Reference to this

The documentation for this struct was generated from the following file:

- include/casc/index_tracker.h

9.6 casc::Orientable Struct Reference

Class representing the orientation.

`#include <Orientable.h>`

Data Fields

- int orientation

  Integer representing +/- 1 orientation.

The documentation for this struct was generated from the following file:

- include/casc/Orientable.h

9.7 util::range< T > Struct Template Reference

A range object to support range based for loops.

`#include <util.h>`

Public Member Functions

- template<class C >
  range (C &&c)

  Construct a range for a container class.
- range (T b, T e)

  Construct a range from an iterator.
- T begin ()

  Get the beginning iterator.
- T end ()

  Get the end iterator.

9.7.1 Detailed Description

template<typename T>
struct util::range< T >

This is a basic data structure which implements a `begin()` and `end()` functions for range based for lopping added in C++11. See also `range-for`.

Template Parameters

T	Typename of the iterator

9.7.2 Constructor & Destructor Documentation

9.7.2.1 range() [1/2]

```
template<typename T >
template<class C >
util::range< T >::range (
            C && c )  [inline]
```

Parameters

in	*c*	Container class which implements begin() and end().

Template Parameters

C	Typename of the container.

9.7.2.2 range() [2/2]

```
template<typename T >
util::range< T >::range (
            T b,
            T e )  [inline]
```

Parameters

in	*b*	Beginning iterator
in	*e*	End iterator.

9.7.3 Member Function Documentation

9.7.3.1 begin()

```
template<typename T >
T util::range< T >::begin ( )  [inline]
```

Returns

Returns an iterator to the beginning.

9.7.3.2 end()

```
template<typename T >
T util::range< T >::end ( )  [inline]
```

Returns

Returns an iterator to the end.

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.8 util::remove_first_val< Integer, IntegerSequence > Struct Template Reference

General template for removing the first value from a type holder.

```
#include <util.h>
```

9.8.1 Detailed Description

template<class Integer, class IntegerSequence>
struct util::remove_first_val< Integer, IntegerSequence >

Template Parameters

Integer	Typename of integer.
IntegerSequence	Sequence of compile time integers.

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.9 util::remove_first_val< Integer, InHolder< Integer, I, Is... > > Struct Template Reference

Specialization for removing first integer from a sequence of compile time integers.

`#include <util.h>`

Public Types

- using type = InHolder< Integer, Is... >
  Type holder with first value removed.

9.9.1 Detailed Description

template<class Integer, template< class, Integer... > class InHolder, Integer I, Integer... Is>
struct util::remove_first_val< Integer, InHolder< Integer, I, Is... > >

Template Parameters

Integer	Typename of integer type.
InHolder	Type holder of integer sequence.
I	The first integer
Is	Remaining integers

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.10 util::reverse_sequence< Integer, IntegerSequence > Struct Template Reference

Reverse an Integer Sequence.

`#include <util.h>`

Public Types

- using type = typename detail::reverse_sequence_helper< Integer, IntegerSequence >::type
  Reversed sequence of types.

9.10.1 Detailed Description

template<class Integer, class IntegerSequence>
struct util::reverse_sequence< Integer, IntegerSequence >

Template Parameters

Integer	Typename of an integer class.
IntegerSequence	Sequence of compile-time integers.

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.11 casc::simplicial_complex< traits >::SimplexID< k > Struct Template Reference

A handle for a simplex object in the complex.

`#include <SimplicialComplex.h>`

Public Types

- using complex = simplicial_complex< traits >
  Typename of the complex.

Public Member Functions

- SimplexID ()
  Default constructor wraps a nullptr.
- SimplexID (NodePtr< k > p)
  Constructor to wrap a NodePtr<k>.
- SimplexID (const SimplexID &rhs)
  Copy constructor.
- SimplexID & operator= (const SimplexID &rhs)
  Assignment operator.
- operator std::uintptr_t () const
  Support casting to uintptr_t for hashing.
- complex::NodeData< k > const & operator* () const
  Dereferencing a SimplexID returns the data stored.
- complex::NodeData< k > & operator* ()
  Dereferencing a SimplexID returns the data stored.
- complex::NodeData< k > const & data () const
  Get a handle to the stored data.
- complex::NodeData< k > & data ()
  Get a handle to the stored data.

- std::array< KeyType, k > indices () const

  Gets the name of a simplex as an std::Array.
- template<class Inserter >
  void cover_insert (Inserter pos) const

  Insert the coboundary keys of a simple into an inserter.
- std::vector< KeyType > cover () const

  Get the coboundary keys of a simplex.
- template<std::size_t j>
  SimplexID< k+j > get_simplex_up (const KeyType(&s)[j]) const

  Get a coboundary simplex.
- template<std::size_t j>
  SimplexID< k+j > get_simplex_up (const std::array< KeyType, j > &arr) const

  Get a coboundary simplex.
- SimplexID< k+1 > get_simplex_up (const KeyType s) const

  Convenience version of get_simplex_up when the name 's' consists of a single character.
- template<std::size_t j>
  SimplexID< k-j > get_simplex_down (const KeyType(&s)[j]) const

  Gets the simplex down.
- template<std::size_t j>
  SimplexID< k-j > get_simplex_down (const std::array< KeyType, j > &arr) const

  Gets the simplex down.
- SimplexID< k-1 > get_simplex_down (const KeyType s) const

  Gets the simplex down.

Data Fields

- friend simplicial_complex< traits >

  SimplexID is a friend of the complex.

Static Public Attributes

- static constexpr std::size_t level = k

  The dimension of the simplex.

Friends

- bool operator== (SimplexID lhs, SimplexID rhs)

  Equality of wrapped pointers.
- bool operator!= (SimplexID lhs, SimplexID rhs)

  Inequality of wrapped pointers.
- bool operator<= (SimplexID lhs, SimplexID rhs)

  Compare wrapped pointers.
- bool operator>= (SimplexID lhs, SimplexID rhs)

  Compare wrapped pointers.
- bool operator< (SimplexID lhs, SimplexID rhs)

  Compare wrapped pointers.
- bool operator> (SimplexID lhs, SimplexID rhs)

  Compare wrapped pointers.
- std ::ostream & operator<< (std::ostream &out, const SimplexID &nid)

  Print the simplex as its name.

9.11.1 Detailed Description

template<typename traits>
template<std::size_t k>
struct casc::simplicial_complex< traits >::SimplexID< k >

SimplexID wraps a Node* for external handling. This way the end users are never exposed to a raw pointer. For all general purposes algorithms should use and pass SimplexIDs over raw pointers.

Template Parameters

k	The Simplex dimension.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 SimplexID() [1/2]

```
template<typename traits >
template<std::size_t k>
casc::simplicial_complex< traits >::SimplexID< k >::SimplexID (
            NodePtr< k > p )  [inline]
```

Parameters

in	*p*	The NodePtr to wrap

9.11.2.2 SimplexID() [2/2]

```
template<typename traits >
template<std::size_t k>
casc::simplicial_complex< traits >::SimplexID< k >::SimplexID (
            const SimplexID< k > & rhs )  [inline]
```

Parameters

in	*rhs*	Another SimplexID to copy.

9.11.3 Member Function Documentation

9.11.3.1 cover()

```
template<typename traits >
template<std::size_t k>
std::vector<KeyType> casc::simplicial_complex< traits >::SimplexID< k >::cover ( ) const
[inline]
```

Returns

A vector of coboundary indices.

9.11.3.2 cover_insert()

```
template<typename traits >
template<std::size_t k>
template<class Inserter >
void casc::simplicial_complex< traits >::SimplexID< k >::cover_insert (
            Inserter pos ) const  [inline]
```

Parameters

in	*pos*	Iterator inserter

Template Parameters

Inserter	Typename of the inserter.

9.11.3.3 get_simplex_up() [1/3]

```
template<typename traits >
template<std::size_t k>
SimplexID<k+1> casc::simplicial_complex< traits >::SimplexID< k >::get_simplex_up (
            const KeyType s ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*s*	The relative single character name of the desired simplex.

Template Parameters

i	The size of simplex 'id'.

Returns

SimplexID of node corresponding to $id \cup s$.

9.11.3.4 get_simplex_up() [2/3]

```
template<typename traits >
template<std::size_t k>
template<std::size_t j>
SimplexID<k+j> casc::simplicial_complex< traits >::SimplexID< k >::get_simplex_up (
            const KeyType(&) s[j] ) const  [inline]
```

Parameters

in	*s*	Array of keys to follow

Template Parameters

j	Number of keys

Returns

The simplex up

9.11.3.5 get_simplex_up() [3/3]

```
template<typename traits >
template<std::size_t k>
template<std::size_t j>
SimplexID<k+j> casc::simplicial_complex< traits >::SimplexID< k >::get_simplex_up (
            const std::array< KeyType, j > & arr ) const  [inline]
```

Parameters

in	*arr*	Array of keys to follow

Template Parameters

j	Number of keys

Returns

The simplex up

9.11.3.6 indices()

```
template<typename traits >
template<std::size_t k>
std::array<KeyType, k> casc::simplicial_complex< traits >::SimplexID< k >::indices ( ) const
[inline]
```

Parameters

in	*id*	SimplexID of the simplex of interest.

Returns

Array containing the name of 'id'.

9.11.4 Friends And Related Function Documentation

9.11.4.1 operator<<

```
template<typename traits >
template<std::size_t k>
std ::ostream& operator<< (
            std::ostream & out,
            const SimplexID< k > & nid )  [friend]
```

Parameters

	out	Handle to the stream
in	*nid*	SimplexID of interest

Returns

Handle to the stream

Example

```
 {(.c)}
mesh.insert<3>({0,1,2});
std::cout « s « std::endl;
s{0,1,2}"
```

The documentation for this struct was generated from the following file:

- include/casc/SimplicialComplex.h

9.12 casc::SimplexMap< Complex > Struct Template Reference

A multimap to represent a map of simplex indices to a set of simplices.

`#include <SimplexMap.h>`

Public Types

- template<std::size_t j>
  using SimplexID = typename Complex::template SimplexID< j >

  Alias for SimplexID.
- using LevelIndex = typename Complex::LevelIndex

  Index sequence of types from the simplicial_complex.
- using cLevelIndex = typename util::remove_first_val< std::size_t, LevelIndex >::type

  Index sequence starting at 1.
- using RevIndex = typename util::reverse_sequence< std::size_t, LevelIndex >::type

  Reversed Index sequence.
- using cRevIndex = typename util::reverse_sequence< std::size_t, cLevelIndex >::type

  Reversed index sequence stops at 1.
- using type_this = SimplexMap< Complex >

  Typename of this object.

Public Member Functions

- SimplexMap ()

  Default constructor.
- template<std::size_t k>
  auto & get ()

  Get the map for a particular simplex dimension.
- template<std::size_t k>
  auto & get () const

Friends

- std ::ostream & operator<< (std::ostream &output, const SimplexMap< Complex > &S)

  Print the SimplexMap.

9.12.1 Detailed Description

template<typename Complex>
struct casc::SimplexMap< Complex >

Template Parameters

Complex	Typename of the simplicial_complex.

9.12.2 Member Function Documentation

9.12.2.1 get() [1/2]

```
template<typename Complex >
template<std::size_t k>
```

```
auto& casc::SimplexMap< Complex >::get ( )  [inline]
```

Template Parameters

k	Simplex dimension to retrieve.

Returns

A map of SimplexID<k> to SimplexSet.

9.12.2.2 get() [2/2]

```
template<typename Complex >
template<std::size_t k>
auto& casc::SimplexMap< Complex >::get ( ) const  [inline]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

9.12.3 Friends And Related Function Documentation

9.12.3.1 operator<<

```
template<typename Complex >
std ::ostream& operator<< (
            std::ostream & output,
            const SimplexMap< Complex > & S )  [friend]
```

Parameters

	output	Handle to the stream to print to.
in	*S*	SimplexMap to print.

Returns

Handle to the stream.

The documentation for this struct was generated from the following file:

- include/casc/SimplexMap.h

9.13 casc::SimplexSet< Complex > Struct Template Reference

A multiset to store simplices in a simplicial_complex.

`#include <SimplexSet.h>`

Public Types

- template<std::size_t j>
  using SimplexID = typename Complex::template SimplexID< j >
  Alias for SimplexID.
- using LevelIndex = typename Complex::LevelIndex
  Index sequence of types from the simplicial_complex.
- using cLevelIndex = typename util::remove_first_val< std::size_t, LevelIndex >::type
  Index sequence starting at 1.
- using RevIndex = typename util::reverse_sequence< std::size_t, LevelIndex >::type
  Reversed index sequence.
- using cRevIndex = typename util::reverse_sequence< std::size_t, cLevelIndex >::type
  Reversed index sequence stops at 1.
- using type_this = SimplexSet< Complex >
  Typename of this.
- using SimplexIDLevel = typename util::int_type_map< std::size_t, std::tuple, LevelIndex, SimplexID >::type
  Tuple of SimplexIDs wrt an integral level.

Public Member Functions

- SimplexSet ()
  Default constructor.
- ~SimplexSet ()
  Default destructor.
- template<std::size_t k>
  auto empty () const noexcept
  Checks if a level has no elements.
- template<std::size_t k>
  auto size () const noexcept
  Return the number of elements in a level.
- void clear ()
  Clear the contents.
- template<std::size_t k>
  void insert (SimplexID< k > s)
  Insert a simplex into the set.
- void insert (const SimplexSet< Complex > &s)
  Insert a SimplexSet into this.
- template<std::size_t k>
  void erase (SimplexID< k > s)
  Remove a simplex from the set.
- void erase (const SimplexSet< Complex > &s)
  Remove a set of simplices.
- template<std::size_t k>
  auto find (const SimplexID< k > s)

Get the simplex of interest.

- template<std::size_t k>
  auto find (const SimplexID< k > s) const

  Get the simplex of interest.
- template<std::size_t k>
  auto end ()

  Get the past-the-end iterator.
- template<std::size_t k>
  auto cend () const

  Get the past-the-end iterator.
- template<std::size_t k>
  auto begin ()

  Get an interator to the first element of the container.
- template<std::size_t k>
  auto cbegin () const

  Get an interator to the first element of the container.
- template<std::size_t k>
  auto & **get** ()
- template<std::size_t k>
  auto & **get** () const

Data Fields

- util::type_map< SimplexIDLevel, NodeSet >::type tupleSet

  Tuple of NodeSets per level.

Friends

- std ::ostream & operator<< (std::ostream &output, const SimplexSet< Complex > &S)

  Print the SimplexSet.

9.13.1 Detailed Description

template<typename Complex>
struct casc::SimplexSet< Complex >

This is really a tuple of sets where each set corresponds to a simplex dimension. Many convenience functions are wrapped so this behaves much like a std::set.

Template Parameters

Complex	Typename of the simplicial_complex.

9.13.2 Member Function Documentation

9.13.2.1 begin()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::begin ( )  [inline]
```

Template Parameters

k	The simplex dimension to get iterator of.

Returns

Returns an iterator to the first element.

9.13.2.2 cbegin()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::cbegin ( ) const  [inline]
```

Template Parameters

k	The simplex dimension to get iterator of.

Returns

Returns an iterator to the first element.

9.13.2.3 cend()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::cend ( ) const  [inline]
```

Template Parameters

k	The simplex dimension to get iterator of.

Returns

Returns an iterator to the element following the last element of the set for the specified simplex dimension.

9.13.2.4 empty()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::empty ( ) const  [inline], [noexcept]
```

Template Parameters

k	Level to check.

Returns

True if the container is empty, false otherwise.

9.13.2.5 end()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::end ( )  [inline]
```

Template Parameters

k	The simplex dimension to get iterator of.

Returns

Returns an iterator to the element following the last element of the set for the specified simplex dimension.

9.13.2.6 erase() [1/2]

```
template<typename Complex >
void casc::SimplexSet< Complex >::erase (
            const SimplexSet< Complex > & s )  [inline]
```

Parameters

in	*s*	SimplexSet to remove.

9.13.2.7 erase() [2/2]

```
template<typename Complex >
template<std::size_t k>
```

```
void casc::SimplexSet< Complex >::erase (
            SimplexID< k > s )  [inline]
```

Parameters

in	*s*	Simplex to remove.

Template Parameters

k	Simplex dimension of 's'.

9.13.2.8 find() [1/2]

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::find (
            const SimplexID< k > s )  [inline]
```

Parameters

in	*s*	The simplex to search for.

Template Parameters

k	Simplex dimension of 's'.

Returns

Iterator to an element with key equivalent to s. If no such element is found, past-the-end iterator (see end()) is returned.

9.13.2.9 find() [2/2]

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::find (
            const SimplexID< k > s ) const  [inline]
```

Parameters

in	*s*	The simplex to search for.

Template Parameters

k	Simplex dimension of 's'.

Returns

Iterator to an element with key equivalent to s. If no such element is found, past-the-end iterator (see end()) is returned.

9.13.2.10 insert() [1/2]

```
template<typename Complex >
void casc::SimplexSet< Complex >::insert (
            const SimplexSet< Complex > & s )  [inline]
```

Parameters

in	*s*	The SimplexSet to insert.

9.13.2.11 insert() [2/2]

```
template<typename Complex >
template<std::size_t k>
void casc::SimplexSet< Complex >::insert (
            SimplexID< k > s )  [inline]
```

Parameters

in	*s*	Simplex to insert.

Template Parameters

k	Simplex dimension of 's'.

9.13.2.12 size()

```
template<typename Complex >
template<std::size_t k>
auto casc::SimplexSet< Complex >::size ( ) const  [inline], [noexcept]
```

Template Parameters

k	Simplex dimension to query

Returns

Returns the number of simplices of dimension `k` are in the set.

9.13.3 Friends And Related Function Documentation

9.13.3.1 operator<<

```
template<typename Complex >
std ::ostream& operator<< (
            std::ostream & output,
            const SimplexSet< Complex > & S )  [friend]
```

See also casc::simplicial_complex::SimplexID::operator<<.

Parameters

	output	Handle to the stream to print to.
in	*S*	SimplexSet to print.

Returns

Handle to the stream.

The documentation for this struct was generated from the following file:

- include/casc/SimplexSet.h

9.14 casc::simplicial_complex< traits > Class Template Reference

The CASC data structure for representing simplicial complexes of arbitrary dimensionality with coloring.

`#include <SimplicialComplex.h>`

Data Structures

- struct EdgeID

  External reference to an edge or a connection within the complex.
- struct SimplexID

  A handle for a simplex object in the complex.

Public Types

- using KeyType = typename traits::KeyType

  Typename of simplex keys.
- using NodeDataTypes = typename traits::NodeTypes

  Typenames of the data stored on simplices.
- using EdgeDataTypes = typename traits::EdgeTypes

  Typenames of the data stored on edges.
- using type_this = simplicial_complex< traits >

  Type of this.
- using LevelIndex = typename std::make_index_sequence< numLevels >

  Index of all simplex dimensions in the complex.
- template<std::size_t k>
  using NodeData = typename util::type_get< k, NodeDataTypes >::type
- template<std::size_t k>
  using EdgeData = typename util::type_get< k, EdgeDataTypes >::type

Public Member Functions

- simplicial_complex ()

  Default constructor.
- ~simplicial_complex ()

  Destruct the simplicial complex.
- template<std::size_t n>
  SimplexID< n > insert (const KeyType(&s)[n])

  Insert a simplex and all sub-simplices into the complex.
- template<std::size_t n>
  SimplexID< n > insert (const KeyType(&s)[n], const NodeData< n > &data)

  Insert a simplex and all sub-simplices into the complex along with data.
- template<std::size_t n>
  SimplexID< n > insert (const std::array< KeyType, n > &s)

  Insert a simplex named and all sub-simplices into the complex.
- template<std::size_t n>
  SimplexID< n > insert (const std::array< KeyType, n > &s, const NodeData< n > &data)

  Insert a simplex and all sub-simplices into the complex along with data.
- KeyType add_vertex ()

  Add a new vertex to the complex.
- KeyType add_vertex (const NodeData< 1 > &data)

  Add a new vertex to the complex with data.
- template<std::size_t n, typename Lambda >
  void get_name (SimplexID< n > id, Lambda fn) const

  Apply a lambda function the name of a simplex.
- template<std::size_t n>
  std::array< KeyType, n > get_name (SimplexID< n > id) const

  Gets the name of a simplex as an std::Array.
- std::array< KeyType, 0 > get_name (SimplexID< 0 >) const

  Gets the name of a simplex.
- template<std::size_t n>
  SimplexID< n > get_simplex_up (const KeyType(&s)[n]) const

  Gets the simplex with name 's'.
- template<std::size_t n>
  SimplexID< n > **get_simplex_up** (const std::array< KeyType, n > &arr) const

- template<std::size_t i, std::size_t j>
  SimplexID< i+j > get_simplex_up (const SimplexID< i > id, const KeyType(&s)[j]) const

  Get the simplex identifier which has the name 's' relative to the simplex 'id'.
- template<std::size_t i, std::size_t j>
  SimplexID< i+j > **get_simplex_up** (const SimplexID< i > id, const std::array< KeyType, j > &arr) const
- template<std::size_t i>
  SimplexID< i+1 > get_simplex_up (const SimplexID< i > id, const KeyType s) const

  Convenience version of get_simplex_up when the name 's' consists of a single character.
- SimplexID< 0 > get_simplex_up () const

  Get the root simplex.
- template<std::size_t i, std::size_t j>
  SimplexID< i-j > get_simplex_down (const SimplexID< i > id, const KeyType(&s)[j]) const

  Get the sub-simplex of the simplex 'id' which does not have 's' in the name.
- template<std::size_t i, std::size_t j>
  SimplexID< i-j > **get_simplex_down** (const SimplexID< i > id, const std::array< KeyType, j > &arr) const
- template<std::size_t i>
  SimplexID< i-1 > get_simplex_down (const SimplexID< i > id, const KeyType s) const

  Convenience version of get_simplex_down when the name 's' consists of a single character.
- SimplexID< 0 > get_simplex_down () const

  Get the root simplex.
- template<std::size_t k, class Inserter >
  void get_cover_insert (const SimplexID< k > id, Inserter pos) const

  Insert the coboundary keys of a simple into an inserter.
- template<std::size_t k, class Lambda >
  void get_cover (const SimplexID< k > id, Lambda fn) const

  Apply a lambda function to the coboundary keys.
- template<std::size_t k>
  std::vector< KeyType > get_cover (const SimplexID< k > id) const

  Get the coboundary keys of a simplex.
- template<std::size_t k>
  std::set< SimplexID< k+1 > > up (const std::set< SimplexID< k > > &&simplices) const

  Get the coboundary of a set of simplices.
- template<std::size_t k>
  std::set< SimplexID< k+1 > > up (const std::set< SimplexID< k > > &simplices) const

  Get the coboundary of a set of simplices.
- template<std::size_t k>
  std::set< SimplexID< k+1 > > up (const SimplexID< k > nid) const

  Get the coboundary of a simplex.
- template<std::size_t k, class InsertIter >
  void **up** (const std::set< SimplexID< k >> &&simplices, InsertIter iter) const
- template<std::size_t k, class InsertIter >
  void **up** (const std::set< SimplexID< k >> &simplices, InsertIter iter) const
- template<std::size_t k, class InsertIter >
  void **up** (const SimplexID< k > simplex, InsertIter iter) const
- template<std::size_t k>
  std::set< SimplexID< k-1 > > down (const std::set< SimplexID< k > > &&simplices) const

  Get the boundary of a set of simplices.
- template<std::size_t k>
  std::set< SimplexID< k-1 > > down (const std::set< SimplexID< k > > &simplices) const

  Get the boundary of a set of simplices.
- template<std::size_t k>
  std::set< SimplexID< k-1 > > down (const SimplexID< k > simplex) const

  Get the boundary of a simplex.

- template<std::size_t k, class InsertIter >
  void **down** (const std::set< SimplexID< k >> &&simplices, InsertIter iter) const
- template<std::size_t k, class InsertIter >
  void **down** (const std::set< SimplexID< k >> &simplices, InsertIter iter) const
- template<std::size_t k, class InsertIter >
  void **down** (const SimplexID< k > simplex, InsertIter iter) const
- template<std::size_t k>
  EdgeID< k+1 > get_edge_up (SimplexID< k > simplex, KeyType a)

  Gets the edge up from a simplex.
- template<std::size_t k>
  EdgeID< k > get_edge_down (SimplexID< k > simplex, KeyType a)

  Gets the edge down from a simplex.
- template<std::size_t k>
  EdgeID< k+1 > get_edge_up (SimplexID< k > simplex, KeyType a) const

  Gets the edge up from a simplex.
- template<std::size_t k>
  EdgeID< k > get_edge_down (SimplexID< k > simplex, KeyType a) const

  Gets the edge down from a simplex.
- template<std::size_t k>
  bool exists (const KeyType(&s)[k]) const

  Check whether a simplex with some name exists.
- template<std::size_t k>
  std::size_t size () const

  Get the number of simplices of dimension 'k'.
- template<std::size_t k>
  auto get_level_id ()

  Create an iterator to traverse the SimplexIDs of a dimension.
- template<std::size_t k>
  auto get_level_id () const

  Create an iterator to traverse the SimplexIDs of a dimension.
- template<std::size_t k>
  auto get_level ()

  Create an iterator to traverse the simplex data of a dimension.
- template<std::size_t k>
  auto get_level () const

  Create an iterator to traverse the simplex data of a dimension.
- template<std::size_t k>
  std::size_t remove (const KeyType(&s)[k])

  Remove a simplex and all dependent simplices by name.
- template<std::size_t k>
  std::size_t remove (const std::array< KeyType, k > &s)

  Remove a simplex and all dependent simplices by name.
- template<std::size_t k>
  std::size_t remove (SimplexID< k > s)

  Remove a simplex and all dependent simplices by SimplexID.
- template<std::size_t k>
  bool onBoundary (const SimplexID< k > s) const

  Checks whether a simplex is on a boundary.
- template<std::size_t level>
  bool nearBoundary (const SimplexID< level > s) const

  Checks whether a simplex is near a boundary.
- template<std::size_t L, std::size_t R>
  bool leq (SimplexID< L > lhs, SimplexID< R > rhs) const

Less than or equal to comparison operator of two SimplexIDs.

- template<std::size_t L, std::size_t R>
  bool eq (SimplexID< L >, SimplexID< R >) const

  Equality comparison of two simplices.
- template<std::size_t k>
  bool eq (SimplexID< k > lhs, SimplexID< k > rhs) const

  Equality comparison of two simplices.
- template<std::size_t L, std::size_t R>
  bool lt (SimplexID< L > lhs, SimplexID< R > rhs) const

  Less than comparison of simplices.

Static Public Attributes

- static constexpr std::size_t numLevels = NodeDataTypes::size

  Total number of levels in the complex.
- static constexpr std::size_t topLevel = numLevels-1

  Dimension of the simplicial complex.
- static constexpr std::size_t bdryLevel = numLevels-2

  Dimension of boundaries.

Friends

- struct SimplexID
- struct EdgeID

9.14.1 Detailed Description

template<typename traits>
class casc::simplicial_complex< traits >

You can create a CASC object by defining a struct containing the traits of the complex. For example:

```
struct complex_traits{
    using KeyType = int;
    using NodeTypes = util::type_holder<int,int,int,int>;
    using EdgeTypes = util::type_holder<int,int,int>;
};
using SurfaceMesh = simplicial_complex<complex_traits>;
```

This is the preferred method for creating a new CASC type. Alternatively you can use the AbstractSimplicialComplex alias to build a struct for you.

Template Parameters

traits	A struct defining the dimension of the complex and data to be stored on each node and edge.

9.14.2 Member Typedef Documentation

9.14.2.1 EdgeData

```
template<typename traits >
template<std::size_t k>
using casc::simplicial_complex< traits >::EdgeData = typename util::type_get<k, EdgeDataTypes>↩
::type
```

Convenience alias for the user specified EdgeData<k> typename

9.14.2.2 NodeData

```
template<typename traits >
template<std::size_t k>
using casc::simplicial_complex< traits >::NodeData = typename util::type_get<k, NodeDataTypes>↩
::type
```

Convenience alias for the user specified NodeData<k> typename

9.14.3 Constructor & Destructor Documentation

9.14.3.1 ~simplicial_complex()

```
template<typename traits >
casc::simplicial_complex< traits >::~simplicial_complex ( )  [inline]
```

Recursively go over the simplices and remove them prior to destructing the CASC object itself.

9.14.4 Member Function Documentation

9.14.4.1 add_vertex() [1/2]

```
template<typename traits >
KeyType casc::simplicial_complex< traits >::add_vertex ( )  [inline]
```

A list of currently unused indices are tracked using a B-tree. This function retrieves a currently unused index and creates a new vertex while returning the new key.

Returns

The key of the new vertex.

9.14.4.2 add_vertex() [2/2]

```
template<typename traits >
KeyType casc::simplicial_complex< traits >::add_vertex (
            const NodeData< 1 > & data )  [inline]
```

Returns

The key of the new vertex.

9.14.4.3 down() [1/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k-1> > casc::simplicial_complex< traits >::down (
            const SimplexID< k > simplex ) const  [inline]
```

Parameters

simplex	The simplex of interest.

Template Parameters

k	The dimension of the simplex.

Returns

Set of (k-1)-simplices of which 'simplex' is a coface of.

9.14.4.4 down() [2/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k-1> > casc::simplicial_complex< traits >::down (
            const std::set< SimplexID< k > > && simplices ) const  [inline]
```

Parameters

simplices	The set of simplicies.

Template Parameters

k	The dimension of the simplices.

Returns

The set of boundary simplices.

9.14.4.5 down() [3/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k-1> > casc::simplicial_complex< traits >::down (
            const std::set< SimplexID< k > > & simplices ) const  [inline]
```

Parameters

simplices	The set of simplicies.

Template Parameters

k	The dimension of the simplices.

Returns

The set of boundary simplices.

9.14.4.6 eq() [1/2]

```
template<typename traits >
template<std::size_t k>
bool casc::simplicial_complex< traits >::eq (
            SimplexID< k > lhs,
            SimplexID< k > rhs ) const  [inline]
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

k	Dimension of the simplices.

Returns

True if the names are the same.

9.14.4.7 eq() [2/2]

```
template<typename traits >
template<std::size_t L, std::size_t R>
bool casc::simplicial_complex< traits >::eq (
            SimplexID< L > ,
            SimplexID< R >  ) const  [inline]
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

L	Dimension of lhs simplex.
R	Dimension of rhs simplex.

Returns

Always false as L != R. The L==R case is overloaded by partial specialization.

9.14.4.8 exists()

```
template<typename traits >
template<std::size_t k>
bool casc::simplicial_complex< traits >::exists (
            const KeyType(&) s[k] ) const  [inline]
```

Parameters

in	*s*	C-style array of the name

Template Parameters

k	The dimension of the simplex.

Returns

True if the simplex is in the complex.

9.14.4.9 get_cover() [1/2]

```
template<typename traits >
template<std::size_t k>
```

```
std::vector<KeyType> casc::simplicial_complex< traits >::get_cover (
            const SimplexID< k > id ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.

Template Parameters

k	The dimension of the simplex.

Returns

A vector of coboundary indices.

9.14.4.10 get_cover() [2/2]

```
template<typename traits >
template<std::size_t k, class Lambda >
void casc::simplicial_complex< traits >::get_cover (
            const SimplexID< k > id,
            Lambda fn ) const  [inline]
```

Parameters

in	*id*	The identifier
in	*fn*	The function

Template Parameters

k	The dimension of the simplex.
Lambda	Typename of a functor which supports operator(KeyType).

9.14.4.11 get_cover_insert()

```
template<typename traits >
template<std::size_t k, class Inserter >
void casc::simplicial_complex< traits >::get_cover_insert (
            const SimplexID< k > id,
            Inserter pos ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*pos*	Iterator inserter

Template Parameters

k	The dimension of the simplex.
Inserter	Typename of the inserter.

9.14.4.12 get_edge_down() [1/2]

```
template<typename traits >
template<std::size_t k>
EdgeID<k> casc::simplicial_complex< traits >::get_edge_down (
            SimplexID< k > simplex,
            KeyType a )  [inline]
```

Parameters

in	*simplex*	The simplex of interest.
in	*a*	Key of the edge to get.

Template Parameters

k	The level of the simplex of interest

Returns

The edge down.

9.14.4.13 get_edge_down() [2/2]

```
template<typename traits >
template<std::size_t k>
EdgeID<k> casc::simplicial_complex< traits >::get_edge_down (
            SimplexID< k > simplex,
            KeyType a ) const  [inline]
```

Parameters

in	*simplex*	The simplex of interest.
in	*a*	Key of the edge to get.

Template Parameters

k	The level of the simplex of interest

Returns

The edge down.

9.14.4.14 get_edge_up() [1/2]

```
template<typename traits >
template<std::size_t k>
EdgeID<k+1> casc::simplicial_complex< traits >::get_edge_up (
            SimplexID< k > simplex,
            KeyType a )  [inline]
```

Parameters

in	*simplex*	The simplex of interest.
in	*a*	Key of the edge to get.

Template Parameters

k	The level of the simplex of interest

Returns

The edge up.

9.14.4.15 get_edge_up() [2/2]

```
template<typename traits >
template<std::size_t k>
EdgeID<k+1> casc::simplicial_complex< traits >::get_edge_up (
            SimplexID< k > simplex,
            KeyType a ) const  [inline]
```

Parameters

in	*simplex*	The simplex of interest.
in	*a*	Key of the edge to get.

Template Parameters

k	The level of the simplex of interest

Returns

The edge up.

9.14.4.16 get_level() [1/2]

```
template<typename traits >
template<std::size_t k>
auto casc::simplicial_complex< traits >::get_level ( )  [inline]
```

Template Parameters

k	The simplex dimension to traverse.

Returns

An iterator across the data of all k-simplices in the complex.

9.14.4.17 get_level() [2/2]

```
template<typename traits >
template<std::size_t k>
auto casc::simplicial_complex< traits >::get_level ( ) const  [inline]
```

Template Parameters

k	The simplex dimension to traverse.

Returns

An iterator across the data of all k-simplices in the complex.

9.14.4.18 get_level_id() [1/2]

```
template<typename traits >
template<std::size_t k>
auto casc::simplicial_complex< traits >::get_level_id ( )  [inline]
```

Template Parameters

k	The simplex dimension to traverse.

Returns

An iterator across all k-simplices of the complex.

9.14.4.19 get_level_id() [2/2]

```
template<typename traits >
template<std::size_t k>
auto casc::simplicial_complex< traits >::get_level_id ( ) const  [inline]
```

Template Parameters

k	The simplex dimension to traverse.

Returns

An iterator across all k-simplices of the complex.

9.14.4.20 get_name() [1/3]

```
template<typename traits >
std::array<KeyType, 0> casc::simplicial_complex< traits >::get_name (
            SimplexID< 0 >  ) const  [inline]
```

This is the explicit specialization which handles the empty set simplex.

Parameters

in	*id*	SimplexID of the simplex of interest.

Returns

Array containing the name of 'id'.

9.14.4.21 get_name() [2/3]

```
template<typename traits >
template<std::size_t n>
std::array<KeyType, n> casc::simplicial_complex< traits >::get_name (
            SimplexID< n > id ) const  [inline]
```

Parameters

in	*id*	SimplexID of the simplex of interest.

Template Parameters

n	Size of the simplex referenced by 'id'.

Returns

Array containing the name of 'id'.

9.14.4.22 get_name() [3/3]

```
template<typename traits >
template<std::size_t n, typename Lambda >
void casc::simplicial_complex< traits >::get_name (
            SimplexID< n > id,
            Lambda fn ) const  [inline]
```

Parameters

in	*id*	SimplexID of the simplex of interest.
in	*fn*	Lambda function to apply to the name of 'id'.

Template Parameters

n	Dimension of simplex 'id'.
Lambda	Functor which supports operator(KeyType).

9.14.4.23 get_simplex_down() [1/3]

```
template<typename traits >
SimplexID<0> casc::simplicial_complex< traits >::get_simplex_down ( ) const  [inline]
```

Returns

The root simplex.

9.14.4.24 get_simplex_down() [2/3]

```
template<typename traits >
template<std::size_t i>
SimplexID<i-1> casc::simplicial_complex< traits >::get_simplex_down (
            const SimplexID< i > id,
            const KeyType s ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*s*	The relative single character name of the desired simplex.

Template Parameters

i	The size of simplex 'id'.

Returns

The node down.

9.14.4.25 get_simplex_down() [3/3]

```
template<typename traits >
template<std::size_t i, std::size_t j>
SimplexID<i-j> casc::simplicial_complex< traits >::get_simplex_down (
            const SimplexID< i > id,
            const KeyType(&) s[j] ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*s*	The relative name of the desired simplex.

Template Parameters

i	The size of simplex 'id'.
j	The length of the name 's'

Returns

The node down.

9.14.4.26 get_simplex_up() [1/4]

```
template<typename traits >
SimplexID<0> casc::simplicial_complex< traits >::get_simplex_up ( ) const  [inline]
```

Returns

The root simplex.

9.14.4.27 get_simplex_up() [2/4]

```
template<typename traits >
template<std::size_t n>
SimplexID<n> casc::simplicial_complex< traits >::get_simplex_up (
            const KeyType(&) s[n] ) const  [inline]
```

Parameters

in	*s*	Name of the simplex to find.

Template Parameters

n	Dimension of simplex s.

Returns

SimplexID of node corresponding to 's'.

9.14.4.28 get_simplex_up() [3/4]

```
template<typename traits >
template<std::size_t i>
SimplexID<i+1> casc::simplicial_complex< traits >::get_simplex_up (
            const SimplexID< i > id,
            const KeyType s ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*s*	The relative single character name of the desired simplex.

Template Parameters

i	The size of simplex 'id'.

Returns

SimplexID of node corresponding to $id \cup s$.

9.14.4.29 get_simplex_up() [4/4]

```
template<typename traits >
template<std::size_t i, std::size_t j>
SimplexID<i+j> casc::simplicial_complex< traits >::get_simplex_up (
            const SimplexID< i > id,
            const KeyType(&) s[j] ) const  [inline]
```

Parameters

in	*id*	The identifier of a simplex.
in	*s*	The relative name of the desired simplex.

Template Parameters

i	The size of simplex 'id'.
j	The length of the name 's'.

Returns

SimplexID of node corresponding to $id \cup s$.

9.14.4.30 insert() [1/4]

```
template<typename traits >
template<std::size_t n>
SimplexID<n> casc::simplicial_complex< traits >::insert (
            const KeyType(&) s[n] )  [inline]
```

Example – insert the simplex {1,2,3}:
```
mesh.insert<3>({1,2,3});
```

Parameters

in	*s*	A C style array of vertices of simplex 's'.

Template Parameters

n	Dimension of simplex 's'.

9.14.4.31 insert() [2/4]

```
template<typename traits >
template<std::size_t n>
SimplexID<n> casc::simplicial_complex< traits >::insert (
            const KeyType(&) s[n],
            const NodeData< n > & data )  [inline]
```

Example – insert the simplex {1,2,3} with data:
```
mesh.insert<3>({1,2,3}, 5);
```

Parameters

in	*s*	A C style array of vertices of simplex 's'.
in	*data*	The data to be stored at the simplex 's'.

Template Parameters

n	Dimension of simplex 's'.

9.14.4.32 insert() [3/4]

```
template<typename traits >
template<std::size_t n>
SimplexID<n> casc::simplicial_complex< traits >::insert (
            const std::array< KeyType, n > & s )  [inline]
```

Parameters

in	*s*	Array of vertices comprising 's'.

Template Parameters

n	Dimension of simplex 's'.

9.14.4.33 insert() [4/4]

```
template<typename traits >
template<std::size_t n>
SimplexID<n> casc::simplicial_complex< traits >::insert (
            const std::array< KeyType, n > & s,
            const NodeData< n > & data )  [inline]
```

Parameters

in	*s*	Array of vertices comprising 's'.
in	*data*	The data to be stored at the simplex 's'.

Template Parameters

n	Dimension of simplex 's'.

9.14.4.34 leq()

```
template<typename traits >
template<std::size_t L, std::size_t R>
bool casc::simplicial_complex< traits >::leq (
            SimplexID< L > lhs,
            SimplexID< R > rhs ) const  [inline]
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

L	Dimension of lhs simplex.
R	Dimension of rhs simplex.

Returns

True if lhs is rhs or a proper face of rhs.

9.14.4.35 lt()

```
template<typename traits >
template<std::size_t L, std::size_t R>
bool casc::simplicial_complex< traits >::lt (
            SimplexID< L > lhs,
            SimplexID< R > rhs ) const  [inline]
```

Parameters

in	*lhs*	The left hand side
in	*rhs*	The right hand side

Template Parameters

L	Dimension of lhs simplex.
R	Dimension of rhs simplex.

Returns

True if lhs is a proper subface of rhs.

9.14.4.36 nearBoundary()

```
template<typename traits >
template<std::size_t level>
bool casc::simplicial_complex< traits >::nearBoundary (
            const SimplexID< level > s ) const  [inline]
```

Parameters

in	*s*	SimplexID of interest

Template Parameters

level	Dimension of the simplex

Returns

True if the simplex or any subsimplices are onBoundary.

9.14.4.37 onBoundary()

```
template<typename traits >
template<std::size_t k>
bool casc::simplicial_complex< traits >::onBoundary (
            const SimplexID< k > s ) const  [inline]
```

Parameters

in	*s*	SimplexID of interest

Template Parameters

k	Dimension of the simplex

Returns

True if the simplex is a member of a topLevel-1 simplex on the boundary or if the simplex is on a boundary or if the simplex is a coboundary of a boundary topLevel-1 simplex.

9.14.4.38 remove() [1/3]

```
template<typename traits >
template<std::size_t k>
std::size_t casc::simplicial_complex< traits >::remove (
            const KeyType(&) s[k] )  [inline]
```

Parameters

in	*s*	C-style array with the name of the simplex to remove.

Template Parameters

k	The dimension of the simplex.

Returns

Integer corresponding to the number of simplices removed.

9.14.4.39 remove() [2/3]

```
template<typename traits >
template<std::size_t k>
std::size_t casc::simplicial_complex< traits >::remove (
            const std::array< KeyType, k > & s )  [inline]
```

Parameters

in	*s*	std::array with the name of the simplex to remove.

Template Parameters

k	The dimension of the simplex.

Returns

Integer corresponding to the number of simplices removed.

9.14.4.40 remove() [3/3]

```
template<typename traits >
template<std::size_t k>
std::size_t casc::simplicial_complex< traits >::remove (
            SimplexID< k > s )  [inline]
```

Parameters

in	*s*	SimplexID of the simplex to remove.

Template Parameters

k	The dimension of the simplex.

Returns

Integer corresponding to the number of simplices removed.

9.14.4.41 size()

```
template<typename traits >
template<std::size_t k>
std::size_t casc::simplicial_complex< traits >::size ( ) const  [inline]
```

Template Parameters

k	The dimension of interest.

Returns

Integer number of k-simplices in the complex.

9.14.4.42 up() [1/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k+1> > casc::simplicial_complex< traits >::up (
            const SimplexID< k > nid ) const  [inline]
```

Parameters

nid	The simplex of interest

Template Parameters

k	The dimension of the simplex.

Returns

Set of (k+1)-simplices of which 'nid' is a face of.

9.14.4.43 up() [2/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k+1> > casc::simplicial_complex< traits >::up (
            const std::set< SimplexID< k > > && simplices ) const  [inline]
```

Parameters

simplices	The set of simplices

Template Parameters

k	The dimension of the simplices.

Returns

The set of coboundary simplices.

9.14.4.44 up() [3/3]

```
template<typename traits >
template<std::size_t k>
std::set<SimplexID<k+1> > casc::simplicial_complex< traits >::up (
            const std::set< SimplexID< k > > & simplices ) const  [inline]
```

Parameters

simplices	The set of simplices

Template Parameters

k	The dimension of the simplices.

Returns

The set of coboundary simplices.

9.14.5 Friends And Related Function Documentation

9.14.5.1 EdgeID

```
template<typename traits >
friend struct EdgeID  [friend]
```

EdgeID is a friend to simplicial_complex

9.14.5.2 SimplexID

```
template<typename traits >
friend struct SimplexID  [friend]
```

SimplexID is a friend of simplicial_complex

The documentation for this class was generated from the following file:

- include/casc/SimplicialComplex.h

9.15 util::type_get< k, T > Struct Template Reference

Helper to get the kth element from a type_holder.

```
#include <util.h>
```

9.15.1 Detailed Description

template<std::size_t k, typename T>
struct util::type_get< k, T >

This is the empty general template which will be later specialized.

Template Parameters

k	Integer index of the type to retrieve
T	A type_holder queue of typenames

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.16 util::type_get< 0, type_holder< Ts... > > Struct Template Reference

Specialization for terminal case.

`#include <util.h>`

Public Types

- using type = typename type_holder< Ts... >::head
  The first type of the type_holder.

9.16.1 Detailed Description

template<typename ... Ts>
struct util::type_get< 0, type_holder< Ts... > >

Template Parameters

Ts	Following typenames

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.17 util::type_get< k, type_holder< Ts... > > Struct Template Reference

Specialization to recursively pop types to get the kth type.

`#include <util.h>`

Public Types

- using type = typename type_get< k-1, typename type_holder< Ts... >::tail >::type
  Recurse after popping the first type off.

9.17.1 Detailed Description

template<std::size_t k, typename ... Ts>
struct util::type_get< k, type_holder< Ts... > >

Template Parameters

k	Integral constant of the type to get
Ts	List of typenames

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.18 util::type_holder< Ts > Struct Template Reference

Queue based data structure to hold list of types.

`#include <util.h>`

Static Public Attributes

- static const std::size_t size = sizeof ... (Ts)

  Length of the list of types.

9.18.1 Detailed Description

template<typename ... Ts>
struct util::type_holder< Ts >

Types in the type_holder can be accessed by accessing the `head` type. Subsequent types are in the `tail`. See also type_get.

Template Parameters

Ts	List of typenames

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.19 util::type_holder< T, Ts... > Struct Template Reference

Partial specialization to allow FIFO access of typenames.

`#include <util.h>`

Public Types

- using head = T

  The first type.
- using tail = type_holder< Ts... >

  The following types.

Static Public Attributes

- static const std::size_t size = 1 + type_holder<Ts...>::size

  Length of the list of types.

9.19.1 Detailed Description

template<typename T, typename ... Ts>
struct util::type_holder< T, Ts... >

Template Parameters

T	The first typename
Ts	The following typenames

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.20 util::type_map< C, V > Struct Template Reference

Map the types in C into `V<T>`.

`#include <util.h>`

Public Types

- using type = typename detail::type_map_helper< C, V >::type

  Tuple of `C<V<T1>, V<T2>, V<T3>, ...>`

9.20.1 Detailed Description

template<class C, template< typename > class V>
struct util::type_map< C, V >

Given a container of types `C<T1,T2,T3,...>` and template template type `V<T>`, this function will apply the types in C to `V<T>`. This produces `C<V<T1>, V<T2>, V<T3>, ...>`.

Template Parameters

C	Container of compile time types.
V	Template template class V<T> to map into.

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.21 util::type_swap< TUPLE, HOLDER_FULL > Struct Template Reference

Move a list of types from one container to another.

`#include <util.h>`

9.21.1 Detailed Description

template<template< class ... > class TUPLE, typename HOLDER_FULL>
struct util::type_swap< TUPLE, HOLDER_FULL >

Template Parameters

TUPLE	Empty container
HOLDER_FULL	Full container

The documentation for this struct was generated from the following file:

- include/casc/util.h

9.22 util::type_swap< TUPLE, HOLDER< Ts... > > Struct Template Reference

Move a list of types from one container to another.

`#include <util.h>`

Public Types

- using type = TUPLE< Ts... >

  Empty container filled with typenames from full container.

9.22.1 Detailed Description

template<template< class ... > class TUPLE, template< class ... > class HOLDER, typename ... Ts>
struct util::type_swap< TUPLE, HOLDER< Ts... > >

Template Parameters

TUPLE	Empty container
HOLDER	Full container
Ts	Typenames in full container

The documentation for this struct was generated from the following file:

- include/casc/util.h

Chapter 10

File Documentation

10.1 include/casc/CASCFunctions.h File Reference

Contains various functions that operate on simplicial complexes.

```
#include <iostream>
#include <fstream>
#include "SimplicialComplex.h"
#include "CASCTraversals.h"
#include "SimplexSet.h"
#include "stringutil.h"
```

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<typename Complex >
  void casc::getStar (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the star of a SimplexSet.
- template<typename Complex , typename Simplex >
  void casc::getStar (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Gets the star of a simplex.
- template<typename Complex >
  void casc::getClosure (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the closure of a simplex set.
- template<typename Complex , typename Simplex >
  void casc::getClosure (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Compute the closure of a simplex.
- template<typename Complex >
  void casc::getLink (Complex &F, casc::SimplexSet< Complex > &S, casc::SimplexSet< Complex > &dest)

  Gets the link of a SimplexSet.
- template<typename Complex , typename Simplex >
  void casc::getLink (Complex &F, Simplex &s, casc::SimplexSet< Complex > &dest)

  Gets the link of a simplex.
- template<typename Complex >
  void casc::writeDOT (const std::string &filename, Complex &F)

  Writes out the topology of an ASC into the dot format.

10.2 include/casc/CASCTraversals.h File Reference

Implementations of various advanced traversals such as by neighborhood and breadth first search.

```
#include <set>
#include <vector>
#include <iostream>
#include <string>
#include <type_traits>
#include <utility>
#include <casc/casc>
```

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<typename Visitor , typename SimplexID >
  void casc::visit_BFS_up (Visitor &&v, typename SimplexID::complex &F, SimplexID s)

  Traverse BFS up the complex and apply a visitor function to each simplex visited.
- template<typename Visitor , typename SimplexID >
  void casc::visit_BFS_down (Visitor &&v, typename SimplexID::complex &F, SimplexID s)

  Traverse BFS down the complex and apply a visitor function to each simplex visited.
- template<typename Visitor , typename EdgeID >
  void casc::edge_up (Visitor &&v, typename EdgeID::complex &F, EdgeID s)

  Traverse across edges BFS.
- template<class Complex , std::size_t level, class InsertIter >
  void casc::neighbors (Complex &F, typename Complex::template SimplexID< level > nid, InsertIter iter)

  Push the immediate face neighbors into the provided iterator.
- template<class Complex , class SimplexID , class InsertIter >
  void casc::neighbors (Complex &F, SimplexID nid, InsertIter iter)

  This is a helper function to assist neighbors to automatically deduce the integral level.
- template<class Complex , std::size_t level, class InsertIter >
  void casc::neighbors_up (Complex &F, typename Complex::template SimplexID< level > nid, InsertIter iter)

  Push the immediate coface neighbors into the provided iterator.
- template<class Complex , class SimplexID , class InsertIter >
  void casc::neighbors_up (Complex &F, SimplexID nid, InsertIter iter)

  This is a helper function to assist neighbors to automatically deduce the integral level.
- template<class Complex , std::size_t level, typename Iterator >
  void casc::kneighbors_up (Complex &F, int ring, std::set< typename Complex::template SimplexID< level > > &nbors, Iterator begin, Iterator end)

  Code for returning a set of k-ring neighbors.
- template<class Complex , class SimplexID >
  void casc::kneighbors_up (Complex &F, SimplexID nid, int ring, std::set< SimplexID > &nbors)

  Helper function to help kneighbors_up to deduce the integral level of SimplexID.
- template<class Complex , std::size_t level, typename Iterator >
  void casc::kneighbors (Complex &F, int ring, std::set< typename Complex::template SimplexID< level > > &nbors, Iterator begin, Iterator end)

  Code for returning a set of k-ring neighbors.
- template<class Complex , class SimplexID >
  void casc::kneighbors (Complex &F, SimplexID nid, int ring, std::set< SimplexID > &nbors)

  Helper function to help kneighbors to deduce the integral level of SimplexID.

10.3 include/casc/decimate.h File Reference

Meta-data aware decimation functions.

```
#include <typeinfo>
#include "SimplexSet.h"
#include "SimplexMap.h"
#include "CASCTraversals.h"
#include "CASCFunctions.h"
```

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<typename Complex >
  void casc::perform_removal (Complex &F, casc::SimplexSet< Complex > &S)

  Remove simplex in SimplexSet S from complex F.
- template<typename Complex >
  void casc::perform_insertion (Complex &F, typename decimation_detail::SimplexDataSet< Complex >::type &S)

  Insert all simplices in SimplexSet S into complex F
- template<typename Complex , template< typename > class Callback>
  void casc::run_user_callback (Complex &F, casc::SimplexMap< Complex > &S, Callback< Complex > &&clbk, typename decimation_detail::SimplexDataSet< Complex >::type &rv)

  Run the user specified callback function.
- template<typename Complex , typename Simplex , template< typename > class Callback>
  void casc::decimate (Complex &F, Simplex s, Callback< Complex > &&clbk)

  Decimate a simplex of any dimension while considering any meta-data stores on decimated simplices.
- template<typename Complex , typename Simplex >
  Complex::KeyType casc::decimateFirstHalf (Complex &F, Simplex s, SimplexMap< Complex > &simplex↩Map)

  Given a simplex to decimate generate a pre-post mapping.
- template<typename Complex >
  void casc::decimateBackHalf (Complex &F, SimplexMap< Complex > &simplexMap, typename decimation↩_detail::SimplexDataSet< Complex >::type &rv)

  Given a simplexMap and mapped resulting data execute the decimation.

10.4 include/casc/index_tracker.h File Reference

B-tree based interval tracker.

```
#include <iostream>
#include <assert.h>
#include <array>
#include <vector>
#include <cstdlib>
#include <limits>
```

Data Structures

- struct index_tracker::index_tracker_detail::Interval< T >

  Interval object represents a range.
- struct index_tracker::index_tracker_detail::BTreeNode< _T, _d >

  An array based BTree.
- class index_tracker::index_tracker< _T, _d >

  Tracker of available indices implemented as a B-tree of intervals.

Namespaces

- index_tracker

  Index tracker namespace.
- index_tracker::index_tracker_detail

  B-tree internal data structures.

Typedefs

- template<typename Node >
  using **index_tracker::index_tracker_detail::Pointer** = typename Node::Pointer
- template<typename Node >
  using **index_tracker::index_tracker_detail::Data** = typename Node::Data
- template<typename Node >
  using **index_tracker::index_tracker_detail::Scalar** = typename Node::Scalar

Functions

- template<typename T >
  bool **index_tracker::index_tracker_detail::operator<** (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator>** (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator<** (T x, const Interval< T > &y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator>** (const Interval< T > &x, T y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator<** (const Interval< T > &x, T y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator>** (T x, const Interval< T > &y)
- template<typename T >
  bool **index_tracker::index_tracker_detail::operator==** (const Interval< T > &x, const Interval< T > &y)
- template<typename T >
  std::ostream & **index_tracker::index_tracker_detail::operator<<** (std::ostream &out, const Interval< T > &x)
- template<typename T >
  int **index_tracker::index_tracker_detail::merge** (Interval< T > &A, T x)
- template<typename Node >
  void **index_tracker::index_tracker_detail::rebalance** (Pointer< Node > head, std::size_t i)
- template<typename Node >
  void **index_tracker::index_tracker_detail::insert_H** (Pointer< Node > head, const Data< Node > &data)

- template<typename Node >
  Pointer< Node > **index_tracker::index_tracker_detail::insert** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  bool **index_tracker::index_tracker_detail::get** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  void **index_tracker::index_tracker_detail::get_replacement** (Pointer< Node > head, Data< Node > &key)
- template<typename Node >
  void **index_tracker::index_tracker_detail::remove_H** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  Pointer< Node > **index_tracker::index_tracker_detail::remove** (Pointer< Node > head, Data< Node > data)
- template<typename Node >
  void **index_tracker::index_tracker_detail::fill_left** (Pointer< Node > head, Data< Node > &x)
- template<typename Node >
  void **index_tracker::index_tracker_detail::fill_right** (Pointer< Node > head, Data< Node > &x)
- template<typename Node >
  void **index_tracker::index_tracker_detail::insert_scalar_H** (Pointer< Node > head, Scalar< Node > data)
- template<typename Node >
  Pointer< Node > **index_tracker::index_tracker_detail::insert_scalar** (Pointer< Node > head, Scalar< Node > data)
- template<typename Node >
  void **index_tracker::index_tracker_detail::insert_left** (Pointer< Node > head, const Data< Node > &x)
- template<typename Node >
  bool **index_tracker::index_tracker_detail::remove_scalar_H** (Pointer< Node > head, Scalar< Node > x)
- template<typename Node >
  bool **index_tracker::index_tracker_detail::remove_scalar** (Pointer< Node > &head, Scalar< Node > data)
- template<typename Node >
  Scalar< Node > **index_tracker::index_tracker_detail::pop_scalar** (Pointer< Node > &head)
- template<typename Node >
  void **index_tracker::index_tracker_detail::destruct** (Pointer< Node > head)
- template<typename Node >
  Data< Node > **index_tracker::index_tracker_detail::check_order** (Pointer< Node > head, Data< Node > curr)
- template<typename T , std::size_t d>
  std::ostream & **index_tracker::operator**<< (std::ostream &out, const index_tracker_detail::BTreeNode< T, d > *head)

10.5 include/casc/Orientable.h File Reference

Data type for orientability.

```
#include <iostream>
#include <queue>
#include <set>
```

Data Structures

- struct casc::Orientable

  Class representing the orientation.

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<typename Complex >
  void casc::init_orientation (Complex &F)

  Initialize the partial ordering of the simplex edges.
- template<typename Complex >
  void casc::clear_orientation (Complex &F)

  Clear the orientation of the facets.
- template<typename Complex >
  std::tuple< int, bool, bool > casc::compute_orientation (Complex &F)

  Initializes and calculates the orientation of a simplicial_complex.
- template<typename Complex >
  std::tuple< int, bool, bool > casc::check_orientation (Complex &F)

  Checks for self consistent orientation and fill in missing orientations.

10.6 include/casc/SimplexMap.h File Reference

SimplexMap data structure and associated convenience functions.

```
#include <array>
#include <map>
#include "util.h"
#include "stringutil.h"
```

Data Structures

- struct casc::SimplexMap< Complex >

  A multimap to represent a map of simplex indices to a set of simplices.

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<std::size_t k, typename Complex >
  static auto & casc::get (SimplexMap< Complex > &S)

  Get the map for a simplex dimension.
- template<std::size_t k, typename Complex >
  static auto & casc::get (const SimplexMap< Complex > &S)

  This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

10.7 include/casc/SimplexSet.h File Reference

SimplexSet data structure and associated convenience functions.

```
#include <algorithm>
#include <unordered_set>
#include "util.h"
```

Data Structures

- struct casc::SimplexSet< Complex >

  A multiset to store simplices in a simplicial_complex.

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<std::size_t k, typename Complex >
  static auto & casc::get (SimplexSet< Complex > &S)

  Get the NodeSet for a simplex dimension from a SimplexSet.
- template<std::size_t k, typename Complex >
  static auto & casc::get (const SimplexSet< Complex > &S)
- template<typename Complex >
  bool casc::operator== (const SimplexSet< Complex > &lhs, const SimplexSet< Complex > &rhs)

  Compare if the sets are equivalent.
- template<typename Complex >
  bool casc::operator!= (const SimplexSet< Complex > &lhs, const SimplexSet< Complex > &rhs)

  Compare if the sets are not equivalent.
- template<typename Complex >
  static void casc::set_union (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

  Compute the set union.
- template<typename Complex >
  static void casc::set_intersection (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

  Compute the set intersection.
- template<typename Complex >
  static void casc::set_difference (const SimplexSet< Complex > &A, const SimplexSet< Complex > &B, SimplexSet< Complex > &dest)

  Compute the set difference.

10.8 include/casc/SimplicialComplex.h File Reference

This header contains the main CASC data structure and associated components.

```
#include <algorithm>
#include <assert.h>
#include <cstdint>
#include <map>
#include <set>
#include <iterator>
#include <array>
#include <vector>
#include <iostream>
#include <fstream>
#include <functional>
#include <type_traits>
#include <ostream>
#include <unordered_set>
#include <unordered_map>
#include <utility>
#include <stdexcept>
#include "index_tracker.h"
#include "util.h"
```

Data Structures

- class casc::simplicial_complex< traits >

  The CASC data structure for representing simplicial complexes of arbitrary dimensionality with coloring.
- struct casc::simplicial_complex< traits >::SimplexID< k >

  A handle for a simplex object in the complex.
- struct casc::simplicial_complex< traits >::EdgeID< k >

  External reference to an edge or a connection within the complex.

Namespaces

- casc

  Namespace for everything CASC.

Typedefs

- template<typename KeyType , typename ... Ts>
  using casc::AbstractSimplicialComplex = simplicial_complex< detail::simplicial_complex_traits_default< KeyType, Ts... > >
- template<typename T >
  using casc::NodeSet = std::unordered_set< T, simplex_set_detail::hashSimplexID< T > >

  Helpful alias defining a unordered_set of simplices. See also hashSimplexID.

10.9 include/casc/stringutil.h File Reference

String utilities for CASC.

```
#include <string>
```

Namespaces

- casc

  Namespace for everything CASC.

Functions

- template<typename T , std::size_t k>
  std::string casc::to_string (const std::array< T, k > &A)

  Returns a string representation of the vertex subsimplicies of a given simplex.

10.10 include/casc/typetraits.h File Reference

Helper functions for debugging template types.

Functions

- template<class T >
  CONSTEXPR14_TN static_string type_name ()

  Print the typename of an object at compile time.

10.10.1 Detailed Description

This is copied directly from this very helpful post from
Stackoverflow.

10.10.2 Function Documentation

10.10.2.1 type_name()

```
template<class T >
CONSTEXPR14_TN static_string type_name ( )
```

Example usage:

```
std::cout « "decltype(i) is " « type_name<decltype(i)>() « '\n';
```

10.11 include/casc/util.h File Reference

Metatemplate pack expansion helpers.

```
#include <utility>
#include <array>
```

Data Structures

- struct util::range< T >
  A range object to support range based for loops.
- struct util::type_holder< Ts >
  Queue based data structure to hold list of types.
- struct util::type_holder< T, Ts... >
  Partial specialization to allow FIFO access of typenames.
- struct util::type_get< k, T >
  Helper to get the kth element from a type_holder.
- struct util::type_get< 0, type_holder< Ts... > >
  Specialization for terminal case.
- struct util::type_get< k, type_holder< Ts... > >
  Specialization to recursively pop types to get the kth type.
- struct util::type_map< C, V >
  Map the types in C into `V<T>`.
- struct util::int_type_map< IntegerType, OutHolder, IntegerSequence, F >
  Maps an integer sequence and typename, F, into outholder.
- struct util::type_swap< TUPLE, HOLDER_FULL >
  Move a list of types from one container to another.
- struct util::type_swap< TUPLE, HOLDER< Ts... > >
  Move a list of types from one container to another.
- struct util::reverse_sequence< Integer, IntegerSequence >
  Reverse an Integer Sequence.
- struct util::remove_first_val< Integer, IntegerSequence >
  General template for removing the first value from a type holder.
- struct util::remove_first_val< Integer, InHolder< Integer, I, Is... > >
  Specialization for removing first integer from a sequence of compile time integers.

Namespaces

- util
  Metatemplate programming utilities namespace.

Functions

- template<typename T >
  range< T > util::make_range (T b, T e)
  Make a range object.
- template<typename T >
  range< T > util::make_range (std::pair< T, T > p)
  Makes a range object.
- template<class Integer , typename IntegerSequence , typename Fn , typename ... Args>
  void util::int_for_each (Fn &&f, Args &&... args)
  Calls a function `f.apply<k>()` for a sequence of integer k's.

Index