Computing in Topology

Seth Simon

September 12, 2017

1 Abstract

An interactive, console-based program that implements the incremental algorithm for calculating the Betti numbers of a simplicial complex was written. The program calculates the first three Betti numbers, and if any of the simplices have a dimension of at least three, the program will warn that β_2 may be inaccurate. The algorithm's complexity ranges from $O(n)$ to $O(n^2)$, depending on the order in which the simplices occur in the input file. The program, released under the GNU GPLv3, is written in C and can be compiled on any platform where gcc is available. The code is available in Appendix A and at `https://github.com/sethsimon/Faces`.

2 Introduction

Topology has many useful applications, such as geometric modeling, medical imaging, and sensor networks. Because of the large amount of data relevant to the problem, it is infeasible to calculate Betti numbers by hand. Thus efficient algorithms are needed.

In 1993, an algorithm called the *incremental algorithm* [1] was developed that improved the state of the art of computational topology. Although the incremental algorithm is only complete up to three dimensions, it is sufficient for many applications.

Section 3 will review some basic concepts from topology. Section 4 will discuss the algorithms used in the program. Section 5 will show an informal proof of the algorithm's correctness. Section 6 will discuss the algorithm's implementation. Section 7 will analyze the algorithm's complexity. Section 8 will discuss the program's shortcomings and future work.

3 Theory

A *topology* on a set X is a subset $\mathrm{T} \subseteq 2^X$ such that:

1. T contains the finite intersection of any of its members.

2. T contains the infinite union of any of its members.

3. T contains the empty set and X.

Because many topologies have an infinite number of points, they are infeasible by themselves for computation. However, any topology that can be derived from a Euclidean space can be triangulated, yielding a simplicial complex.

A *simplicial* complex K is a finite set of simplices such that:

1. K contains all of the faces of all of its members.

2. The intersection of any two members σ and τ of K are faces of both σ and τ.

A *k-simplex* is the convex hull of k+1 linearly independent points, each of which is called a vertex of the simplex. Its dimension is k. Given two simplices σ and τ, τ is a face of σ if the set of vertices of τ is a subset of those of σ. σ is a coface of τ if τ is a face of σ. These relationships are denoted by $\tau \leq \sigma$ and $\sigma \geq \tau$.

All of the algorithms discussed in this paper require a filtered complex as input. A *filtered complex* is a sequence of simplicial complexes such that $\emptyset = K^0 \subseteq K^1 \subseteq ... \subseteq K^M = K$, where $K^i \subseteq K^j$ indicates that the set of vertices of K^i is a subset of the set of vertices of K^j.

A *p-chain* c of a simplicial complex K is a linear combination of the simplices of K such that $c = \sum_{i=1}^{m} a_i * b_i$, where each a_i is either 0 or 1, and $b_1 ... b_m$ are the p-simplices of K. A p-boundary is always a (p-1)-chain. A *p-cycle* is a p-chain with an empty boundary. The Fundamental Law of Homology states that the boundary of a boundary is always empty. Thus all p-boundaries are p-cycles.

The Betti numbers capture the topology of a simplicial complex up to homology. The nth Betti number is the number of n-cycles that are not boundaries. Intuitively, the zeroeth Betti number, β_0, is the number of connected components, β_1 is the number of holes, and β_2 is the number of enclosed spaces, or voids. In the next section, we discuss an algorithm for computing these first three Betti numbers.

4 Algorithms

To find the first three Betti numbers, an incremental algorithm [1, 2] is used. If the newly added simplex, with dimension k, belongs to a k- cycle, then β_k is incremented. Otherwise, β_{k-1} is decremented.

To determine whether a k-simplex σ belongs to a k-cycle, each simplex has an id number. Let f be the set of (k-1)-faces of σ, let m be the minimum of the ids of the simplices in f, and let n be the maximum of the ids of the simplices in f. If $m == n$ or f is empty, then σ belongs to a k-cycle, and β_k is incremented.

Otherwise, σ does not belong to a k-cycle, and β_{k-1} is decremented. Furthermore, the id of every (k-1)-simplex that is connected to any simplex in f via any number of k-simplices is set to m, as shown in Figure 1. Whether σ

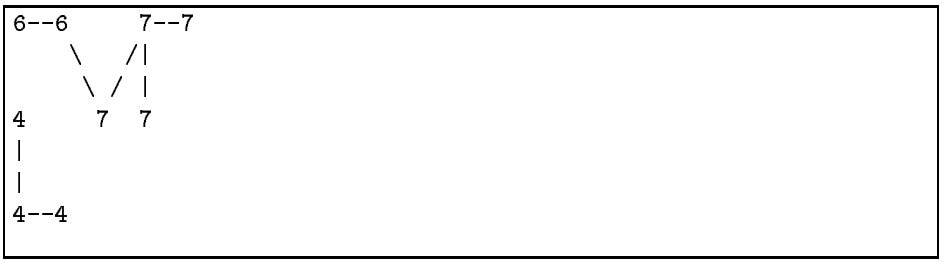

Figure 1: When encountering the edge denoted by two backslashes, the ids of the vertices with id 7 are changed to 6.

belongs to a k-cycle or not, its id is set to a number that's larger than every id in use.

In psuedo-code, this algorithm can be expressed as follows:

```
add_simplex(simp) {
    dim := dimension of simp
    // B[x] is the xth Betti number of the simplicial complex

    if(d > 2) {
        Warn that B[2] is inaccurate
        return
    }

    f := faces of simp with dimension d - 1
    if(f is empty or every element in f has the same id) {
        B[d]++
    } else {
        B[d - 1]--
        m := minimum of the ids in f
        for(each element q in f) {
            if(q.id != m) set_id(q, m)
        }
    }
    simp->id = next_unused_id++
}

set_id(simp, n) {
    d := dimension of simp
    simp->id = n
    for(each (d+1)-simplex c of simp) {
        for(each d-simplex f of c) {
            if(f->id != n) set_id(f, n)
        }
    }
}
```

5 Proof of Correctness

In this section, we give an informal argument that in terms of detecting 1-cycles, the psuedo-code in the previous section is equivalent to the incremental algorithm that's discussed in [1, 2]. The formal proof and an argument for correctness in terms of detecting 2-cycles are left as exercises for the reader.

The incremental algorithm can be stated in terms of three primitive operations: *ADD*, *FIND*, and *UNION*. *ADD(u)* adds a new component that consists of one component- u. *FIND(u)* returns the component that contains u. *UNION(A,B)* merges components A and B.

The incremental algorithm operates on a filtered complex in sequential order. When a vertex u is encountered, *ADD(u)* is called and β_0 is incremented. When an edge e connecting vertices v_1 and v_2 is added, $FIND(v_1) = FIND(v_2)$ if and only if e created a 1-cycle, in which case β_1 is incremented. Otherwise, $UNION(v_1, v_2)$ is called and β_0 is decremented.

In the psuedo-code, $FIND(x) = FIND(y)$ if and only if x and y have the same id. Adding a vertex creates a new component, indicated by a unique id, and increments β_0.

Given $m = min(FIND(v_1), FIND(v_2))$ and $x \in \{v_1, v_2 | FIND(x) \neq m\}$, $UNION(v_1, v_2)$ is equivalent to $set\_id(x, m)$. When $set\_id(x, m)$ is called, each vertex in the 1-skeleton containing x is visited exactly once. Adjacent vertices are accessed via x's cofaces' faces. Because no node's id in the 1-skeleton is m (if any node's id were m, all of them would be m and *set_id* would not need to be called), each vertex is visited at least once. Because each vertex's id is set to m immediately upon visiting it, no vertex is visited more than once (this prevents infinite recursion).

6 Implementation

The program, called *Faces*, is written in C and can run on any platform where *gcc* is available. *Faces* reads an input file containing a filtration, such as the following:

```
# A line with a # in column 0 is a comment,
# but end-of-line comments are NOT supported.

# Blank lines or lines with only whitespace are ignored.

# Some vertices
foo1
bar1
quux1

# Edges that connect every vertex to every vertex
e0 foo1 bar1
e1 bar1 quux1
```

```
e2 quux1 foo1

face e0 e1 e2
```

When the program begins execution, the input file is processed and the Betti numbers are calculated as discussed previously. Then, the program enters an interactive command loop until EOF is reached. A complete listing of the available commands is available in Appendix A.7.

Faces utilizes the following structs:

```
struct simplex {
    int nfaces;
    struct simplex **faces;

    int ncofaces;
    struct simplex **cofaces;

    char *id;

    struct simplex *next; // for the hash table

    // Preprocessing -> id number (see research paper)
    // Interactive   -> Whether it's been printed or not (showface.c)
    unsigned processed;
};

struct scomplex {
    int table_size;
    struct simplex **simplices;

    int max_dim; // used in showface.c

    int betti[3];
    int betti2_unreliable;
};
```

The dimension of a simplex can be found based on its number of faces: 0-simplices have no faces, 1-simplices have two faces, and 2-simplices have three faces (vertices are not counted). The simplices in the simplicial complex are stored in a hash table. The number of buckets is *table_size* and the hash table itself is *simplices*.

The *set_id* algorithm is implemented as follows:

```
static void set_id(struct simplex *simp, const unsigned val) {
    simp->processed = val;
    for(int coidx = 0; coidx < simp->ncofaces; coidx++) {
        struct simplex *cosimp = simp->cofaces[coidx];
```

```
        for(int faidx = 0; faidx < cosimp->nfaces; faidx++) {
            if(cosimp->faces[faidx]->processed != val) {
                set_id(cosimp->faces[faidx], val);
            }
        }
    }
}
```

7 Analysis

Although a full analysis of the running time is left as an exercise for the reader, we will show an example of how the best-case and worst-case scenarios can differ by as much as $O(n)$ and $O(n^2)$. Consider the simplicial complex, whose vertices' ids are labelled, shown below:

```
0   0   0   0   0
0   1   2   3   4
```

Suppose we were to add edges in order to create the following simplicial complex:

```
0---0---0---0---0
0   0   0   0   0
```

If the edges are added in order from left to right, four additional *set_id* operations are required, as the following diagram shows:

```
0---0   0   0   0
0   0   2   3   4

0---0---0   0   0
0   0   0   3   4

0---0---0---0   0
0   0   0   0   4

0---0---0---0---0
0   0   0   0   0
```

However, if the edges are added from right to left, ten additioonal *set_id* operations are required, as the following diagram shows:

```
0   0   0   0---0
0   1   2   3   3

0   0   0---0---0
0   1   2   2   2
```

```
0   0---0---0---0---0
0   1   1   1   1

0---0---0---0---0---0
0   0   0   0   0
```

In general this situation requires e operations when adding edges from left to right, and $\frac{(e(e+1))}{2}$ when adding edges from right to left, where e is the number of edges. To further illustrate this, two test files were creating with $e = 10000$ using the program in Appendix B. Each input file was timed by executing `upct faces.exe testfile < nul` [1] on a Compaq Presario with a 667MHz processor and 64MB of RAM running FreeDOS 1.2:

	Left-to-right	Right-to-left
Trial 1	0.40 seconds 476,071 cycles	41.65 seconds 121,286,420 cycles
Trial 2	0.36 seconds 423,713 cycles	101.93 seconds 121,626,389 cycles

8 Conclusion

This paper presents an example of the implementation of the incremental algorithm for calculating the first three Betti numbers of a simplicial complex. Because it is a console program with no external dependencies, it can be compiled on any platform where gcc is available. So far it has been tested on FreeDOS 1.2 (compiled with DJGPP [2]) and 32-bit Debian Jessie.

The program is far from complete. As the timing test in the analysis section showed, the worst-case performance is atrocious. The hotspots should be identified with a profiler and optimized. It may be worthwhile to sort the input file beforehand or investigate alternative data structures and/or algorithms. A DOS-specific bug is that redirecting output (e.g. `!echo foo > foo.txt`) creates an empty file. Also, the size of the hash table is based on the size of the input file when it should be based on the number of simplices in the input file, as in `grep -v "^#" foo | wc -l`.

We were unable to implement a feature where simplices can be added and removed dynamically- that is, without quitting *Faces*, editing the input file, and rerunning it. We had hoped that the Betti numbers would not have to be recalculated from scratch after a dynamic addition or removal, but implementing that proved to be more challenging than expected, especially when accounting for error handling.

[1] Ultra Precision Command Timer 1.6 by Erik de Neve (1993) is available via anonymous ftp at `ftp.sac.sk/pub/sac/utilmisc/upct16.zip`.

[2] DJGPP is a set of software, including ports of `gcc`, for developing 32-bit programs that run in protected mode under DOS. DJGPP was created and is maintained by DJ Delorie (`http://www.delorie.com`).

The error checking code is in need of bugfixes and refactoring. If two different simplices have identical faces, the error is ignored and erroneous results are produced. Lines in the input file should be able to be arbitrarily long rather than limited to 127 characters. If any line exceeds this limit, incorrect results may be silently produced or erroneous errors may be raised. From a performance standpoint, it may be beneficial to add a flag that causes error checking to be skipped (for advanced users only).

References

[1] C. J. A. Delfinado and H. Edelsbrunner. "An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere." Computer Aided Geometric Design, 12:771-784, 1995.

[2] Zomorodian, Afra J. Topology for Computing. New York: Cambridge University Press, 2005.

9 Appendix A.0: Code Listing (COPYING)

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                              Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, 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
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.
```

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users

can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a

copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or

specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same

material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and

propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

  If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

  A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If 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 convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may

not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "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 PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS

THE PROGRAM 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 PROGRAM (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 PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state 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 program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program 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 General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

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

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate

parts of the General Public License. Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.

10 Appendix A.1: Code Listing (main.c)

```
/**
* This program (Faces), is an interactive program that calculates
* the first three Betti numbers.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "scomplex.h"
#include "command.h"

#include <stdio.h>
#include <string.h>
#include <sys/stat.h>

static void usage(FILE *f) {
    fprintf(f, "Usage: faces <file>\n\n"
               "Each line of the file (max 127 chars) is formatted "
               "as follows:\n"
               "<id> <face1> <face2> ... <facen>\n"
               "\nExamples:\n"
               "'v0' declares a vertex named v0\n"
               "'foo bar quux' declares an edge whose vertices "
```

```
                "are bar and quux\n"
                "'f1 e0 e1 e2' is a face whose edges are "
                "e0, e1, and e2\n"
                "\n"
                "A line beginning with '#' is a comment.\n"
                "End of line comments are NOT supported.\n");
}

int main(int argc, char **argv) {
    printf("Faces: Copyright 2017 Seth Simon (s.r.simon@csuohio.edu)\n"
           "This program comes with ABSOLUTELY NO WARRANTY; for "
           "details, see the license.\nThis is free software, and "
           "you are welcome to redistribute it\nunder certain "
           "conditions; "
           "see the license for details.\nYou should have received "
           "a copy "
           "of the GNU General Public License\n(version 3) along "
           "with this program. If not, see "
           "<http://www.gnu.org/licenses/>.\n\n");

    if(argc != 2) {
        usage(stderr);
        return 1;
    } else if(!strcmp(argv[1], "?") || !strcmp(argv[1], "-h") ||
              !strcmp(argv[1], "--help") || !strcmp(argv[1], "/?") ||
              !strcmp(argv[1], "/h") || !strcmp(argv[1], "/help")) {
        usage(stdout);
        return 0;
    }

    int ret = 1;
    struct stat st;
    FILE *file;
    if(stat(argv[1], &st) || !(file = fopen(argv[1], "r"))) {
        fprintf(stderr, "Failed to open '%s' for reading\n", argv[1]);
        return 1;
    }
    struct scomplex scomplex = SCOMPLEX_DEFAULTS;
    if(init_scomplex(&scomplex, st.st_size)) goto done;

    char line[128];
    for(int lineno = 1; fgets(line, 128, file); lineno++) {
        if(process_line(&scomplex, line, lineno)) goto done;
    }
    get_betti(scomplex.betti, &scomplex.betti2_unreliable);
    fclose(file);
    file = NULL;

    printf("Type ? for help, CTRL-D (UNIX) or CTRL-Z + ENTER (DOS) "
           "to quit.\n\n? ");
```

```
    char cmd[128];
    while(fgets(cmd, 128, stdin)) {
        do_command(&scomplex, cmd);
        printf("\n? ");
    }

    ret = 0;
done:
    if(file) fclose(file);
    free_scomplex(&scomplex);
    return ret;
}
```

11 Appendix A.2: Code Listing (simplex.h)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef SIMPLEX_H
#define SIMPLEX_H

#include <stdio.h>

#define SIMPLEX_DEFAULTS (struct simplex) {\
        .nfaces = 0,\
        .faces = NULL,\
        .ncofaces = 0,\
        .cofaces = NULL,\
        .id = NULL,\
        .next = NULL,\
        .processed = 0\
    }
struct simplex {
```

```
    int nfaces;
    struct simplex **faces;

    int ncofaces;
    struct simplex **cofaces;

    char *id;

    struct simplex *next; // for the hash table

    // Preprocessing -> id number (see research paper)
    // Interactive   -> Whether it's been printed or not (showface.c)
    unsigned processed;
};

#define DIMENSION(simp) (((simp)->nfaces) ? ((simp)->nfaces - 1) : 0)

void free_simplex(struct simplex *simp);
int insert_face(struct simplex *simp, struct simplex *face);
int insert_coface(struct simplex *simp, struct simplex *coface);

void reset_processed_flags(struct simplex *simp, const int co);

void add_betti(struct simplex *simp);
void get_betti(int *b, int *unreliable);

#endif
```

12 Appendix A.3: Code Listing (simplex.c)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "simplex.h"
```

```
#include <stdlib.h>
#include <string.h>
#include <limits.h>

#define FACES_AT_ONCE 4
#define PTRSIZE (sizeof(void *))

void free_simplex(struct simplex *simp) {
    free(simp->faces);
    free(simp->cofaces);
    free(simp->id);
    free(simp);
}

int insert_face(struct simplex *simp, struct simplex *face) {
    if(!simp->faces) {
        simp->faces = malloc(FACES_AT_ONCE * PTRSIZE);
        if(!simp->faces) return 1;
    } else if(simp->nfaces % FACES_AT_ONCE == 0) {
        void *tmp = realloc(simp->faces, PTRSIZE * simp->nfaces * 2);
        if(!tmp) return 1;
        simp->faces = tmp;
    }
    simp->faces[simp->nfaces++] = face;
    return 0;
}

int insert_coface(struct simplex *simp, struct simplex *coface) {
    if(!simp->cofaces) {
        simp->cofaces = malloc(FACES_AT_ONCE * PTRSIZE);
        if(!simp->cofaces) return 1;
    } else if(simp->ncofaces % FACES_AT_ONCE == 0) {
        void *tmp =
            realloc(simp->cofaces, PTRSIZE * simp->ncofaces * 2);
        if(!tmp) return 1;
        simp->cofaces = tmp;
    }
    simp->cofaces[simp->ncofaces++] = coface;
    return 0;
}

void reset_processed_flags(struct simplex *simp, const int co) {
    simp->processed = 0;
    const int count = co ? simp->ncofaces : simp->nfaces;
    struct simplex **arr = co ? simp->cofaces : simp->faces;
    for(int i = 0; i < count; i++) {
        reset_processed_flags(arr[i], co);
    }
}
```

```
// ====================
//      Betti
// ====================

/**
 * Sets all of the n-simplices connected to simp via an
 * (n+1)-simplex to val (where n == DIMENSION(simp))
*/
static void set_id(struct simplex *simp, const unsigned val) {
    simp->processed = val;
    for(int coidx = 0; coidx < simp->ncofaces; coidx++) {
        struct simplex *cosimp = simp->cofaces[coidx];
        for(int faidx = 0; faidx < cosimp->nfaces; faidx++) {
            if(cosimp->faces[faidx]->processed != val) {
                set_id(cosimp->faces[faidx], val);
            }
        }
    }
}

static int betti[3] = {0, 0, 0};
static int betti2_unreliable = 0;
static unsigned next_free_id = 0;

static int faces_have_same_id(const struct simplex *simp) {
    if(!simp->nfaces) return 1;

    const unsigned id = simp->faces[0]->processed;
    for(int i = 1; i < simp->nfaces; i++) {
        if(simp->faces[i]->processed != id) return 0;
    }
    return 1;
}

static unsigned min_face_id(const struct simplex *simp) {
    unsigned ret = UINT_MAX;
    for(int i = 0; i < simp->nfaces; i++) {
        if(simp->faces[i]->processed < ret) {
            ret = simp->faces[i]->processed;
        }
    }
    return ret;
}

// TODO: Don't use processed because of [add]/[remove] commands
void add_betti(struct simplex *simp) {
    if(DIMENSION(simp) > 2) {
        betti2_unreliable = 1;
        return;
```

```
    }

    if(faces_have_same_id(simp)) {
        betti[DIMENSION(simp)]++;
    } else {
        betti[DIMENSION(simp) - 1]--;
        const unsigned min = min_face_id(simp);
        for(int i = 0; i < simp->nfaces; i++) {
            if(simp->faces[i]->processed != min) {
                set_id(simp->faces[i], min);
            }
        }
    }
    simp->processed = next_free_id++;
}

void get_betti(int *b, int *unreliable) {
    b[0] = betti[0];
    b[1] = betti[1];
    b[2] = betti[2];
    *unreliable = betti2_unreliable;
}
```

13 Appendix A.4: Code Listing (scomplex.h)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef SCOMPLEX_H
#define SCOMPLEX_H

#include "simplex.h"

#include <stdio.h>
```

```
#define SCOMPLEX_DEFAULTS (struct scomplex) {\
        .table_size = 0,\
        .simplices = NULL,\
        \
        .max_dim = 0,\
        \
        .betti = { 0, 0, 0 },\
        .betti2_unreliable = 0\
    }
struct scomplex {
    int table_size;
    struct simplex **simplices;

    int max_dim; // used in showface.c

    int betti[3];
    int betti2_unreliable;
};

int init_scomplex(struct scomplex *scomplex, const size_t fsize);
int process_line(struct scomplex *scomplex, char *line,
                 const int lineno);
void free_scomplex(struct scomplex *scomplex);

struct simplex *get_simplex(struct scomplex *scomplex,
                            const char *id);

#endif
```

14 Appendix A.5: Code Listing (scomplex.c)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
```

```
#include "scomplex.h"

#include <stdlib.h>
#include <string.h>

#define PTRSIZE (sizeof(void *))

void free_scomplex(struct scomplex *scomplex) {
    for(int i = 0; i < scomplex->table_size; i++) {
        struct simplex *simp = scomplex->simplices[i];
        while(simp) {
            struct simplex *next = simp->next;
            free_simplex(simp);
            simp = next;
        }
    }
}

static unsigned calc_hash(struct scomplex *scomplex, const char *id) {
    // djb2 hashing algorithm by Dan Bernstein
    // See http://www.cse.yorku.ca/~oz/hash.html
    unsigned ret = 5381;
    for( ; *id; id++) {
        ret = 33 * ret + *id;
    }
    return ret % scomplex->table_size;
}

struct simplex *get_simplex(struct scomplex *scomplex,
                            const char *id) {
    const unsigned hash = calc_hash(scomplex, id);
    struct simplex *ret = scomplex->simplices[hash];

    while(ret && strcmp(ret->id, id)) {
        ret = ret->next;
    }
    return ret;
}

static struct simplex *add_simplex(struct scomplex *scomplex,
                                   const char *id) {
    struct simplex *ret = malloc(sizeof(struct simplex));
    if(!ret) return NULL;
    *ret = SIMPLEX_DEFAULTS;

    ret->id = malloc(strlen(id) + 1);
    if(!ret->id) {
        free(ret);
        return NULL;
```

```
    }
    strcpy(ret->id, id);

    const unsigned hash = calc_hash(scomplex, id);
    if(!scomplex->simplices[hash]) {
        scomplex->simplices[hash] = ret;
    } else {
        struct simplex *cur = scomplex->simplices[hash];
        while(cur->next) cur = cur->next;
        cur->next = ret;
    }
    return ret;
}

int init_scomplex(struct scomplex *scomplex, const size_t fsize) {
    // Assume ~10 chars/simplex, aim for load factor = 0.5
    // TODO: grep -v "^#" foo | wc -l would be better
    scomplex->table_size = fsize / 5 + 1;

    scomplex->simplices = malloc(scomplex->table_size * PTRSIZE);
    if(!scomplex->simplices) {
        fprintf(stderr, "Malloc failed in init_scomplex\n");
        return 1;
    }

    memset(scomplex->simplices, 0, scomplex->table_size * PTRSIZE);
    return 0;
}

int process_line(struct scomplex *scomplex, char *line,
                 const int lineno) {
    // \n can be included in the line
    char *token = strtok(line, " \r\t\n");
    if(!token || !*token || *token == '#') return 0;

    if(get_simplex(scomplex, token)) {
        fprintf(stderr, "Line %d: Duplicate id '%s'\n", lineno,
                token);
        return 1;
    }

    struct simplex *const simp = add_simplex(scomplex, token);
    if(!simp) {
        fprintf(stderr, "Line %d: Malloc failed\n", lineno);
        return 1;
    }

    while((token = strtok(NULL, " \r\t\n"))) {
        struct simplex *const face = get_simplex(scomplex, token);
        if(!face) {
```

```
            fprintf(stderr, "Line %d: Couldn't find a simplex with "
                    "id '%s'\n", lineno, token);
            return 1;
        }
        if(insert_face(simp, face) || insert_coface(face, simp)) {
            fprintf(stderr, "Line %d: Malloc failed\n", lineno);
            return 1;
        }
    }

    if(simp->nfaces == 1) {
        fprintf(stderr, "Line %d: Malformed face with exactly one "
                "simplex\n", lineno);
        return 1;
    }
    for(int i = 0; i < simp->nfaces; i++) {
        if(DIMENSION(simp->faces[i]) + 1 != DIMENSION(simp)) {
            fprintf(stderr, "Line %d: Since %s has %d faces, %s "
                    "must have dimension %d, not %d\n", lineno,
                    simp->id, simp->nfaces, simp->faces[i]->id,
                    DIMENSION(simp) - 1, DIMENSION(simp->faces[i]));
            return 1;
        }
    }

    if(DIMENSION(simp) > scomplex->max_dim) {
        scomplex->max_dim = DIMENSION(simp);
    }
    add_betti(simp);
    return 0;
}
```

15 Appendix A.6: Code Listing (command.h)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
```

```
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef COMMAND_H
#define COMMAND_H

#include "scomplex.h"

void do_command(struct scomplex *scomplex, char *cmd);
void command_help();

#endif
```

16 Appendix A.7: Code Listing (command.c)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "command.h"
#include "showface.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h> // INT_MIN and INT_MAX

void command_help() {
    printf("faces <id> [mindim] [maxdim]\n"
           "    Show id's faces with a dimension of at least "
           "mindim and at most maxdim\n"
           "cofaces <id> [mindim] [maxdim]\n"
           "    Show id's cofaces with a dimension of at least "
           "mindim and at most maxdim\n"
           "betti [n]\n"
```

```
            "    Show the Nth betti number, or the first 3 if n is "
            "omitted\n"
            "dimension [id1] [id2] ... [idn]\n"
            "    Show the dimension(s) of some simplices\n"
            "hash\n"
            "    Show the hash table's statistics\n"
            "!<cmd>\n"
            "    Execute a shell command\n"
            "CTRL-D (UNIX) or CTRL-Z + ENTER (DOS)\n"
            "    Quit\n"
            "help, ?\n"
            "    Show this message\n");
}

static void show_hash_statistics(struct scomplex *scomplex) {
    printf("%d buckets\n", scomplex->table_size);

    int occupants = 0;
    int collisions = 0;
    for(int i = 0; i < scomplex->table_size; i++) {
        struct simplex *cur = scomplex->simplices[i];
        if(cur) {
            occupants++;
            cur = cur->next;
        }
        while(cur) {
            collisions++;
            cur = cur->next;
        }
    }

    printf("%d occupants\n", occupants);
    printf("Load factor = %.2f\n",
           occupants / (float)scomplex->table_size);
    printf("%d collisions\n", collisions);
}

static void show_betti(struct scomplex *scomplex, int n) {
    if(n == INT_MAX) {
        printf("%d %d %d\n", scomplex->betti[0], scomplex->betti[1],
               scomplex->betti[2]);
    } else {
        printf("%d\n", scomplex->betti[n]);
    }

    if(scomplex->betti2_unreliable && (n == INT_MAX || n == 2)) {
        fprintf(stderr, "Warning: Betti2 is unreliable\n");
    }
}
```

```
static int garbage_at_end() {
    if(strtok(NULL, " \n")) {
        fprintf(stderr, "Error: too many arguments\n");
        return 1;
    }
    return 0;
}

static int get_num(int *num, char **token) {
    char *tok = strtok(NULL, " \n");
    char *unconverted = NULL;
    if(tok) {
        *num = strtol(tok, &unconverted, 10);
        if(unconverted && *unconverted) {
            fprintf(stderr, "%s is not a number\n", tok);
            if(token) *token = tok;
            return 1;
        }
    }
    if(token) *token = tok;
    return 0;
}

void do_command(struct scomplex *scomplex, char *cmd) {
    if(*cmd == '!') {
        // TODO: Redirection (>) doesn't work, it just creates
        // an empty file!
        system(cmd + 1);
        return;
    }

    char *token = strtok(cmd, " \n");
    if(!token || !*token) return;

    if(!strcmp(token, "faces") || !strcmp(token, "cofaces")) {
        const int faces = !strcmp(token, "faces");

        int min = INT_MIN;
        int max = INT_MAX;
        char *id = strtok(NULL, " \n");
        if(!id) {
            fprintf(stderr, "Missing id\n"); return;
        }

        if(get_num(&min, NULL)) return;
        if(get_num(&max, NULL)) return;

        if(min > max) {
            printf("The minimum of %d cannot be bigger than "
                   "the maximum of %d\n", min, max);
```

```
        } else if(!garbage_at_end()) {
            if(faces) show_faces(scomplex, id, min, max);
            else show_cofaces(scomplex, id, min, max);
        }
    } else if(!strcmp(token, "help") || !strcmp(token, "?")) {
        if(!garbage_at_end()) command_help();
    } else if(!strcmp(token, "hash")) {
        if(!garbage_at_end()) show_hash_statistics(scomplex);
    } else if(!strcmp(token, "betti")) {
        int n;
        if(get_num(&n, &token)) return;
        if(token && (n < 0 || n > 2)) {
            fprintf(stderr, "Only Betti0 through Betti2 are "
                    "supported\n");
            return;
        }
        if(garbage_at_end()) return;
        show_betti(scomplex, token ? n : INT_MAX);
    } else if(!strcmp(token, "dimension")) {
        while((token = strtok(NULL, " \n"))) {
            struct simplex *s = get_simplex(scomplex, token);
            if(s) printf("%d\n", DIMENSION(s));
            else fprintf(stderr, "No simplex named '%s'\n", token);
        }
    } else {
        fprintf(stderr, "Unknown command '%s', type '?' for "
                "help\n", token);
    }
}
```

17 Appendix A.8: Code Listing (showface.h)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
```

```
#ifndef SHOWFACE_H
#define SHOWFACE_H

#include "scomplex.h"

void show_faces(struct scomplex *scomplex, char *id, int mindim,
                int maxdim);
void show_cofaces(struct scomplex *scomplex, char *id, int mindim,
                  int maxdim);

#endif
```

18 Appendix A.9: Code Listing (showface.c)

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "showface.h"

#include <stdio.h>
#include <string.h>

#define PRINT_HEADER printf("Simplex    Dimension\n" \
                            "====================\n")
#define PRINT_SIMPLEX(simp) printf("%-10s %d\n", simp->id,\
                            DIMENSION(simp))

static void show_faces_internal(struct simplex *simp, int dim) {
    if(DIMENSION(simp) == dim) {
        if(!simp->processed) {
            simp->processed = 1;
            PRINT_SIMPLEX(simp);
        }
```

```
        return;
    }
    for(int idx = 0; idx < simp->nfaces; idx++) {
        show_faces_internal(simp->faces[idx], dim);
    }
}

void show_faces(struct scomplex *scomplex, char *id, int mindim,
                int maxdim) {
    struct simplex *simp = get_simplex(scomplex, id);
    if(!simp) {
        fprintf(stderr, "No simplices have id '%s'\n", id);
        return;
    }
    reset_processed_flags(simp, 0);
    PRINT_HEADER;

    if(maxdim > DIMENSION(simp)) maxdim = DIMENSION(simp);
    if(mindim < 0) mindim = 0;
    for(int i = mindim; i <= maxdim; i++) {
        show_faces_internal(simp, i);
    }
}

static void show_cofaces_internal(struct simplex *simp, int dim) {
    if(DIMENSION(simp) == dim) {
        if(!simp->processed) {
            simp->processed = 1;
            PRINT_SIMPLEX(simp);
        }
        return;
    }
    for(int idx = 0; idx < simp->ncofaces; idx++) {
        show_cofaces_internal(simp->cofaces[idx], dim);
    }
}

void show_cofaces(struct scomplex *scomplex, char *id,
                  int mindim, int maxdim) {
    struct simplex *simp = get_simplex(scomplex, id);
    if(!simp) {
        fprintf(stderr, "No simplices have id '%s'\n", id);
        return;
    }
    reset_processed_flags(simp, 1);
    PRINT_HEADER;

    if(mindim < DIMENSION(simp)) mindim = DIMENSION(simp);
    if(maxdim > scomplex->max_dim) maxdim = scomplex->max_dim;
    for(int i = mindim; i <= maxdim; i++) {
```

```
        show_cofaces_internal(simp, i);
    }
}
```

19 Appendix B: Code that Creates a File that Illustrates the Best and Worst Case Running Time

```
/**
* This file is part of Faces.
* Copyright (C) 2017 Seth Simon (s.r.simon@csuohio.edu)
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program 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 General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include <stdlib.h>

/**
 * runtime.c
 *
 * Creates two test files for analyzing the running time
 * of the algorithm in different scenarios.
*/

int main(int argc, char **argv) {
    if(argc != 4) {
        printf("Usage: runtime.exe <# vertices> <left-to-right-file> "
               "<right-to-left-file>\n");
        return 1;
    }

    char *unconverted = NULL;
    const int nvertices = (int)strtol(argv[1], &unconverted, 10);
    if((unconverted && *unconverted) || nvertices <= 0) {
        printf("Invalid vertex count\n");
```

```
        return 1;
    }

    FILE *ltr = fopen(argv[2], "w");
    FILE *rtl = fopen(argv[3], "w");
    if(!ltr || !rtl) {
        printf("Failed to open output file\n");
        if(rtl) fclose(rtl);
        return 1;
    }

    fprintf(ltr, "# This is the left-to-right file created by "
            "runtime.c\n");
    fprintf(rtl, "# This is the right-to-left file created by "
            "runtime.c\n");

    for(int i = 0; i < nvertices; i++) {
        fprintf(ltr, "v%d\n", i);
        fprintf(rtl, "v%d\n", i);
    }

    for(int i = 0; i < nvertices - 1; i++) {
        fprintf(ltr, "e%d v%d v%d\n", i, i, i + 1);
        fprintf(rtl, "e%d v%d v%d\n", i, nvertices - 1 - i,
                nvertices - 2 - i);
    }

    fclose(rtl);
    fclose(ltr);
    return 0;
}
```