COMPUTATIONAL TOPOLOGY WITH REGINA: ALGORITHMS, HEURISTICS AND IMPLEMENTATIONS

BENJAMIN A. BURTON

Dedicated to J. Hyam Rubinstein

Abstract. Regina is a software package for studying 3-manifold triangulations and normal surfaces. It includes a graphical user interface and Python bindings, and also supports angle structures, census enumeration, combinatorial recognition of triangulations, and high-level functions such as 3-sphere recognition, unknot recognition and connected sum decomposition.

This paper brings 3-manifold topologists up-to-date with *Regina* as it appears today, and documents for the first time in the literature some of the key algorithms, heuristics and implementations that are central to *Regina*'s performance. These include the all-important simplification heuristics, key choices of data structures and algorithms to alleviate bottlenecks in normal surface enumeration, modern implementations of 3-sphere recognition and connected sum decomposition, and more. We also give some historical background for the project, including the key role played by Rubinstein in its genesis 15 years ago, and discuss current directions for future development.

1. Introduction

Algorithmic problems run to the heart of low-dimensional topology. Prominent amongst these are *decision problems* (e.g., recognising the unknot, or testing whether two triangulated manifolds are homeomorphic); *decomposition problems* (e.g., decomposing a triangulated manifold into a connected sum of prime manifolds); and *recognition problems* (e.g., "naming" the manifold described by a given triangulation).

For 3-manifolds in particular, such problems typically have highly complex and expensive solutions—running times are often exponential or super-exponential, and implementations are often major endeavours developed over many years (if they exist at all). This is in contrast to dimension two, where many such problems are easily solved in small polynomial time, and dimensions ≥ 4, where such problems can become provably undecidable [21, 40].

Regina [5, 14] is a software package for low-dimensional topologists and knot theorists. Its main focus is on triangulated 3-manifolds, though it is now branching into both two and four dimensions also (see Section 7). It provides powerful

2000 *Mathematics Subject Classification.* Primary 57-04, 57N10; Secondary 68W05, 57Q15.

Key words and phrases. 3-manifolds, algorithms, software, simplification, normal surfaces, recognition, angle structures.

The author is supported by the Australian Research Council under the Discovery Projects funding scheme (projects DP1094516 and DP110101104).

algorithms and heuristics to assist with decision, decomposition and recognition problems; more broadly, it includes a range of facilities to study and manipulate 3-manifold triangulations. It offers rich support for normal surface theory, a major algorithmic framework that recurs throughout 3-manifold topology.

One of Regina's most important uses is for *experimentation*: it can analyse a single example too large to study by hand, or millions of examples "in bulk". Other uses include *computer proofs*, such as the recent resolution of Thurston's old question of whether the Weber-Seifert dodecahedral space is non-Haken [18], and *calculation* of previously-unknown topological invariants, such as recent improvements to crosscap numbers in knot tables [13, 15].

The remainder of this introduction gives a brief overview of Regina and some history behind its early development, back to its genesis with Rubinstein 15 years ago. Following this, we select some of Regina's most important features and study the modern heuristics, algorithms and implementations behind them. Some of these details are crucial for Regina's fast performance, but have not appeared in the literature to date. Overall, this paper offers a fresh update on the significant progress and performance enhancements in the eight years since Regina's last major report in the literature [5].

The specific features that we focus on are:

- the process of simplifying triangulations, which plays a critical role in many of Regina's high-level algorithms;
- the enumeration of normal and almost normal surfaces, including the high-level 0-efficiency, 3-sphere recognition and connected sum decomposition algorithms;
- combinatorial recognition, whereby Regina attempts to recognise a triangulation by studying its combinatorial structure;
- the enumeration of angle structures, taut angle structures and veering structures on triangulations.

With respect to these features, there are interesting new developments in the pipeline. Examples include "breadth-first simplification" of triangulations, the tree traversal algorithm for enumerating vertex normal surfaces, and new 0-efficiency and 3-sphere recognition algorithms based on linear programming that exhibit experimental polynomial-time behaviour. The corresponding code is already written (it is now being tested and prepared for integration into Regina), and we outline these new developments in the relevant sections of this paper.

To finish, Section 6 gives concrete examples of how Regina can be used for experimentation, including an introduction to Regina's in-built Python scripting, so that interested readers can use these as launching points for their own experiments. Section 7 concludes with additional future directions for Regina, including new work with Ryan Budney on triangulated *4-manifolds*, much of which is already coded and running in the development repository.

1.1. **Overview of Regina.** Regina is now 13 years old, with over 190 000 lines of source code. It is released under the GNU General Public License, and contributions from the research community are welcome. It adheres to the following broad development principles, in order of precedence:

(1) *Correctness:* Having correct output is critical, particularly for applications such as computer proofs. Regina is extremely conservative in this regard:

for example, it will use arbitrary precision integer arithmetic if it cannot be proven unnecessary, and the API documentation makes thorough use of preconditions and postconditions.

(2) *Generality:* Algorithms operate in the broadest possible scenarios (within reason), and do not require preconditions that cannot be easily tested. For instance, unknot recognition runs correctly for both bounded and ideal triangulations, and even when the input triangulation is not known to be a knot complement (whereupon it generalises to solid torus recognition).

(3) *Speed:* Because many of its algorithms run in exponential or super-exponential time, speed is imperative. Regina makes use of sophisticated algorithms and heuristics that, whilst adhering to the constraints of correctness and generality, make it practical for real topological problems.

Regina is multi-platform, and offers a drag-and-drop installer for MacOS, an MSI-based installer for Windows, and ready-made packages for several GNU/Linux distributions. It is thoroughly documented, and stores its data files in a compressed XML format. Regina provides three levels of user interface:

- a full graphical user interface, based on the Qt framework [48];
- a scripting interface based on Python, which can interact with the graphical interface or be used a standalone Python module;
- a programmers' interface offering native access to Regina's mathematical core through a C++ shared library.

There are facilities to help new users learn their way around, including an illustrated users' handbook, context-sensitive "what's this?" help, and sample data files that can be opened through the *File* $\longrightarrow$ *Open Example* menu.

Regina's core strengths are in working with triangulations, normal surfaces and angle structures. It only offers basic support for hyperbolic geometries, for which the software packages SnapPea [60] and its successor SnapPy [23] are more suitable. In addition to its range of low-level operations and 3-manifold invariants, Regina includes implementations of high-level decision and decomposition algorithms, including the only known full implementations of 3-sphere recognition and connected sum decomposition. For a comprehensive list of features, see `http://regina.sourceforge.net/docs/featureset.html`.

1.2. **History.** In the 1990s, Jeff Weeks' software package SnapPea had opened up enormous possibilities for computer experimentation with hyperbolic 3-manifolds. In contrast, outside the hyperbolic world many fundamental 3-manifold algorithms remained purely theoretical, including high-profile algorithms such as 3-sphere recognition, connected sum decomposition, Haken's unknot recognition algorithm, and testing for incompressible surfaces.

More broadly, *normal surface theory*—a ubiquitous technology in algorithmic 3-manifold topology—had no concrete software implementation (or at least none that had been publicised). Many of the algorithms that used normal surfaces were so complex that it was believed that any attempt at implementation would be both painstaking to develop and impractically slow to run.

Nevertheless, Rubinstein sought to move these algorithms from the theoretical realm to the practical, and in 1997 he assembled a team for a new software project that would make *computational* normal surface theory a reality. This initial team consisted of David Letscher, Richard Rannard, and myself. Much planning was

done and a little preliminary code was written, but then the team members became occupied with different projects.

Letscher later resumed the project on his own, and at a 1999 workshop on computational 3-manifold topology at Oklahoma State University he presented *Normal*, a Java-based application that could modify and simplify triangulations, and enumerate vertex normal surfaces. This was an exciting development, but the software was proof-of-concept only, and later that month Letscher and I sat down to begin again from scratch. The successor would be a fast and robust software package with a C++ engine, would be designed for extensibility and portability, and would support community-contributed "add-ons".

Letscher stayed with the project for the first year, and then moved on to pursue different endeavours. I remained with the project (a PhD student at the time, with more time on my hands), and the project—now known as *Regina*—saw its first public release in December 2000. Development at this stage was taking place at Oklahoma State University, where Bus Jaco was both a strong supporter and a significant influence on the project in its formative years.

Around that time, a significant phase shift was taking place in normal surface theory: Jaco and Rubinstein were developing their theory of *0-efficient triangulations*. Their landmark 2003 paper [33] showed that key problems such as 3-sphere recognition and connected sum decomposition could be solved without the expensive and highly problematic operation of *cutting* along a normal surface, and that instead this could be replaced by a *crushing* operation that always reduces the number of tetrahedra. For the first time, the key algorithms of 3-sphere recognition and connected sum decomposition became both feasible and practical to implement. Shortly after, in March 2004, Regina became the first software package to implement robust, conclusive solutions to these fundamental problems.

Regina has come a long way since development began in 1999. It continues to grow in its mathematics, performance, usability, and technical infrastructure, and it enjoys contributions from a wide range of people (as noted in the following section). For a list of highlights over the years, the reader is invited to peruse the "abbreviated changelog" at `http://regina.sourceforge.net/docs/history.html`.

1.3. **Acknowledgements.** Regina is a mature project and has benefited from the expertise of many people and the resources of many organisations.

Ryan Budney and William Pettersson have both made significant and ongoing contributions, and are both now primary developers for the project. A host of other people have contributed either code or expertise, including Bernard Blackham, Marc Culler, Dominique Devriese, Nathan Dunfield, Matthias Goerner, William Jaco, David Letscher, Craig Macintyre, Melih Ozlen, Hyam Rubinstein, Jonathan Shewchuk, and Stephan Tillmann. In addition, the open-source software libraries Normaliz [3] (by Winfried Bruns, Bogdan Ichim and Christof Soeger) and SnapPea [60] (by Jeff Weeks) have been grafted directly into Regina's calculation engine.

Organisations that have contributed to the development of Regina include the American Institute of Mathematics, the Australian Research Council (Discovery Projects DP0208490, DP1094516 and DP110101104), the Institute for the Physics and Mathematics of the Universe (University of Tokyo), Oklahoma State University, the Queensland Cyber Infrastructure Foundation, RMIT University, the University of Melbourne, The University of Queensland, the University of Victoria (Canada), and the Victorian Partnership for Advanced Computing (Australia).

2. Simplifying triangulations

The most basic object in Regina is a *3-manifold triangulation*. Regina does not restrict itself to simplicial complexes; instead it uses *generalised triangulations*, a broader notion that can represent a rich array of 3-manifolds using very few tetrahedra, as described in Section 2.1 below.

One of Regina's most important functions is *simplification*: retriangulating a 3-manifold to use very few tetrahedra (ideally, the fewest possible). Regina includes a rich array of local simplification moves, as outlined in Section 2.2; together it combines these moves into a strong simplification algorithm, which we describe in full in Section 2.3. Moreover, this algorithm is fast: in Section 2.4 we outline some important implementation details, and show that it runs in small polynomial time.

This simplification algorithm gives no guarantee of achieving a *minimal* triangulation (where the number of tetrahedra is the smallest possible). However, the algorithm is extremely powerful in practice: for instance, of the 652 635 906 combinatorially distinct triangulations of the 3-sphere with $3 \leq n \leq 10$ tetrahedra [10], this simplification algorithm quickly reduces *all but 26* of them.

Having a powerful simplification algorithm is essential in computational 3-manifold topology, for several reasons:

- Many significant algorithms have *running times that are exponential or worse* in the number of tetrahedra, and so reducing this number is of great practical importance.
- Good simplification may allow us to *avoid these expensive algorithms entirely*. For instance, in 3-sphere recognition we could simplify the triangulation and then compare it to the known trivial ($\leq$ 2-tetrahedron) triangulations of S^3: if they match then the expensive steps involving normal and almost normal surfaces can be avoided altogether [12].
- More generally, simplification offers good heuristics for solving the *homeomorphism problem*. Simplified triangulations are often well-structured, which makes Regina's combinatorial recognition routines more likely to identify the underlying 3-manifold (see Section 4). Moreover, if two triangulations $\mathcal{T}$ and $\mathcal{T}'$ represent the same 3-manifold, then after simplification it often becomes relatively easy to convert one into the other using Pachner moves (i.e., bistellar flips) [12].

Most of the individual local moves that we describe in Section 2.2 are well-known, and were implemented by Letscher in his early software package *Normal*. Moreover, Matveev [43, 46] and Martelli and Petronio [41] perform similar moves in the dual setting of special spines, and the author has described several in the context of census enumeration [4]. We therefore outline these moves very briefly, although in our general setting there are prerequisites for preserving the underlying 3-manifold that earlier authors have not required (either because they worked in the more flexible dual setting of spines, or they were simply proving criteria for minimality and/or irreducibility). We pay particular attention to the *edge collapse* move, whose prerequisites are complex and require non-trivial algorithms to test quickly. The combined simplification algorithm, whose underlying heuristics have been fine-tuned over many years, is described here in Section 2.3 for the first time.

In Section 2.5, we conclude our discussion on simplification with a new technology that will soon make its way into Regina: simplification by *breadth-first search*

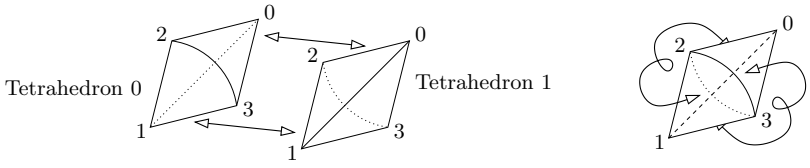


FIGURE 1. A triangulation of the real projective space $\mathbb{R}P^3$

through the Pachner graph. Unlike the algorithm of Section 2.3, this is not polynomial-time; however, it is found to be extremely effective in handling the (very rare) pathological triangulations that cannot be simplified otherwise.

2.1. **Generalised triangulations.** A *generalised 3-manifold triangulation*, or just a *triangulation* from here on, is formed from n tetrahedra by affinely identifying (or "gluing") some or all of their $4n$ faces in pairs. A face is allowed to be identified with another face of the same tetrahedron. It is possible that several edges of a single tetrahedron may be identified together as a consequence of the face gluings, and likewise for vertices. It is common to work with *one-vertex triangulations*, in which all vertices of all tetrahedra become identified as a single point.

Generalised triangulations can produce rich constructions with very few tetrahedra. For instance, Matveev obtains 103 042 distinct closed orientable irreducible 3-manifolds from just ≤ 13 tetrahedra [44], including representatives from all of Thurston's geometries except for $S^2 \times \mathbb{R}$ (which cannot yield a manifold of this type [54]).

Figure 1 illustrates a two-tetrahedron triangulation of the real projective space $\mathbb{R}P^3$. The two tetrahedra are labelled 0 and 1, and the four vertices of each tetrahedron are labelled 0, 1, 2 and 3. Faces 012 and 013 of tetrahedron 0 are joined directly to faces 012 and 013 of tetrahedron 1, creating a solid ball; then faces 023 and 123 of tetrahedron 0 are joined to faces 132 and 032 of tetrahedron 1, effectively gluing the top of the ball to the bottom of the ball with a 180° twist.

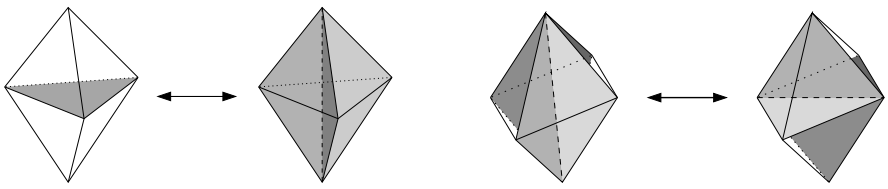
All of this information can be encoded in a table of face gluings, which is how Regina represents triangulations:

Tetrahedron	Face 012	Face 013	Face 023	Face 123
0	1 (012)	1 (013)	1 (132)	1 (032)
1	0 (012)	0 (013)	0 (132)	0 (032)

Consider the cell in the row for tetrahedron t and the column for face abc. If this cell contains u (xyz), this indicates that face abc of tetrahedron t is identified with face xyz of tetrahedron u, using the affine gluing that maps vertices a, b and c of tetrahedron t to vertices x, y and z of tetrahedron u respectively.

For any vertex V of a triangulation, the *link* of V is defined as the frontier of a small regular neighbourhood of V. This mirrors the traditional concept of a link in a simplicial complex, but is modified to support the generalised triangulations that we use in Regina.

A *closed triangulation* is one that represents a closed 3-manifold (like the example above): every tetrahedron face must be glued to a partner, and every vertex link must be a 2-sphere. A *bounded triangulation* is one that represents a 3-manifold with boundary: some tetrahedron faces are not glued to anything (together these form the boundary of the manifold), and every vertex link must be a 2-sphere or a



(a) The 2-3 and 3-2 Pachner moves (b) The "aggregate" 4-4 move

FIGURE 2. Three Pachner-type moves

disc. In either case, it is important that no edge be identified with itself in reverse as a consequence of the face gluings.

Regina can also work with *ideal* triangulations, such as Thurston's famous two-tetrahedron triangulation of the figure eight knot complement [58]: these are triangulations in which vertex links can be higher-genus closed surfaces. For simplicity, in this paper we focus our attention on *closed and bounded triangulations only*, though most of our results apply equally well to ideal triangulations also.

Users can enter tetrahedron gluings directly into Regina, or create triangulations in other ways: importing from SnapPea [60] or other file formats; building "pre-packaged" constructions such as layered lens spaces or Seifert fibred spaces; entering *dehydration strings* [20] or the more flexible *isomorphism signatures* [12] (short pieces of text that completely encode a triangulation); or accessing large ready-made censuses that hold tens of thousands of triangulations of various types.

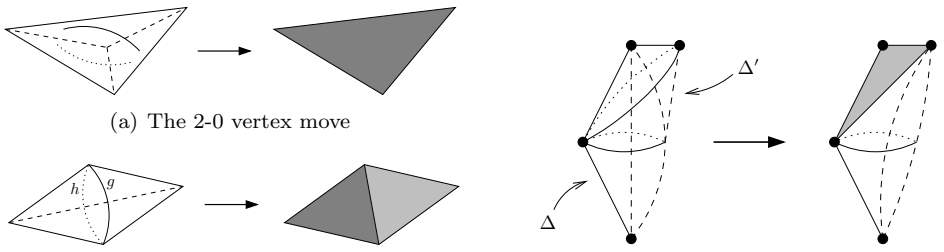
2.2. **Individual simplification moves.** In this section we outline several individual local moves on triangulations. Following this, in Section 2.3 we piece these moves together to build Regina's full simplification algorithm.

We describe each move in full generality, as applied to either closed or bounded triangulations, and without any restrictions such as orientability or irreducibility. For each move we give sufficient conditions under which the move is "safe", i.e., does not change the underlying 3-manifold.

To keep the exposition as short as possible, we simply state these conditions without proof. However, the proofs are simple: the key idea is that, throughout the intermediate stages of each move, we never crush an edge, flatten a bigon or flatten a triangular pillow whose bounding vertices, edges or faces respectively are either identified or both in the boundary. For detailed examples of these types of arguments, see (for instance) the proof of Lemma 3.7 in [6].

2.2.1. *Pachner-type moves.* Our first moves are simple combinations of the well-known *Pachner moves* [49], also known as *bistellar flips*. All of these moves, when performed locally within some larger triangulation, preserve the underlying 3-manifold with no special preconditions required.

Definition 2.1. Consider a triangular bipyramid: this can be triangulated with (i) two distinct tetrahedra joined along an internal face, or with (ii) three distinct tetrahedra joined along an internal degree three edge, as illustrated in Figure 2(a). A *2-3 Pachner move* replaces (i) with (ii), and a *3-2 Pachner move* replaces (ii) with (i).



(a) The 2-0 vertex move

(b) The 2-0 edge move

(c) The 2-1 edge move

FIGURE 3. Local moves around low-degree edges and vertices

Consider a square bipyramid (i.e., an octahedron). This can be triangulated with four distinct tetrahedra joined along an internal degree four edge; moreover, there are three ways of doing this (since the internal edge could follow any of the three main diagonals of the octahedron). A *4-4 move* replaces one such triangulation with another, as illustrated in Figure 2(b).

There are two additional Pachner moves: the *1-4 move* and the *4-1* move. Regina does not use either of these: the 1-4 move complicates the triangulation more than is necessary (since it introduces a new vertex, which is never needed to simplify a triangulation [42, 43]), and the 4-1 move is a special case of an *edge collapse* (described later in this section). The 4-4 move is not a Pachner move, but it can be expressed as an "aggregate" of a 2-3 move followed by a 3-2 move.

2.2.2. *Moves around low-degree edges and vertices.* It is well-known that (under the right conditions) minimal triangulations cannot have low-degree edges or vertices. One typically proves this using local moves around such edges or vertices that either reduce the number of tetrahedra or break some prior assumption on the manifold.

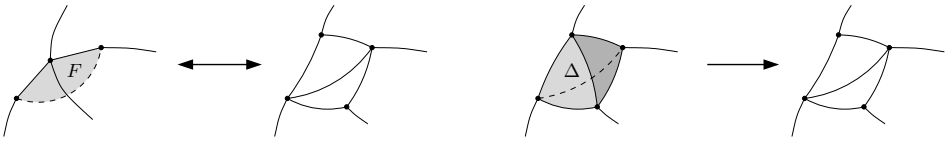
Here we outline three such moves, which Regina uses in a more general setting to simplify arbitrary triangulations. Unlike the earlier Pachner-type moves, these moves *might* change the underlying 3-manifold; after describing the moves, we list sufficient conditions under which they are "safe" (i.e., the 3-manifold is preserved).

Definition 2.2. A *2-0 vertex move* operates on a triangular pillow formed from two distinct tetrahedra surrounding an internal degree two vertex, as illustrated in Figure 3(a), and flattens this pillow to a single face. A *2-0 edge move* operates on a bigon bipyramid formed from two distinct tetrahedra surrounding an internal degree two edge, as illustrated in Figure 3(b), and flattens this to a pair of faces.

Consider a tetrahedron Δ, two of whose faces are folded together around an internal degree one edge, and let Δ' be some distinct adjacent tetrahedron as illustrated in Figure 3(c) (for clarity, the vertices of both tetrahedra are marked in bold). A *2-1 edge move* flattens the two uppermost faces of Δ' together and retriangulates the remaining region with a single tetrahedron to yield a new degree one edge, as shown in the illustration.

To ensure that these moves do not change the underlying 3-manifold, the following conditions are sufficient:

- For the 2-0 vertex move, the two faces that bound the pillow must be distinct (i.e., not identified) and not both simultaneously in the boundary.



(a) The book opening and closing moves (b) One type of boundary shelling move

FIGURE 4. Moves on the boundary of a 3-manifold triangulation

- For the 2-0 edge move, let g and h denote the two edges that we flatten together, as marked in Figure 3(b). These edges must be distinct and not both boundary. Likewise, on each side of g and h, the two faces that we flatten together must be distinct and not both boundary. Finally, although we may identify a face on one side of g and h with a face on the other, we do not allow *all four* faces to be identified in pairs, and we do not allow two of them to be identified if the other two are both in the boundary.
- For the 2-1 edge move, the two edges of Δ' that we flatten together must be distinct and not both boundary, and likewise the two faces of Δ' that we flatten together must be distinct and not both boundary. Note that the second constraint comes "for free" as a corollary of the first.

2.2.3. *Moves on the boundary.* Our next moves apply only to bounded triangulations. As before, they have the potential to change the underlying 3-manifold, and after presenting the moves we give sufficient conditions under which they do not.

Definition 2.3. An *book opening move* operates on a face with precisely two of its three edges in the boundary of the triangulation, as illustrated in Figure 4(a), and "unfolds" the two tetrahedra on either side (which need not be distinct) to create two new boundary faces. A *book closing move* is the inverse move: it operates on two distinct adjacent faces in the boundary and "folds" them together so that they become identified as a single internal face.

A *boundary shelling move* operates on a tetrahedron Δ that has precisely one, two or three faces in the boundary, and simply removes the tetrahedron from the triangulation (effectively "plucking it off" the boundary). This move is illustrated in Figure 4(b) for the case where Δ has two faces in the boundary.

The book opening move will always preserve the underlying 3-manifold. The book closing move will preserve the 3-manifold if (i) the two boundary faces in question are not the *only* faces in that boundary component, and (ii) the two vertices opposite the common edge between these faces are not already identified. The behaviour of the boundary shelling moves depends on the number of faces of the tetrahedron Δ that lie in the boundary:

- If Δ has three boundary faces, removing Δ always preserves the 3-manifold.
- If Δ has two boundary faces, we preserve the 3-manifold if (i) the one edge of Δ not on these faces is internal to the triangulation, and (ii) the two remaining (non-boundary) faces of Δ are not identified.
- If Δ has one boundary face, we preserve the 3-manifold if (i) the one vertex of Δ not on this face is internal to the triangulation, and (ii) no two of the three remaining (non-boundary) edges of Δ are identified.

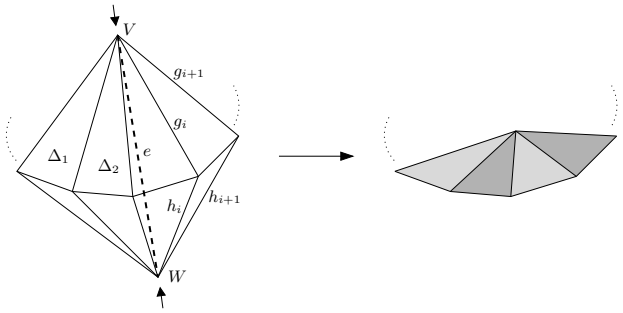


FIGURE 5. Collapsing an edge of a triangulation

2.2.4. *Collapsing edges.* Our final move is the most powerful: it collapses an edge of the triangulation to a single point, and if the edge has high degree then it can eliminate many tetrahedra. The conditions for preserving the underlying 3-manifold are complex, both to describe and to test algorithmically. The details are as follows.

Definition 2.4. An *edge collapse* operates on an edge e of the triangulation that joins two distinct vertices. It crushes edge e to a point and flattens every tetrahedron containing e to a face, as illustrated in Figure 5.

Suppose the edge e has degree d: we denote the d tetrahedra that contain it by $\Delta_1, \ldots, \Delta_d$, and we denote the two endpoints of e by V and W.

If e is an *internal* edge of the triangulation (i.e., its endpoints may lie on the boundary but its relative interior must not), then the following conditions are sufficient to preserve the underlying 3-manifold. Section 2.4 gives details on how we can test these conditions efficiently.

(1) The tetrahedra $\Delta_1, \ldots, \Delta_d$ must all be distinct.
(2) The two endpoints V and W (which we already know to be distinct) must not both be in the boundary.
(3) Denote the d "upper" edges in the diagram that touch V by $g_1, \ldots, g_d$, and denote the corresponding "lower" edges that touch W by $h_1, \ldots, h_d$ (so each g_i, h_i pair will be merged together by the edge collapse). We form a multigraph Γ (allowing loops and/or multiple edges) as follows:
   - each distinct edge of the triangulation becomes a node of Γ;
   - for each $i = 1, \ldots, d$, we add an arc of Γ between the two nodes corresponding to edges g_i and h_i;
   - we add an extra node ∂ to represent the boundary, and add an arc from ∂ to every node that represents a boundary edge.

   Then this multigraph Γ must not contain any cycles.
(4) In a similar way, we build a multigraph whose nodes represent *faces* of the triangulation, whose arcs join corresponding "upper" and "lower" faces that touch V and W respectively, and with an extra boundary node connected to every boundary face. Again, this multigraph must not contain any cycles.

In essence, condition (3) ensures that we never flatten a chain of bigons whose outermost edges are both identified or both boundary, and condition (4) ensures that we never flatten a chain of triangular pillows whose outermost faces are both identified or both boundary.

If the edge e lies in the boundary of the triangulation then we may still be able to perform the move: the sufficient conditions for preserving the 3-manifold are similar but slightly more complex, and we refer the reader to Regina's well-documented source code for the details.

2.3. **The full simplification algorithm.** Now that we are equipped with our suite of local simplification moves, we can present the full details of Regina's simplification algorithm. This algorithm is designed to be both fast and effective, in that order of priority, and its underlying mechanics have evolved over many years according to what has been found to work well in practice.

Of course, other software packages—such as SnapPea [60] and the 3-Manifold Recogniser [45]—have simplification algorithms of their own. To date there has been no comprehensive comparison between them; indeed, such a comparison would be difficult given the ever-present trade-off between speed and effectiveness.

Algorithm 2.5. *Given an input triangulation $\mathcal{T}$, the following procedure attempts to reduce the number of tetrahedra in $\mathcal{T}$ without changing the underlying 3-manifold:*

(1) *Greedily reduce the number of tetrahedra as far as possible. We do this by repeatedly applying the following moves, in the following order of priority, until no such moves are possible:*
   - *edge collapses;*
   - *low-degree edge moves (3-2 Pachner moves, or 2-0 or 2-1 edge moves);*
   - *low-degree vertex moves (2-0 vertex moves);*
   - *boundary shelling moves.*

(2) *Make up to $5R$ successive random 4-4 moves, where R is the maximum number of available 4-4 moves that could be made from any single triangulation obtained during this particular iteration of step (2). If we ever reach a triangulation from which we can greedily reduce the number of tetrahedra (as defined above) then return immediately to step (1).*

(3) *If the triangulation has boundary, then perform book opening moves until no more are possible. If this enables us to collapse an edge then do so and return to step (1). Otherwise undo the book openings and continue.*

(4) *If a book closing move can be performed, then do it and return to step (1). Otherwise terminate the algorithm.*

The greedy reduction in step (1) prioritises edge collapses, because these can remove many tetrahedra at once, and because we typically aim for a one-vertex triangulation. In step (2), the coefficient 5 is chosen somewhat arbitrarily; note also that the quantity R might increase as this step progresses. The book openings in step (3) aim to increase the number of vertices without adding new tetrahedra, in the hope that an edge collapse becomes possible. The book closures in step (4) aim to leave us with the smallest boundary possible, which becomes advantageous during other expensive algorithms (such as normal surface enumeration).

Note that we never explicitly *increase* the number of tetrahedra (e.g., we never perform an explicit 2-3 Pachner move). This is only possible because we have a large suite of moves available: if we reformulate our algorithm in terms of Pachner moves alone then almost every step would require both 2-3 and 3-2 moves.

One of the more prominent simplification techniques that we do *not* use is the 0-efficiency reduction of Jaco and Rubinstein [33]. This is because the best known algorithms for testing for 0-efficiency run in worst-case exponential time. Moreover,

experimental observation suggests that—ignoring well-known exceptions, such as reducible manifolds and $\mathbb{R}P^3$—after running Algorithm 2.5 we typically find that the triangulation is already 0-efficient. We discuss algorithms for 0-efficiency testing further in Section 3.3.

2.4. **Time complexity and performance.** Simplification is one of the most commonly used "large-scale" routines in Regina's codebase, and it is imperative that it runs quickly. In this section we analyse the running time, which includes a discussion of key implementation details for the edge collapse. All time complexities are based on the word RAM model of computation (so, for instance, adding two integers is considered $O(1)$ time as long as the integers only require $\log n$ bits).

Throughout this discussion, we let n denote the number of tetrahedra in the triangulation. Some key points to note:

- *Adding a new tetrahedron* takes $O(1)$ time. However, *deleting a tetrahedron* takes $O(n)$ time for Regina since we must reindex the tetrahedra that remain (i.e., if we delete tetrahedron Δ_i then we must rename Δ_j to Δ_{j-1} for all $j > i$).[1] Deleting *many* tetrahedra can be done in combined $O(n)$ time, since if we are careful in our implementation then each leftover tetrahedron only needs to be reindexed at most once.
- *Computing the skeleton* of the triangulation (i.e., identifying and indexing the distinct vertices, edges, faces and boundary components of the triangulation, and linking these to and from the corresponding tetrahedra) can be done in $O(n)$ time using standard depth-first search techniques.

Theorem 2.6. *Algorithm 2.5 (the full simplification algorithm) runs in time $O(n^4 \log n)$, where n is the number of tetrahedra in the input triangulation.*

The worst culprit in raising the time complexity is the edge collapse move, and specifically, testing its sufficient conditions. In practice running times are much faster than quartic, and with more delicacy we could bring down the theoretical time complexity to reflect this; we return to such issues after the proof.

Proof. First, we observe that for every move type except for the edge collapse, we can test the sufficient conditions in $O(1)$ time and perform the move in $O(n)$ time (where the dominating factor is deleting tetrahedra and rebuilding the skeleton, as outlined above).

For the edge collapse move, performing the move takes $O(n)$ time but testing the sufficient conditions is a little slower. Recall that we must build a multigraph Γ and ensure that it contains no cycles. We can do this by adding one arc at a time, and tracking connected components: if an arc joins two distinct components then we merge them into a single component, and if an arc joins some component with itself then we obtain a cycle and the sufficient conditions fail.

To track connected components, we use the well-known *union-find* data structure [22]. With union-find, the operations of (i) identifying which component a node belongs to and (ii) merging two components together each take $O(\log n)$ time. Therefore testing the multigraph Γ for cycles takes $O(n \log n)$ time overall, and testing sufficient conditions for an edge collapse likewise becomes $O(n \log n)$.

[1] This is Regina's own implementation constraint: many moves could be $O(1)$ if we ignored Regina's need to consecutively index tetrahedra and other objects. Either way, however, the time complexity for the edge collapse—and hence the full simplification algorithm—would be the same.

From here the running time is simple to obtain. The algorithm works through "stages", where in each stage we either reduce the number of tetrahedra, or we reduce the number of boundary faces through a book closing move. Either way, it is clear there can be at most $O(n)$ such stages in total (since the triangulation and its boundary cannot disappear entirely).

Within each stage we *perform* $O(n)$ moves in total: at most $5R \leq 5(\#\text{ edges}) \leq 5 \cdot 6n$ successive 4-4 moves, and at most $(\#\text{ internal faces}) \leq 2n$ successive book opening moves. This gives us a total of $O(n^2)$ moves throughout the life of the algorithm, each requiring $O(n)$ time to perform.

Between each pair of moves, we might *test* a large number of potential moves whose sufficient conditions ultimately fail. The number of moves that we test on a given triangulation is clearly $O(n)$ (since there are $O(n)$ possible "local regions" in which each type of move could be performed), and so throughout the entire algorithm with its $O(n^2)$ moves we run a total of $O(n^3)$ tests. In the worst case (edge collapses), each test could take $O(n \log n)$ time.

It is clear now that the total time spent *performing* moves is $O(n^3)$ and the total time spent *testing* sufficient conditions is $O(n^4 \log n)$, yielding a running time of $O(n^4 \log n)$ overall. □

Some further remarks on this running time:

- We noted earlier that *in practice* running times are faster than $O(n^4 \log n)$. This is because the powerful edge collapse moves typically reach the fewest possible number of vertices very quickly (i.e., one vertex for a closed manifold, or else one vertex for each boundary component). Once we achieve this, testing sufficient conditions for an edge collapse move becomes $O(1)$ time, which eliminates an $n \log n$ factor from our running time.
- In theory, we can remove a factor of n as follows: once greedy simplification fails and we move on to steps (2) and (3), we only test for *new* simplification moves in the immediate neighbourhood of the *last* 4-4 or book opening move. The implementation becomes more subtle and the bookkeeping more complex, and this is planned for future versions of Regina.
- Finally, we note that the $\log n$ factor can be stripped down to "almost constant": essentially an inverse of the Ackermann function, and ≤ 4 in all conceivable situations. We can do this by applying the *path compression* optimisation to the union-find data structure; see [22, 56] for details.

We finish this section with a practical demonstration. Let $\mathcal{T}$ be the 23-tetrahedron triangulation of the Weber-Seifert dodecahedral space presented in [18], and let S be an arbitrarily-chosen vertex normal surface of the highest possible genus (here we choose vertex surface #1733, which is an orientable genus 16 surface). If we cut $\mathcal{T}$ open along the surface S using Regina's `cutAlong()` procedure, we obtain a (disconnected) bounded triangulation $\mathcal{T}'$ with 1990 tetrahedra. The simplification algorithm reduces this to 135 tetrahedra in roughly 1.9 seconds (as measured on a 2.93 GHz Intel Core i7).

If we perform a *barycentric subdivision* on $\mathcal{T}'$, we obtain a new triangulation $\mathcal{T}''$ with 47 760 tetrahedra (and a large number of vertices, which means a large number of expensive edge collapse moves). Again the simplification algorithm reduces this to 135 tetrahedra, but this time takes 780 seconds, suggesting (for this arbitrary example) only a quadratic—not quartic—growth rate in n.

2.5. **Exhaustive simplification via the Pachner graph.** We finish our discussion on simplification with a new technology: *breadth-first search through the Pachner graph*. The key idea is, instead of using greedy simplification heuristics, to try *all possible sequences* of Pachner moves (up to a user-specified limit). The result is a much slower, but also much stronger, simplification algorithm. This algorithm is based on ideas from the large-scale experimental study of Pachner graphs described in [12], and the code will soon be merged into Regina's main source tree.

Definition 2.7. For a closed 3-manifold triangulation $\mathcal{M}$, the *restricted Pachner graph* $\mathscr{P}_1(\mathcal{M})$ is the infinite graph whose nodes correspond to isomorphism classes of one-vertex triangulations of $\mathcal{M}$ (where by *isomorphism* we mean a relabelling of tetrahedra and/or their vertices), and where two nodes are joined by an arc if there is a 2-3 or 3-2 Pachner move between the corresponding triangulations. The nodes of $\mathscr{P}_1(\mathcal{M})$ are partitioned into finite *levels* $1, 2, \ldots$ according to the number of tetrahedra in the corresponding triangulations.

The basic idea is as follows. By a result of Matveev [42, 43], if we exclude level 1 then the restricted Pachner graph $\mathscr{P}_1(\mathcal{M})$ is always connected, and so we should be able to simplify a non-minimal triangulation of $\mathcal{M}$ by finding a path through $\mathscr{P}_1(\mathcal{M})$ from the corresponding node to some other node at a lower level. In detail:

Algorithm 2.8. *Given a one-vertex triangulation $\mathcal{T}$ with n tetrahedra representing a closed 3-manifold $\mathcal{M}$, as well as a user-defined "height parameter" h, the following procedure attempts to find a triangulation of $\mathcal{M}$ with $< n$ tetrahedra:*

1. *Conduct a breadth-first search through $\mathscr{P}_1(\mathcal{M})$ starting from the node representing $\mathcal{T}$, but restrict this search to consider only nodes at levels $\leq n+h$.*
2. *If we ever reach a node at level $< n$, this yields a simpler triangulation (which we try to simplify further with the fast Algorithm 2.5). Otherwise we advise the user to try again with a larger h (if they can afford to do so).*

The height parameter is needed because $\mathscr{P}_1(\mathcal{M})$ is infinite, and because even the individual levels grow extremely quickly (for $\mathcal{M} = S^3$ the growth rate is at least exponential, and it is open as to whether it is super-exponential [2]). Extremely small height parameters work very well in practice: for $\mathcal{M} = S^3$ there are no known cases for which $h = 2$ will not suffice [11].

Because of the enormous number of nodes involved, a careful implementation of Algorithm 2.8 is vital. We cannot afford to build even an entire single level of $\mathscr{P}_1(\mathcal{M})$ beforehand; instead we construct nodes as reach them, and cache them using their isomorphism signatures (polynomial-time computable strings that also manage isomorphism testing [11]). The algorithm lends itself well to *parallelisation* (using multithreading with shared memory, not large-scale distributed processing).

Unlike the earlier Algorithm 2.5, this new Algorithm 2.8 is certainly not polynomial-time. Even if we are able to simplify the triangulation using k Pachner moves for small k, we still test a total of $O((n+h)^k)$ potential moves; moreover, the growth rate of k itself is unknown, and so even the exponent could become exponential. See [11] for experimental measurements of k for a range of different 3-manifolds.

In practice, with height parameter $h = 2$, Algorithm 2.8 easily simplifies the 26 "pathological" triangulations of S^3 with ≤ 10 tetrahedra that the faster Algorithm 2.5 could not. For these cases we need $5 \leq k \leq 9$ moves, with CPU times ranging from 0.8 to 14 seconds. This is quite slow for just $n \leq 10$ tetrahedra, and highlights the ever-present trade-off between speed and effectiveness.

(a) Triangles and quadrilaterals (b) An octagonal piece

FIGURE 6. A tetrahedron intersecting a normal or almost normal surface

3. Normal and almost normal surfaces

One of Regina's core strengths is its ability to enumerate and work with normal and almost normal surfaces. A *normal surface* in a 3-manifold triangulation $\mathcal{T}$ is a properly embedded surface in $\mathcal{T}$ that meets each tetrahedron in a (possibly empty) collection of disjoint curvilinear triangles and/or quadrilaterals, as illustrated in Figure 6(a). An *octagonal almost normal surface* is defined in the same way, but also requires that exactly one tetrahedron contains exactly one additional octagonal piece, as illustrated in Figure 6(b).

Normal surfaces are a powerful tool for high-level recognition and decomposition algorithms; prominent examples include unknot recognition [26], connected sum decomposition [33], and testing for incompressible surfaces [31]. Almost normal surfaces were introduced by Rubinstein, and play a central role in algorithms such as 3-sphere recognition [52], Heegaard genus [37], and recognising small Seifert fibred spaces [53]. Rubinstein originally defined almost normal surfaces to include either a single octagonal piece or a single *tube* piece, but Thompson later showed that for 3-sphere recognition, only octagons need to be considered [57].

We begin in Section 3.1 with a very brief overview of the necessary concepts from normal surface theory; for more context we refer the reader to [27].

In Section 3.2 we discuss the all-important problem of *enumerating* vertex and fundamental normal surfaces, and introduce a new trie-based optimisation to alleviate the most severe bottlenecks in the enumeration algorithm. We follow in Section 3.3 with a brief discussion of 0-efficiency and the important problem of locating normal *spheres*, describing the rationale behind Regina's choice of algorithm, and explaining why other well-known options are not effective.

Section 3.4 outlines Regina's current implementations of 3-sphere recognition, 3-ball recognition and connected sum decomposition. Although the key ideas are already known, the implementations have evolved to the point where all three algorithms are now surprisingly simple, and we present them here as a useful reference in a modern algorithmic form that is "ready for implementation".

We finish in Section 3.5 with a brief discussion of *tree traversal* algorithms, a new technology soon to appear in Regina based on backtracking and linear programming, and with enormous potential for improving performance on large problems.

Beyond the algorithms described here, Regina offers many ways to analyse normal surfaces, both "at a glance" and in detail. It supports the complex operation of cutting a triangulation open along a normal surface and retriangulating, and it supports the Jaco-Rubinstein operation of crushing a surface to a point [33] (which may introduce additional changes in topology).

3.1. **Preliminaries from normal surface theory.** In an n-tetrahedron triangulation $\mathcal{T}$, normal surfaces correspond to integer vectors in a cone of the form $\{\mathbf{x} \in \mathbb{R}^{7n} \mid A\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$, where the matrix A of *matching equations* is derived from $\mathcal{T}$. The $7n$ coordinates are grouped into $4n$ *triangle coordinates*, which count the triangles at each corner of each tetrahedron, and $3n$ *quadrilateral coordinates*, which count the quadrilaterals passing through each tetrahedron in each of the three possible directions. Such vectors must also satisfy the *quadrilateral constraints*, which require that at most one quadrilateral coordinate within each tetrahedron can be non-zero. These constraints map out a (typically non-convex) union of faces of the cone above.

An important observation is that non-trivial connected normal surfaces can be reconstructed from their $3n$ quadrilateral coordinates alone [59]. We can therefore identify such surfaces with integer points in a smaller-dimensional cone of the form $\{\mathbf{x} \in \mathbb{R}^{3n} \mid B\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$. We refer to $\mathbb{R}^{7n}$ and $\mathbb{R}^{3n}$ as working in *standard coordinates* and *quadrilateral coordinates* respectively.

A normal surface is called a (standard or quadrilateral) *vertex surface* if its vector in (standard or quadrilateral) coordinates lies on an extreme ray of the corresponding cone, and it is called a (standard or quadrilateral) *fundamental surface* if its vector lies in the Hilbert basis of the cone. The quadrilateral vertex and fundamental surfaces are typically a strict subset of their standard counterparts.

Throughout this paper, we use the phrase *almost normal surface* to refer exclusively to the case where the extra piece is an octagon (not a tube). For almost normal surfaces we introduce three additional octagon coordinates for each tetrahedron, yielding a cone in *standard almost normal coordinates* of the form $\{\mathbf{x} \in \mathbb{R}^{10n} \mid C\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$. As before, non-trivial connected surfaces can be reconstructed from their $3n$ quadrilateral and $3n$ octagon coordinates [9], yielding a cone in *quadrilateral-octagon coordinates* of the form $\{\mathbf{x} \in \mathbb{R}^{6n} \mid D\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$. We can likewise define vertex and fundamental surfaces in these coordinate systems.

To finish, we make the well-known observation that Euler characteristic is a linear function in standard normal and almost normal coordinates [34], though it is not linear in quadrilateral or quadrilateral-octagon coordinates.

3.2. **Enumeration.** Many high-level algorithms are based on locating particular surfaces, which—if they exist—can be found as vertex normal surfaces, or for some more difficult algorithms, fundamental normal surfaces. Regina comes with heavily optimised algorithms for enumerating all vertex normal surfaces [8] or fundamental normal surfaces [13] in a triangulation, in all of the coordinate systems listed above.

Here we focus on the vertex enumeration algorithm, which is based on the double description method for enumerating extreme rays of polyhedral cones [24, 47]. We outline the double description method very briefly, and then introduce a new trie-based optimisation that yields significant improvements in its running time.

In brief, the double description method enumerates the extreme rays of the cone $\{\mathbf{x} \in \mathbb{R}^d \mid A\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$ by constructing a series of cones $\mathcal{C}_0, \mathcal{C}_1, \ldots$, where each $\mathcal{C}_i$ is defined only using the first i rows of A. The initial cone $\mathcal{C}_0$ is simply the non-negative orthant, with extreme rays defined by the d unit vectors, and each subsequent cone $\mathcal{C}_i$ is obtained inductively from $\mathcal{C}_{i-1}$ by intersecting with a new hyperplane H_i. The extreme rays of $\mathcal{C}_i$ are obtained from (i) extreme rays of $\mathcal{C}_{i-1}$ that lie on H_i; and (ii) convex combinations of *pairs* of adjacent extreme rays of $\mathcal{C}_{i-1}$ that lie on either side of H_i.

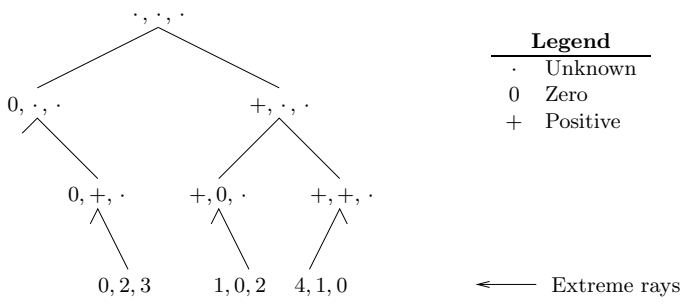


FIGURE 7. An example in $\mathbb{R}^3$ of storing extreme rays in a trie

There are significant optimisations that can be applied to the double description method in the context of normal surface theory; see [7, 8] for details. However, a major problem remains: the intermediate cones $\mathcal{C}_i$ can have a great many extreme rays (the well-known "combinatorial explosion" in the double description method), and the combinatorial algorithm[2] for identifying all pairs of *adjacent* extreme rays of $\mathcal{C}_i$ is cubic in the total number of extreme rays.

This cubic procedure stands out as the most severe bottleneck in the algorithm. The adjacency test is simple: two extreme rays $\mathbf{x}_1, \mathbf{x}_2$ of $\mathcal{C}_i$ are adjacent if and only if there is no other extreme ray $\mathbf{z}$ of $\mathcal{C}_i$ for which, whenever the ith coordinates of $\mathbf{x}_1$ and $\mathbf{x}_2$ are both zero, then the ith coordinate of $\mathbf{z}$ is zero also. The cubic algorithm essentially just iterates through all possibilities for $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{z}$.

We improve this procedure by storing the extreme rays of $\mathcal{C}_i$ in a *trie* (also known as a *radix tree*) [55]. In our case this is a binary tree of depth d, illustrated in Figure 7, where at depth i the left and right branches contain all extreme rays for which the $(i+1)$th coordinate is zero and non-zero respectively.[3] The extreme rays themselves correspond to leaves of the tree at depth d. We only store those portions of the tree that contain extreme rays as descendants, so the total number of nodes is $O(d \cdot \#\text{ extreme rays})$ and not $2^{d+1} - 1$.

To identify all pairs of adjacent extreme rays of $\mathcal{C}_i$, we first insert all extreme rays into the trie: each insertion takes $O(d)$ time (just follow a path down and create new nodes as needed). We then iterate through all pairs of extreme rays $\mathbf{x}_1, \mathbf{x}_2$, and to test *adjacency* we walk through the trie by (i) starting at the root, and (ii) whenever we reach a node at level i, if the $(i+1)$th coordinates of $\mathbf{x}_1$ and $\mathbf{x}_2$ are both zero then we follow the left branch, and otherwise we follow both the left and right branches in a depth-first manner. We declare $\mathbf{x}_1$ and $\mathbf{x}_2$ to be adjacent if and only if we do *not* locate some other extreme ray $\mathbf{z} \neq \mathbf{x}_1, \mathbf{x}_2$ during our walk.

We can optimise this trie further:

- At each node, we store the number of extreme rays in the corresponding subtree. This allows us to avoid the "false positives" $\mathbf{x}_1$ and $\mathbf{x}_2$: if we are in a subtree containing one or both of $\mathbf{x}_1, \mathbf{x}_2$ and the number of extreme rays is one or two respectively, then we can backtrack immediately.

[2]There is an alternate algebraic algorithm; see [24] for theoretical and practical comparisons.

[3]This is well-defined because each extreme ray is of the form $\{\lambda \mathbf{v} \mid \lambda \geq 0\}$ for some vector $\mathbf{v}$.

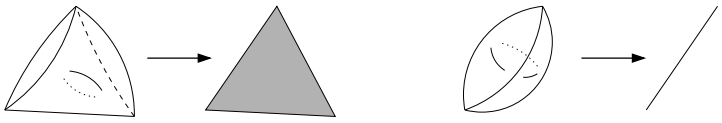


FIGURE 8. Examples of collapsing non-tetrahedron pieces

- We can "compress" the trie by storing extreme rays at the nodes corresponding to their last *non-zero* coordinate, instead of at depth d, a useful optimisation given that our extreme rays may contain many zeroes.

The appeal of this data structure is that we are able to target our search by only looking at "promising" candidates for $\mathbf{z}$, instead of scanning through all extreme rays. It is difficult to pin down the theoretical complexity of our trie-based search; certainly it might be exponential in d (because of the branching), but it is clearly no worse than $O(d \cdot \#$ extreme rays), i.e., the total number of nodes.

In practice, it serves us very well. Consider again the 23-tetrahedron triangulation of the Weber-Seifert dodecahedral space from [18]. With the original implementation of the double description method (including all optimisations except for the trie-based search), enumerating all 698 quadrilateral vertex normal surfaces requires 174 minutes (measured on a 2.93 GHz Intel Core i7). The new trie-based algorithm reduces this to 62 minutes, cutting the running time from roughly three hours down to just one.

3.3. **0-efficiency.** An important problem in normal surface theory is searching for normal *spheres*: this is at the heart of Jaco and Rubinstein's 0-efficiency machinery [33], and features in all of the high-level algorithms listed in Section 3.4.

A closed orientable 3-manifold is *0-efficient* if its only normal 2-spheres are the trivial vertex linking spheres (which contain only triangles, and which must always be present). If a triangulation is not 0-efficient, then we can use Jaco and Rubinstein's "destructive crushing" procedure [33]: we crush the non-trivial sphere to a point, and then collapse away any degenerate non-tetrahedron pieces (such as footballs, pillows and so on) to become edges and faces, as illustrated in Figure 8.

The result is that every tetrahedron that contains a quadrilateral from the non-trivial sphere will disappear entirely, and the final triangulation (which might be disconnected) will be closed and have strictly fewer tetrahedra than the original. The crushing process might introduce topological changes, but these are limited to pulling apart connected sums, adding new 3-sphere components, and deleting S^3, $\mathbb{R}P^3$, $S^2 \times S^1$ and/or $L_{3,1}$ components. The crushing process is simple to implement (in stark contrast to the messy procedure of *cutting* along a normal surface), and with help from first homology groups any topological changes are easy to detect.

In order to test for 0-efficiency (and to explicitly identify a non-trivial normal sphere if one exists), Regina uses the following result:

Lemma 3.1. *If a closed orientable triangulation $\mathcal{T}$ contains a non-vertex-linking normal sphere, then it contains one as a quadrilateral vertex normal surface.*

Proof. This result is widely known, but (to the author's best knowledge) does not appear in the literature, and so we outline the simple proof here. When we convert to vectors in *standard* coordinates, any connected non-vertex-linking normal surface F can be expressed as a positive rational combination of one or more *quadrilateral*

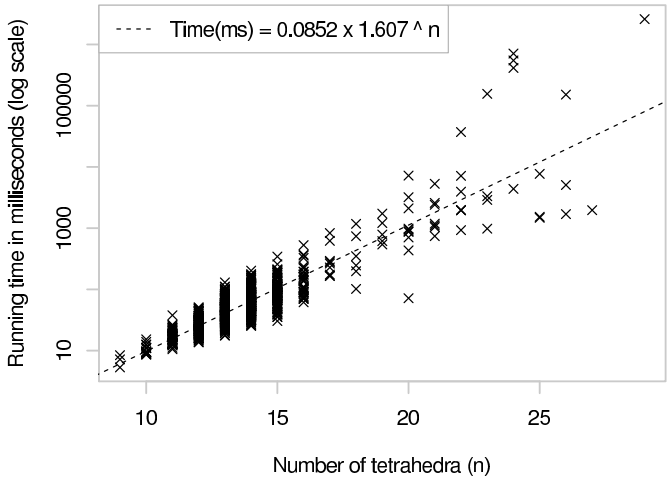


FIGURE 9. Growth of the running time for the double description method

vertex normal surfaces, minus zero or more vertex linking spheres. Since the Euler characteristic χ is linear in standard coordinates and $\chi(S^2) > 0$, it follows that if F is a non-trivial normal sphere then some quadrilateral vertex normal surface Q must have $\chi(Q) > 0$, whereupon some rational multiple of Q must be a non-trivial normal sphere also. □

At present, Regina tests for 0-efficiency by enumerating *all* quadrilateral vertex normal surfaces (up to multiples) and testing each. This of course is more work than we need to do, since we only need to locate *one* non-trivial sphere. We outline some tempting alternatives now, and explain why Regina does not use them.

The first alternative is that, in standard coordinates, we could restrict our polyhedral cone by adding the homogeneous linear constraint $\chi \geq 0$. This has been tried in Regina, but yields a substantially *slower* algorithm. Experimentation suggests that this is because (i) we are forced to work in the higher-dimensional $\mathbb{R}^{7n}$ instead of $\mathbb{R}^{3n}$; and (ii) the constraint $\chi \geq 0$ slices through the cone in a way that creates significantly more extreme rays, exacerbating the combinatorial explosion in the double description method.

The second alternative is based on linear programming. Casson and Jaco et al. [30] have suggested (in essence) that, for each of the 3^n choices of which quadrilateral coordinate we allow to be non-zero in each tetrahedron, we could solve a linear program (in polynomial time) to maximise χ over a corresponding sub-cone in $\mathbb{R}^{7n}$. This is a promising approach, but it has a significant problem: for "good" triangulations, which typically *are* 0-efficient, we must attempt all 3^n linear programs before we can terminate. That is, 3^n becomes a *lower bound* on the running time.

Although the best known theoretical time complexity for a full vertex enumeration is slower than this [17], in *practice* a full enumeration is typically much faster. For example, when enumerating quadrilateral vertex normal surfaces for the first 1000 triangulations in the Hodgson-Weeks closed hyperbolic census [29] (a good source of "difficult" manifolds for normal surface enumeration), the optimised double description method gives a running time that grows roughly like 1.6^n, as shown in Figure 9.

3.4. **High-level algorithms.** Here we present Regina's current implementations of the high-level 3-sphere recognition, 3-ball recognition and connected sum decomposition algorithms. As noted earlier, the key ideas are already known; the purpose of this description is to give a useful reference for these algorithms in a modern "ready to implement" form.

It should be noted that all three algorithms guarantee both correctness and termination (i.e., they are not probabilistic in nature). The 3-sphere recognition and connected sum decomposition algorithms include developments from many authors [9, 32, 33, 34, 36, 52, 57], and the 3-ball recognition algorithm is a trivial modification of 3-sphere recognition. See [43] for a related but non-equivalent variant of 3-sphere recognition based on special spines.

Algorithm 3.2 (3-sphere recognition)**.** *The following algorithm tests whether a given triangulation $\mathcal{T}$ is a triangulation of the 3-sphere.*

(1) *Test whether $\mathcal{T}$ is closed, connected and orientable. If $\mathcal{T}$ fails any of these tests, terminate and return* `false`*.*
(2) *Simplify $\mathcal{T}$ using Algorithm 2.5.*
(3) *Test whether $\mathcal{T}$ has trivial homology. If not, terminate and return* `false`*.*
(4) *Create a list $\mathcal{L}$ of triangulations to process, initially containing just $\mathcal{T}$. While $\mathcal{L}$ is non-empty:*
   - *Let $\mathcal{N}$ be the next triangulation in the list $\mathcal{L}$. Remove $\mathcal{N}$ from $\mathcal{L}$, and test whether $\mathcal{N}$ has a quadrilateral vertex normal sphere F.*
     - *If so, then perform the Jaco-Rubinstein crushing procedure on F. For each connected component $\mathcal{N}'$ of the resulting triangulation, simplify $\mathcal{N}'$ and add it back into the list $\mathcal{L}$.*
     - *If not, and if $\mathcal{N}$ has only one vertex, then search for a quadrilateral-octagon vertex almost normal sphere in $\mathcal{N}$. If none exists then terminate and return* `false`*.*
(5) *Once there are no more triangulations in $\mathcal{L}$, terminate and return* `true`*.*

The key invariant in the algorithm above is that the original 3-manifold is always the connected sum of all manifolds in $\mathcal{L}$. The homology test in step (3) is crucial, since the Jaco-Rubinstein crushing procedure could silently delete $S^2 \times S^1$, $\mathbb{R}P^3$ and/or $L_{3,1}$ components.

Algorithm 3.3 (3-ball recognition)**.** *The following algorithm tests whether a given triangulation $\mathcal{T}$ is a triangulation of the 3-ball.*

(1) *Test whether $\mathcal{T}$ is connected, orientable, has precisely one boundary component, and this boundary component is a 2-sphere. If $\mathcal{T}$ fails any of these tests, terminate and return* `false`*.*
(2) *Simplify $\mathcal{T}$ using Algorithm 2.5.*
(3) *Cone the boundary of $\mathcal{T}$ to a point by attaching one new tetrahedron to each boundary face, and simplify again.*
(4) *Run 3-sphere recognition over the final triangulation, and return the result.*

For our final algorithm, we note that by "connected sum decomposition" we mean a decomposition into *non-trivial* prime summands (i.e., no unwanted S^3 terms).

Algorithm 3.4 (Connected sum decomposition)**.** *The following algorithm computes the connected sum decomposition of the manifold described by a given triangulation $\mathcal{T}$. We assume as a precondition that $\mathcal{T}$ is closed, connected and orientable.*

(1) *Simplify $\mathcal{T}$ using Algorithm 2.5.*
(2) *Compute the first homology of $\mathcal{T}$, and let r, t_2 and t_3 denote the rank, $\mathbb{Z}_2$ rank and $\mathbb{Z}_3$ rank respectively.*
(3) *Create an input list $\mathcal{L}$ of triangulations to process, initially containing just $\mathcal{T}$, and an output list $\mathcal{O}$ of prime summands, initially empty.*
While $\mathcal{L}$ is non-empty:
   - *Let $\mathcal{N}$ be the next triangulation in the list $\mathcal{L}$. Remove $\mathcal{N}$ from $\mathcal{L}$, and test whether $\mathcal{N}$ has a quadrilateral vertex normal sphere F.*
     - *If so, then perform the Jaco-Rubinstein crushing procedure on F. For each connected component $\mathcal{N}'$ of the resulting triangulation, simplify $\mathcal{N}'$ and add it back into the list $\mathcal{L}$.*
     - *If not, then append $\mathcal{N}$ to the output list $\mathcal{O}$ if either (i) $\mathcal{N}$ has non-trivial homology, or (ii) $\mathcal{N}$ has only one vertex and no quadrilateral-octagon vertex almost normal sphere.*
(4) *Compute the first homology of each triangulation in the output list $\mathcal{O}$, sum the ranks, $\mathbb{Z}_2$ ranks and $\mathbb{Z}_3$ ranks, and append additional copies of $S^2 \times S^1$, $\mathbb{R}P^3$ and $L_{3,1}$ to $\mathcal{O}$ so that these ranks sum to r, t_2 and t_3 respectively.*

On termination, the output list $\mathcal{O}$ will contain triangulations of the (non-trivial) prime summands of the input manifold.

The key invariants of this algorithm are that (i) the input manifold is always the connected sum of all manifolds in $\mathcal{L}$ and $\mathcal{O}$, plus zero or more $S^2 \times S^1$, $\mathbb{R}P^3$ and/or $L_{3,1}$ summands; and that (ii) every output manifold in $\mathcal{O}$ is prime and not S^3.

3.5. **Tree traversal algorithms.** There have been recent interesting developments in computational normal surface theory that could allow us to move away from the double description method entirely. These are algorithms based on traversing a search tree [17, 16], and they combine aspects of linear programming, polytope theory and data structures.

The resulting algorithms avoid the dreaded combinatorial explosion of the double description method; moreover, they offer incremental output and are well-suited to parallelisation, progress tracking and early termination. Most importantly, experimentation suggests that they are significantly faster and less memory-hungry—even when run in serial—for larger and more difficult problems. The code is already up and running, and will be included in the next release of Regina.

Such tree traversal algorithms can be used for either a full enumeration of vertex normal surfaces [17], or to locate a single non-trivial normal or almost normal sphere (for 0-efficiency testing and/or 3-sphere recognition) [16]. The key idea is to build a search tree according to which quadrilateral coordinates are non-zero in each tetrahedron, and to run incremental linear programs that enforce the quadrilateral constraints for those tetrahedra where decisions have been made, but *ignore* the quadrilateral constraints for those tetrahedra that we have not yet processed.

For the full enumeration of vertex normal surfaces, details of the tree traversal algorithm can be found in [17]. To illustrate, we return again to the 23-tetrahedron triangulation of the Weber-Seifert dodecahedral space: whereas the trie-based double description method enumerates all 698 quadrilateral vertex normal surfaces in 62 minutes, the tree traversal algorithm does this in just 32 minutes.

For locating just a single normal or almost normal sphere, the tree traversal algorithm becomes extremely powerful: it can prove that this same triangulation of

the Weber-Seifert dodecahedral space is 0-efficient in *under 10 seconds*. Note that there is no early termination here—the tree traversal algorithm conclusively proves in under 10 seconds that no non-trivial normal sphere exists.

This latter algorithm for locating normal and almost normal spheres relies on a number of crucial heuristics, and full details can be found in [16]. Perhaps most interesting is the following experimental observation: for "typical" inputs, this algorithm appears to require only a *linear* number of linear programs; that is, the typical behaviour appears to be *polynomial-time*. One should be quick to note that this is in experimentation only, and that the algorithm is not polynomial-time in the worst case. Nevertheless, this is a very exciting computational development.

4. Combinatorial recognition

In this brief section we outline Regina's combinatorial recognition code. Despite significant advances in 3-manifold algorithms, we as a community are still a long way from being able to implement the full homeomorphism algorithm—even simpler problems such as JSJ decomposition have never been implemented, and Hakenness testing (which plays a key role in the homeomorphism problem) has only recently become practical [18]. These are the issues that we aim to address (or rather work around) here.

In addition to slower but always-correct and always-conclusive algorithms such as 3-sphere recognition and connected sum decomposition, Regina offers a secondary means for identifying 3-manifolds: *combinatorial recognition*. The central idea is that we "hard-code" a large number of general constructions for infinite families of 3-manifolds (such as Seifert fibred spaces, surface bundles and graph manifolds). Then, given an input triangulation $\mathcal{T}$, we test whether $\mathcal{T}$ follows one of these hard-coded constructions, and if it does, we "read off" the parameters to name the underlying 3-manifold.

The advantages of this technique are:

- It is extremely fast—all of Regina's hard-coded constructions of infinite families can be recognised in small polynomial time.
- It allows Regina to recognise a much larger range of 3-manifolds than would otherwise be practically possible.

There are, of course, clear disadvantages:

- Such techniques require a *lot* of code if we wish to recognise each construction in its full generality: Regina currently has over 25 000 lines of source code devoted to combinatorial recognition alone.
- The code is only as powerful and general as the constructions that are implemented. For instance, with a handful of exceptions, Regina's recognition routines do not include any hyperbolic manifolds.
- To be recognised, a triangulation must be *well-structured*—an arbitrary triangulation of even a simple manifold such as a lens space will not be recognised unless it follows one of the known constructions.

Despite these drawbacks, combinatorial recognition is enormously useful in practice. Regina's recognition code is particularly strong for non-orientable manifolds: of the 366 manifolds in the $\leq$ 11-tetrahedron closed non-orientable census [10], Regina is able to recognise *all* minimal triangulations for 334 of them, and is able to recognise at least *one* minimal triangulation for 357—that is, all but nine.

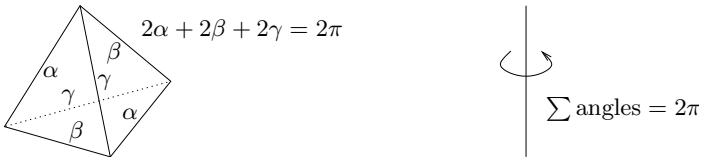


FIGURE 10. The conditions for an angle structure on a triangulation

If Regina cannot recognise the manifold from an input triangulation $\mathcal{T}$ (i.e., the combinatorial recognition is inconclusive), then often a good strategy is to modify $\mathcal{T}$ so that the triangulation becomes more "well-structured". This might include (i) simplifying $\mathcal{T}$ using Algorithm 2.5, or (ii) the slower but stronger technique of performing a breadth-first from $\mathcal{T}$ through Pachner graph until we reach a triangulation that *can* be recognised, following the discussion in Section 2.5.

Regina is not the only software package to employ combinatorial recognition: there is also the 3-Manifold Recogniser by Matveev et al., which has extremely powerful recognition heuristics that can recognise a much wider range of 3-manifolds than Regina can. See [43, 44, 45] for details.

5. Angle structures

In addition to normal surfaces, Regina can also enumerate and analyse *angle structures* on a triangulation $\mathcal{T}$. An angle structure assigns non-negative internal dihedral angles to each edge of each tetrahedron of $\mathcal{T}$, so that (i) opposite edges of a tetrahedron are assigned the same angle; (ii) all angles in a tetrahedron sum to 2π; and (iii) all angles around any internal edge of $\mathcal{T}$ likewise sum to 2π (see Figure 10). Such structures are often called *semi-angle structures* [35], to distinguish them from *strict angle structures* in which all angles are strictly positive. Note that, by a simple Euler characteristic computation, an angle structure can only exist if $\mathcal{T}$ is an ideal triangulation with every vertex link a torus or Klein bottle.

Angle structures were introduced by Rivin [50, 51] and Casson, with further development by Lackenby [38], and are a simpler (but weaker) combinatorial analogue of a complete hyperbolic structure. Some angle structures are of particular interest: these include *taut angle structures*[4] in which every angle is precisely 0 or π (representing "flattened" tetrahedra) [28, 39], and *veering structures* which are taut angle structures with powerful combinatorial constraints [1, 28]. All of these objects have an interesting role to play in building a complete hyperbolic structure on $\mathcal{T}$ [25, 28, 35].

Conditions (i)–(iii) above map out a polytope in $\mathbb{R}^{3n}$, where n is the number of tetrahedra; the vertices of this polytope are called *vertex angle structures*, and their convex combinations generate all possible angle structures on $\mathcal{T}$. For many years now, Regina has been able to enumerate all vertex angle structures using the double description method, as outlined in Section 3.2. Moreover, it can detect taut angle structures and (more recently) veering structures when they are present.

A newer development is that Regina can enumerate *only* taut angle structures. Detecting even a single taut angle structure is NP-complete [19]; nevertheless,

[4]We follow the nomenclature of Hodgson et al. [28]—these are slightly more general than the original taut structures of Lackenby [39], who also adds a coorientation constraint.

Regina can enumerate all taut angle structures for relatively large triangulations—in ad-hoc experiments it can do this for 70–80 tetrahedra in a matter of minutes.

The underlying algorithm is based on the following simple observation:

Lemma 5.1. *Every taut angle structure is also a vertex angle structure.*

Proof. Describing angle structures by vectors in $\mathbb{R}^{3n}$ as outlined above, suppose that $\tau = \lambda\alpha_1 + (1-\lambda)\alpha_2$ where $\lambda \in (0,1)$, and where τ, α_1, α_2 are angle structures with τ taut. Then both α_1 and α_2 must have dihedral angles of zero wherever τ has a dihedral angle of zero, whereupon it follows that $\alpha_1 = \alpha_2 = \tau$. □

Algorithm 5.2. *Given an n-tetrahedron triangulation $\mathcal{T}$, the following algorithm enumerates all taut angle structures on $\mathcal{T}$.*

First, we projectivise the polytope described by conditions (i)–(iii) above. This is a standard construction: we add a $(3n+1)$th coordinate, embed the original polytope $\mathcal{P}$ in the hyperplane $x_{3n+1} = 1$, and build the cone *from the origin through $\mathcal{P}$. This replaces our bounded polytope $\mathcal{P} \subseteq \mathbb{R}^{3n}$ with a polyhedral cone $\mathcal{C} \subseteq \mathbb{R}^{3n+1}$ of the form $\mathcal{C} = \{\mathbf{x} \in \mathbb{R}^{3n+1} \mid A\mathbf{x} = 0,\ \mathbf{x} \geq 0\}$. The vertex angle structures in the original polytope $\mathcal{P}$ now correspond to the extreme rays of the cone $\mathcal{C}$.*

We run the double description method to enumerate all extreme rays of $\mathcal{C}$. Recall that this inductively constructs cones $\mathcal{C}_0, \mathcal{C}_1, \ldots$, where $\mathcal{C}_i$ is obtained from $\mathcal{C}_{i-1}$ by intersecting with a new hyperplane H_i, and where each extreme ray of $\mathcal{C}_i$ is either (a) an extreme ray of $\mathcal{C}_{i-1}$ that lies on H_i, or (b) the convex combination of two adjacent extreme rays $\mathbf{x}_1, \mathbf{x}_2$ of $\mathcal{C}_{i-1}$ that lie on opposite sides of H_i.

Here we introduce a simple but crucial optimisation: in case (b) above, we only consider pairs $\mathbf{x}_1, \mathbf{x}_2$ that together *do not have positive values in more than one coordinate position per tetrahedron.*

This optimisation is similar to Letscher's filtering method for normal surface enumeration [8]. It works because, if some pair $\mathbf{x}_1, \mathbf{x}_2$ fails the final condition above, then (by virtue of the fact that all angles are non-negative and we always perform convex combinations) any vertex angle structure that we eventually obtain from a combination of $\mathbf{x}_1$ and $\mathbf{x}_2$ must have multiple non-zero coordinates in some tetrahedron, and so cannot be taut.

Finally, we note that we can improve the enumeration algorithm further, both for enumerating taut angle structures and enumerating *all* vertex angle structures, by employing the same trie-based optimisation that we introduce in Section 3.2.

6. Experimentation

In this penultimate section we illustrate how Regina can be used for both small-scale and large-scale experimentation, in the hope that readers can use this as a template for beginning their own experiments.

In addition to its graphical user interface, Regina offers a powerful scripting facility, in which most of the C++ classes and functions in its mathematical engine are made available through a dedicated Python module. Python is a popular scripting language that is easy to write and easy to read, and the Python module in Regina makes it easy to quickly prototype new algorithms, run tests over large bodies of census data, or perform complex tasks that would be cumbersome through a point-and-click interface.

Users can access Regina's Python module in two ways:

- by opening a Python console from within the graphical user interface, which allows users to study or modify data in the current working file;
- by starting the command-line program `regina-python`, which brings up a standalone Python prompt.

Users can also run their own Python scripts directly via `regina-python`, embed scripts within data files as *script packets*, or write their own libraries of frequently-used routines that will be loaded automatically each time a Regina Python session starts.

The following sample Python session constructs the triangulation of $\mathbb{R}P^3$ that was illustrated in Section 2.1, prints its first homology group, enumerates all vertex normal surfaces, and then locates and prints the coordinates of the vectors that represent vertex normal projective planes.

```
bab@rosemary:~$ regina-python
Regina 4.93
Software for 3-manifold topology and normal surface theory
Copyright (c) 1999-2012, The Regina development team
>>> tri = NTriangulation()
>>> t0 = tri.newTetrahedron()
>>> t1 = tri.newTetrahedron()
>>> t0.joinTo(0, t1, NPerm4(1,0,3,2))       # Glues 0 (123) -> 1 (032)
>>> t0.joinTo(1, t1, NPerm4(1,0,3,2))       # Glues 0 (023) -> 1 (132)
>>> t0.joinTo(2, t1, NPerm4(0,1,2,3))       # Glues 0 (013) -> 1 (013)
>>> t0.joinTo(3, t1, NPerm4(0,1,2,3))       # Glues 0 (012) -> 1 (012)
>>> print tri.getHomologyH1()
Z_2
>>> s = NNormalSurfaceList.enumerate(tri, NNormalSurfaceList.STANDARD, 1)
>>> print s
5 vertex normal surfaces (Standard normal (tri-quad))
>>> for i in range(s.getNumberOfSurfaces()):
...     if s.getSurface(i).getEulerCharacteristic() == 1:
...         print s.getSurface(i)
...
0 0 0 0 ; 0 1 0 || 0 0 0 0 ; 0 1 0
0 0 0 0 ; 0 0 1 || 0 0 0 0 ; 0 0 1
>>>
```

One of Regina's most useful facilities for experimentation is its ability to create *census data*: exhaustive lists of all 3-manifold triangulations (up to combinatorial isomorphism) that satisfy some given set of constraints. The census algorithms are heavily optimised [4, 6, 10], and can be run in serial on a desktop or in parallel on a large supercomputer.

The simplest way for users to create their own census data is through the command-line `tricensus` tool. The following example constructs all 532 closed orientable 3-manifold triangulations with $n = 4$ tetrahedra:

```
bab@rosemary:~$ tricensus --tetrahedra=4 --internal --orientable --finite
                --sigs output.txt
Starting census generation...
0:1 0:0 1:0 1:1 | 0:2 0:3 2:0 2:1 | 1:2 1:3 3:0 3:1 | 2:2 2:3 3:3 3:2
0:1 0:0 1:0 1:1 | 0:2 0:3 2:0 3:0 | 1:2 2:2 2:1 3:1 | 1:3 2:3 3:3 3:2
0:1 0:0 1:0 1:1 | 0:2 0:3 2:0 3:0 | 1:2 3:1 3:2 3:3 | 1:3 2:1 2:2 2:3
0:1 0:0 1:0 2:0 | 0:2 1:2 1:1 3:0 | 0:3 2:2 2:1 3:1 | 1:3 2:3 3:3 3:2
```

```
0:1 0:0 1:0 2:0 | 0:2 1:2 1:1 3:0 | 0:3 3:1 3:2 3:3 | 1:3 2:1 2:2 2:3
0:1 0:0 1:0 2:0 | 0:2 2:1 2:2 3:0 | 0:3 1:1 1:2 3:1 | 1:3 2:3 3:3 3:2
0:1 0:0 1:0 2:0 | 0:2 2:1 3:0 3:1 | 0:3 1:1 3:2 3:3 | 1:2 1:3 2:2 2:3
1:0 1:1 1:2 2:0 | 0:0 0:1 0:2 3:0 | 0:3 3:1 3:2 3:3 | 1:3 2:1 2:2 2:3
1:0 1:1 2:0 2:1 | 0:0 0:1 3:0 3:1 | 0:2 0:3 3:2 3:3 | 1:2 1:3 2:2 2:3
1:0 1:1 2:0 3:0 | 0:0 0:1 2:1 3:1 | 0:2 1:2 3:2 3:3 | 0:3 1:3 2:2 2:3
Finished.
Total triangulations: 532
```

The triangulations are converted to text-based isomorphism signatures [12] and written to the text file `output.txt`, one per line.

We can now (as an illustration) search through this census for all two-vertex 0-efficient triangulations of the 3-sphere:

```
bab@rosemary:~$ regina-python
Regina 4.93
Software for 3-manifold topology and normal surface theory
Copyright (c) 1999-2012, The Regina development team
>>> f = open('output.txt', 'r')
>>> sig = f.readline()
>>> while sig:
...     sig = sig[0:-1]    # Strip off trailing newline
...     tri = NTriangulation.fromIsoSig(sig)
...     if tri.getNumberOfVertices() == 2:
...         if tri.isZeroEfficient() and tri.isThreeSphere():
...             print sig
...     sig = f.readline()
...
eLAkaccddjgjqc
>>>
```

Here we see that, for $n = 4$ tetrahedra, there is one and only one such triangulation. To study it in more detail, we can open up Regina's graphical user interface and create a new triangulation from the isomorphism signature `eLAkaccddjgjqc`.

These small examples illustrate how, using the census facility and Python scripting combined, Regina can be an invaluable tool for experimentation, testing conjectures, and searching for pathological examples.

7. The future of Regina

Regina continues to enjoy active development and regular releases. The developers are currently working towards a major version 5.0 release, which will also work with triangulated *4-manifolds* and normal hypersurfaces (in joint work with Ryan Budney). Other coming developments include richer operations on triangulated 2-manifolds, visualisation of vertex links in 3-manifold triangulations (by Budney and Samuel Churchill), and much more sophisticated algebraic machinery in 2, 3 and 4 dimensions (by Budney). Much of this code is already running and well-tested.

Users are encouraged to contribute code and offer feedback. For information on new releases, interested parties are welcome to subscribe to the low-traffic mailing list `regina-announce@lists.sourceforge.net`.

References

1. Ian Agol, *Ideal triangulations of pseudo-Anosov mapping tori*, Topology and Geometry in Dimension Three, Contemp. Math., vol. 560, Amer. Math. Soc., Providence, RI, 2011, pp. 1–17.
2. Bruno Benedetti and Günter M. Ziegler, *On locally constructible spheres and balls*, Acta Math. **206** (2011), no. 2, 205–243.
3. Winfried Bruns and Bogdan Ichim, *Normaliz: Algorithms for affine monoids and rational cones*, J. Algebra **324** (2010), no. 5, 1098–1113.
4. Benjamin A. Burton, *Face pairing graphs and 3-manifold enumeration*, J. Knot Theory Ramifications **13** (2004), no. 8, 1057–1101.
5. ———, *Introducing Regina, the 3-manifold topology software*, Experiment. Math. **13** (2004), no. 3, 267–272.
6. ———, *Enumeration of non-orientable 3-manifolds using face-pairing graphs and union-find*, Discrete Comput. Geom. **38** (2007), no. 3, 527–571.
7. ———, *Converting between quadrilateral and standard solution sets in normal surface theory*, Algebr. Geom. Topol. **9** (2009), no. 4, 2121–2174.
8. ———, *Optimizing the double description method for normal surface enumeration*, Math. Comp. **79** (2010), no. 269, 453–484.
9. ———, *Quadrilateral-octagon coordinates for almost normal surfaces*, Experiment. Math. **19** (2010), no. 3, 285–315.
10. ———, *Detecting genus in vertex links for the fast enumeration of 3-manifold triangulations*, ISSAC 2011: Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation, ACM, 2011, pp. 59–66.
11. ———, *The Pachner graph and the simplification of 3-sphere triangulations*, SCG '11: Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry, ACM, 2011, pp. 153–162.
12. ———, *Simplification paths in the Pachner graphs of closed orientable 3-manifold triangulations*, Preprint, `arXiv:1110.6080`, October 2011.
13. ———, *Enumerating fundamental normal surfaces: Algorithms, experiments and invariants*, ALENEX 2014: Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, SIAM, 2014, pp. 112–124.
14. Benjamin A. Burton, Ryan Budney, William Pettersson, et al., *Regina: Software for 3-manifold topology and normal surface theory*, `http://regina.sourceforge.net/`, 1999–2012.
15. Benjamin A. Burton and Melih Ozlen, *Computing the crosscap number of a knot using integer programming and normal surfaces*, ACM Trans. Math. Software **39** (2012), no. 1, 4:1–4:18.
16. ———, *A fast branching algorithm for unknot recognition with experimental polynomial-time behaviour*, Preprint, `arXiv:1211.1079`, November 2012.
17. ———, *A tree traversal algorithm for decision problems in knot theory and 3-manifold topology*, Algorithmica **65** (2013), no. 4, 772–801.
18. Benjamin A. Burton, J. Hyam Rubinstein, and Stephan Tillmann, *The Weber-Seifert dodecahedral space is non-Haken*, Trans. Amer. Math. Soc. **364** (2012), no. 2, 911–932.
19. Benjamin A. Burton and Jonathan Spreer, *The complexity of detecting taut angle structures on triangulations*, SODA '13: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2013, pp. 168–183.
20. Patrick J. Callahan, Martin V. Hildebrand, and Jeffrey R. Weeks, *A census of cusped hyperbolic 3-manifolds*, Math. Comp. **68** (1999), no. 225, 321–332.
21. Maurice Chiodo, *Finding non-trivial elements and splittings in groups*, J. Algebra **331** (2011), 271–284.
22. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.
23. Marc Culler, Nathan M. Dunfield, and Jeffrey R. Weeks, *SnapPy, a computer program for studying the geometry and topology of 3-manifolds*, `http://snappy.computop.org/`, 1991–2011.
24. Komei Fukuda and Alain Prodon, *Double description method revisited*, Combinatorics and Computer Science (Brest, 1995), Lecture Notes in Comput. Sci., vol. 1120, Springer, Berlin, 1996, pp. 91–111.

25. David Futer and François Guéritaud, *From angled triangulations to hyperbolic structures*, Interactions Between Hyperbolic Geometry, Quantum Topology and Number Theory, Contemp. Math., vol. 541, Amer. Math. Soc., Providence, RI, 2011, pp. 159–182.
26. Wolfgang Haken, *Theorie der Normalflächen*, Acta Math. **105** (1961), 245–375.
27. Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger, *The computational complexity of knot and link problems*, J. Assoc. Comput. Mach. **46** (1999), no. 2, 185–211.
28. Craig D. Hodgson, J. Hyam Rubinstein, Henry Segerman, and Stephan Tillmann, *Veering triangulations admit strict angle structures*, Geom. Topol. **15** (2011), no. 4, 2073–2089.
29. Craig D. Hodgson and Jeffrey R. Weeks, *Symmetries, isometries and length spectra of closed hyperbolic three-manifolds*, Experiment. Math. **3** (1994), no. 4, 261–274.
30. William Jaco, David Letscher, and J. Hyam Rubinstein, *Algorithms for essential surfaces in 3-manifolds*, Topology and Geometry: Commemorating SISTAG, Contemporary Mathematics, no. 314, Amer. Math. Soc., Providence, RI, 2002, pp. 107–124.
31. William Jaco and Ulrich Oertel, *An algorithm to decide if a* 3*-manifold is a Haken manifold*, Topology **23** (1984), no. 2, 195–209.
32. William Jaco and J. Hyam Rubinstein, *PL equivariant surgery and invariant decompositions of 3-manifolds*, Adv. Math. **73** (1989), no. 2, 149–191.
33. ———, *0-efficient triangulations of 3-manifolds*, J. Differential Geom. **65** (2003), no. 1, 61–168.
34. William Jaco and Jeffrey L. Tollefson, *Algorithms for the complete decomposition of a closed* 3*-manifold*, Illinois J. Math. **39** (1995), no. 3, 358–406.
35. Ensil Kang and J. Hyam Rubinstein, *Ideal triangulations of 3-manifolds II; Taut and angle structures*, Algebr. Geom. Topol. **5** (2005), 1505–1533.
36. Hellmuth Kneser, *Geschlossene Flächen in dreidimensionalen Mannigfaltigkeiten*, Jahresbericht der Deut. Math. Verein. **38** (1929), 248–260.
37. Marc Lackenby, *An algorithm to determine the Heegaard genus of simple 3-manifolds with nonempty boundary*, Algebr. Geom. Topol. **8** (2008), no. 2, 911–934.
38. ———, *Word hyperbolic Dehn surgery*, Invent. Math. **140** (2000), no. 2, 243–282.
39. ———, *Taut ideal triangulations of 3-manifolds*, Geom. Topol. **4** (2000), 369–395.
40. A. A. Markov, *Insolubility of the problem of homeomorphy*, Proc. Internat. Congress Math. 1958, Cambridge Univ. Press, New York, 1960, pp. 300–306.
41. Bruno Martelli and Carlo Petronio, *Three-manifolds having complexity at most 9*, Experiment. Math. **10** (2001), no. 2, 207–236.
42. S. V. Matveev, *Transformations of special spines, and the Zeeman conjecture*, Izv. Akad. Nauk SSSR Ser. Mat. **51** (1987), no. 5, 1104–1116, 1119.
43. Sergei Matveev, *Algorithmic topology and classification of 3-manifolds*, Algorithms and Computation in Mathematics, no. 9, Springer, Berlin, 2003.
44. ———, *3-Manifold Recognizer and 3-Manifold Atlas*, Oberwolfach Rep. **9** (2012), no. 2, 1409–1410.
45. Sergei Matveev et al., *Manifold recognizer*, `http://www.matlas.math.csu.ru/?page=recognizer`, accessed August 2012.
46. Sergei V. Matveev, *Complexity theory of three-dimensional manifolds*, Acta Appl. Math. **19** (1990), no. 2, 101–130.
47. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, *The double description method*, Contributions to the Theory of Games, Vol. II (H. W. Kuhn and A. W. Tucker, eds.), Annals of Mathematics Studies, no. 28, Princeton University Press, Princeton, NJ, 1953, pp. 51–73.
48. Nokia Corporation, *Qt – cross-platform application and UI framework*, `http://qt.nokia.com/`, 1992–2012.
49. Udo Pachner, *P.L. homeomorphic manifolds are equivalent by elementary shellings*, European J. Combin. **12** (1991), no. 2, 129–145.
50. Igor Rivin, *Euclidean structures on simplicial surfaces and hyperbolic volume*, Ann. of Math. (2) **139** (1994), no. 3, 553–580.
51. ———, *Combinatorial optimization in geometry*, Adv. in Appl. Math. **31** (2003), no. 1, 242–271.
52. J. Hyam Rubinstein, *An algorithm to recognize the* 3*-sphere*, Proceedings of the International Congress of Mathematicians (Zürich, 1994), vol. 1, Birkhäuser, 1995, pp. 601–611.
53. ———, *An algorithm to recognise small Seifert fiber spaces*, Turkish J. Math. **28** (2004), no. 1, 75–87.

54. Peter Scott, *The geometries of 3-manifolds*, Bull. London Math. Soc. **15** (1983), no. 5, 401–487.
55. Robert Sedgewick, *Algorithms in C++*, Addison-Wesley, Reading, MA, 1992.
56. Robert Endre Tarjan, *Efficiency of a good but not linear set union algorithm*, J. ACM **22** (1975), no. 2, 215–225.
57. Abigail Thompson, *Thin position and the recognition problem for S^3*, Math. Res. Lett. **1** (1994), no. 5, 613–630.
58. William P. Thurston, *The geometry and topology of 3-manifolds*, Lecture notes, Princeton University, 1978.
59. Jeffrey L. Tollefson, *Normal surface Q-theory*, Pacific J. Math. **183** (1998), no. 2, 359–374.
60. Jeffrey R. Weeks, *SnapPea: Hyperbolic 3-manifold software*, `http://www.geometrygames.org/SnapPea/`, 1991–2000.

School of Mathematics and Physics, The University of Queensland, Brisbane QLD 4072, Australia

E-mail address: `bab@maths.uq.edu.au`