Polymorph Segmentation Representation for Medical Image Computing

Csaba Pinter[1], Andras Lasso[1], Gabor Fichtinger[1]

[1]Laboratory for Percutaneous Surgery, School of Computing, 557 Goodwin Hall, Queen's University, K7L 2N8, Kingston, Ontario, Canada

Corresponding author: Csaba Pinter, csaba.pinter@queensu.ca, +1-613-893-2706

Abstract

Background and Objective: Segmentation is a ubiquitous operation in medical image computing. Various data representations can describe segmentation results, such as labelmap volumes or surface models. Conversions between them are often required, which typically include complex data processing steps. We identified four challenges related to managing multiple representations: conversion method selection, data provenance, data consistency, and coherence of in-memory objects. **Methods:** A complex data container preserves identity and provenance of the contained representations and ensures data coherence. Conversions are executed automatically on-demand. A graph containing the implemented conversion algorithms determines each execution, ensuring consistency between various representations. The design and implementation of a software library are proposed, in order to provide a readily usable software tool to manage segmentation data in multiple data representations. A low-level core library called PolySeg implemented in The Visualization Toolkit (VTK) manages the data objects and conversions. It is used by a high-level application layer, which has been implemented in the medical image visualization and analysis platform 3D Slicer. The application layer provides advanced visualization, transformation, interoperability, and other functions. **Results:** The core conversion algorithms comprising the graph were validated. Several applications were implemented based on the library, demonstrating advantages in terms of usability and ease of software development in each case. The Segment Editor

application provides fast, comprehensive, and easy-to-use manual and semi-automatic segmentation workflows. Clinical applications for gel dosimetry, external beam planning, and MRI-ultrasound image fusion in brachytherapy were rapidly prototyped resulting robust applications that are already in use in clinical research. The conversion algorithms were found to be accurate and reliable using these applications. **Conclusions:** A generic software library has been designed and developed for automatic management of multiple data formats in segmentation tasks. It enhances both user and developer experience, enabling fast and convenient manual workflows and quicker and more robust software prototyping. The software's BSD-style open-source license allows complete freedom of use of the library.

Keywords: Segmentation, Software library, Open-source, 3D Slicer, Voxelization, DICOM.

1. Introduction

Segmentation, which is the process of delineating structures of interest, is a critically important task in many medical image computing and visualization workflows: it enables quantitative analysis inside structures such as organs and malignancies, facilitates therapy planning, and provides better understanding of patient anatomy. Various data structures are available to represent segmentation results. Unfortunately none of them are optimal for storage, analysis, and real-time visualization at the same time, so a trade-off is typically made to choose the most suitable representation for the main purpose of a specific application.

Most commonly, segmentation results are stored in 3D binary **labelmap volume** (shown in Fig. 1/A) where the value of each voxel indicates if that point is inside or outside the structure. This representation is the usual output of manual segmentation and preferred input for image processing and analysis algorithms. However, for 3D visualization of structures a **surface model** (Fig. 1/B) is more optimal, as it can be rendered by graphics cards efficiently. A surface model is a mesh consisting of tens or hundreds of

thousands of connected triangles. Certain segmentation algorithms yield **fractional labelmaps** (Fig. 1/C) with voxels indicating probabilities (Iglesias and Sabuncu, 2015) instead of a binary decision. For radiation therapy, the DICOM standard (Mildenberger et al., 2002) requires the structures to be stored as "structure sets" - a series of **planar contours** (Fig. 1/D). Structure sets may also be represented as **ribbons** (Fig. 1/E), which can be easily computed from contours, indicate slice thickness, and make 3D shapes more recognizable.

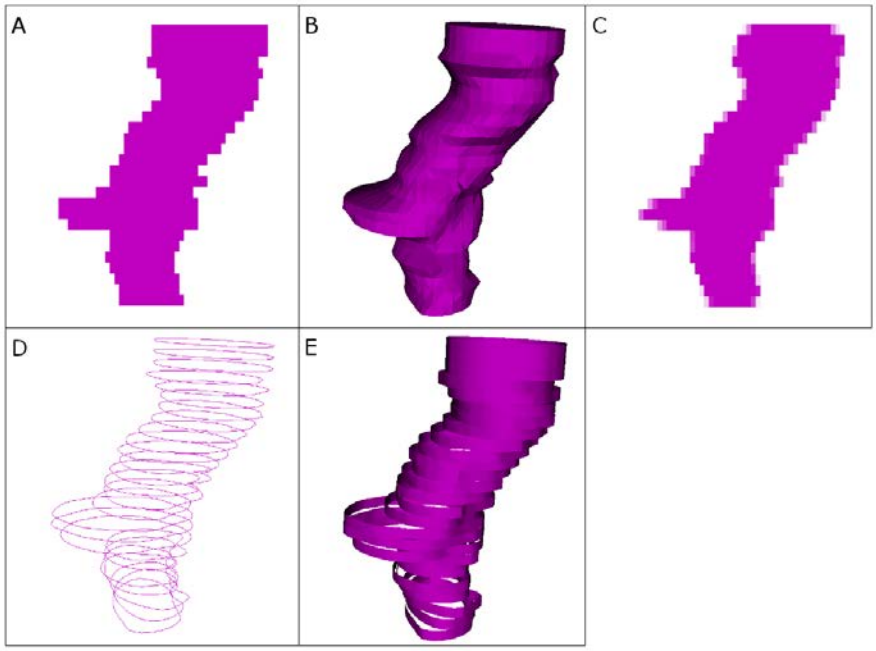

Fig. 1: Different representations of the same brain stem structure. A: Binary labelmap, B: Closed surface, C: Fractional labelmap, D: Planar contours, E: Ribbon model

In most workflows, it is not sufficient to use a single representation of segmentation data. An example for the need of multiple representations is the calculation of dose volume histograms (DVH) for radiation therapy, which are used for treatment plan evaluation and optimization. Targets to treat and organs to avoid are segmented by delineating their cross-section in multiple slices of the image, which results in a list of planar contours. Contours are then voxelized into labelmaps, so that they can be used as masks for dose distribution volumes and computing the histograms. For 3D visualization closed surfaces are shown, which can be created either from contours or labelmaps.

There are challenges involved in the design and implementation of storage and analysis of anatomical structures that must be addressed in all software applications:

1. Conversion method selection: The need for conversion in order to prepare segmentation data for certain operations is something to be recognized by the user. Further, some details are often lost or altered during conversion. It is important to make users aware of these changes, while at the same time keep the software simple and easy to use.
2. Provenance: The identity and origin of the structures and what they represent. In a typical research workflow not driven by a rigid processing pipeline it is up to the user to maintain the objects participating in the workflow. Often, this results in an unorganized set of objects with arbitrary names. When working with more than a few structures it quickly becomes unclear where each object originates from and what transformations they went through.
3. Consistency: Representations may change after conversions (*e.g.* by manual editing), in which case the other representations of the same structure become invalid, and the data scene inconsistent. It is imperative to make sure that no invalid data is accessible at any time.
4. Coherence: Structure sets typically correspond to the same entity (i.e., a patient), so when objects are stored in memory or disk, processed, or visualized it should be possible to manage them as a unified whole.

Extensive research has been performed towards developing fast and accurate conversion algorithms between representations (Congalton, 1997; Fuchs et al., 1977; Jian Huang et al., 2014; Meyers et al., 1992; Novotný et al., 2010; Schroeder et al., 2015, etc.) for various applications ranging from geography, through computer graphics to medical image computing. To our knowledge, however, no investigation has been done in terms of a complex workflow involving multiple data representations and the challenges identified earlier. Keeping the data in good shape is essential, especially if a human user is involved, or the workflow steps' order is not fixed. By making these conversion methods more accessible and robust, the time spent

in the workflow can be reduced considerably. An example is the dosimetry workflow by Alexander et al. (2018), in which the time of analysis was reduced from hours to minutes.

This paper proposes a software system methodology and implementation for *Polymorphic Segmentation Representation* to facilitate dynamic management of multiple data formats representing the same structure in a way that addresses the identified challenges involved in the process. It aspires to open the way to rapid prototyping of more robust and user-friendly software for medical image analysis research and related clinical applications.

2. Computational methods and theory

A complex data structure called a *segmentation object* is proposed that unites different representations of segmented structures. It provides automatic conversion between representation types. New representation types can be dynamically added to the system. The software design is the result of a consensus reached through discussions with numerous researchers and developers participating in the Project Week open-source programming event series (Kapur et al., 2016).

2.1. Segmentation object

A *segmentation object* contains multiple structures and representations in one object, as shown in Fig. 2. Each structure in a segmentation object is a *segment*, which can contain multiple representations. To simplify software implementation and user interfaces, we decided to constrain each segment to contain the same set of representations. Segments contain their basic properties such as name and color, as well as a dictionary for storing any additional metadata, such as standard medical terminology.

One representation is chosen as *master representation*, which is typically the one that the data was originally created in. For example, when segmenting manually using a paintbrush-like tool, it is a binary labelmap. The significance of master representation is that it is the only data representation that must be

persistently stored, and all other representations are derived from it via conversion. Since no conversion was performed to create the master representation, it is a lossless representation, which most closely represents the experts' opinion or the original algorithm output.

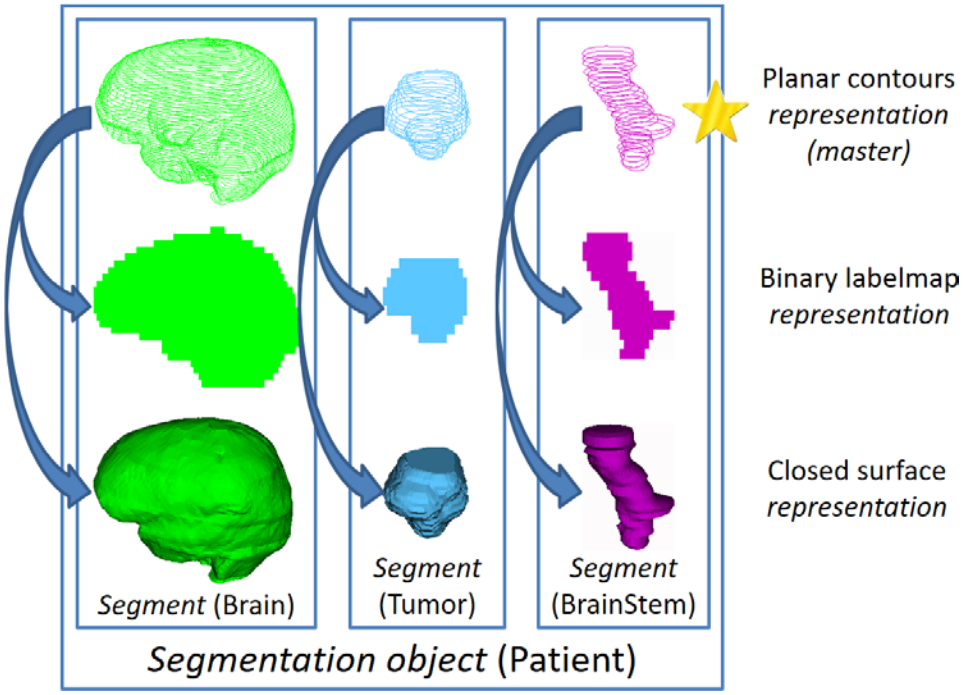

Fig. 2. Segmentation object storing each structure of an entity (patient) with each representation in one data object. The example segmentation of a patient contains three segments: brain, tumor, and brain stem. Each segment contains three representations: planar contour, binary labelmap, and closed surface. The master representation, which is the original representation of the structures, is marked with a star.

2.2. Automatic conversion

Most clinical or research workflows dealing with segmented structures include steps that make conversion of the segmentation data necessary. In commercial software, conversions are built-in steps of the fixed processing workflow. However, in research software, usually there are no fixed workflows, and the users can perform steps in arbitrary order. They are responsible for performing conversions as needed. It is a tedious and potentially error-prone task involving manual specification of the input, output, and conversion parameters. This manual step is the source of both the Conversion method selection (1) and Provenance (2) challenges, which can be solved by automation. When a representation is requested by a

workflow step, then automatic conversion takes place. Examples of such steps: a) Execution of a processing pipeline which uses a yet absent representation in one of its steps, b) Change of display settings such as showing surface mesh or adjusting the level of smoothing, c) User exporting segmentation for 3D printing. The Conversion method selection challenge is thus solved by either obscuring the fact of conversion from the users or reducing the number of steps to take. Storing all representations within the same segmentation object solves the Provenance challenge by taking the burden of managing the atomic representation objects off the user. Encapsulation solves the Coherence (4) problem as well, because it makes managing the representations as one entity (*e.g.* a patient) straightforward.

Automatic conversion between the different representations is driven by a *conversion graph*. The graph's nodes are the representations, and the directed edges are conversion algorithms; a typical graph is shown in Fig. 3. Each algorithm – called a *converter rule* – has the following properties: source and target representation type, cost, and parameters. Cost is an approximate relative value roughly representing the duration of a typical conversion step using a particular converter and settings. *Parameters* are a set of key-value pairs for each converter rule. For example, the binary labelmap to closed surface rule has three parameters: strength of smoothing, strength of decimation, and computation of surface normal vectors. Upon a conversion request the graph performs a search for the cheapest path from the master representation (which is always the source) to the requested one. The graph then executes the converter rules comprising the path one by one. The user has the option to manually override both the path and the parameters. It could be desirable to perform the search for maximum fidelity instead of conversion time. In this case, the cost metric should represent how lossy the particular algorithm is with the current parameters. This option will be explored in future work.

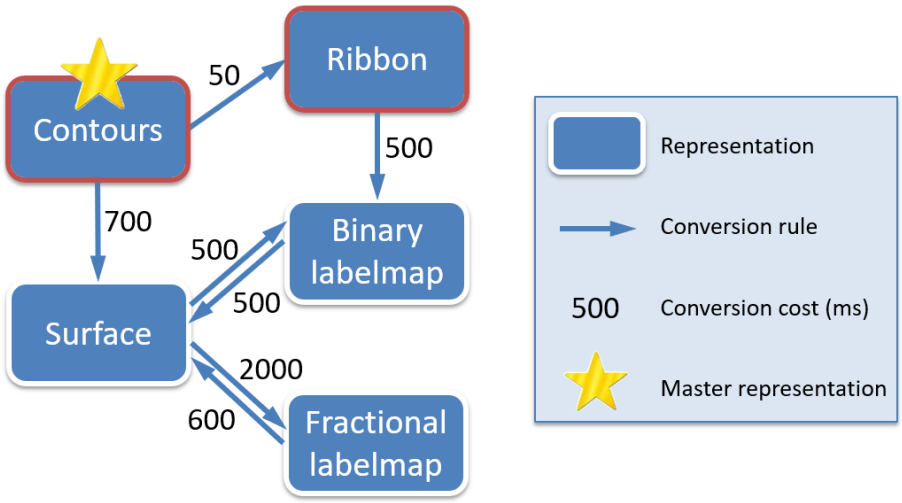

Fig. 3. Example of a conversion graph. The nodes are the representations and the edges are the conversion algorithms (rules). The numbers are the approximate costs of each conversion rule. Star marks the master representation. Red frame indicates the rules that are added as an extension.

Another important task is to ensure consistency of the encapsulated data representations. When a representation changes, then all the derived (*i.e.* converted) representations become invalid, as they contain outdated information. This is prevented by removing the outdated representations, which is enough as any absent representation is automatically re-converted on demand. Thus, the Consistency (3) challenge is also addressed. User experience could be improved by re-converting the invalid representations (preferably in the background) after their removal. This will be addressed in future work.

Researchers and developers may work with data representations not supplied in the default implementation, and may wish to add their representations to the graph. In this case, they need to implement conversion rules from/to another representation. Once it is registered in the graph, any conversion targeting or originating from their representation can be performed. Planar contours and ribbons serve as examples for such representations, which are an extension to the core library (see representations with red border in Fig. 3). If a new conversion rule parallel to an existing one is added, then the cost will determine which one gets selected. To force using a conversion rule, it is possible to unregister the undesired rule from the graph.

3. System description

3.1. Core library

The basic storage and conversion features are implemented in a software library called *PolySeg*[1]. It includes the segmentation object and segment classes, as well as those responsible for conversion. The core library is written entirely in C++ and is based on the Visualization Toolkit (VTK) (Schroeder et al., 2004). VTK is well-established and one of the most widely used software libraries in the field of medical image computing: MITK (Nolden et al., 2013), MeVisLab (Ritter et al., 2011), medInria (Toussaint et al., 2006), NIfTI (Cox et al., 2004), etc. VTK is also used outside the field of medical applications, such as in geography and chemistry.

VTK being the only dependency of PolySeg facilitates its integration into various software applications. The core library contains its own set of automated tests, verifying the correct operation of the conversion graph, and the coherence of the segmentation object after various operations. Thus, PolySeg is a full-fledged software library ready to be used in software solutions.

3.2. Application layer – integration into 3D Slicer

The initial motivation for creating PolySeg was to be able to manage the increasingly complex configuration of data objects for the continually emerging use cases for the SlicerRT open-source radiation therapy research toolkit (Pinter et al., 2012). SlicerRT is an extension of the widely used medical image visualization and analysis application platform 3D Slicer (Fedorov et al., 2012). Thus, the first end-user application adopting this library is 3D Slicer.

[1] https://github.com/PerkLab/PolySeg

In order for the library to be utilized in an end-user application, higher-level features were required, such as visualization, file storage, etc. These functions were implemented as an application layer within the 3D Slicer platform, mostly written in C++ and partially Python. The medium of data management in 3D Slicer is the medical reality modeling language (MRML). Thus, the segmentation object was embedded in a MRML node (*vtkMRMLSegmentationNode*, see architecture in Fig. 4). Reusable user interface (UI) widgets have been implemented for handling certain aspects of a segmentation object: list of segments and their properties, representations and conversions, conversion paths and parameters, and display properties. These elements comprise an application module in which a segmentation node can be explored and the segment properties and representations managed. A logic class provides operations such as import/export to other formats, bulk transformation using rigid or deformable transforms, etc. This layer also includes classes for 2D and 3D display, file storage, and automated testing. Application level testing covers data import/export, legacy operations such as creation or merged labelmap, and correct operation of certain widgets. SlicerRT includes tests that exercise PolySeg indirectly through various DVH calculation scenarios, validating them against other software expecting to exceed an accuracy threshold.

An important component of the application layer is the Segment Editor UI widget, which provides manual and semi-automated segmentation methods to create and edit segmentations. Numerous editor "effects" have been included, each of which is a tool for segmentation, such as Paint, Threshold, Grow from seeds, Smoothing, etc. New editor effects can be – and have been – added in 3D Slicer extensions.

The main contribution of these components in 3D Slicer is tighter integration of the segmentation related features and centralized management of segmentation data. The traditional data types storing segmentation data were labelmap nodes for one or more structures, and model nodes containing one structure in planar contours, ribbons, or closed surface. Conversion between these nodes was only possible using individual modules for one particular conversion step. Modules performing analysis had to

have a fixed input representation type. PolySeg facilitates easier data selection and reduces the number of modules needed to be used for a single task by allowing modules to simply work on a segmentation node, and by providing low-level functionality such as conversions internally and automatically.

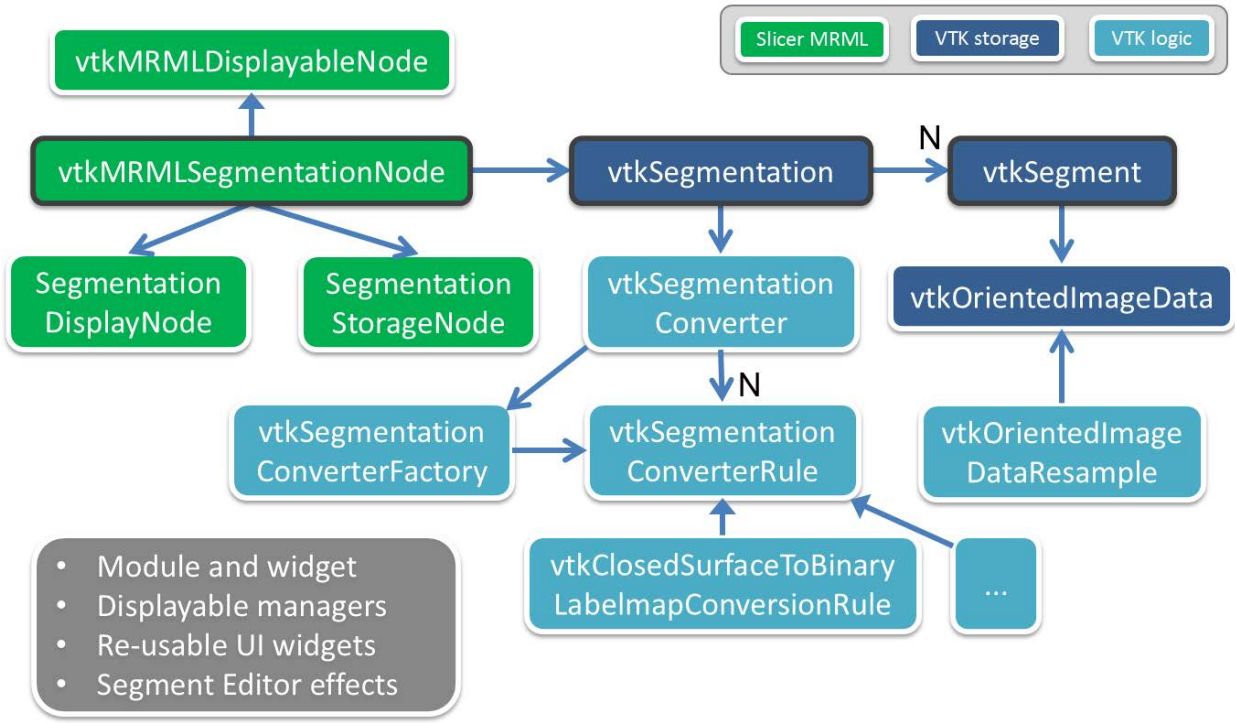

Fig. 4. Simplified software architecture diagram of PolySeg. Dark blue: Storage classes representing a segmentation object. Light blue: Logic classes taking part in the automatic conversion. The two blue groups comprise PolySeg core. Green: Application classes in 3D Slicer for MRML storage and display. Grey: Miscellaneous class groups not included in the diagram, such as the various classes comprising the Segmentations module, classes displaying the representations in the desired form in 2D and 3D, and the widgets listed in 3.2. 'N' denotes associations between one object to multiple objects.

3.3. Interoperability

Because VTK is the only dependency of the core library, it is possible to integrate it to other platforms and applications as well. To our knowledge, the MITK developers have also started adapting PolySeg into MITK-based applications. VTK's Python wrapping function provides full Python accessibility for PolySeg, allowing complex end-user applications or automated workflows to be written using the Python language. Using the Matlab Bridge extension 3D Slicer can connect to Matlab, and send and receive data seamlessly between the two applications. Interoperability with the Insight Toolkit (ITK) (Yoo et al., 2002) is currently

reached via conversion functions in SlicerRT's vtkSlicerRtCommon class. In future work, segmentation node support will be added to the ITK-based Command Line Processing interface adapted by 3D Slicer, MITK, MeVisLab, etc. Support of standard clinical dictionaries (*i.e.* terminologies) such as SNOMED Clinical Terms (Stearns et al., 2001) provides unambiguous identification of the segments, and facilitates lookup of structures.

Segmentations can be imported from and exported to DICOM SEG and DICOM RT, providing direct connection to clinical software systems. Research formats are also supported, such as nrrd for labelmaps, and stereolithography (STL) for surfaces. Geometry in each case is unambiguously described in the file headers for DICOM and nrrd. The STL files contain the vertex coordinates in 3D Slicer's mm-based world coordinate system. Data provenance tracking at export is ensured within the capabilities of the output format: DICOM objects keep their patient and study information, while nrrd and STL files reflect the names of the MRML nodes and/or segments.

3.4. Conversion algorithms

The edges in the graph of Fig. 3 show the conversion rules implemented. This set of conversion rules cover all use cases encountered in radiation therapy workflows, and based on the feedback of the 3D Slicer community between 2011 and 2018.

Binary labelmap to closed surface is the most frequently used conversion rule. It is executed every time the operator makes changes to a segmentation using Segment Editor while 3D visualization is enabled. The converter employs the discrete flying edges algorithm (Schroeder et al., 2015), followed by optional decimation (to keep number of vertices reasonably low) and smoothing (to remove staircase artifacts).

It may be desirable to keep the segments non-overlapping if they were touching each other in the input labelmap. As the conversion algorithms apply on individual segments, it can only be achieved in regular

conversion with smoothing turned off. If it is necessary to keep this property while doing smoothing, the *joint smoothing* option in Segment Editor can be used (see 4.1).

The **closed surface to binary labelmap** algorithm is typically used when the input data is a surface mesh, but labelmap is needed for analysis, *e.g.* when anatomy atlases stored as meshes need to be edited or used for statistical analysis. After pre-processing steps ensuring that the surface mesh only contains triangles with correct normal vectors, a VTK surface to image algorithm performs the conversion.

Fractional labelmap is a special labelmap, in which the voxels contain a fractional value instead of a binary in/out state. The fractional values may indicate partial occupancy at the edge of a structure, the probability of being inside the structure, or the probability of the presence of abnormal tissue (Iglesias and Sabuncu, 2015). Sunderland et al. (2017) developed the conversion rules between **fractional labelmap and closed surface** representations.

The **planar contours to closed surface** conversion provides interpolation of the contours in the direction perpendicular to the contour planes for accurate recovery of the original structure. This is the most complex of the provided converters, as triangulation needs to support special cases such as branching structures, hollow “keyhole” contours, sudden changes, and sealing the mesh with “end-capping” (Sunderland et al., 2015).

The **planar contour to ribbon model** algorithm is a conversion method for fast and crude visualization of planar contours. It creates ribbon-like triangulations from contours with width equal to the distance between the contours (Fig. 1/E). The last two algorithms were implemented as extensions to PolySeg within SlicerRT.

3.5. Accuracy and performance of the conversion algorithms

Quantitative validation of the two most critical conversion rules – binary labelmap to and from closed surface – is possible through a round-trip conversion from labelmap to closed surface and back, and comparing the input and output labelmaps. Hausdorff distance (Huttenlocher et al., 1993) was measured on three radiation therapy plans[2]; an ear-nose-throat ("ENT") and a prostate RANDO® phantom ("PROS"), and a brain fractionated stereotactic clinical plan ("FSRT"), as shown in Table 1.

	ENT	***PROS***	***FSRT***
Mean Hausdorff Max. (mm)	2.16	1.18	0.92
Mean Hausdorff 95% (mm)	1.16	0.98	0.66
Mean Hausdorff Avg. (mm)	1.13	0.11	0.22

Table 1. Comparison of the input and output binary labelmaps of a round-trip labelmap-surface-labelmap conversion

The reference grid used for voxelization was the CT image of each study, and the voxel size for the ENT, PROS, and FSRT binary labelmaps were 1.07 x 1.07 x 2.5mm^3, 0.98 x 0.98 x 2.5mm^3, and 0.68 x 0.68 x 1mm^3, respectively. Default conversion parameters were used, meaning moderate smoothing on the created closed surface. The Hausdorff distances suggest that the difference between the input and output of the round-trip is less than one voxel, meaning that both conversions were accurate.

The fractional labelmap to and from closed surface converters were validated in the paper introducing fractional labelmaps (Sunderland et al., 2017). To evaluate the closed surface to fractional labelmap conversion the fractional labelmaps were compared to the binary labelmaps, and were found to be more accurate representing structures than binary labelmaps at the same resolution: an average of 7% increase in total volume accuracy was measured. Derived DVHs were also assessed, and an average accuracy improvement of 5% and 3.6% were found against a commercial treatment planning system and a radiation therapy research toolkit, respectively. A round-trip conversion helped evaluate the fractional labelmap to

[2] All data used for validation are available in the SlicerRT data repository: https://github.com/SlicerRt/SlicerRtData

closed surface algorithm, finding an average of 0.7mm improvement in maximum point-to-point distance considering all structures, compared to round-trip conversion through binary labelmaps.

4. Samples of typical system

It has been shown through various applications that building on PolySeg yields robust and versatile software in a short development time. The following applications are based on 3D Slicer, being the first fully supported platform utilizing the PolySeg features. However, similar applications are possible to develop using PolySeg directly, or after integration to a similar medical image computing toolkit.

4.1. Segment editor

The most intuitive use case is manual and semi-automated segmentation from anatomical volumes, which has a wide range of applications, such as visualization, quantification, 3D printing, surgical planning, machine learning training, etc. Labelmap volume – *i.e.* a single 3D image containing multiple structures (labels) as its voxel values – are typically used in segmentation tools, such as 3D Slicer's legacy Editor module, or ITK-SNAP (Yushkevich et al., 2006). While it is tempting to choose labelmap volume as master representation due to its simplicity and memory efficiency, the fact that it does not support overlapping labels makes it impossible to use it as a generic storage format. Using individual volumes for segments allows having overlapping segments and storing masks or intermediate processing results in the same segmentation object. Segments may contain each other (*e.g.* body > brain > planning target volume > clinical target volume > gross tumor volume), and margins can be added without the risk of losing data in other structures. Segmentation objects have comparable storage cost to labelmap volumes in most cases thanks to only storing the "effective extent" in memory (the region that contains non-zero voxels).

We developed the *Segment Editor* module within 3D Slicer. Besides the advantages provided by overlapping structure support, a major additional feature is real-time visualization of the 3D surface

created from the edited binary labelmap. The automatic conversion mechanism makes sure that every time the master binary labelmap is changed, the surface is also updated, allowing the user to see changes almost in real-time without user interaction. The reworked architecture enabled a general speed-up and richer features for traditionally existing Editor "effects" such as *Paint*, which was given a 3D brush and the possibility to paint on non-axis-aligned (oblique) slices, or *Grow from seeds* (a.k.a. GrowCut (Zhu et al., 2014)), which now features preview and correction. It also enabled the developers to add novel tools, such as *Scissors*, which cuts the space using a projection through a drawn shape in 2D or 3D, or *Surface cut*, which enables creating surfaces from points in space. Various smoothing operations are available such as *Gaussian smoothing* or *Joint smoothing*, which achieves smoothing without shrinking the structure and thus preserve touching surfaces (Taubin et al., 1996). Segment Editor is a reusable widget that can be added into any 3D Slicer module. Additionally, the *Terminologies* feature allows assigning standard terminology entries to the segments, meaning that instead of relying on arbitrary names, standard codes can be assigned (*e.g.* type: Morphologically altered structure / Necrosis, region: Cerebral cortex / Left), which allows automatic look-up and grouping of structures in a database without curation or heuristics.

Since its release in June 2016, we experienced a high interest in Segment Editor. The majority of the topics in the 3D Slicer forum (https://discourse.slicer.org, introduced in April 2017) are related to segmentation and the usage of Segment Editor. This is illustrated by the tags given to the topics: "segmentation" is the most frequent tag with 169 occurrences, followed by tags of 68, 65, and 46 occurrences (as of July 2018). The release also correlates with an apparent steepening of the 3D Slicer download statistics, see Fig. 5.

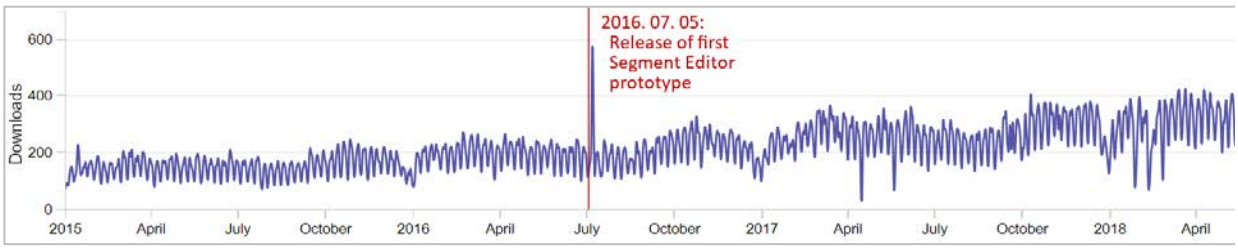

Fig. 5: 3D Slicer downloads between January 2015 and July 2018. The frequent jumps corresponds to workdays and weekends. Downloads spiked when the Segment Editor module was released, and the increase of downloads per day accelerated.

4.2. Batch structure set conversion

Conversion of a large number of datasets from one format to another is a frequent operation. A command-line script for a particular conversion from DICOM RT structure sets (planar contours) to nrrd (labelmap volumes) has been implemented within SlicerRT. It uses PolySeg to convert from planar contours to binary labelmap through closed surface, and populate the output images. The script demonstrates how the method and implementation scales and extends to larger collections of segmentations across a population, or to statistical anatomical atlases.

4.3. MRI-US contour propagation for prostate cancer biopsy and brachytherapy planning

Brachytherapy is a treatment modality based on placing radioactive point sources into a tumor. An alternative to treating the whole prostate is to irradiate the intraprostatic lesion. Many clinics use ultrasound for planning, which is not effective in identifying lesions, however MRI allows accurate segmentation of the gland, lesion, and urethra. A clinical application has been developed and evaluated (Poulin et al., 2017) for automated fusion of the MRI and US delineations. Prostate contours are used to register the modalities, after which the MRI structures are warped to the ultrasound frame of reference. Finally, the warped MRI structures are exported to DICOM for use in treatment planning and catheter insertion guidance. PolySeg greatly facilitated the development process in terms of geometric pre-processing, conversions, import/export, bulk transformation, resampling, and advanced visualization.

4.4. Gel dosimetry analysis

An emerging 3D dosimetry technique called gel dosimetry aspires to replace the conventional one- and two-dimensional methods, so that dose delivery verification can be performed on complex radiation

techniques. A streamlined clinical application has been developed and validated (Alexander et al., 2018) to replace the lengthy and complex in-house scripts previously used, reducing the duration of the analysis from 3-4 hours to 10 minutes. PolySeg helped reduce code complexity and development time.

4.5. Open-source external beam radiation therapy treatment planning system

Commercial radiation therapy treatment planning systems (TPS) are single-purpose applications optimized for fixed workflows in an extremely robust and directed way. This makes them challenging to use for testing and evaluating new techniques. A SlicerRT module called External Beam Planning (EBP) aspires to be a generic platform for evaluating cutting edge radiotherapy techniques. EBP provides tools for radiation plan creation and setup, and a plugin mechanism so that clinics only need to supply their dose engine. An example is the proton engine from Massachusetts General Hospital (Sharp et al., 2017).

4.6. User and developer experience

Based on experience during the development of the above applications and the feedback of clinicians using them, the most important outcome of PolySeg is the improved user and developer experience. The chance for both user and developer errors are reduced by offering a solution for the four challenges described in section 1.

User workflows involving conversions in 3D Slicer have been made considerably more convenient and straightforward using PolySeg compared to the former methods. The previous approach included manual conversions in individual UI modules involving input selection and output naming, both potentially leading to errors or ambiguities. Especially as conversions need to be performed on each structure, and repeated after any changes, automation provided by PolySeg brings considerable simplification. In addition, the need to use converter modules in a workflow increased the number of modules and thus interactions involved. An example is the Segment Comparison module (in SlicerRT), which requests the necessary

conversions based on the representation needed for the comparison algorithm. Similarly in data export, the only necessary step is the selection of the MRML node and the output file format. This improvement in reliability and user interaction applies to all research applications with flexible workflows instead of a fixed stream of steps.

PolySeg also improves the development process compared to having a set of individual conversion algorithms that need to be used to manage the versatile data the application works with. With a library automatically managing all structure representation related operations, the number of lines of code is largely decreased. Thus, the opportunity to write faulty, inefficient, or unstable code is reduced. Similarly to the user experience, many aspects cannot be quantified, such as the correctness or stability of the application code that is developed on top of PolySeg. However, we can provide a simple example to illustrate the simplified process. Two code snippets were implemented to convert the PROS dataset, one using 3D Slicer's binary labelmap to closed surface conversion module ModelMaker, and the other using PolySeg. The number of lines of (reasonably compact) Python code starting with the input variable present in the context are 11 for ModelMaker and 2 for PolySeg. It is also worth mentioning that the input of ModelMaker was a simple labelmap volume that does not support overlaps, so the code that preserves overlaps would be even longer. Furthermore, the developer needs to pay attention to the names of the output objects if correspondence between the representations is to be preserved. Besides the reduced code, having access to a validated set of conversion algorithms that are automatically executed when needed allows researchers to focus on the task at hand. This has been demonstrated during the development of the Segment Editor: each new effect took between a few hours and a few days to develop, and they showed good initial robustness. PolySeg's support for some of the most popular programming languages (C++, Python, and Matlab through 3D Slicer) makes the library accessible in a wide variety of projects.

5. Hardware and software specifications

PolySeg is platform-independent, and is tested on Windows, MacOS, and Linux. Nightly automated testing ensures error-free operation on each operating system. The end-user application prototypes implemented using PolySeg and 3D Slicer work on the same three operating systems.

Hardware requirements are specific to the end-user applications built on top of PolySeg. However, the general requirement is a configuration that has the graphics capabilities to display and memory to hold the original medical image datasets, as well as the converted data structures as specified in the workflow employed by the application. Due to the potentially large datasets used, a 64-bit architecture is typically required.

6. Mode of availability of the system and programs

The PolySeg library is distributed under the BSD 2-clause open-source license, which contains no restrictions on the use of the software. The source code and documentation are accessible freely on GitHub: https://github.com/PerkLab/PolySeg.

The implemented end-user applications are also under the BSD 2-clause license, and can be freely accessed at the following locations:

- Segment Editor within the 3D Slicer application: https://www.slicer.org
- Batch structure set conversion and open-source external beam radiation therapy treatment planning system within the SlicerRT toolkit: http://slicerrt.org
- MRI-US contour propagation for prostate cancer biopsy and brachytherapy planning: https://github.com/SlicerRt/SegmentRegistration
- Gel dosimetry analysis: https://github.com/SlicerRt/GelDosimetryAnalysis

Acknowledgement

This work was supported by Cancer Care Ontario through the Applied Cancer Research Unit and Research Chair in Cancer Imaging grant, The Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO) grant funded by the Ontario Research Fund, and the Research Software Program provided by CANARIE.

Conflict of Interest Statement

The authors have no relevant conflicts of interest to disclose.

References

Alexander, K.M., Pinter, C., Fichtinger, G., Olding, T., Schreiner, L.J., 2018. Streamlined Open-Source Gel Dosimetry Analysis in 3D Slicer. Biomed. Phys. Eng. Express 1–23.

Congalton, R.G., 1997. Exploring and Evaluating the Consequences of Vector-to-Raster and Raster-to-Vector Conversion. Photogramm. Eng. Remote Sens. 63, 425–434.

Cox, R.W., Ashburner, J., Breman, H., Fissell, K., Haselgrove, C., Holmes, C.J., Lancaster, J.L., Rex, D.E., Smith, S.M., Woodward, J.B., Strother, S.C., 2004. A (Sort of) New Image Data Format Standard: {NIfTI-1}. Neuroimage 22, 1.

Fedorov, A., Beichel, R., Kalpathy-Cramer, J., Finet, J., Fillion-Robin, J.-C., Pujol, S., Bauer, C., Jennings, D., Fennessy, F., Sonka, M., Buatti, J., Aylward, S., Miller, J. V, Pieper, S., Kikinis, R., 2012. 3D Slicer as an image computing platform for the Quantitative Imaging Network. Magn. Reson. Imaging 30, 1323–41. https://doi.org/10.1016/j.mri.2012.05.001

Fuchs, H., Kedem, Z.M., Uselton, S.P., 1977. Optimal surface reconstruction from planar contours. ACM SIGGRAPH Comput. Graph. 11, 236–236. https://doi.org/10.1145/965141.563899

Huttenlocher, D.P., Klanderman, G. a., Rucklidge, W.J., 1993. Comparing images using the Hausdorff distance. IEEE Trans. Pattern Anal. Mach. Intell. 15, 850–863. https://doi.org/10.1109/34.232073

Iglesias, J.E., Sabuncu, M.R., 2015. Multi-atlas segmentation of biomedical images: A survey. Med. Image Anal. 24, 205–219. https://doi.org/10.1016/j.media.2015.06.012

Jian Huang, Roni Yagel, Vassily Filippov, Yair Kurzion, 2014. An accurate method for voxelizing polygon meshes. IEEE Symp. Vol. Vis. (Cat. No.989EX300) 119–126,. https://doi.org/10.1109/SVV.1998.729593

Kapur, T., Pinter, C., Lasso, A., 2016. Increasing the Impact of Medical Image Computing Using Community-Based Open- Access Hackathons : the NA-MIC and 3D Slicer Experience.

https://doi.org/10.1016/j.media.2016.06.035

Meyers, D., Skinner, S., Sloan, K., 1992. Surfaces from contours. ACM Trans. Graph. 11, 228–258. https://doi.org/10.1145/130881.131213

Mildenberger, P., Eichelberg, M., Martin, E., 2002. Introduction to the DICOM standard. Eur. Radiol. 12, 920–927. https://doi.org/10.1007/s003300101100

Nolden, M., Zelzer, S., Seitel, A., Wald, D., Müller, M., Franz, A.M., Maleike, D., Fangerau, M., Baumhauer, M., Maier-Hein, L., Maier-Hein, K.H., Meinzer, H.-P., Wolf, I., 2013. The Medical Imaging Interaction Toolkit: challenges and advances : 10 years of open-source development. Int. J. Comput. Assist. Radiol. Surg. 8, 607–20. https://doi.org/10.1007/s11548-013-0840-8

Novotný, P., Dimitrov, L.I., Šrámek, M., 2010. Enhanced voxelization and representation of objects with sharp details in truncated distance fields. IEEE Trans. Vis. Comput. Graph. 16, 484–498. https://doi.org/10.1109/TVCG.2009.74

Pinter, C., Lasso, A., Wang, A., Jaffray, D., Fichtinger, G., 2012. SlicerRT: radiation therapy research toolkit for 3D Slicer. Med. Phys. 39, 6332–8. https://doi.org/10.1118/1.4754659

Poulin, E., Boudam, K., Pinter, C., Kadoury, S., Lasso, A., Fichtinger, G., Ménard, C., 2017. Validation of MRI to US Registration for Focal HDR Prostate Brachytherapy. Brachytherapy 16, S56--S57.

Ritter, F., Boskamp, T., Homeyer, A., Laue, H., Schwier, M., Link, F., Peitgen, H.O., 2011. Medical image analysis. IEEE Pulse 2, 60–70. https://doi.org/10.1109/MPUL.2011.942929

Schroeder, W.J., Lorensen, B., Martin, K., 2004. The visualization toolkit: an object-oriented approach to 3D graphics. Kitware.

Schroeder, W.J., Maynard, R., Geveci, B., 2015. Flying Edges: A High-Performance Scalable Isocontouring Algorithm, in: IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV). https://doi.org/10.13140/RG.2.1.3415.9609

Sharp, G.C., Pinter, C., Fichtinger, G., Unkelbach, J., 2017. Proceedings to the 56 th Annual Meeting of the Particle Therapy Cooperative Group (PTCOG), 8-13 May 2017. Int. J. Part. Ther. 4, 14–83. https://doi.org/10.14338/IJPT.17-PTCOG-1.1

Stearns, M.Q., Price, C., Spackman, K.A., Wang, A.Y., Authority, I., Cross, A., 2001. SNOMED Clinical Terms : Overview of the Development Process and Project Status 662–666.

Sunderland, K., Pinter, C., Lasso, A., Fichtinger, G., 2017. Fractional labelmaps for computing accurate dose volume histograms, in: SPIE Medical Imaging. p. 101352Y--101352Y. https://doi.org/10.1117/12.2254978

Sunderland, K., Woo, B., Pinter, C., Fichtinger, G., 2015. Reconstruction of surfaces from planar contours through contour interpolation, in: Progress in Biomedical Optics and Imaging - Proceedings of SPIE. https://doi.org/10.1117/12.2081436

Taubin, G., Zhang, T., Golub, G., 1996. Optimal surface smoothing as filter design 283–292. https://doi.org/10.1007/BFb0015544

Toussaint, N., Souplet, J.-C., Fillard, P., 2006. Medical Image Navigation and Research Tool by INRIA (MedINRIA). Enseignement 1–39.

Yoo, T.S., Ackerman, M.J., Lorensen, W.E., Schroeder, W., Chalana, V., Aylward, S., Metaxas, D., Whitaker, R., 2002. Engineering and Algorithm Design for an Image Processing API: A Technical Report on ITK - the Insight Toolkit, in: Proc. of Medicine Meets Virtual Reality, J. Westwood, Ed. pp. 586–592.

Yushkevich, P.A., Piven, J., Hazlett, H.C., Smith, R.G., Ho, S., Gee, J.C., Gerig, G., 2006. User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. Neuroimage 31, 1116–1128. https://doi.org/10.1016/j.neuroimage.2006.01.015

Zhu, L., Kolesov, I., Gao, Y., Kikinis, R., Tannenbaum, A., 2014. An Effective Interactive Medical Image Segmentation Method Using Fast GrowCut. Int. Conf. Med. Image Comput. Comput. Assist. Interv. submitted.