
Semi-Automatic Mandible Segmentation (SAMS) Pipeline Documentation

Release 1.0

Vocal Tract Development Laboratory

Feb 18, 2019

1	Introduction	1
2	Semi-Automatic Mandible Segmentation (SAMS) Pipeline	3
2.1	Overview	3
2.1.1	Background	3
2.1.2	Dataset	4
2.1.3	Flowchart	5
2.2	Installation	5
2.2.1	Operating System Requirements	6
2.2.2	Executable Software Dependencies	6
2.2.3	Cluster Computing Environment	6
2.3	Setting up Reference Templates	6
2.3.1	Template Selection	6
2.3.2	Template Preparation Process	7
2.4	Basic Workflow	7
2.4.1	Input Preparation	8
2.4.2	Pre-processing	8
2.4.3	Automatic Segmentation and Compositing	10
2.4.4	Post-processing	12
2.5	Cluster Computing: Overview	13
2.5.1	Workflow Map	13
2.6	Cluster Computing: Workflow	14
2.6.1	Terminology	14
2.6.2	Main Components	16
2.7	Authors	19
2.8	Acknowledgement	19
2.9	License	19
2.10	Contact	20
3	Indices and tables	21

CHAPTER 1

Introduction

This semi-automatic mandible segmentation (SAMS) pipeline was developed at our [Vocal Tract Development Lab](#) for the purpose of generating 3D mandible/jaw objects from head and neck multi-detector Computed Tomography (MDCT) images. The ultimate goal of this pipeline is to accelerate the process of consistently segmenting large number of 3D mandibles needed for analysis to quantify morphological changes of the mandible during the lifespan but more specifically during the first two decades of life. This pipeline is semi-automatic with the initial (pre-processing) and final (post-processing) steps require manual interventions.

The purpose of this documentation is to provide the background and a walkthrough of the SAMS pipeline.

Semi-Automatic Mandible Segmentation (SAMS) Pipeline

2.1 Overview

This tutorial outlines the method developed by the [Vocal Tract Development Lab \(VTLab\)](#) to segment 3D mandible/jaw objects from head and neck multi-detector Computed Tomography (MDCT) images. This pipeline is designed to reduce the effort spent on manually segmenting 3D mandibles from raw CT images. Due to morphological variation of the mandible across age and individuals, manual segmentation is a tedious and time consuming process, especially in research setting requiring large input of 3D mandible models, eg: developmental study on mandibular morphology. Therefore the VTLab developed this semi-automatic mandible segmentation (SAMS) pipeline to speed up the segmentation process, using a relatively easy setup. Other research groups facing similar issues may therefore adopt and adapt this pipeline for their use.

2.1.1 Background

This semi-automated mandible segmentation (SAMS) pipeline uses image registration to compare an unknown case (referred to as a *test scan* in this documentation, see next figure for flowchart), with a set of previously-segmented CT images and their respective 3D mandible models (this dataset is referred to as *reference templates*). This process is then used to infer the location of the mandibular region on the *test scan*.

This pipeline was developed with using mainly open-source software with the exception of [Analyze 12.0](#) (AnalyzeDirect, Overland Park, KS). The VTLab uses Analyze 12.0 (AnalyzeDirect, Overland Park, KS) for image preparation, editing and threshold determination. It can however be replaced with alternative software listed under the Installation section of this documentation.

[FSL \(FMRIB Software Library\)](#) and [Advanced Normalization Tools \(ANTs\)](#) are used for thresholding, trimming, and image registration. ITK-SNAP's [Convert3D \(C3D\)](#) tool is used to convert output into binary form. [AFNI \(Analysis of Functional NeuroImages\)](#) is used for post-processing padding and file conversion as needed. [MATLAB](#) is used for inspection of the segmented mandibles. See Installation page for links, alternatives and additional information.

This pipeline consists of three major steps: 1) Pre-processing, 2) Automatic Segmentation and Compositing, and 3) Post-processing. This pipeline is semi-automated since step 1 and step 3 require manual intervention.

In summary, an input CT series stored in Digital Imaging and Communications in Medicine (DICOM) format is first converted into Analyze-compatible 3D format for the user to determine a threshold in Hounsfield Unit (HU) that will remove all non-osseous tissues from the CT image (*test scan*). Then the user will determine the minimal enclosing box that contains the entire mandible in the *test scan*. The input *test scan* is then processed through the pipeline to be thresholded and cropped based on input user values. The pre-processed *test scan* is then sent to cluster computing to be registered against multiple *reference templates*. Once the mathematical transformation between the *test scan* and each reference is established, the inverse transformation is applied to the known reference's 3D mandible models (*template model*) to yield predictive mandible model for the *test scan*. These resulting "segmented 3D mandibles" are then compiled and averaged to produce a final model. This final model is then rendered in 3D for visual inspection. Mandibles with regions requiring correction/enhancement are converted back into Analyze format for manual editing in Analyze 12.0 (see Installation page for software options).

2.1.2 Dataset

The dataset and the *reference templates* used to design this pipeline are from our VTLab imaging database which was collected retrospectively following [University of Wisconsin-Madison Institutional Review Board \(IRB\)](#) approval (IRB number: 2011-0036 and 2016-1052).

2.1.3 Flowchart

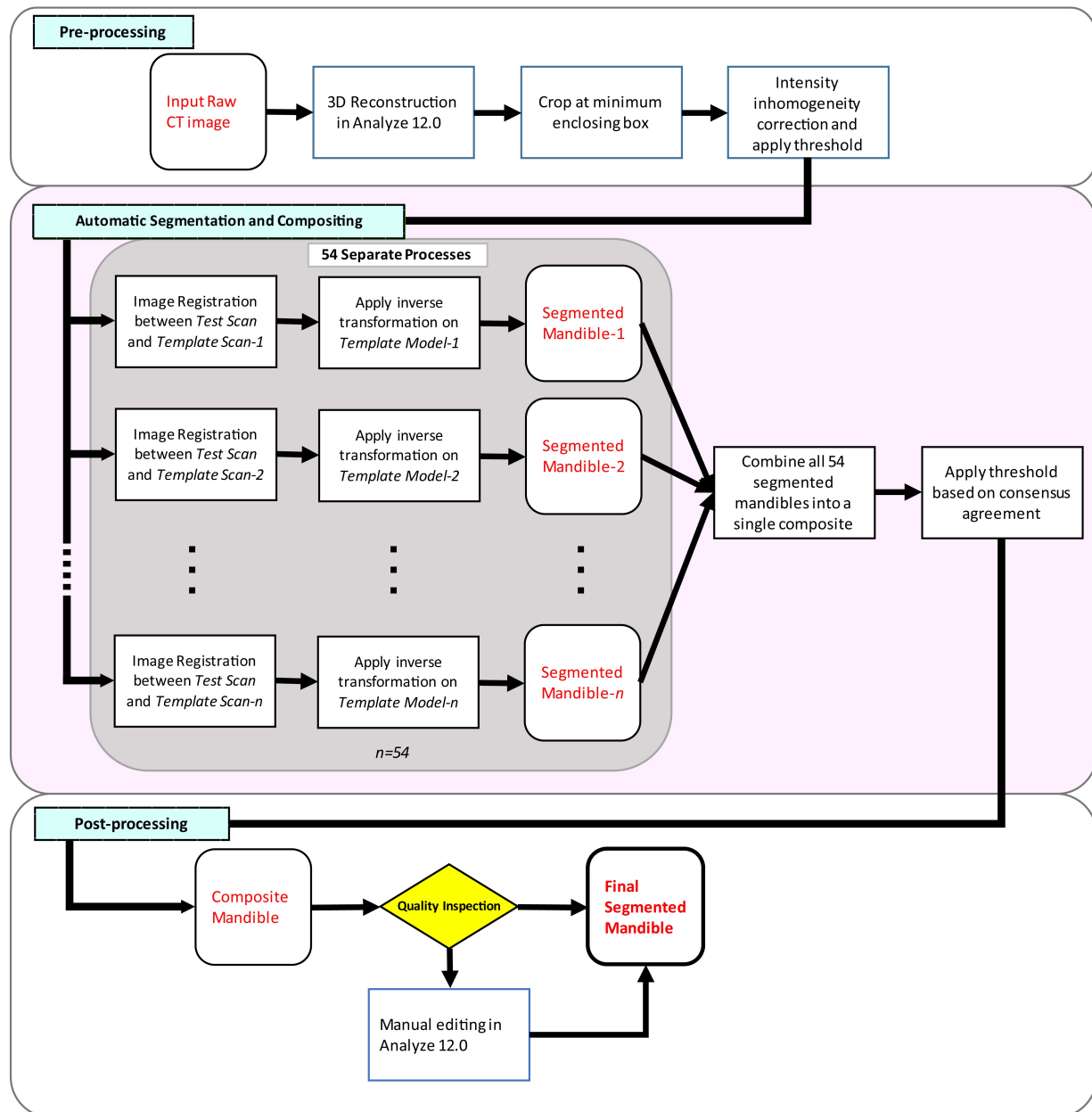


Fig. 1: Flowchart of the SAMS Pipeline

2.2 Installation

The SAMS pipeline consists of shell scripts requiring several executable programs as dependencies.

2.2.1 Operating System Requirements

The SAMS pipeline is set up to run in a Linux environment (Mac is possible). It has been tested on the Ubuntu 14.04 platform. Window users with access to Linux server can execute the pipeline using a terminal emulator, or set up a Linux environment using [VirtualBox](#).

2.2.2 Executable Software Dependencies

The following software are needed to operate the SAMS pipeline:

- [AFNI](#) - Analysis of Functional NeuroImages
- [Analyze](#)- Analyze 12.0
- [ANTs](#) - Advanced Normalization Tools
- [C3D](#) - Convert3D
- [FSL](#) - FMRIB Software Library
- [MATLAB](#) - MATLAB is preferred by the VTLab to render resulting SAMS-processed mandible for visual inspection.

Alternatives

The following is a list of open-source software that can be used as alternatives to Analyze 12.0 and/or MATLAB:

- [ImageJ](#)
- [ITKSNAP](#)
- [Slicer](#)

2.2.3 Cluster Computing Environment

This pipeline is designed to work with cluster computing for optimal performance. The Vocal Tract Development Lab employs High Throughput Computing (HTC) resources from the [Center for High Throughput Computing](#) at the University of Wisconsin-Madison. It can be adapted to use on other cluster computing environment for users who may have access to similar resources.

2.3 Setting up Reference Templates

The SAMS pipeline requires a set of CT images and their respective segmented 3D mandible models as the **reference templates**. These templates are the CT image (*template scan*) that the *test scan* will be mapped on to during the image registration process. Their respective 3D mandible models are the deformable models used to create the automatically-segmented mandibles for each registration. The following sections showed steps the VTLab implemented to build the *reference templates* needed for this pipeline.

2.3.1 Template Selection

The 54 *reference templates* (27 Males, 27 Females) we selected consisted of typically-developing individuals from birth to age of 20 years old. Each of the *reference templates* have the following components prepared and pre-processed:

1. CT Image (Referred to as *Template Scan* in the rest of this documentation)
2. 3D Mandible Model (Referred to as *Template Model* in the rest of this documentation)

This set of *reference templates* can be altered according to user preferences and research goals. Parameters presented in this documentation were optimized based on this set of *reference templates*. The user should perform parameter optimization to allow for best-performance of the reference templates built. Refer to Basic Workflow section for details of parameters set for this pipeline.

2.3.2 Template Preparation Process

The *template scans* are first processed through a cycle of registration pipeline with their 3D mandible models (that are segmented manually prior) as the deformable models, generating a set of automatically-segmented mandible models for each of the *template scan*. The segmented mandible models are then post-processed for enhancement and eventually used as the final *template models* in the SAMS pipeline.

2.4 Basic Workflow

After setting up the required dependencies and *reference templates*, users may begin segmenting 3D mandibles from an input raw CT image (*test scan*).

What follows is a list of all commands used in the SAMS pipeline. Variable names/values/files needed to be modified by users are enclosed between `<... >`.

For example, below is a command applying threshold to a mandible image

Example

```
$ fslmaths <input_mandible_image> -thr <user_decided_threshold_value> -uthr 3000  
↪<output_thresholded_mandible_image>
```

where

- *fslmaths* is the executing function,
- *<input_mandible_image>*, as stated in the variable name, refers to the file of the input mandible image, so users should replace this value with the filename (eg: mandibleA.nii),
- *thr* is the setting to the executing function, in this case the value following this will set the value to the minimum threshold,
- *<user_decided_threshold_value>* is a numerical value the user assigns to be the minimum threshold,
- *uthr* is also the setting to the executing function, stands for “upper threshold”, the value following this will set the value to the maximum / upper threshold limit,
- *3000* is the upper threshold limit,
- *<output_thresholded_mandible_image>*, as stated in the variable name, this will be the output of the above setting, users should enter the desired output filename here (eg: mandibleThresholded.nii)

Note: Refer to related documentations of functions and commands described in this documentation for detailed explanation.

2.4.1 Input Preparation

Before starting the SAMS pipeline, the desired input files should be converted from DICOM series into Analyze75 file format [.hdr/.img] or equivalent format.

In Analyze 12.0, user needs to inspect the scan and collect the threshold needed to remove all non-osseous tissues from the 3D renderings.

If an alternative software is used for inspection and threshold collection, the input file can also be saved as NIfTI file format [.nii or .nii.gz].

The following pre-processing step accepts both Analyze75 and NIfTI file format as the input *test scan*.

2.4.2 Pre-processing

In pre-processing, the input *test scan* will be inspected by user to:

1. Find a threshold value in Hounsfield Unit (HU) that provides best quality of the mandible.
2. Find the minimal enclosing box that contains the mandible.

In Analyze 12.0, the threshold value is determined by adjusting for the minimum threshold that will render a 3D mandible free of all non-osseous tissue. Typical threshold range between 80-350 HU.

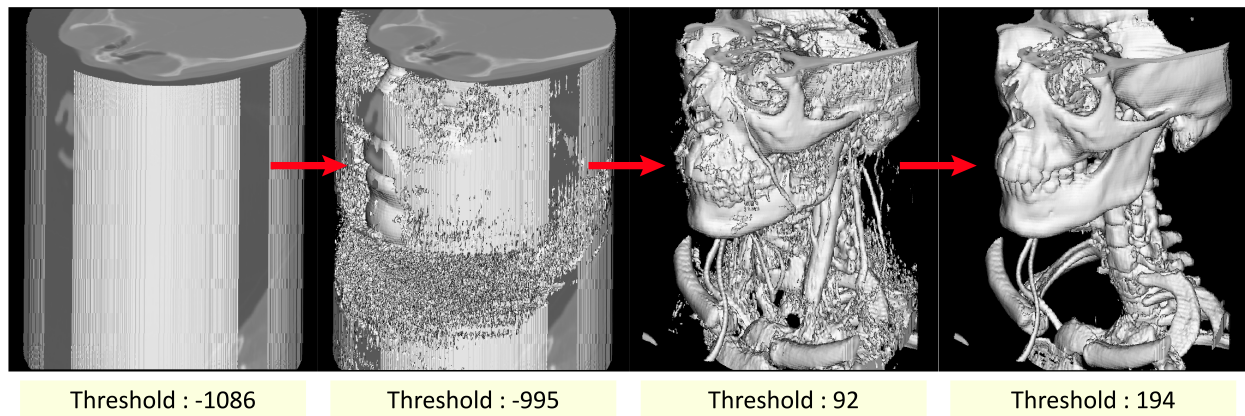


Fig. 2: From left to right: Effect of increasing minimum threshold values on a 3D rendering to allow best visualization of boney structure.

Next, user will record the smallest and largest slice number of the image in all X, Y and Z direction. These dimensions information will be used as input values to crop the image into its minimal enclosing box:

```
$ fsroi <trim_image> <input_image> <xlower> <xupper> <ylower> <yupper> <zlower>
↪<zupper>
```

After cropping the image the following is used to set origin of the image to 0 0 0

```
$ SVAdjustVoxelspace -in <trim_image> -origin 0 0 0
```

The image will be then be thresholded according to the threshold value the user collected:

```
$ fslmaths <trim_image> -thr <threshold_value> -uthr 3000 <trim_image>
```

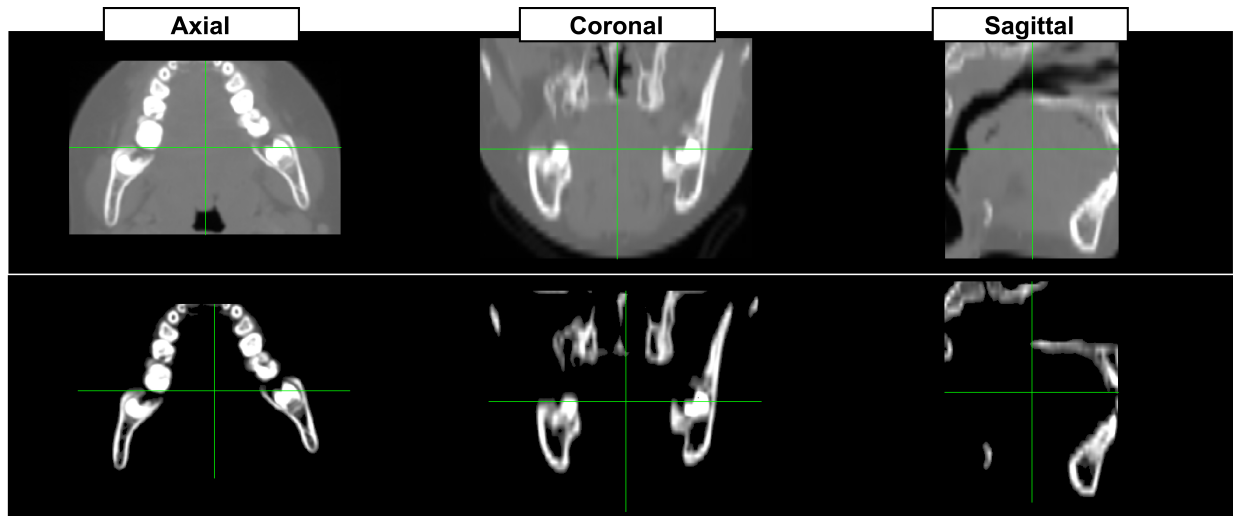



Fig. 3: Effect of applying a threshold value to a cropped *test scan* in all anatomical orientation. Top panel: cropped, raw *test scan*; Bottom panel: scan of bony mandible after applying the threshold.

Shown below are illustrations of cropping input *test scan* to the minimum enclosing box containing a complete mandible:

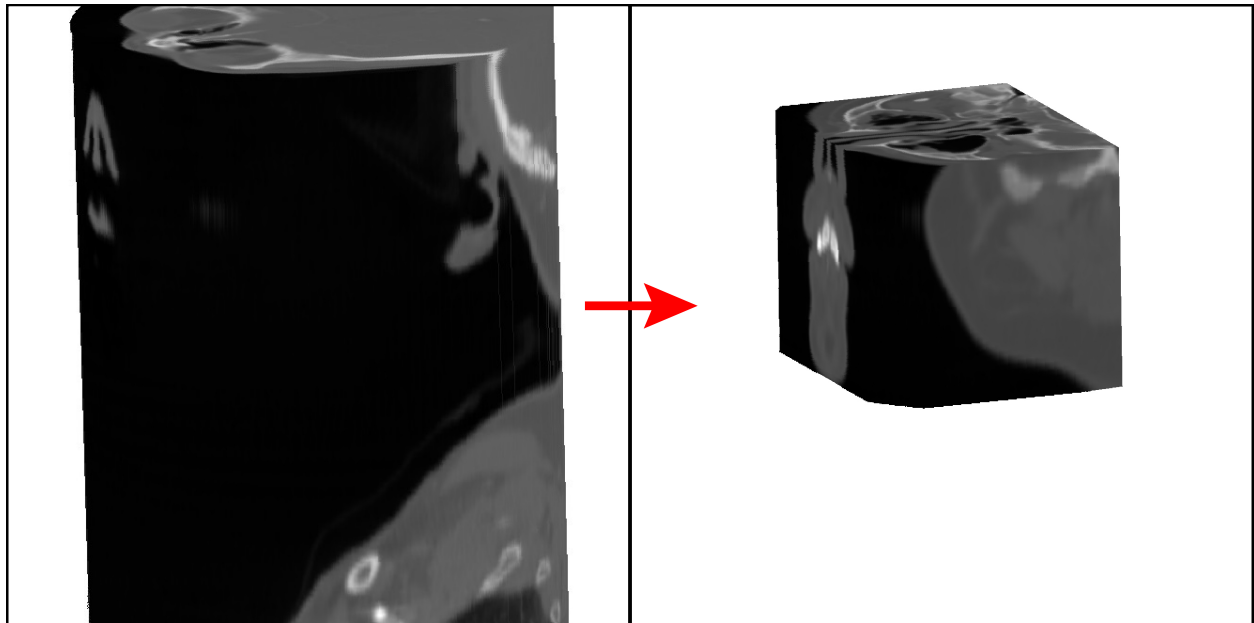


Fig. 4: Cropping a 512x512x660 volume into its minimal enclosing box of the dimension 332x270x262.

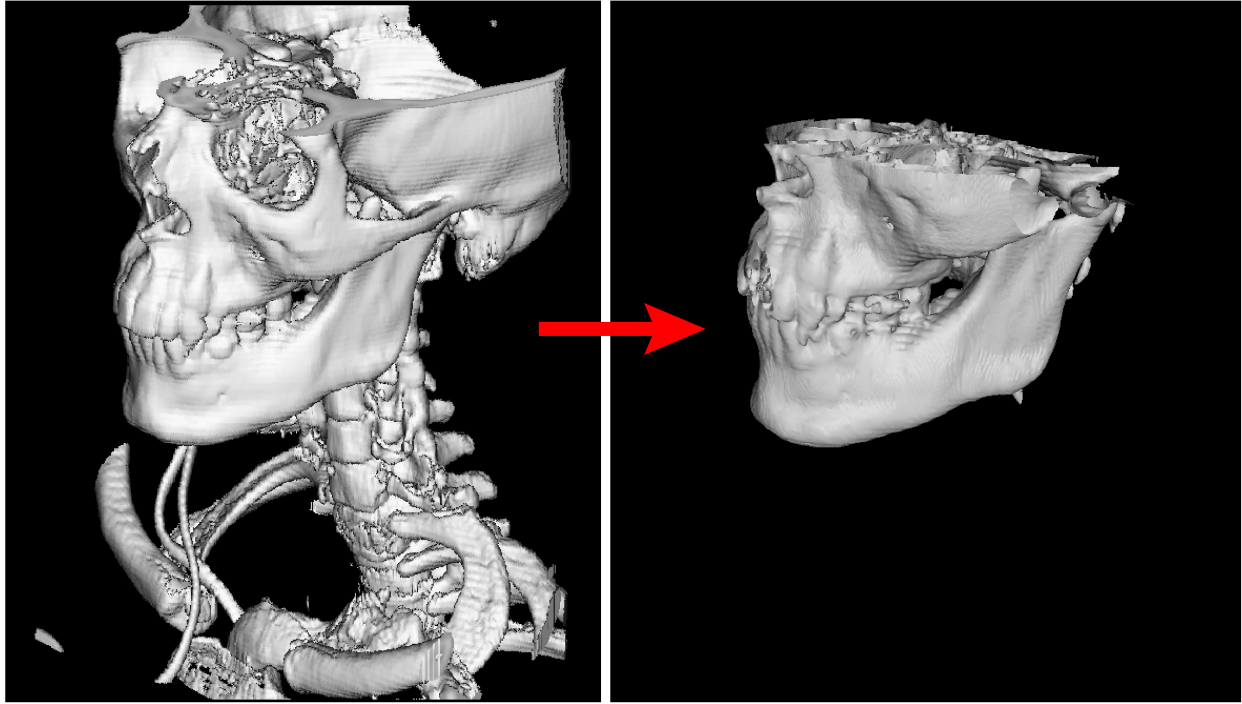


Fig. 5: Illustration of the cropping effect shown above on boney structure.

Bias correction is then applied to decrease scan intensity inhomogeneity:

```
$ N4BiasFieldCorrection -d 3 -i <trim_image> -o <trim_image>
```

The resulting trimmed *test scan* is now ready for the automatic portion of the pipeline.

2.4.3 Automatic Segmentation and Compositing

This step is an entirely automatic process completed on cluster computing (if available to user). The preprocessed input-*test scan*, will be registering against 54 *template scans*, followed by the application of the resulting inverse transformation on the *template models* to generate 54 separate segmented mandibles of the input. The 54 segmented mandibles are then composited and normalized to generate the final mandible.

Major commands used are as follow:

Automatic Segmentation

ANTs registration (shown are the command using ANTS instead of antsRegistration)

```
$ ANTS 3 -m MSQ[<referenceScan>, <testScan>, 1, 0] -o <output>.nii.gz -i 10x20x5 -r_
↪Gauss[4,0] -t SyN[4] --affine-metric-type MSQ --number-of-affine-iterations_
↪2000x2000x2000 <output>.log
```

The ANTS parameters listed here are as follow:

1. Similarity Metric = MSQ[<referenceScan>, <testScan>, 1, 0]
2. Deformable iteration = 5 x 50 x 10

3. Regularizer = Gauss[4,0]
4. Transformation = SyN[0.4]
5. Affine metric type = MSQ
6. Number of affine iterations = 2000 x 2000 x 2000

These values were obtained after parameter sweeping was performed in our VTLab. Users can alter the values according to their targeted reference templates' age range and demographics. For detailed explanation on each of the parameter functions, refer to the [ANTs documentation](#).

Followed by

```
$ WarpImageMultiTransform 3 <referenceModel> <affineInverseWarp>.nii.gz -i
  ↳<ANTSAffine>.txt <ANTSIInverseWarp>.nii.gz --use-NN -R <testScan>
```

Note: affineInverseWarp.nii.gz, ANTSAffine.txt and ANTSInverseWarp.nii.gz are output from the first step.

Binarization is performed to ensure that segmented mandibles are in binary form

```
$ c3d <affineInverseWarp>.nii.gz -binarize -o <segmented_mandible>.nii.gz
```

Compositing

All segmented mandibles from Automatic Segmentation steps will be compiled into one single composite and normalized:

```
$ fslmerge -t <allModels>.nii.gz "all <segmented_mandibles>.nii.gz separated by space"
$ fslmaths <allModels>.nii.gz -thr <normalization_value> -uthr <total_number_of_
  ↳mandibles_in_composite> -bin <allModels>.nii.gz
```

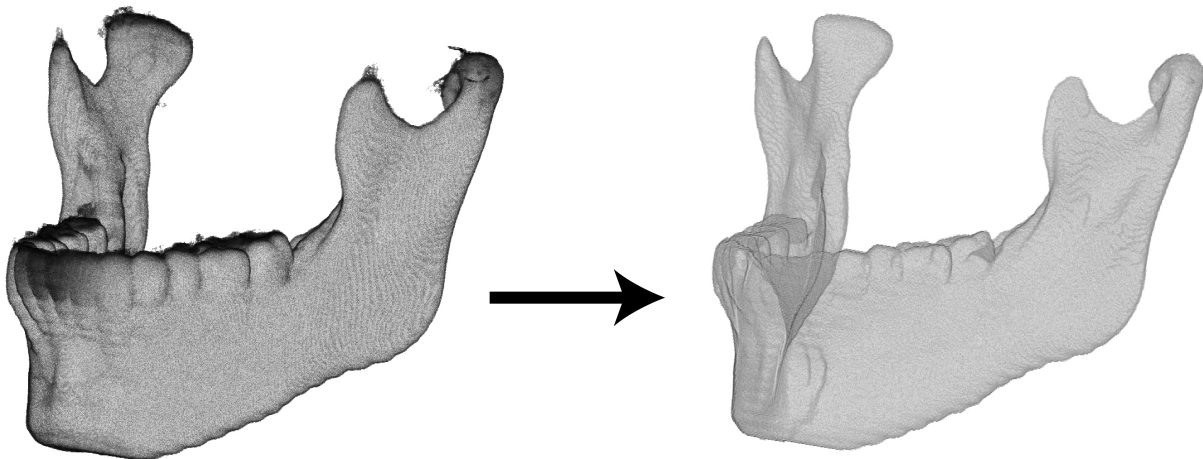


Fig. 6: **Left:** Normalized mandible binary after compositing all 54 mandible binaries from registration processes. Mandibles are rendered on a grayscale here. Darker color represents low voxel/overlap intensity while lighter color represents high voxel/overlap intensity; **Right:** Mandible composite after normalization and threshold.

Note: 3D Mandibles in figure above was rendered using the Volume Viewer module in ImageJ.

2.4.4 Post-processing

Once all compositing and averaging are completed, a single final composite mandible is generated. This composite mandible is viewed in MATLAB to determine if further manual touch-up is needed. In our case, the output from step 2 is in NIfTI file format so the `load_nii` function from the [Tools for NIfTI and ANALYZE image](#) package on MathWorks File Exchange is used to load accordingly

```
nii = load_nii('<allModels>.nii.gz')
mandible = isosurface(nii.img, 0.5)
```

Now you can view the 3D mandible:

```
p = patch(mandible)
set(p, 'FaceColor', '<ColorValues>', 'EdgeColor', 'none')
camlight
```

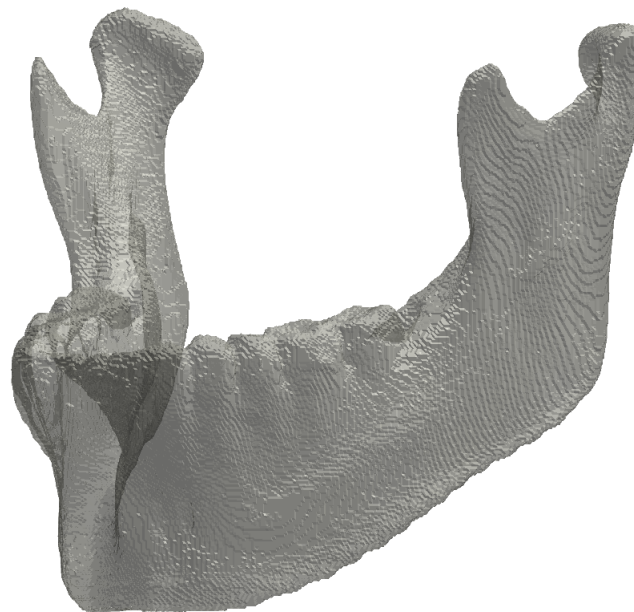


Fig. 7: 3D mandible model rendered in MATLAB using a color matrix value of [0.75 0.75 0.70].

Rotate the mandible to inspect any regions requiring further enhancement

For more MATLAB parameters and specifications, refer to MATLAB documentation on [Patch Properties](#).

Manual Editing

If the mandible needs to be edited manually, the NIfTI file will be padded into its original dimension using AFNI. Once padded, the mandible will be imported back into Analyze 12.0 for manual editing.

The following are used only if the user is using Analyze 12.0 as the editing software.

The padding values needed here are values recorded during pre-processing's cropping step:

```
$ 3dZeropad -I <zlower> -S <zupper> -P <ylower> -A <yupper> -L <xlower> -R <xupper> -  
↪prefix <outputName> <allModels>.nii.gz  
$ 3dAFNItoANALYZE <outputName> <outputName>+orig
```

When reloading the scan into the Analyze, users should flip the scan in X direction.

2.5 Cluster Computing: Overview

The VTLab uses resources from the [Center of High Throughput Computing \(CHTC\)](#) at the University of Wisconsin-Madison to execute the automatic segmentation and compositing step of the pipeline. Due to the multiple template-based CT image registration design of this portion of the pipeline, a parallel computing environment provides more computing resources to complete multiple registration within the same time frame. Automatic processing, however, is not limited to the CHTC environment described here.

This pipeline is integrated to work with HTCondor for higher throughput. HTCondor is a tool for HTC on large collections of distributed computing resources, consisting of machines (typically a physical computer) that may have multiple cores and slots (portion of a machine, often corresponds to one core). It is a specialized workload management system for jobs that are computationally intensive or projects that require parallel computing power. It provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring and management. Users submit their jobs and HTCondor places them into a queue, chooses when and where to run the jobs based on the specification designed by the user, monitors the progress, and ultimately returns the result of the job to the user.

For more information regarding HTCondor, refer to the [HTCondor Manual](#).

2.5.1 Workflow Map

The registration and compositing commands described in the Basic Workflow section are put into a workflow designed to run on the HTCondor scheduler. DAGMan (Directed Acyclic Graph Manager), a meta-scheduler for HTCondor, is utilized for the SAMS pipeline to connect the registration and compositing steps together as dependencies. Refer to the [DAGMan documentation](#) in the HTCondor manual for detailed information.

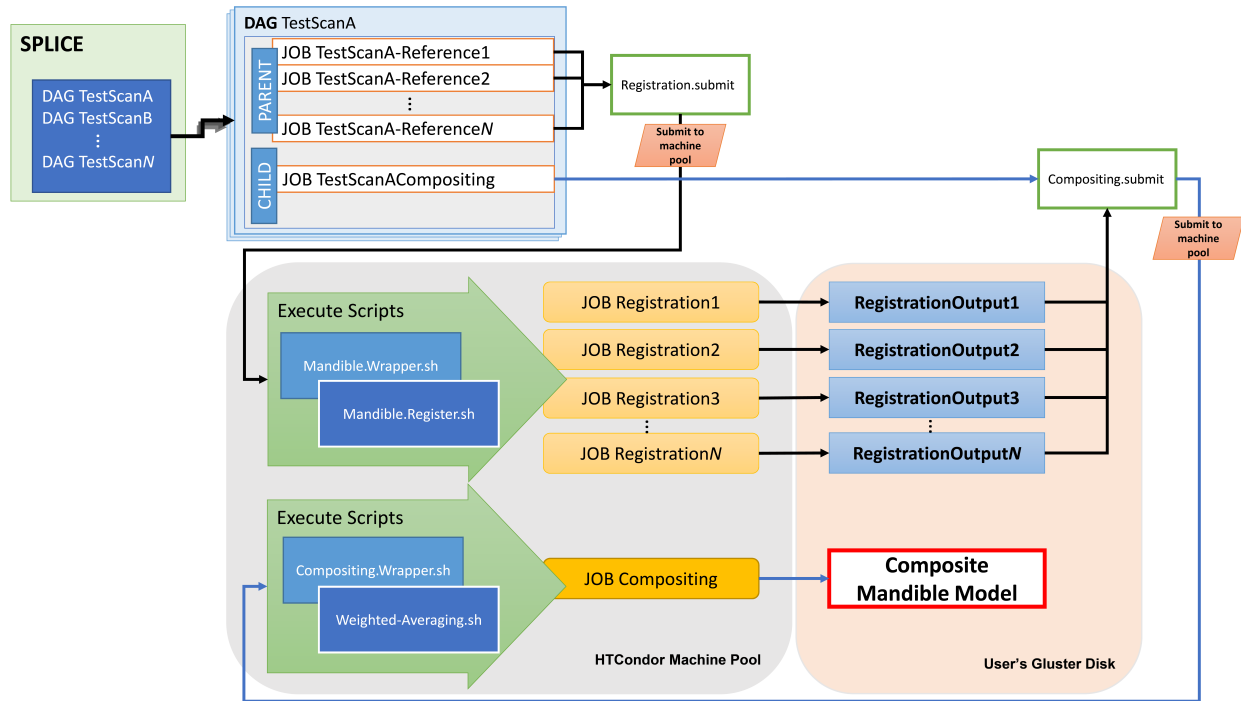


Fig. 8: Flowchart demonstrating steps, scripts and components of running Automatic Segmentation and Compositing step using CHTC resources.

2.6 Cluster Computing: Workflow

The explanation and examples provided in this section are HTCondor-specific terminologies and components, consisting extensive excerpts from the [HTCondor Manual](#). The description provided here are brief snippets of their actual and potential functions. Refer to the [HTCondor Manual](#) for detailed explanations. All HTCondor functions and commands displayed here are work of the [Center for High Throughput Computing](#), Computer Sciences Department, University of Wisconsin-Madison, Madison, WI.

2.6.1 Terminology

JOB : The *JOB* keyword specifies a job to be managed by HTCondor. This job is usually a *submit file*, which will be described later in the upcoming section.

```
JOB JobName SubmitFileName
```

PARENT ... CHILD : The *PARENT* and *CHILD* keywords specify the dependencies within the DAG. A parent node must be completed successfully before any of the children nodes may be started. We use this to ensure the compositing job starts only after all specified registration jobs are completed.

```
PARENT ParentJobName... CHILD ChildJobName...
```

DAG : Directed Acyclic Graph (DAG) is used to represent a set of computations where the input, output or execution of one or more computations is dependent on one or more other computations. A DAG input file describes the DAG, and further submit description files are used by DAGMan when submitting programs to run under Condor. DAGMan

is itself executed as a scheduler universe job within Condor. See HTCondor [Manual](#) for more information. A DAG file will be submitted using the tool `condor_submit_dag` by running the syntax as follow:

```
> condor_submit_dag <dagfilename>.dag
```

A very simple DAG input file:

```
JOB A A.sub
JOB B B.sub
JOB C C.sub
JOB D D.sub
PARENT A CHILD B C
PARENT B C CHILD D
```

VARs : The `VARs` keyword allows variables specific to a *JOB* node's *submit file* to be defined. The variables hence can be stated in the submit description file as a *macro*, contained between `$(...)`. The `VARs` syntax can be used in a *DAG* file as follow:

```
VARs Jobname variablename = "string"
```

A simple example of a *DAG* file incorporating `VARs`:

```
# File name: example.dag
#
JOB A A.sub
JOB B B.sub
JOB C C.sub
VARs A testName = "TestA"
PARENT A CHILD B C
```

JOB A's submit file - A.sub may then use the above-defined macro `testName` in an example as follow :

```
# File name: A.sub
executable = A.exe
log         = A.log
arguments   = "$(testName) "
queue
```

Note: See upcoming section for format of submit files used in the SAMS pipeline. Refer to the HTCondor [Manual](#) for detailed explanation.

SPLICE : A splice is an instance of a subgraph which is specified in a separate DAG file. This creates a named instance of a DAG as specified in another file as an entity which may have PARENT-CHILD dependencies.

```
SPLICE SpliceName DAGFileName [DIR directory]
```

A simple DAG spliced input file is

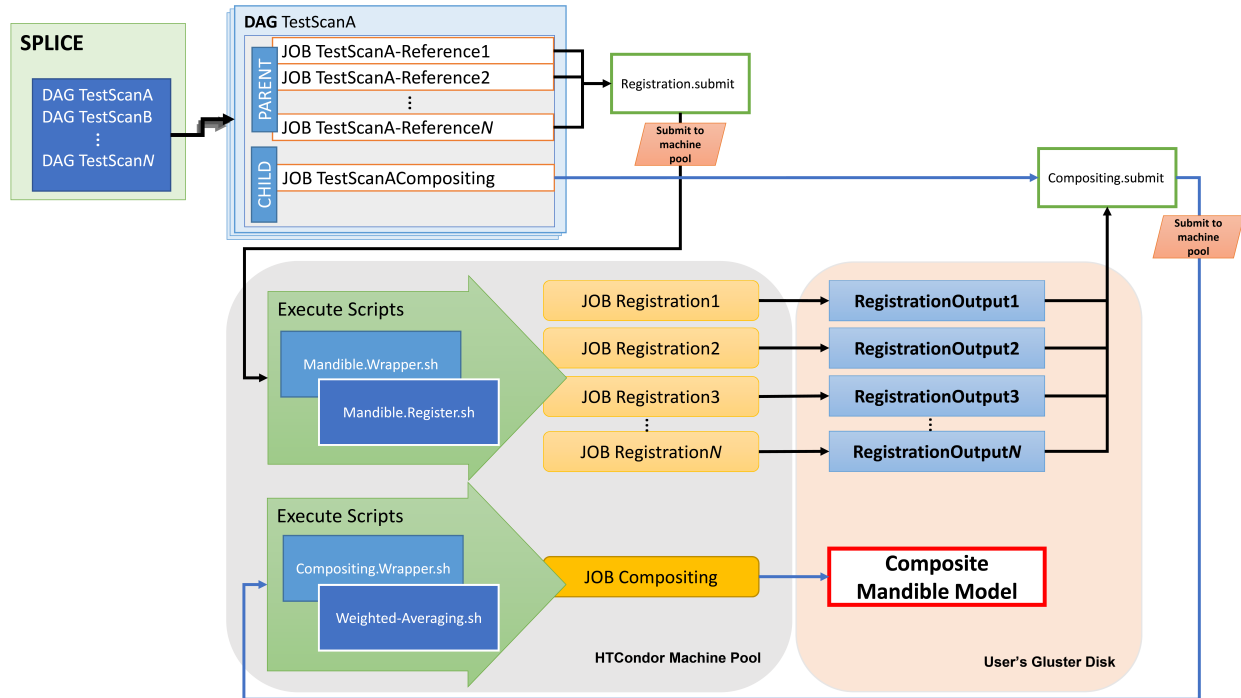
```
SPLICE file1 dagFile1.dag
SPLICE file2 dagFile2.dag
...
SPLICE file5 dagFile5.dag
```

Gluster : Gluster is a file share maintained by CHTC, and used for files or software that are too large for HTCondor file transfer. See <http://chtc.cs.wisc.edu/file-avail-gluster.shtml> for explanation. Note: This is an attribute unique only to

CHTC and not generally available. *As of September 25, 2017, Gluster is no longer used for the methodology described in this protocol.*

2.6.2 Main Components

In this section we go through the main components used in setting up the SAMS pipeline with the CHTC machine pool. Shown below is the workflow set up for SAMS pipeline when working with CHTC resources.



DAG File

Sample dag file

```
JOB TestA-RefA Registration.submit
VARS TestA-RefA id="mandible.submit"
VARS TestA-RefA test_name="F001-00-01-002"
VARS TestA-RefA test_model="F001-00-01-002-M-trim.nii.gz"
VARS TestA-RefA refName="F155-10-00-002_trimmed.nii.gz"
VARS TestA-RefA refModel="F155-10-00-002-M_trimmed.nii.gz"
VARS TestA-RefA Now("<timenow>")
...
JOB TestA-Compositing Compositing.submit
VARS TestA-Compositing id="mandible.compositing"
...
PARENT TestA-RefA TestA-RefB ... CHILD TestA-Compositing
```

Submit File

The HTCondor scheduler relies on a *Submit File* that communicates everything about our job(s) to the scheduler. A submit file is a text file that specifies the executing job/script, arguments, variables, etc. (Refer to HTCondor Manual

for detailed explanation).

The SAMS pipeline consists of two main submit files:

1. Registration.submit - A submit file for the *Registration* Job
2. Compositing.submit - A submit file for the *Compositing* Job

The following are samples of the two submit files. Users should replace variables indicated between <... > accordingly.

Registration.submit

```
universe=vanilla
getenv=True
environment="ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS=1"
executable=<location>/mandible.wrapper.sh
should_transfer_files=YES
when_to_transfer_output=ON_EXIT
request_cpus=1
request_memory=4000
request_disk=6G

requirements = (HasGluster == true)
periodic_release=((JobStatus==5)&&((CurrentTime - EnteredCurrentStatus)>360))

transfer_input_files=<location>/mandible.register.sh,<location>/bin/c3d,$(test_dicom),
↪$(test_model),$(refImg),$(refMod),<location>/fsl-5.0.8-htc-built.tgz,<location>/
↪ants-htc-built.tgz

arguments="$(now) $(test_name) $(refName) $(Cluster) $(useModel) "
log=$(id)_T-$(test_name)_R-$(refName)_$(Cluster)_$(now).log
output=$(id)_T-$(test_name)_R-$(refName)_$(Cluster)_$(now).out
error=$(id)_T-$(test_name)_R-$(refName)_$(Cluster)_$(now).err
notification=Error
notify_user=<user@email.com>
stream_output=True

queue
```

Compositing.submit

```
universe=vanilla
getenv=True
environment="ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS=1"
executable=<location>/compositing.wrapper.sh
should_transfer_files=YES
when_to_transfer_output=ON_EXIT
request_cpus=1
request_memory=10000
request_disk=8G
requirements = (HasGluster == true)

transfer_input_files=<location>/weighted-averaging.sh,<location>/mandible.unpack.sh,
↪<location>/bin/c3d,<location>/$(test_model),<location>/fsl-5.0.8-htc-built.tgz

arguments="$(now) $(test_name) $(dir_name) $(Cluster) $(comp) "
log=$(id)_T-$(test_name)_O-$(dir_name)_$(Cluster).log
output=$(id)_T-$(test_name)_O-$(dir_name)_$(Cluster).out
error=$(id)_T-$(test_name)_O-$(dir_name)_$(Cluster).err
```

(continues on next page)

(continued from previous page)

```
notification=Error
notify_user=<user@email.com>
stream_output=True

queue
```

Note: The line *requirements = (HasGluster == true)* is a CHTC-only attribute and not generally available.

Executing Scripts

As shown in the flowchart above, there are two executing scripts for the Registration step and Compositing step respectively. These two steps are linked through a PARENT-CHILD dependency listed in the submit DAG file. The two scripts consist of a “wrapper” script and an executing script.

Wrapper

- Initiate and make referral to executing environment
- Specify all variables and arguments need for executing script
- Unzip and install software prebuilt on machine
- Run executing script
- Compile output into tarball and export to *gluster*

Executing Script

- Run software with wrapper-specified arguments

Registration

Scripts used are

```
mandible.wrapper.sh
mandible.registration.sh
```

The executing script here will consist of commands specified in [Basic Workflow/Automatic Segmentation and Compositing/Automatic Segmentation](#) section.

Compositing

Scripts used are

```
compositing.wrapper.sh
weighted-averaging.sh
mandible.unpack.sh
```

The executing script here will consist of commands specified in [Basic Workflow/Automatic Segmentation and Compositing/Compositing](#) section.

Samples scripts will be provided here in late July

2.7 Authors

The semi-automatic mandible segmentation (SAMS) pipeline was developed as a tool to aid in the goal of quantifying the multidimensional growth of the human mandible during the course of development. This project started in the [Vocal Tract Development Lab](#), Waisman Center at the University of Wisconsin-Madison in 2013 under the supervision of VTLab Director and Principal Investigator [Dr. Houri K. Vorperian](#) and with guidance from co-investigator [Dr. Moo K. Chung](#).

The development of this pipeline is a culmination of efforts of the following VTLab team members who worked on this project in chronological order: [Dr. Nagesh Adluru](#), Simon Lank, Benjamin M. Doherty and Ying Ji Chuang. The refinement of this pipeline and development of this documentation are the direct outcomes of Ying Ji Chuang's effort, with valuable guidance and input from Dr. Nagesh Adluru.

2.8 Acknowledgement

We would like to acknowledge Lauren Michael of the [Center of High-Throughput Computing \(CHTC\)](#) at the University of Wisconsin-Madison, for the support and guidance she provided throughout this project. We also thank Greg Thain for his comments on an earlier version of this documentation. This research was performed using the compute resources and assistance of the UW-Madison Center For High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by UW-Madison, the Advanced Computing Initiative, the Wisconsin Alumni Research Foundation, the Wisconsin Institutes for Discovery, and the National Science Foundation, and is an active member of the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science.

This work was supported by NIH research grant R01 DC006282 (MRI and CT Studies of the Developing Vocal Tract, Houri K. Vorperian, Principal Investigator) from the National Institute on Deafness and other Communication Disorders (NIDCD), and by a core grant P30 HD03352 and U54 HD090256 to the Waisman Center (Albee Messing, PI) for research support from the National Institute of Child Health and Human Development (NICHD).



2.9 License

Copyright (c) 2017 Vocal Tract Development Laboratory

This documentation by the Vocal Tract Development Laboratory is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, and/or merge copies of the Software, and to permit persons to whom the Software is furnished to do

so, subject to the following conditions listed under the [Creative Commons Attribution-ShareAlike 4.0 International License](#).

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.10 Contact

Vocal Tract Development Lab

Waisman Center
University of Wisconsin-Madison
1500 Highland Ave., Room 429
Madison, WI-53705

Direct questions to vtlab@waisman.wisc.edu

Help is available throughout the duration of the grant that made this work possible.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`