Framework for Automatic Identification of Critical Regions of the Head and Neck for Refined Dose-Toxicity Analysis in Radiotherapy

Chis Micek. Arun Raghavan, Julie Shade
Mentors: Dr. Todd McNutt and Dr. Russell Taylor

May 18, 2017

Computer Integrated Surgery II
Dr. Russell Taylor
Spring 2017
Johns Hopkins University, Baltimore, MD

1. Introduction

1a. Background

Radiation therapies for head and neck tumors are fairly effective, but lead to unpleasant oral side effects in almost all patients [1]. These include swallowing difficulty, xerostomia (decreased saliva production), inflamed mucosal glands, and speech impairments. Research has indicated that some anatomical regions of the head and neck are more sensitive to radiation exposure than others and lead to more of these undesired effects, so it is desirable for clinicians to minimize radiation dose to these areas. This is possible with modern linear accelerators and treatment planning software, which allow for targeted radiotherapy, although it is not possible to completely avoid radiation exposure to areas outside the tumor volume. Thus, one must design a treatment that maximizes radiation dose to the tumor while minimizing dose outside the tumor, especially to known “critical areas” that are particularly susceptible to radiation damage.

Luijk, et al. found that radiation exposure to the stem-cell containing ductal region of the parotid (salivary) gland is associated with a decrease in saliva production post-radiotherapy. They also found that radiation therapy designed to spare this region is associated with a recovery of salivary function over time [2]. Preliminary work by our group involving manual contouring of the ductal area of the parotid in many radiotherapy patients, dose-volume analysis in this region from a treatment planning database, and machine-learning methods indicates that dose to this region is predictive of xerostomia, confirming Luijk et al.’s results. Identification of other anatomical regions of the head and neck most associated with adverse effects during and after radiation therapy would allow clinicians to prioritize dose minimization in these regions during treatment planning, potentially leading to a decrease in undesired side effects in radiotherapy patients.

1b. Problem

It would take too long to manually contour every possible region of the head and neck and perform dose-toxicity analysis on each since there are an infinite number of possible anatomical regions and manual segmentation is time consuming. Evidence indicates that this type of analysis could improve patient outcomes, so a more efficient method is required. We aim to provide a framework for automatic identification of anatomical regions of the head and neck that are most predictive of adverse effects after radiotherapy, eliminating the need for manual contouring of each region and allowing for dose-toxicity analysis of sub-organ structures and inter-organ space. This will greatly improve clinicians’ ability to plan radiotherapy treatments with minimal side effects.

2. Approach

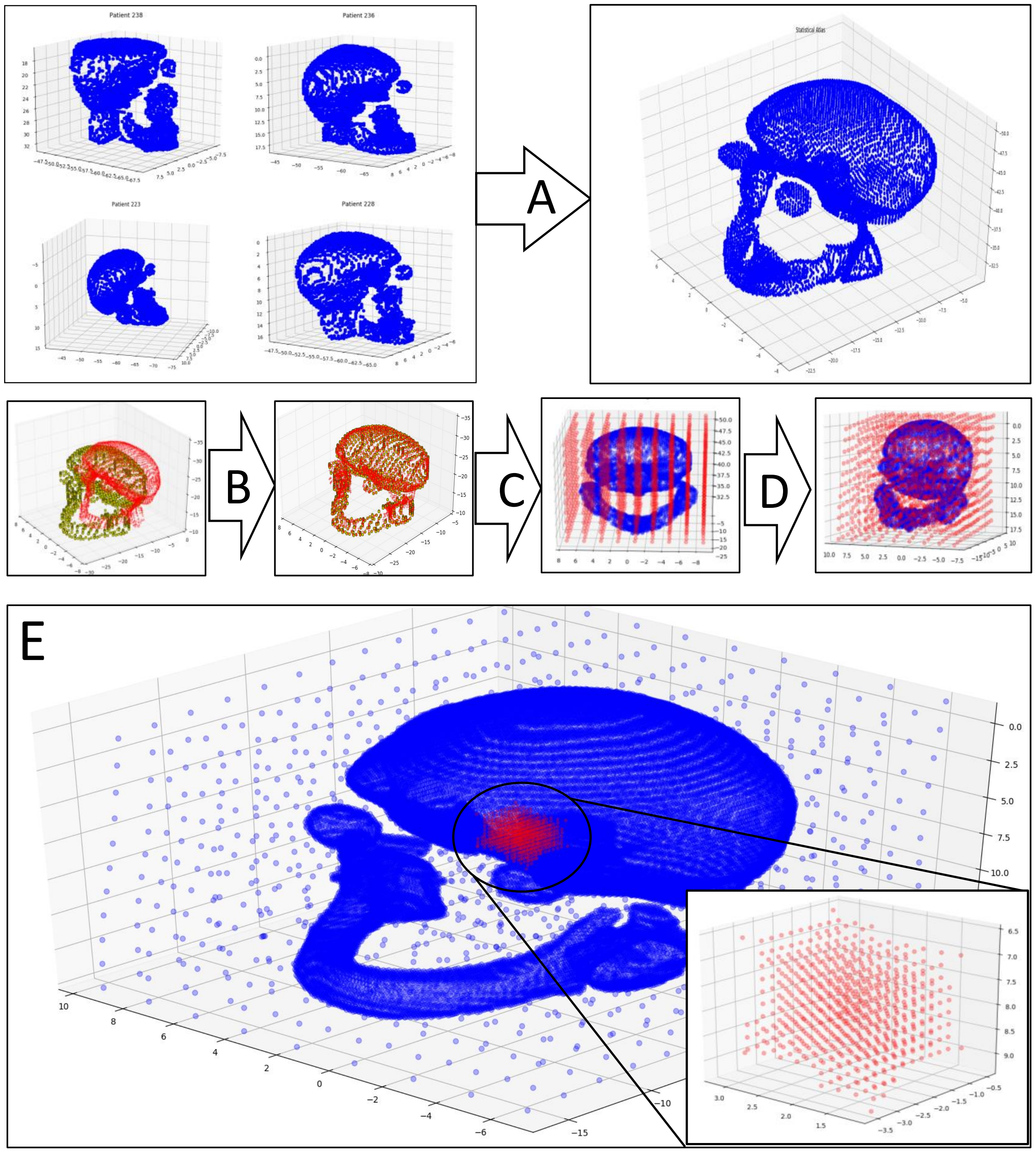

Figure 1: The process of accumulating dose statistics. A. Patient representations are queried from the database and put through our AtlasCreation algorithm, which performs registration to a common coordinate space to create an atlas. B. The atlas is then registered to an individual patient. C. The results of registration can then be used to fit a Thin Plate Spline transformation. D. In this example we use TPS to transform a uniform grid to an individual patient. E. We can then pick a transformed section of the grid and query for dose statistics from the database. This queried data corresponds to the location specified by the original section of the grid in the atlas.

The overall approach to the region identification framework is summarized in Fig. 1. We first create a statistical atlas representing a point cloud of mean patient anatomy. We then define the desired anatomical region for dose-toxicity analysis in this patient space. Note that this can be any arbitrary region, as shown in red in Fig. 1, but will likely be an evenly spaced grid over some region of the patient space and dose-toxicity analysis will be performed for each section of the grid individually. Next, we query the Oncospace database, an in-house database of patient CT and dose data stored as run-length encoded binary masks, to find a point cloud of all anatomical regions that are contoured in the desired patient and included in the atlas. We use coherent point drift to register the atlas to the patient and fit a thin-plate spline to create a mapping for any (x, y, z) point in the atlas space to the patient space. We transform the desired region to the patient space using the thin-plate spline and calculate dose inside the desired region by querying the dose map in the Oncospace database. Each step of this workflow will be described in more detail and code can be found in the technical appendices. With this framework, our group will be able to use machine learning methods to determine whether radiation dose to any given anatomical region is predictive of adverse effects and avoid these regions in future radiotherapy plans.

2a. Registration: Coherent Point Drift

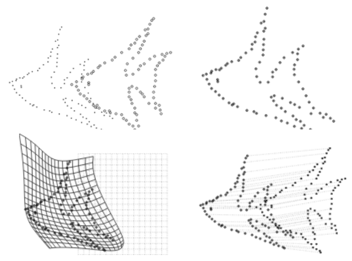
Figure 2: Visual representation of coherent point drift registration method as described by Myronenko et al. on artificially created point clouds. [3]

We chose to use a coherent point drift registration method to perform the necessary registrations. These include an iterative registration process to create the statistical atlas and registrations between the atlas and other patients, used to transform regions in the atlas coordinate system to the patient coordinate system. The method we used is described in the 2010 paper by Myronenko, A., & Song, X, Point Set Registration: Coherent Point Drift. This involves creation of a smooth vector field transformation between two point sets, which is ideal for our application involving anatomical structures. A visual representation of the method is shown in Fig. 2., which was taken from their paper. The detailed mathematical method can be found in their paper, but the pseudocode of the method is [3]:

```
Initialize parameters: $\lambda$, $\beta$, $\sigma$
Construct G matrix, initialize $\mathbf{Y} = \mathbf{Y}_o$
Deterministic annealing:
    EM optimization, until convergence:
        E-step: Compute $\mathbf{P}$
        M-step: Solve for $\mathbf{W}$ from:
                                        $(\text{diag}(\mathbf{P} \cdot \mathbf{1})\mathbf{G} + \lambda\sigma^2\mathbf{I})\mathbf{W} = \mathbf{PX} - \text{diag}(\mathbf{P} \cdot \mathbf{1})\, \mathbf{Y}_o$
                                    Update $\mathbf{Y} = \mathbf{Y}_o + \mathbf{GW}$
                               Anneal $\sigma = \alpha * \sigma$
                          Compute the velocity field: $v(\mathbf{z}) = \mathbf{G}(\mathbf{z}, \cdot)\mathbf{W}$
```

2b. Statistical Atlas Creation

Our framework allows users to create an atlas representing a point cloud of mean patient anatomy using any number of patients from the Oncospace database and any number of anatomical regions. The documentation for code used to create this atlas is provided in the technical appendices. The procedure begins by user selection of a list of anatomical regions and number of patients to include. The database is queried for a set of point clouds representing the desired number of patients that have all the desired regions contoured. The centroids of each data set are aligned and a random patient's coordinate system is chosen as the base coordinate system. All patient data sets are registered to this "fixed patient" coordinate system using CPD. Next, a statistical average of the data sets after registration is taken. The resulting average point cloud is set as the fixed patient coordinate system and the process is repeated until the registration error in below a given threshold value.

2c. Transformation: Thin-Plate Splines

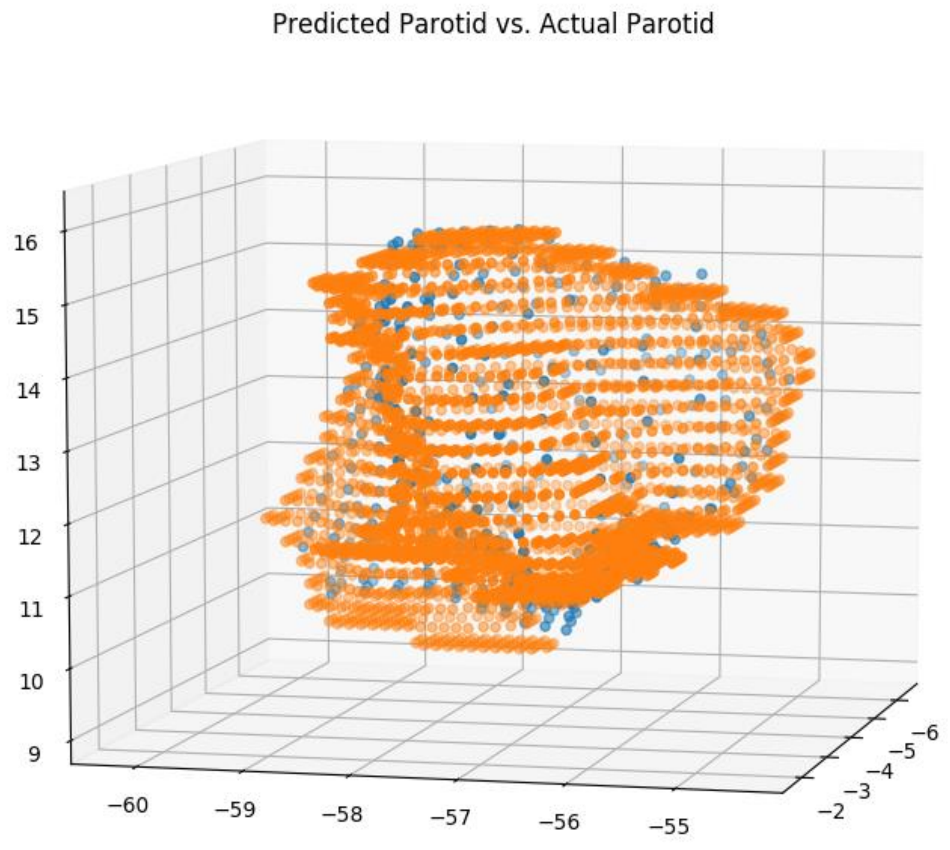

Figure 3: A comparison of the Thin Plate Spline predicted location of the right Parotid gland (in blue) to the location of the actual organ in patient 228. During the registration used to produce the parameters for Thin Plate Spline, the right Parotid was left out of the patient point cloud, thus this prediction is a visual representation of the accuracy of predicted regions with no corresponding structure in patient coordinate space.

To create a mapping function between the patient coordinate frame and the coordinate system of the atlas, we fit a Thin Plate Spline (TPS) transformation to the registration between the atlas and the patient. This is done by calculating the kernel matrix across the set of points given to the function. The kernel function $\Phi(z)$ for each point z consists of a 1 x K (where K is the number of control points chosen). From the kernel function a surface function is calculated and represented as a matrix. The pseudo-inverse of this matrix then can be used to transform any set of points from atlas coordinate space to patient coordinate space. Figure 3 shows the result of TPS in predicting the location of the right parotid gland using Registration with the Parotid gland left out.

2d. Dose Extraction

A TPS transformation from atlas space from patient space allows us to find the dose in a patient of any region we might define in atlas space. We first define a dense, regular grid of points in the atlas with voxel-width spacing, and then use the computed TPS transformation to deform this grid onto the patient coordinate system. The dose information for each patient is defined in Oncospace as a coarse grid over a subset of the patient volume, with each “voxel” containing the dose administered in the corresponding area of the grid. Functions already exist to interpolate this dose information onto the binary mask representing the patient anatomy; we convert the deformed grid into a mask contained in the patient volume, and use these functions to get dose data for each voxel in the region of interest. Then it is simple to calculate all manner of statistics on the calculated dose, e.g. to obtain the maximum, minimum, and mean dose. A sample dose mask is shown in Figure 4.

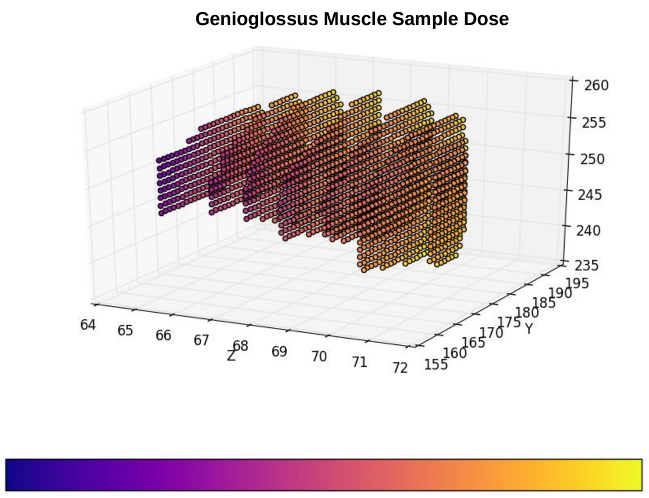

Figure 4: Sample dose mask for a genioglossus muscle in a random patient from Oncospace. The heat map represents normalized dose; blue indicates the lowest dose administered, and yellow indicates the highest.

3. Results

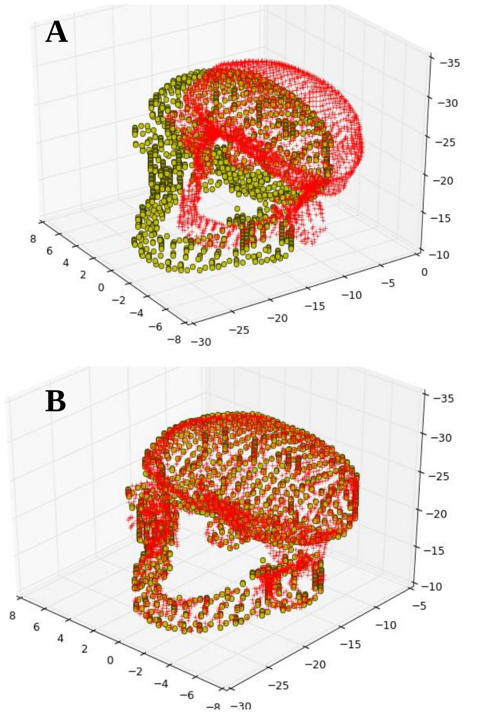

Figure 5: A. Atlas point set (red) and patient point set (yellow) before registration with CPD, B. After registration of atlas points to patient coordinate system.

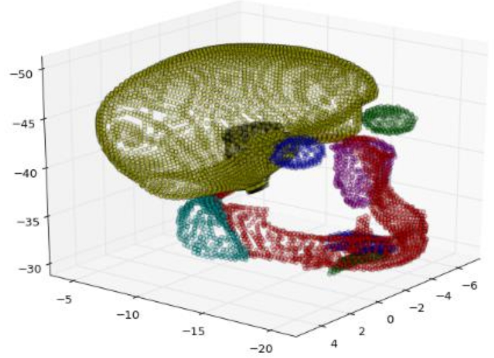

Figure 6: Point cloud representing statistical atlas created using left eye (blue), right eye (green), mandible (red), left parotid (cyan), right parotid (magenta), brain (yellow), brainstem (blue), genioglossus muscle (green), geniohyoid muscle (red), and hyoglossus muscle (cyan).

We were able to use code for coherent point drift registration developed by the Oncospace group to register point clouds representing patient anatomy with minimal error. One such registration is shown in Fig. 3, representing the registration between a statistical atlas and a patient data set including the left eye, right eye, left parotid, right parotid, mandible, and brain. This registration method allowed us to create several statistical atlases, as shown in Fig. 5. This atlas was developed using data from 30 patients whose data sets included the 10 contoured regions of interest described in the figure. It took approximately 36 hours to create on a laptop with 8GB RAM and an Intel i7 processor. We used this atlas for validation purposes.

A leave-one-out method was used to validate the atlas and region prediction/dose accumulation methods. The process was as follows:

1. Select a region to leave out
2. Query Oncospace database for point clouds of patient anatomy including all regions in atlas except left out region for 30 patients
3. Down-sample atlas to increase speed and register atlas point cloud to each patient point cloud, save registration parameters
4. Fit thin-plate spline for each atlas-patient pair
5. Use thin-plate spline to transform points of left-out organ in atlas frame to patient frame
6. Query Oncospace database to find radiation dose inside transformed point cloud (predicted left-out organ location) in each patient
7. Query Oncospace database to find radiation dose inside left-out organ in each patient
8. Calculate percent error between dose in predicted organ location and actual organ location in each patient
9. Calculate statistics for percent error

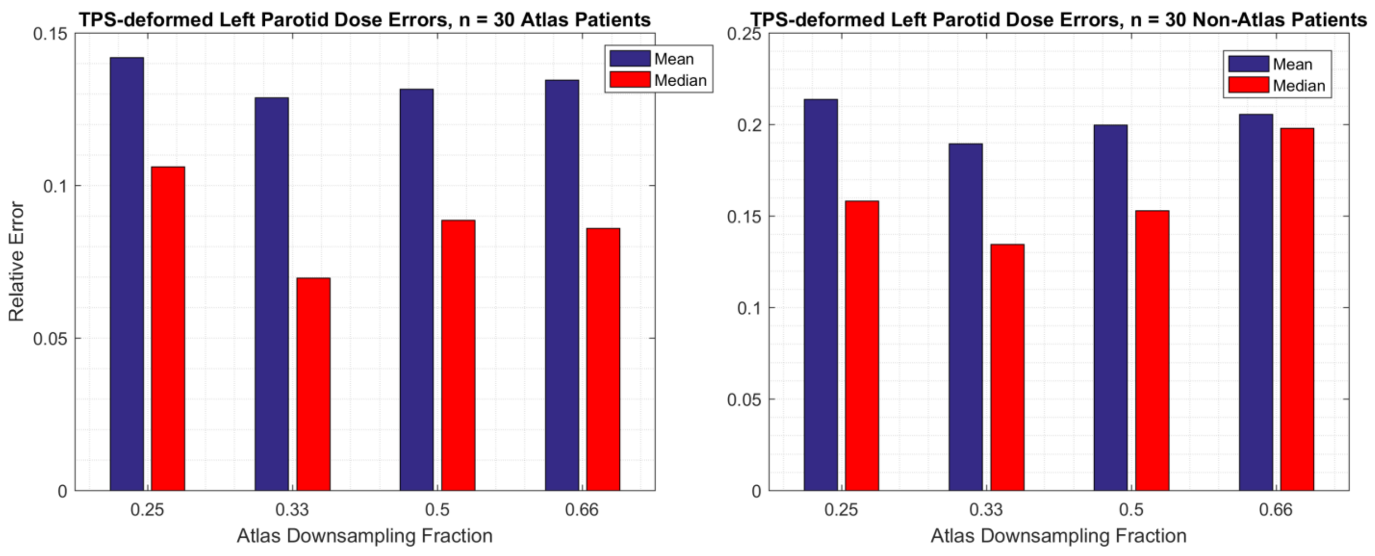

Figure 7: Mean and median dose errors for the left parotid with various fractions of atlas down-sampling, shown for 30 patients from the atlas and 30 patients not in the atlas, all with the anatomy from Figure 6.

As shown in Fig. 7, the downsampling fraction used to increase efficiency by reducing the number of points used for atlas registration and TPS transformation effects the accuracy of the mean-dose measurement obtained from validation. In all cases, the median is significantly less than the mean relative error, indicating the presence of some outliers. As runtime increases exponentially with the number of points used, however, we found that the best sampling fraction to use that minimizes the tradeoff between accuracy and runtime is 0.33. This trend is true regardless of whether the patients measured were included in our atlas, although for patients not in the atlas the error is higher for all downsampling fractions. This would likely be reduced if more patients were included in the atlas [4]. All numerical results are shown in Table 1. While in most instances the error for the left parotid was above our threshold value of 0.05, when the analysis was repeated for the genioglossus muscle errors improved dramatically, reaching a mean error below 0.05 for all downsampling fractions (see Figure 7). This is likely because the more central location of the genioglossus muscle results in TPS deformations less influenced by large variations in patient anatomy, in addition to the genioglossus exhibiting lower variation itself. Numerical results are shown in Table 2.

Statistic	Patient in Atlas?	Downsampling Fraction			
		0.25	0.33	0.5	0.66
Minimum	Yes	0.001795	0.002759	0.002366	0.001724
	No	0.005126	0.001754	0.002366	0.000146
Maximum	Yes	0.442518	0.447097	0.481809	0.409589
	No	0.491422	0.464304	0.560780	0.471700
Mean	Yes	0.141925	0.128821	0.131650	0.134488
	No	0.213695	0.189596	0.199720	0.205467
Median	Yes	0.106169	0.069733	0.088682	0.086025
	No	0.158248	0.134405	0.152835	0.198056
Standard Deviation	Yes	0.114689	0.127032	0.130911	0.116822
	No	0.156433	0.161554	0.156602	0.139194

Table 1: All measured statistics on relative error between mean dose of the left parotid in ground-truth patient data and the left parotid deformed from the atlas. Dose calculation was performed for the 30 patients included in the atlas, as well as 30 not included in the atlas with the same anatomy contoured.

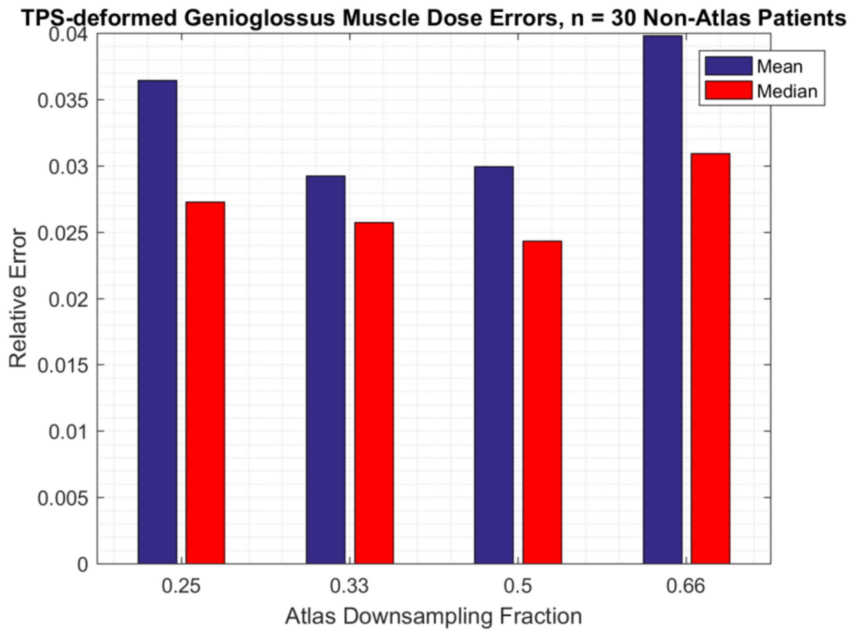

Figure 8: Mean and median dose errors for the genioglossus muscle with various fractions of atlas down-sampling, shown for 30 not in the atlas, all with the anatomy from Figure 6.

	Downsampling Fraction			
	0.25	0.33	0.5	0.66
Minimum	0.000589	0.000073	0.000296	0.002429
Maximum	0.145107	0.104386	0.100659	0.133717
Mean	0.036435	0.029256	0.029937	0.039817
Median	0.027277	0.025742	0.024334	0.030902
Standard Deviation	0.035446	0.025740	0.027952	0.033018

Table 2: All measured statistics on relative error between mean dose of the genioglossus muscle in ground-truth patient data and the genioglossus muscle deformed from the atlas. Dose calculation was performed for the 30 patients not included in the atlas with the same anatomy contoured.

4. Significance

In order to reduce harmful side effects, doctors want to know which areas of the head and neck are most sensitive to radiation exposure and thus should absolutely be avoided in radiotherapy. Prior to computerized treatment planning, doctors may have been able to make assumptions based on anecdotal evidence, such as “If the radiation beam is pointed in the general direction of the parotid gland for too long, the patient is likely to lose salivary function,” but there were no ethical methods to test these assumptions in humans. With advances in treatment planning and machine learning methods, clinicians can now manually contour a region of the head and neck, compute the radiation dose to this region in hundreds of patients, and see if increased dose to the region is predictive of adverse effects or not. However, this method is time consuming because it requires manual contouring of each region of interest by an expert, so it is not practical to test every possible region of interest in this way.

We have created a framework that will allow researchers to use machine learning methods to automatically detect regions of the head and neck that are most predictive of adverse effects. Users can create a statistical atlas using any number of patients and contoured organs (this is limited only by computational resources) and use the atlas to predict the location of regions of interest in all patients in the Oncospace database. We have shown that the method used is able to predict the location of non-contoured regions with reasonable success, averaging a best-case error of about 3%.

5. Management Summary

Who did what: Chris worked on validation methods, region definition and transformation (sectors.py), and dose accumulation. Arun wrote the method for thin-plate spline transformation and worked on region definition and dose accumulation. Julie worked on statistical atlas creation, edits to existing visualization methods, and region definition.

Accomplished vs. Planned: Our initial deliverables were as follows:

> **Min**: *Well-documented* API for creation of a statistical atlas of contoured anatomy and script to register a new patient to this atlas using coherent point drift. Script to predict the location of a defined cube-shaped region from the atlas in this patient, written in Python.
>
> **Expected:** *Well-documented* expansion on methods from minimum deliverable to define the same arbitrary volume in multiple patients, corresponding to the same anatomical soft-tissue region. Statistical validation of this method showing minimal error between arbitrarily selected regions.
>
> Creation of multiple statistical atlases using different anatomical regions to determine which gives most accurate prediction of the location of the arbitrary volume in patients not included in the atlas.
>
> **Max:** GUI to select anatomical region in statistical atlas and view ROI estimates along with dose volume histogram of carved regions in addition to the expected and minimum deliverables.

We accomplished all minimum and expected deliverables. We ended up changing the maximum deliverable as the goals of the project were clarified during weekly meetings with our mentors. Instead of creating a GUI to manually select an anatomical region, we developed methods to map a 3D grid from the atlas coordinate space to the patient coordinate space and calculate the radiation dose in one tetrahedron at a time. This would allow the end users to calculate dose in each rectangular prism in the atlas space transformed onto many patients and see to which regions or combination of regions dose is are most predictive of negative side effects, while requiring minimal manual input. This replaced the original maximum deliverable.

What might be next: The Oncospace group hopes to combine our dose accumulation pipeline with machine learning methods to determine which anatomical regions of the head and neck are most sensitive to radiation exposure. They will then use this information to improve radiotherapy treatment planning by avoiding these regions (if any dose-sensitive regions are discovered), hopefully leading to a reduction in side effects.

What we learned: This project allowed us to apply many of the methods we learned about in CIS I, such are non-rigid registration, statistical atlas creation, and transformation of points between coordinate systems using thin-plate splines. While we had learned about these methods in theory, none of us had actually tried to apply them to a real-life project before this.

6. Acknowledgements

We would like to thank our mentors, Dr. Todd McNutt and Dr. Russ Taylor for their support and guidance. We would also like to thank Pranav Lakshminarayana for his help, especially in understanding the existing Oncospace codebase.

7. References

1. Tolentino, E. de S., Centurion, B. S., Ferreria, L. H. C., de Souza, A. P., Damante, J. H., & Rubira-Bullen, I. R. F. (2011) Oral adverse effects of head and neck radiotherapy: literature review and suggestion of a clinical oral care guideline for irradiated patients. *Journal of Applied Oral Science*, *19*(5), 448–454. http://doi.org/10.1590/S1678-77572011000500003
2. Luijk, P. V., Pringle, S., Deasy, J. O., Moiseenko, V. V., Faber, H., Hovan, A, … Coppes, R. P. (2015). Sparing the region of the salivary gland containing stem cells preserves saliva production after radiotherapy for head and neck cancer. *Science Translational Medicine, 7*(305). doi:10.1126/scitranslmed.aac4441
3. Myronenko, A., & Song, X. (2010). Non-Rigid Point Set Registration: Coherent Point Drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 32*(12), 2262-2275. doi:10.1109/tpami.2010.46
4. Chintalapani, G. (2010). *Statistical Atlas Validation*. Lecture.

8. Technical Appendices

All documentation for code is below, in alphabetical order by file. Full code can be found on GitHub.

AtlasCreate.py
AtlasValidation.py
db_helper.py
DoseAccumulation.py
Sectors.py
SectorsUtils.py
TPS.py

```
"""
AtlasCreate.py
Authors: Pranav Lakshminarayanan, Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Atlas creation using methods from registration-cpd-atlas-2.ipynb. Methods to
create and save a statistical atlas representing point clouds of mean patient
anatomy from desired anatomical regions and number of patients. Resulting atlas
can be separated by organ.
```

```
"""

def point_to_organ_map(db, fixed_patient, reg_names, base):
    """
    Returns a list of integers (organmap). For each point in the base point cloud, the
index of its corresponding ROI in reg_names is returned at the same index in the organ
map.

    :param db: database
    :param fixed_patient: ID of fixed patient
    :param reg_names: list of ROI names
    :param base: base point cloud

    :type db: OncospaceConnect.Database
    :type fixed_patient: int
    :type reg_names: [str]
    :type base: np.array npoints x 3

    :return: A list of integers mapping each point in the base cloud to a ROI in
reg_list
    :rtype: np.array[int]
    """

def point_cloud_one_organ(db, fixed_patient, ROIname):
    """
    Finds the point cloud for one organ with extra high sampling.

    :param db: database
    :param fixed_patient: ID of fixed patient
    :param ROIname: ROI to return point cloud for

    :type db: Oncospace.Database
    :type fixed_patient: int
    :type ROIname: str

    :return: Point cloud of desired ROI in patient with ID = fixed_patient
    :rtype: np.array[float64]
    """

def sort_atlas_by_organ(atlas, reg_names, organmap):
    """
    Sorts the points in the atlas according to what organ they belong to. Points are
returned in one point cloud, with two additional arrays. The first array tells the
index of the returned array at which the set of points belonging to each ROI in
reg_names begins and the second array tells the number of points in the atlas from
each organ in reg_names (in order) in the atlas.

    :param atlas: statistical atlas representing ROIs in reg_names
    :param reg_names: list of ROIs in atlas
    :param organmap: list of integers mapping each point in the base cloud to a ROI in
reg_list

    :type atlas: AtlasCalculator from statistical_atlas.py
    :type reg_names: [str]
    :type organmap: [int]

    :return atlas_by_organ_cloud: atlas sorted by organ (same order as in reg_names)
    :return indices: index in atlas_by_organ_cloud at which each organ in reg_names
```

```
begins
    :return npoints_per_organ: number of points in atlas belonging to each organ (in
order of reg_names)

    :rtype atlas_by_organ_cloud: np.array[float64]
    :rtype indices: np.array[int]
    :rtype npoints_per_organ: np.array[int]
    """

def visualize_atlas_and_creation_process(atlas):
    """
    Visualize the given atlas and several of the steps in the atlas creation process
to ensure atlas was created correctly.

    :param atlas: atlas to visualize
    :type atlas: AtlasCalculator

    :return: None
    """

def save_atlas(output_name, reg_names, num_patients, atlas, base, clds,
test_reading=True):
    """
    Save atlas with either procedurally generated name or given name, and optionally
attempt to read atlas file to test whether save was successful. If attempting file
read, visualize point clouds (together and separated by organ)    from atlas file.

    :param output_name: file output name
    :param reg_names: list of ROIs included in atlas
    :param num_patients: number of patients included in atlas
    :param atlas: atlas to save
    :param base: base cloud of atlas
    :param clds: patient clouds included in atlas
    :param test_reading: whether to test atlas file reading

    :type output_name: str
    :type reg_names: [str]
    :type num_patients: int
    :type atlas: AtlasCalculator
    :type base: np.array[float64]
    :type clds: [np.array[float64]]
    :type test_reading: boolean

    :return: None
    """

def create_atlas(reg_names, num_patients, save=False, output_name=None, pat_list=None,
fixed_patient=None):
    """
    Creates a statistical atlas from the organs listed in reg_names for the given
number of patients. Default returns but does not save atlas file, gives procedurally
generated name based on organ names and number of patients.
    Metadata of atlas:
    atlas.metadata[1] = reg_names
    atlas.metadata[0][0] = atlas_by_organ_cloud (sorted atlas by organ, in one point
cloud)
    atlas.metadata[0][1] = indices (index in atlas_by_organ_cloud where each organ's
point cloud begins)
    atlas.metadata[0][2] = npoints_per_organ (number of points in each organ's point
cloud)
```

```
    :param reg_names: List of names of regions to include in registration
    :param num_patients: Number of patients to use to create atlas
    :param save_atlas: if True, atlas will be saved in ./output
    :param output_name: name of output file (optional, will be created automatically
if needed)
    :param pat_list: Optional list of patient IDs to use for atlas creation
    :param fixed_patient: Optional provided fixed patient

    :type reg_names: [str]
    :type num_patients: Integer
    :type save_atlas: boolean
    :type output_name: None | str
    :type pat_list: list[int]
    :type fixed_patient: int

    :return: statistical atlas representing average patient
    :rtype: AtlasCalculator.atlas

    """

"""
AtlasValidation.py
Authors: Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Set of functions used to validate atlas creation and dose accumulation.

"""

class Validator:
    """
    Class to automate the validation of statistical atlases built with different
regions of interest.
    """

    def __init__(self, patient_list, roi_list=None):
        """
        Initializer for Validator. Currently assumes one Validator per atlas, with
patient and roi lists provided on initialization.

        :param patient_list: List of patient representation IDs to use in atlas
creation
        :param roi_list: List of ROI names to use for atlases

        :type patient_list: list[str]
        :type roi_list: list[str]
        """


    def validate(self, atlas, leftout_patient, leftout_roi, thresh,
sampling_params=((0.25, 0.25, 1), True), display=False, fname=None, mode='whole',
ignoreFN=True):
        """
        Single validation pass. Validates registration, deformation and dose
extraction for the provided atlas following several different paradigms. Accepted
modes are:
        'whole': Given an atlas composed of N - 1 patients from self.patient_list,
registers the atlas to the frame of the left-out patient, and computes the Dice
coefficient of the whole atlas to determine accuracy. leftout_patient should be None.
If this value is greater than the threshold provided, the 'Success' flag of the output
```

```
is True. Can also be used on patients not in the atlas.

        'roi': Same as above, but tests thin-plate spline (TPS) deformation of regions
of the atlas missing in the patient space. First omits the region from the atlas,
registers it to the patient, and then deforms the left-out region using TPS onto the
patient coordinate system, and computed the Dice coefficient as a metric of validity.

        'dose': Same as above, but uses relative error between the mean dose
administered to the left-out roi in the patient and that from the transformed region
from the atlas. The threshold parameter given here is the maximum admissible error
(i.e. if at most 5% error is allowed, the threshold value is 0.05).

        The final result is a dictionary containing:

          'Validator': A copy of this Validator instance;
          'Leftout_patient': The patient representation ID of the patient left
out;
          'Leftout_ROI': The name of the ROI left out of the atlas, if any;
          'Mode': The mode used when validating;
          'Threshold': The provided Dice coefficient threshold for 'whole' and 'roi'
modes, or the maximum admissible error for 'dose' mode;
          'Success': A boolean flag that is True if the calculated Dice coefficient is
greater than the provided threshold, and False else for 'whole' and 'roi' modes. For
'dose' mode, returns True if the calculated error is below the
provided threshold;

        Specific to 'whole' and 'roi' modes:
          'Dice': The calculated Dice coefficient. 0.0 indicates no overlap, and 1.0
is perfect overlap;
          'IgnoreFN': Boolean flag indicating whether to ignore false negatives in the
Dice coefficient calculation;

        Specific to 'dose' mode:
          'Error': The relative error between the mean dose to the left-out region in
the patient and that of the region deformed onto the patient from the atlas.

         :param atlas: An AtlasCalculator object that is composed of N - 1 patients
from self.patient_list and uses the regions in self.roi_list

        :param leftout_patient: The patient representation ID of the patient left out
of the atlas

        :param leftout_roi: A region to leave out of the atlas registration to the
left-out patient, if desired; else None.

        :param thresh: Threshold value for the calculated Dice coefficient. If the
coefficient is above this value, the registration is considered a success.

        :param sampling_params: For modes 'whole' and 'roi', a tuple ((x, y, z), crop)
indicating the fraction of points to use for downsampling the x, y, and z axes of the
patient mask, and a boolean value indicating whether to crop the mask. See
transform.py for details.

        :param display: If true, plots the patient mask (blue) superimposed with the
deformed atlas mask (red)

        :param fname: If provided, the file the result will be written to (should end
in .dat)

        :param mode: The validation mode to use: 'whole', 'roi', or 'dose'

        :param ignoreFN: Flag for 'whole' and 'roi' modes indicating whether to ignore
false negatives in the Dice coefficient calculation. Note that depending on the
```

```
sampling of the atlas, the patient mask may be much denser than the transformed atlas,
leading to an artificially low Dice coefficient value.

        :type atlas: framework.statistical_atlas.AtlasCalculator
        :type leftout_patient: int
        :type leftout_roi: str | None
        :type thresh: float
        :type sampling_params: tuple(tuple(float, float, float), bool)
        :type display: bool
        :type fname: str
        :type mode: str
        :type ignoreFN: bool

        :return: A dictionary with info summarizing the validation (see above)
        :rtype: dict(str: Validator, str: int, str: str | None, str: str, str: float,
str: bool, str: float, str: bool) | dict(str: Validator, str: int, str: str | None,
str: str, str: float, str: bool, str: float)
        """

    @staticmethod
    def _dice(mask1, mask2, ignoreFN=True):
        """
        Helper method that calculates the Dice coefficient, a measure of overlap used
in several segmentation applications, between two mask objects (representative of
volumes from the Oncospace Database). 0.0 indicates no overlap, and 1.0 indicates
perfect overlap. Note that the calculation assumes that masks are representative of
the same volume, and that the data in their respective buffers have the same order for
memory access.

        :param mask1: The first mask, the mask registered
        :param mask2: The second mask, the mask to compare to
        :param ignoreFN: Flag indicating whether to ignore false negatives in the Dice
coefficient calculation. Note that depending on the sampling of the atlas, the patient
mask may be much denser than the transformed atlas, leading to an artificially low
Dice coefficient value.

        :type mask1: framework.Utils.image.mask
        :type mask2: framework.Utils.image.mask
        :type ignoreFN: bool

        :return: The Dice coefficient
        :rtype: float
        """

    @staticmethod
    def _tofile(data, fname):
        """
        Helper method that writes 'data' to a binary file specified by 'fname' using
Pickle. The file should end in .dat.

        :param data: The data to write
        :param fname: The file to write to (should end in .dat)

        :type data: T
        :type fname: str

        :return: None
        """
    @staticmethod
    def fromfile(fname):
        """
        Helper method to read Pickled data from a file.
        :param fname: The file to read from (should end in .dat)

        :return: The data read
        :rtype: T
        """

def single_atlas(roi_list, num_pats, fdir, fixed_patient=None):
    """
    Given the representation IDs of N patients, creates N atlases, each created with a
different combination of N - 1 patients. The fixed patient to use for atlas creation
is fixed_patient if provided; else it is random, but guaranteed to include the
required ROIs. The same fixed patient is used for each atlas. Saves the atlases in the
provided directory 'fdir' as 'lo_xxx.oaf', where xxx is the rep ID of the left-out
patient of the atlas.

    :param roi_list: List of ROIs to use
    :param num_pats: The number of patients to include in the set for atlas creation
    :param fdir: Location to save the atlases
    :param fixed_patient: Optional patient rep ID of the fixed patient to use

    :type roi_list: list(str)
    :type num_pats: int
    :type fdir: str
    :type fixed_patient: int

    :return: None
    """

def multi_atlas(roi_list, num_pats, fdir, fixed_patient=None):
    """
    Given the representation IDs of N patients, creates N atlases, each created with a
different combination of N - 1 patients. The fixed patient to use for atlas creation
is fixed_patient if provided; else it is random, but guaranteed to include the
required ROIs. The same fixed patient is used for each atlas. Saves the atlases in the
provided directory 'fdir' as 'lo_xxx.oaf', where xxx is the rep ID of the left-out
patient of the atlas.

    :param roi_list: List of ROIs to use
    :param num_pats: The number of patients to include in the set for atlas creation
    :param fdir: Location to save the atlases
    :param fixed_patient: Optional patient rep ID of the fixed patient to use

    :type roi_list: list(str)
    :type num_pats: int
    :type fdir: str
    :type fixed_patient: int

    :return: None
    """

def validate_whole(rootdir, roi_list, sampling_params=((0.25, 0.25, 1), True),
return_dice=False):
    """
    Creates and runs instances of the Validator class for each atlas in the directory
'rootdir', where each atlas is formed using N-1 patients from a larger atlas (i.e.
using multi_atlas), and is given the name lo_###.oaf, where ### is the patient
representation ID of the patient left out of the original atlas. Evaluates the CPD
registration and deformation of the entire atlas onto the entire patient anatomy, with
no regions missing. The results are saved in the folder 'validator', which is created
in 'rootdir' if it does not exist, as the file 'atlasname_validator.dat'.

    :param rootdir: The directory containing the atlases to validate
    :param roi_list: The list of ROIs in the atlases
    :param sampling_params: A tuple ((x, y, z), crop) indicating the fraction of
points to use for downsampling the x, y, and z axes of the patient mask, and a boolean
value indicating whether to crop the mask. See transform.py for details.
    :param return_dice: Flag indicating whether to return a list of Dice coefficients
for each validation as the result of this function. By default, there is no return
value.

    :type rootdir: str
    :type roi_list: list(str)
    :type sampling_params: tuple(tuple(float, float, float), bool)
    :type return_dice: bool

    :return: None, or a list of Dice coefficients from each validation
    :rtype: None | list(float)
    """

def validate_roi(rootdir, roi_list, leftout_roi, sampling_params=((0.25, 0.25, 1),
True)):
    """
    Creates and runs instances of the Validator class for each atlas in the directory
'rootdir', where each atlas is formed using N-1 patients from a larger atlas (i.e.
using multi_atlas), and is given the name lo_###.oaf, where ### is the patient
representation ID of the patient left out of the original atlas. Evaluates the TPS
transformation of a region left out of the atlas during initial registration to the
patient onto patient space. The results are saved in the folder 'validator', which is
created in 'rootdir' if it does not exist, as the file 'atlasname_roi_validator.dat'.

    :param rootdir: The directory containing the atlases to validate
    :param roi_list: The list of ROIs in the atlases
    :param leftout_roi: The name of the ROI to leave out of the initial atlas
registration to the patient, which will then be deformed onto the patient using TPS.
    :param sampling_params: A tuple ((x, y, z), crop) indicating the fraction of
points to use for downsampling the x, y, and z axes of the patient mask, and a boolean
value indicating whether to crop the mask. See transform.py for details.

    :type rootdir: str
    :type roi_list: list(str)
    :type leftout_roi: str
    :type sampling_params: tuple(tuple(float, float, float), bool)

    :return: None
    """

def validate_dose(atlas_path, output_dir, roi, sampling=0.25, use_atlas_pats=True,
num_pats=30):
    """
    Same as validate_roi above, but instead calculated the dose masks for the left-out
region in the patient and the corresponding region in the deformed atlas, and compares
their mean doses for validation. Follows the validate 'dose' method. Output is stored
in output_dir as 'atlasname_###_xxxx_dose_validator.dat', where ### is the patient
representation ID of the patient used in that Validator instance, and xxxx is the name
of the left-out region.

    :param atlas_path: Relative or absolute path to the atlas to use
    :param output_dir: The directory to save output
    :param roi: The ROI to use for validation
    :param sampling: The fraction of points to use from the atlas; see
Sectors.downsample_pc for more info
    :param use_atlas_pats: If True, uses patients from the atlas to validate (the
default); else, choose a random num_pats-length subset of all patients containing the
same regions as the atlas
    :param num_pats: If use_atlas_pats is False, the number of random patients to use
for validation
```

```
    :type atlas_path: str
    :type output_dir: str
    :type roi: str
    :type sampling: tuple(float)
    :type use_atlas_pats: bool
    :type num_pats: int

    :return: None
    """

def _pats_not_in_atlas(atlas_pats, roi_list, num_pats):
    """
    Returns a num-pat-length list of patient representation IDs for patients
containing all regions in roi_list.

    :param atlas_pats: List of patient rep IDs in the atlas, to omit in the result
    :param roi_list: List of ROIs in the atlas
    :param num_pats: Number of patient rep IDs to return

    :type atlas_pats: list(int)
    :type roi_list: list(str)
    :type num_pats: int

    :return: A list of patient rep Ids of length num_pats, where each patient is not
in atlas_pats but contains the regions in roi_list
    :rtype: list(int)
    """

def process_dose_validator(validator_dir):
    """
    Calculates summary statistics on the Validators saved from validate_dose, returned
in a dictionary.

    :param validator_dir: The directory containing the dose Validator files to use for
analysis.
    :type validator_dir: str

    :return: A tuple containing the Validator results, and a dictionary containing:
         'Threshold': The relative error threshold used when validating;
        'Successes': Number of patients who had error less than or equal to the
threshold;
        'Failures': Number of patients who had error above the threshold;
        'min': Minimum observed error;
        'max': Maximum observed error;
        'mean': Mean error;
        'median': Median error;
        'std': Error standard deviation.

    :rtype: tuple(dict(str: Validator, str: int, str: str | None, str: str, str:
float, str: bool, str: float), dict(str: float, str: int, str: int, str: float, str:
float, str: float, str: float, str: float))
    """

def subdir_stats(rootdir, stat_names, outfile=None):
    """
    Starting at rootdir, walks through all subdirectories and accumulates the
statistics in stat_names on all dose_validator.dat files therein. Only statistics
returned by process_dose_validator are valid.

    :param rootdir: The directory containing all subdirectories to search
    :param stat_names: The names of any summary statistics desired. Valid names are
those in the dictionary returned by rocess_dose_validator, above.
```

```
    :param outfile: If specified, saves output as 'outfile' in 'rootdir'
    :type rootdir: str
    :type stat_names: list(str)
    :type outfile: None | str

    :return: A list of dictionaries, one per subdirectory, containing key-value pairs
of the statistics in stat_names and the associated results.
    :rtype: list(dict(str: float|int))
    """


"""
db_helper.py
Authors: Oncospace group, Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Helper functions for querying database and working with patient masks.
"""

def mult_ROI_masks(roi_list, pRepID_list=None):
    """
    Helper method to combine several ROIs into one mask per patient.

    :param roi_list: The list of ROIs to include in the masks
    :param pRepID_list: Optional list of patient representation IDs

    :type roi_list: list[str]
    :type pRepID_list: list[int]

    :return: A dictionary of patientRepID: mask object key-value pairs, where the mask
is the mask combining the provided ROIs
    :rtype: dict[int: framework.Utils.image.mask]
    """

def cloud_to_mask(cloud, size=None, patient_mask=None):
    """
    Given a point cloud, turns it into a binary mask (image.mask). Either size or a
reference patient mask must be provided (providing both is acceptable as well). The
function either treats the points as a multi_index to the volume with shape 'size' if
patient_mask is None, or generates the necessary indices in the volume of the
patient_mask for the points in the cloud if it is provided. If the point cloud is from
an atlas, the mask of the fixed patient is an ideal choice for the patient_mask
parameter.

    :param cloud: The point cloud to convert, an Nx 3 array of 3D points
    :param size: The shape of the desired volume
    :param patient_mask: An image.mask object, whose volume will be used to host the
points from 'cloud'

    :type cloud: N x 3 numpy.ndarray
    :type size: tuple(int) (x, y, z)
    :type patient_mask: framework.Utils.image.mask

    :return: An image.mask object generated from the points in the point cloud.
    :rtype: framework.Utils.image.mask
    """

def plot_mask(mask1, mask2=None):
    """
    Generates a 3D scatter plot of the points in mask1, and likewise for mask2, on the
same set of axes, if provided.
```

```
    :param mask1: image.mask object to plot
    :param mask2: second mask to plot

    :type mask1: framework.Utils.image.mask
    :type mask2: framework.Utils.image.mask

    :return: None
    """

def test():
    """
    Make sure the utility functions work as expected!

    :return: None
    """


"""
DoseAccumulation.py
Authors: Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Functions to accumulate dose inside a given region (Defined in atlas coordinate
space) in corresponding anatomical
regions in multiple patients.
"""

def accumulateDose(region, atlasfilename, atlasname, numberpats, roi_list):
    """
    Returns dose masks for a given region in multiple patients.

    :param region: Point cloud representing boundaries of region (should be
tetrahedral)
    :param atlasfilename: File name of atlas
    :param atlasname: Name of atlas
    :param numberpats: Number of patients to query dose grid for
    :param roi_list: List of regions included in atlas

    :type region: np.array(float64)
    :type atlasfilename: str
    :type atlasname: str
    :type numberpats: int
    :type roi_list: [str]

    :return: Dictionary of dose masks of desired region in numberpats patients
    :rtype: dict
    """

def getPatIDS(roi_list, num=-1):
    """
    Returns patient IDs for desired number of patients that have contoured ROIs for
all ROIs in roi_list, or all
    patients with desired ROIs if no number specified.

    :param roi_list: list of desired region of interest
    :param num: number of patients
    :type roi_list: [str]
    :type num: int

    :return: List of patients with desired ROIs
    :rtype: [int]
    """

def get_dosemap(t_region, patID, radID=-1):
    """
    Get dose inside given region for patient with patID for radiotherapy session
radID.

    :param t_region: Desired region. Should be tetrahedral (8 points representing
corners).
    :param patID: Patient ID
    :param radID: Radiotherapy session ID, default = most recent
    :type t_region: np.array(float64) 3 x 8
    :type patID: int
    :type radID: int

    :return: dose inside desired region
    :rtype: framework.Utils.dose_map.dose_mask
    """

"""
Sectors.py
Authors: Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Functions to define sectors and grids as point clouds in atlas space, and provides
utilities to perform registrations between the atlas and specified patients, as
well as the utilities to find the dose within a defined sector.
"""

def get_atlas(filename):
    """
    Read an atlas from a file and return it as an atlas object.
    :param filename: .oaf file containing atlas
    :type filename: str
    :return: AtlasCalculator object of stored atlas
    :rtype: AtlasCalculator
    """


def query_test_patient(reg_names, num=1):
    """
    Returns a random patient from the Oncospace db that has all of the regions of
interest contoured.
    :param reg_names: list of regions of interest
    :type reg_names: [str]
    :return: random patient that has all of regions of interest contoured
    :rtype: int

def query_dose_grid(patientRepID, rad_ID_num=-1):
    """
    Returns the dose grid for a given patient ID, default is first radiotherapy
session if no session specified.
    :param patientRepID: Patient ID from Oncospace database
    :param rad_ID_num: Radiotherapy session number
    :type patientRepID: int
    :type rad_ID_num: int
    :return: Dose grid for specified patient/radiotheapy session
    :rtype: Dictionary with:
                :key:   RadiotherapySession.ID (rad_ID_num)
                :value: a dose.dose() instance
    """
```

```
def register_atlas_to_patient(fixed_patient_cld, patID, roi_list, saveToMat=False, 
atlasname=None):
    """
    Register an atlas to a patient and return registered point cloud and information 
from registration method.
    Note: uses registration method from /normalization/cpd/cpd_nonrigid.py where t = y 
+ g*wc
    :param fixed_patient_cld: point cloud representation atlas ("fixed patient")
    :param patientID: ID of patient to register atlas to
    :param roi_list: list of organs from moving patient to include
    :param saveToMat: Flag indicating whether to save the atlas point cloud and 
registration return values to a .mat
                      file. The file holds a dict containing:
                      'atlas': The atlas point cloud
                      'transformed': The transformed points (patient transformed to 
atlas)
                      'g': G matrix from CPD calculation; see 
framework.Normalization.cpd.cpd_nonrigid for details
                      'wc': matrix of coefficients
                      'errors': Variance of position differences between point 
correspondences
    :param atlasname: If not None, includes this atlas name in the name of the .mat 
file saved.

    :type fixed_patient_cld: numpy.ndarray(numpy.float64), N x 3
    :type patientID: int
    :type roi_list: list(str)
    :type saveToMat: bool
    :type atlasname: None | str

    :return: A tuple containing:
            T: transformed points (patient transformed to atlas)
            g: G matrix from CPD calculation; see 
framework.Normalization.cpd.cpd_nonrigid for details
            wc: matrix of coefficients
            errors: Variance of position differences between point correspondences

    :rtype: tuple(numpy.ndarray(numpy.float64)) N x 3, numpy.ndarray(numpy.float64), 
numpy.ndarray(numpy.float64),
            list(numpy.ndarray(numpy.float64)))
    """

def get_mask(patID, roi_list, sampling=(0.25, 0.25, 1), crop=True):
    """
    Get a combined binary mask with all the desired ROIs for the patient.
    :param patID: The patient representation ID of the patient to get a mask from
    :param roi_list: The list or ROI names to include in the mask
    :param sampling: Tuple (x, y, z) of fraction of points in each dimension to use 
for downsampling
    :param crop: Boolean indicating whether to crop the resulting mask

    :type patID: int
    :type roi_list: list(str)
    :type sampling: tuple(float)
    :type crop: bool

    :return: The binary mask representation of the patient
    :rtype: numpy.ndarray(numpy.float64) X x Y x Z
    """
```

```
def simple_grid(x, y, z, center, spacing):
    """
    Return a grid with specified center and x, y, and z lengths
    :param x: length along x axis
    :param y: length along y axis
    :param z: length along z axis
    :param center: center of the volume
    :param spacing: spacing between each point
    :type x: float
    :type y: float
    :type z: float
    :type center: list(float)
    :type spacing: tuple(float)
    :return: grid
    :rtype: numpy.ndarray
    """


def simple_grid_helper(center, spacing, min_y, max_y, min_z, max_z):
    """
    Appends an array around the given center points from the min_y to max_y and min_z
to max_z points with the given spacing.
    :param center: center point to build array around
    :param spacing: distance between points
    :param min_y: minimum y point
    :param max_y: maximum y point
    :param min_z: minimum z point
    :param max_z: maximum z point

    :type center: list(float)
    :type spacing: tuple(float)
    :type min_y: float
    :type max_y: float
    :type min_z: float
    :type max_z: float

    :return: array around given center point
    :rtype: np.array(float64)
    """

def getSectorMap(filename, sector, roi_list, visualize=False, patID=-1,
saveToMat=False):
    """
    Get an atlas point cloud registered to a patient with a transformed sector
delineation
    :param filename: the filename of the catlas
    :param sector:  a point cloud representing the sector in atlas coordinate frame
    :param roi_list: a list of the regions of interest for the patient
    :param visualize: if True, the method will plot the before and after
transformations
    :param patID: patient ID, if left empty, a random patient will be chosen using the
roi_list
    :param saveToMat: by default, false, but if true, will save to a .mat file called
sectors.mat
    :type filename: str
    :type sector: numpy.array N x 3
    :type roi_list: list[str]
    :type visualize: bool
    :type patID: int
    :type saveToMat: bool
    :return: A point cloud of the atlas and the sector together and as separate objects
    :rtype: numpy.array N x 3
"""
```

```
def get_dose_inside_sector(patID, mode='explicit', *args, **kwargs):
    """
    Returns the dose inside a rectangular prism (or tetrahedral prism). This can
either be defined implicitly in the
    patient space, or explicitly after an external deformation.

    :param patID: patient representation ID
    :param mode: Choice of sector grid definition mode. Default is 'explicit', which
accepts a dense point cloud
                 representing the region to be queried as a positional argument.
Alternative is 'implicit',
                 which defines a center point and distances along x, y, and z axes.
    :param args: Positional arguments. If mode is 'explicit', the only argument is the
point cloud defining the
                 queried region:
                    sector_cloud: The dense point cloud representing the sector where
dose information is desired.
                                  (likely from SectorUtils.find_points_in_def_cube)
                 If the mode is 'implicit', the accepted arguments are:
                    x: length along x axis
                    y: length along y axis
                    z: length along z axis
                    center: center of the volume
    :param kwargs: Keyword arguments:
        rad_ID_num: Radiation therapy session number to query; default is the first

    :type patID: int
    :type mode: str
    :type args: list(numpy.ndarray([[numpy.float64]])) | list(float, float, float,
tuple(float))
    :type kwargs: None | dict(str, int)

    :return: A dose_map.dose_mask object instance containing dose information within
the region specified by
             cornerpoints
    :rtype: framework.Utils.dose_map.dose_mask
    """

def _get_mask_helper(prepID, reg):
    """
    Helper function for get_dose_inside_sector. Returns a mask object at full
resolution for a single region of interest in the given patient.

    :param prepID: The patient representation ID of the patient to query
    :param reg: The ROI name of the region to get

    :type prepID: int
    :type reg: str

    :return: A fully-sampled mask of the queried region for the given patient
    :rtype: framework.Utils.image.mask
    """


def get_corner_points(indices, grid, x, y, z, spacing):
    """
    Returns the 8 corner points for the cube in the grid indicated by cube index.
Indices should be given in x, y, z
    order and begin from the -x, -y, -z corner of the grid.
    :param indices: [x, y, z] indices of the desired section of the grid. indexed 0 to
```

```
n-1
    :param grid: point cloud representing a once-regular grid that has been
transformed into patient coordinate space
                or a regular grid
    :param x: length along x axis (BEFORE transformation)
    :param y: length along y axis (BEFORE transformation)
    :param z: length along z axis (BEFORE transformation)
    :param spacing: tuple representing spacing along x, y, z axes. can also be int
representing same spacing along each
                    axis.
    :return: point cloud with the 8 corner points of the desired section
    """

def downsample_pc(msk_pts, samplefrac):
    """
    Uniformly down-sample a point cloud
    :param msk_pts: point cloud to be down-sampled
    :param samplefrac: fraction of points to be left in msk_pts, either a single float
or a tuple(x, y, z)

    :type msk_pts: numpy.ndarray(numpy.float64) N x 3
    :type samplefrac: float | tuple(float)

    :return: down-sampled msk_pts
    :rtype: numpy.ndarray(numpy.float64) N x 3
    """

def getRegistrationFile(filename):
    """
    Load registration data from a .mat file
    :param filename: location and name of file
    :type filename: str
    :return: all the data within the file
    """

def getGridDataFromfile(filename):
    """
    Get the grid data from a saved file
    :param filename: the name and location of the file
    :type filename: str
    :return: transformed grid and original grid as well as runtime
    """

"""
SectorsUtils.py
Authors: Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Utility for trimming a grid to the borders of a deformed polyhedral shape,
necessary for densely populating a given sector in order to query dose from the
database.
"""

def find_points_in_def_cube(d_cube, grid):
    """
    Return the points from a grid that are inside an arbitrary shape
    :param d_cube: the arbitrary shape point cloud
    :param grid: the grid
    :type d_cube: numpy.array Nx3
    :type grid: numpy.array Dx3
    :return: the points that lie inside the arbitrary shape
    :rtype: numpy.array Kx3
"""



"""
TPS.py
Authors: Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Utility for fitting a Thin Plate Spline transform to a given Atlas-to-Patient
registration, providing a function to take arbitrary points from atlas space to
patient coordinate space. It allows the saving of TPS parameters which greatly
increases the speed of the function.
"""

def register(atlcld_before, atlcld_after, ptcld, loadFromMat=False, filename=None,
saveToMat=False, patID=-1, atlasname=None):
    """
    Perform a thin plate spline warping to gain a mapping function
    :param patlcld_before: the atlas point cloud before registration to the patient
    :param atlcld_after: the atlas point cloud after registration to target patient
    :param ptcld: the sector point cloud in atlas coordinate frame
    :param loadFromMat: if True, loads parameters from file
    :param filename: filename to load parameters from
    :param saveToMat: if True, saves parameters to file
    :param patID: patient ID, used for creating filename
    :param atlasname: atlas name used for creating the filename for saving parameters
    :type patcld_before: numpy.array Nx3
    :type atlcld_after: numpy.array Nx3
    :type ptcld: numpy.array Dx3
    :type loadFromMat: bool
    :type filename: str
    :type saveToMat: bool
    :type patID: int
    :type atlasname: str
    :return: the sector point cloud in patient coordinate frame
   :return: runtime
    :rtype: numpy.array Dx3
    :rtype: tim: float
    """
```

EDITS TO EXISTING CODE ARCHITECTURE:

```
"""
visualize.py
Authors: Oncospace Group, Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

Updated Functions: visualizePointCloud - updated to allow the display of different
colors for up to two different regions
"""

def visualizePointCloud(cloud, title=None, alpha=1.0, num_points=0, state=0,
pt_size=0):
"""
    Visualize a point cloud with certain points highlighted with a different color
    :param cloud: a Nx3 array of all the points to be visualized
    :param title: Title for the graph displayed
    :param alpha: The radius of the sphere representing each point in the scatterplot
    :param num_points: The number of points to highlight (these points to be
highlighted must be at the end of the array.
    :param state: If set to 1, highlighting will occur, if 0, no highlighting will
occur.
    :param pt_size: At 0 the highlighted points will be displayed at the same alpha as
the specified beforehand. When toggled to 1, the highlighted points will have a
smaller alpha than the other points.
    :return: None
    """

def visualizePointCloudByOrgan(cloud, npoints_per_organ, title=None, alpha=1.0):
    """
    Visualizes a point cloud representing multiple organs by plotting each organ in a
different color.
    :param cloud: sorted point clouds containing points from multiple organs
    :param npoints_per_organ: number of points in each organ, in same order as they
are sorted in the point cloud
    :param title: plot title
    :param alpha: opacity of scatter plot points

    :type cloud: np.array npoints x 3
    :type npoints_per_organ: np.array norgans x 1
    :type title: str
    :type alpha: float

    :return: None
    """

"""
dose_map.py
Authors: Oncospace Group, Chris Micek, Arun Raghavan, Julie Shade
Updated: May 2017

New Functions: apply_function - New function in dose_mask class that allows the
application of an arbitrary function to the dose data of the mask.
"""


def apply_function(self, f, *args, **kwargs):
    """
    Apply an arbitrary function f to the dose data of the mask. If the dose mask data
should not be the first argument of f, f may be a lambda function with the correct
arrangement of arguments.

    :param f: The function to apply; must accept a numpy array as a parameter.
    :param args: The positional arguments for f
    :param kwargs: The keyword arguments for f

    :type f: types.FuntionType
    :type args: list
    :type kwargs: dict

    :return: The result of the application of f on the dose mask data.
    :rtype: T
    """
```