3D Medical Imaging

Achintya Gopal
agopal2

Riley Scott
rscott39

Paul Watson
pwatso14

William Watson
wwatso13

Abstract

This project seeks to take CT scans of a patient and create a 3D mesh of the head. In addition, we will discover pertinent facial features to develop a feature vector encoding for a database to identify the patient.

1 Introduction

To create a database to classify 3D meshes by patient identification number, we had to first collect data. The data we used was a collection of axial CT scans of the body for patients with head and neck cancer undergoing chemo-radiation therapy. However, the data was not sorted. We first had to spend time to sort the data from the top of the head to the top of the shoulders since for our project we only used CTs of the head and neck.

After spending time sorting the data, we processed the images to remove noise and any extraneous details not concerning the head. After we stacked the processed images, we applied the Marching Cubes algorithm to convert this into a mesh of the head.

Since we have the meshes, we had to figure out how to create feature vectors out of these meshes. We used two methods: a Fast Fourier Transform taken from the Trimesh library and a descriptor of the eyes and nose of the mesh. For the descriptor of the eyes and nose, to keep the value invariant under transformations, we used ratios.

After constructing these feature vectors, we applied a simple nearest neighbor algorithm to label test data.

2 Data Processing

2.1 What is the Data?

Axial CT scans of cancer patients taken in the late 1990s to early 2000s (1,) (2,).

2.2 What Parts of the Data did we use?

We isolated the images from the forehead to the base of the neck. The end goal was to be able to identify a patients ID number from a 3D mesh, so we chose the face because it is the most a differentiating feature between people.

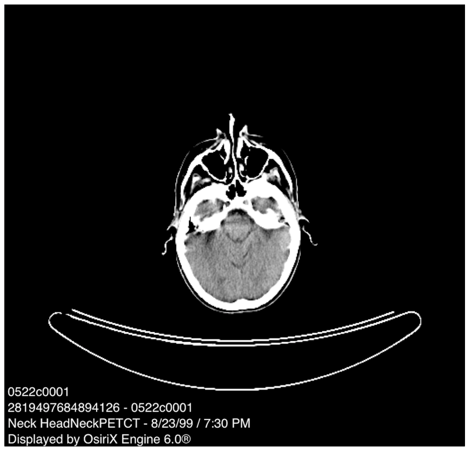
0522c0001
2819497684894126 - 0522c0001
Neck HeadNeckPETCT - 8/23/99 / 7:30 PM
Displayed by OsiriX Engine 6.0®

Figure 1: Sample CT Scan

2.3 Voxel Image Creation

We need to create a voxel image of the head out of the individual CT images in order to create a mesh. The CT scans contain objects and noise that we want to ignore when creating our meshes. Things like the platform that the head rests on and other equipment show up in the images that need to be removed.

The first step in preparing the images is to perform a binary thresholding using Otsu. Otsu binarization automatically calulated a threshold value from the image histogram for a bimodal image, or an image where the histogram has two peaks. In our implementaion, we used OpenCV's Otsu binarization. OpenCV's implmentation finds the optimal threshold value that minimizes the weighted within-class variance, and finds a threshold that lies inbetween the two peaks in the histogram. This minimizes the variances to both classes.

This removes the majority of the background noise and some of the smaller parts of the equipment that only faintly appear in the raw scans. With this binary image, we label the connected components with the assumption that all parts of the head will be connected in each image, and that most of the machinery in the image will be separated. To find which label represents the head, we look at a window in the center of the image and count the number of times each label appears, saving the most common label as the head. Each patient is mostly aligned in the center of the image, so the label for the head is always the most common if there is a head in the image.

Once the heads label is identified, we perform an in-range threshold on the labeled image to isolate the head. This removes almost all the extraneous information from the image. The only exceptions are any objects that might be touching the head, and also are bright enough to not be erased by the initial binary thresholding.

In general, the objects that are touching the head are only outlines. We use this to our advantage by opening the image. This erases any thin areas such as, most notably, any outlines of objects touching the head. To return the head to its original shape, we dilate the image and then we have a binary image where the white pixels are exactly the pixels of the head in the original image, and the black pixels are the extraneous information in the background.

The final step is to perform a bitwise AND of the original image and the binary mask to isolate the head. This final image is then appended to a list of images. After all the images in a single set of CT scans are processed, the list is converted to a Numpy array, creating a single 3D voxel image.

3 Marching Cubes

The Marching Cubes algorithm is a commonly used algorithm to generate meshes from voxel-based images (3,). Since in the last step we stacked the images of the head and neck and calculated the distance between slices to create a voxel-based image, we were able to apply the Marching Cubes algorithm.

The Marching Cubes method processes eight voxels at a time. The algorithm sees how the object, in our case the head and neck, intersects the voxels. Since each voxel can either have an object or not, there are $2^8 = 256$ different cases. However, due to rotational symmetries, these cases can be reduced to 15 cases (4,). For the case where there are more than 4 intersections, the algorithm looks at the voxels that do not have an object.

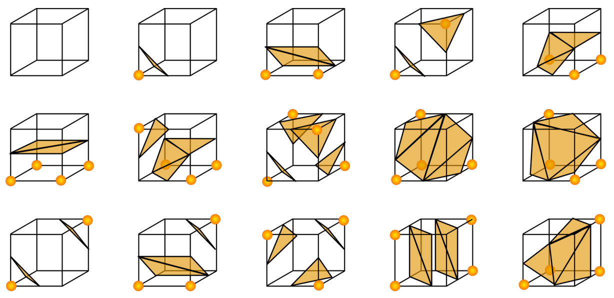

Figure 2: Marching Cubes Base Cases

We created a configuration lookup table which we could use to create all the triangles needed to form a mesh. We then saved the mesh in a Wavefront .obj file. To create a Wavefront file, the algorithm prints a list of all the points and then a list of the triangles based off the indices of the points.

4 Nose and Eye Detection

The overall process of detecting the eyes and nose in a given mesh is to form a 2-dimensional frontal projection, smooth the image, and perform an algorithm we call ”minimum descent” to discover the bounding ellipses for the right eye, left eye, and the nose.

4.1 Mesh Projection

From the marching cubes algorithm, we have created a 3-dimensional triangular mesh of our patient's

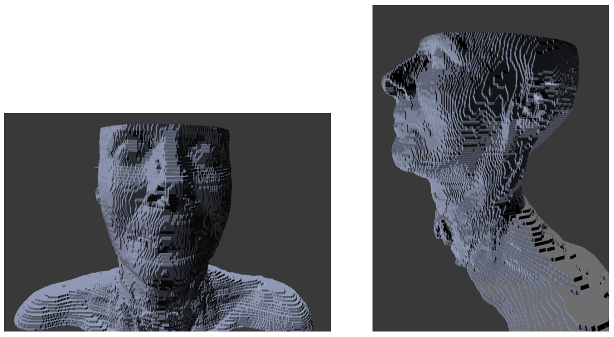

Figure 3: Sample Mesh Results

head. However, traversing the mesh to find the nose and eyes is not trivial. Therefore, to simplify the computational costs and accuracy of our detection, we project the front of the mesh onto the 2-dimensional image plane.

For this projection algorithm, we had to define the axis for our projection. Hence, our columns correspond to the x value of the mesh point. Our rows, however, correspond to the z value of our mesh point. Therefore, the pixel value at $I(z, x)$ will contain information on the y value of our mesh point.

Our algorithm looks at every face and then every point on that face to determine if we will fill the corresponding pixel value. Due to implementation details, we had to scale our x and z values by two to preserve the original patients' ratios.

Since we want the minimum y value, we construct our depth image as follows:

$$y_{val} = 2 \times y_{dim} - y_{point} + 1$$

Where y_{val} will be the pixel value to place in the image, y_{dim} is the size of the y dimension of the mesh, and y_{point} is the y coordinate of the face point currently being processed.

To set a pixel value, if the current value at $I(z, x) = 0$, we overwrite the value with y_{val}. Otherwise if $I(z, x) < y_{val}$, we set the pixel value to y_{val}. Otherwise, we do not update the pixel value.

After traversing all the faces and points, we have a 2-dimensional depth image. We normalize the image to the maximum value in the image. It is important to note that because we are dealing with triangular meshes, not every pixel value is filled, and hence our image has holes in it.

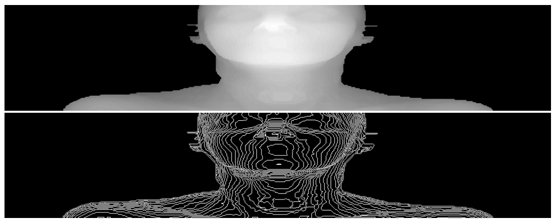

Figure 4: Projection after smoothing, Laplacian to detail contours

4.2 Projection Smoothing

Our resulting projection has holes in it. We solve this by applying a morphological closing with a 4×4 elliptical kernel. This smooths the image and fills any holes with a similar value to the original projection. This allows the depth image to become dense and form a contour map of the face. We also scale every pixel value in the image by a constant multiplier which for our purposes is 700.

4.3 Nose Detection

A general assumption made for our projections is that the pixel value that has the largest value corresponds to the point of the patient's face that is closest to the point of reference. Since our image is a depth map of the frontal face, we can assume that the maximum value in the image is the nose. Hence, in order to find the nose in a given frontal face projection, we simply find the maximum value and location in the image. However, if the maximum point in the image is not the nose, which in some cases the maximum value was the chin, then it cannot properly find the nose.

4.4 Minimum Descent and Eye Detection

This step is predicated on the nose's position. We found that if we found the patient's nose, then we always could find the eyes properly. However, if the nose is not the maximum value, then eye detection is not possible.

We developed an algorithm similiar to the idea of gradient descent to find both the right and left eyes. The idea comes from the fact that the eye sockets are local minimums in the depth image, and when we start from the global maximum, we can decend by following the natural contours of the nose to the eye sockets.

The first step of our minimum descent is setting a seed point. This seed point is found by a constant offset from the nose. For our purposes, if (r, c) is the nose point, then our seed point for the left eye descent is $(r-20, c-20)$ and for the right eye $(r-20, c+30)$. This ensures that our algorithm descends properly.

We descend while our local minimum is not the center of the window we are looking at, so when given a patch around our current point, we find the minimum value location and then update our window size. The update function for the window is related to the size of the descent, so it could never jump more than it previously moved. This method is used to find the right and left eyes.

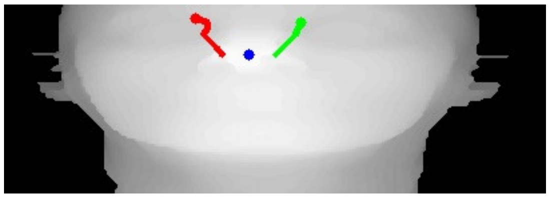
Figure 5: Minimum Descent Pathway

4.5 Flood Fill

After the minimum descent places our keypoints in the eye sockets (local minimums), we flood fill the cavity to highlight the eyes and nose as distinct facial features. There are also slight pertubations in the seed points for the eyes. This is done for more accurate results.

One challenge to flood filling the eyes was that, in some patients, the gradients between the eyes did not significantly change, and flood fill would mistakenly color both eyes at once. In order to resolve this issue, we blacked out the entire image except for a square region around the seed point. We do this to prevent flood fill from coloring more of the image than what is necessary. We then add the original image minus the square region to the flooded image to get the original image back with coloring. This is done for all three facial features.

4.6 Fitting Ellipses

To find the bounding ellipses, we need to find our facial features. Since we flood filled our facial features, we set the nose to blue, the left eye to green,

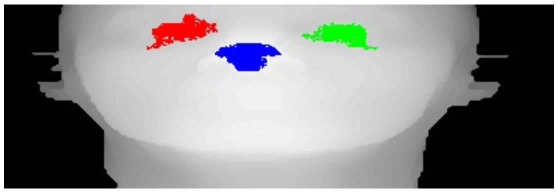
Figure 6: Flood Filled Facial Features

and the right eye to red. Since they are distinct colors, we do a simple in-range threshold for each color. Then, we find the bounding contour. Afterwards, we take the largest area contour and bound an ellipsis to the contour. These contours contain the center, major and minor axis lengths, and the angle of the ellipse.

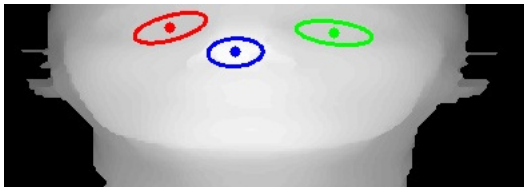
Figure 7: Fitted Ellipticals

5 Feature Vector Construction

The first features we use are the roundedness of both the eyes and the nose. We defined the roundedness as the ratio of length of the major axis of the ellipse representing the body part over the length of the minor axis of the ellipse. We also used the ratio of the distance between the eyes over the distance between the center of the eyes to the nose. We used the ratio of the distance between the eyes over the width of the face. To find the width of the face, we found the line that intersects the centers of both of the eyes in the projection image. We follow this line from each eye, moving outward, until we encounter a blank pixel. We then use the distance between the furthest non-blank pixels on this line as the width. We use the ratio of the distance between the right eye and the nose over the distance between the left eye and the nose. The last feature is the angle between the rays extending from the nose to each of the eyes.

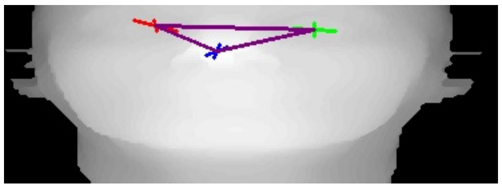

Figure 8: Ratios used for our Feature Vectors

6 Results

6.1 How we Tested

We used two feature vector encodings from each patient as our database. We used the remaining encodings for testing. For each testing encoding, we compared it to all the encodings in the database by finding the Euclidean distance between them. We used the label of the closest point as our prediction. We did this with our five patients, and we correctly predicted one of them.

6.2 What we tested against

We compared our results with the results from a third party feature vector encoding. We used the rotationally invariant feature function from the Trimesh library. This function uses the Fast Fourier Transform to compute the features. Using this vector, we also correctly predicted one out of five patients.

6.3 The Results

These results were expected. We encountered difficulties because of the inconsistencies between the CT scans. The patients were not always laying with the same postures which changed the locations of the extrema. Some patients had their upper lips or chin as the most foward point of the face, i.e. the highest point in the projection image. This not only affects finding the nose, but also the eyes. We use the position of the nose to find the eyes, so a bad starting point led to bad results for the eyes as well. These different positions also changed the contours that the flood fill function uses to calculate the areas to fill.

7 Conclusion

7.1 Mesh Generation

Our mesh generation algorithm using Marching Cubes which gives us distinguishable faces regardless of the inconsistent quality of the original CT scans. Every patient had a recognizable mesh that could be distinguished from the others.

7.2 Feature Vectorization

The 2D frontal projection with smooting always resulted in a high quality image. The depth map always modelled the contours of the face. However, due to the original posture of the patient, our method for finding the eyes and nose only worked when the nose was the global maximum. This led to faulty results when identifying the patient as the ratios were not reflective of their intended meaning. We believe that having access to higher quality CT or MRI scans would improve our results.

Github Repository

All of our code can be found on our Github repository:
`https://github.com/achintyagopal/3DMedicalImaging`

References

[1] Bosch, Walter R., Straube, William L., Matthews, John W., and Purdy, James A. 2015. *Data From Head-Neck Cetuximab.* The Cancer Imaging Archive. Link

[2] Clark K, Vendt B, Smith K, Freymann J, Kirby J, Koppel P, Moore S, Phillips S, Maffitt D, Pringle M, Tarbox L, Prior F. 2015. The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository, Journal of Digital Imaging, Volume 26, Number 6, December, 2013, pp 1045-1057. (paper)

[3] Chernyaev, Evgeni V. 1995. *Marching Cubes 33: Construction of Topologically Correct Isosurfaces.* Institute for High Energy Physics, 142284, Protvino, Moscow Region, Russia. (paper)

[4] Marching Cubes Graphic on Original 15 Cases. Image Link