The use of Tcl/Tk in the AnatLab Virtual Anatomy Laboratory

Cyndy Lilagan, Mike Doyle, Steve Huntley, Steve Landers

Iomas Research LLC
Wheaton, Illinois

ABSTRACT

The AnatLab system provides an interactive Web environment for exploring human anatomy, including the locations, identities, relationships and extents of anatomical structures, as well as associated textual information. An early version of AnatLab was demonstrated at Tcl2008[1], and this paper goes beyond that to describe a number of the Tcl-based technologies used to create AnatLab.

1 Introduction and Background

The National Library of Medicine Visible Human [2] male data set includes a series of 1,871 anatomical images representing 1mm axial (i.e. transverse or cross-section[3]) color cryosections through a male cadaver.

The Iomas AnatLab system is a Web-based navigator that allows users to browse through these sections. It also contains an atlas of the structures on each section, allowing the user to click on any area of any section through the body to have the system immediately identify the anatomical structure and provide a variety of information related to that structure.

AnatLab uses a technique called zMapping to achieve this. An x,y coordinate in a section image is used to look up the the color value at the same location in a secondary colormap of 24-bit voxels. AnatLab uses the RGB triplet found in the secondary map as a unique 24-bit object index into a database containing related anatomical structure information.

The process of identifying the boundaries of a structure on a particular section is called annotation. AnatLab has over 2500 structures identified in this way – a total of over 150,000 individual annotations in the axial plane alone. But AnatLab also supports sections in the coronal (front-to-back) [4] and sagittal (left-to-right) [5] planes using images derived from the axial sections. This involves a total of over 4600 sections and 700,000 annotations overall.

The AnatLab client is a browser-based application that uses modern Web techniques such as Ajax[6] and the HTML5 canvas[7] to provide a rich and responsive user experience.

The AnatLab back-end infrastructure comprises a number Tcl-based technologies, including:

- a Tk-based graphical anatomical annotator
- a Tcl-scripted annotation processor for generating the zMap files and databases
- Ajax Web services based on a Tcl Web server
- Tcl services that support the flash-based search plugin
- a database for fast hashed access to anatomical structure information
- a Tcl anatomy name server that replaced a slower and more complex Java version
- a rendering system for generating 3D virtual reality models

It is the use of Tcl/tk in this back-end infrastructure that is the focus of this paper. The conference presentation will also include an extended demonstration of the AnatLab client.

2 The Anatomical Annotator

The AnatLab annotator is a Tk-based graphical annotation tool used by anatomical experts to define the boundaries of structures on each section in the Visible Human data set.

Annotation is an ongoing activity: the members of the 2009 summer annotation team were the head anatomist, a neuro-anatomist, seven medical students, a biomedical visualization student, a programmer and the quality control team. Over 11,000 annotations were created involving approximately 100 new anatomical objects.

The user (i.e. the anatomy expert) chooses an anatomical object name (or creates a new one) and identifies the object by drawing polygons on magnified cross-sectional images. The polygons can comprise multiple line segments or smoothed splines; and can be edited, copied or pasted to other sectional images.

The polygons are saved to a text file (called a map file) for post-processing into the zMap colormap. The map files contain the name and coordinates of each anatomical object, a time stamp, the author, the start point and sectional image number. The map file also includes information to assist in subsequent processing, e.g. whether the polygon is segmented or smooth, or if there are restrictions such as only overwriting particular structures like connective tissue or subcutaneous fat.

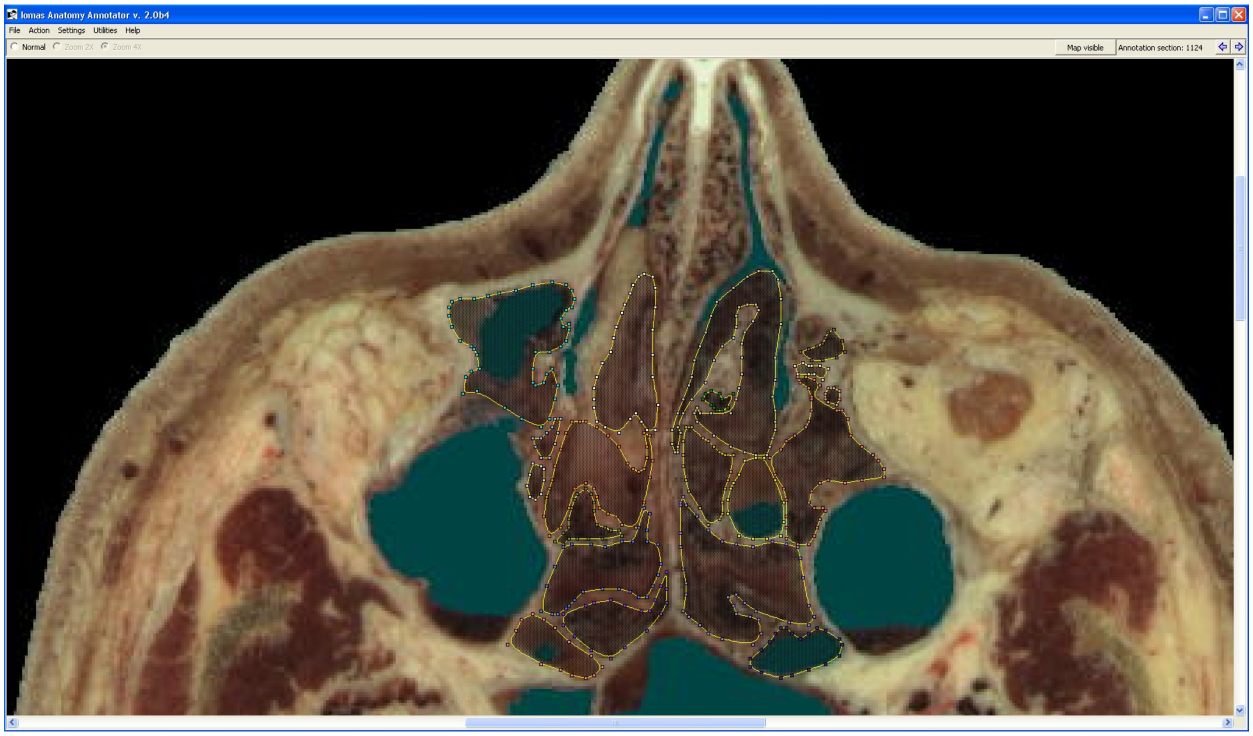
The annotator uses a number of Tcl and Tk extensions and techniques (including a number from the Tclers' Wiki) including:

- Canvas item selections [8]
- Simple zooming and scaling in a canvas [9]
- Image scaling [10]
- Drawing and editing polygons [11]
- Canvas Zooming [12]
- Colors with Names [13]
- Maze Solver [14]
- Canvas: Polygons from lines [15]
- Getting the Canvas View Area in Pixels [16]

The drawing commands were enhanced to support magnification and scaling. An accurate representation of the magnified drawing was achieved by allowing the drawing of nodes only onto pixel locations that would persist once the image was scaled back down to its original resolution. The copy/paste commands for canvas objects were enhanced to allow storage of the identification and processing information for each object.

An edge walking utility is used to define the polygons for already-defined objects, and to create map files (i.e. polygons) to be used in the Annotator program. Given a structure's RGB value, the edge walker will detect the boundaries of the object within a given section and then generates polygons to represent these.

The following screen image shows the annotator being used to annotate the ethmoid sinuses.


Iomas Anatomy Annotator v. 2.0b4
File Action Settings Utilities Help
Normal Zoom 2X Zoom 4X
Map visible
Annotation section: 1124

3 The Annotation Post-processing

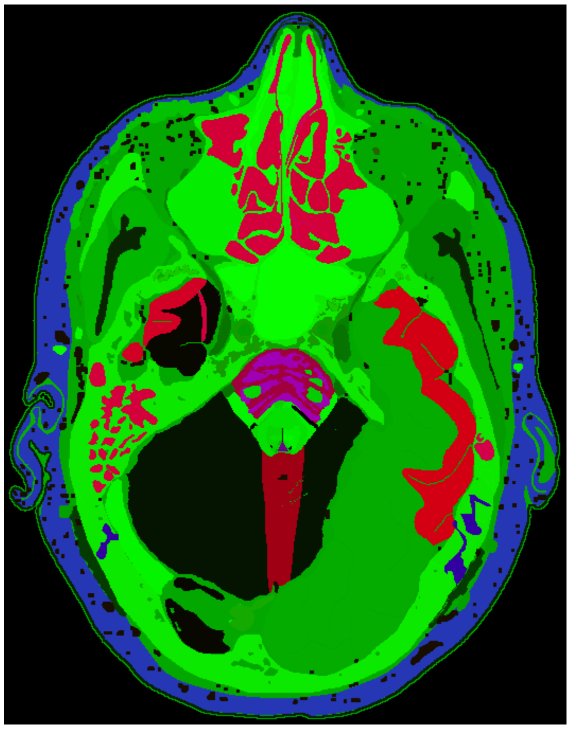
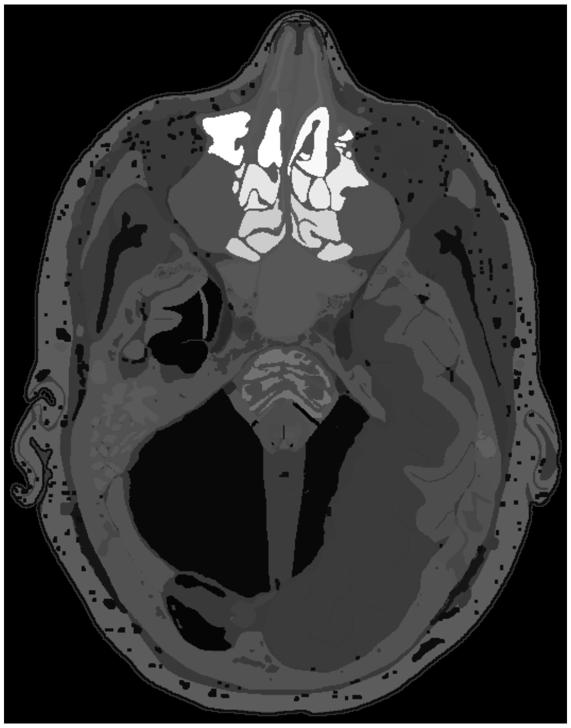
The Annotation processor is a Tcl program that takes the annotator map files and generates zMap colormap images corresponding to each section. A separate imagemap is generated for each section. Each one contains the RGB color images for all the structures identified on the section.

The processor performs quality control and consistency checking along the way, including checks for overwrite restrictions (some structures such as connective tissue are routinely overwritten as annotations improve).

Automated quality and consistency checks also include identifying:

- objects (i.e. annotated structures) that do not have a contiguous range of sections
- objects without names
- colors in the colormap images that do not have an associated structure name in the database
- unused colors in the RGB space
- colors in the database that do not appear in colormap images

The following show the output from the annotation post-processing: the left-most image shows a colormap image corresponding to the above Annotator screenshot, and the right image shows the grayscale equivalent with the ethmoid sinuses highlighted.

The annotation processor uses the SQLite database [17] for storage of structure and polygon information during processing.

4 Generating Derived Sagittal and Coronal Sections

It is helpful to be able to view anatomical structures from various angles, to get a better sense of their orientation in the body. Although the Visible Human dataset only contains sections in the axial plane, the AnatLab zMap database contains the coordinates of structure boundaries as a collection of three-dimensional data. So extraction of coordinates for a given structure along any desired plane is possible. Given this, it was decided to add coronal and sagittal support to the AnatLab client: a feature requested by a number of the AnatLab users.

The AnatLab user interface was enhanced to support coronal (front) and sagittal (side) orientation for viewing features of interest, and section images in these two planes were generated (i.e. derived) from the axial section images. This was done by:

- decompressing the axial section image files
- assigning each raster line of the images to individual files
- re-assembling the rasters in new configurations to produce contiguous image files in the desired orientations
- re-translating the resulting image files to JPEG format.

Several utilities from the Netpbm toolset [18] were used to accomplish this task, coordinated by a Tcl script. In order to take maximum advantage of available multiple core computer resources, key calls to Netpbm utilites to process large sub-groupings of files were put into makefiles, since GNU make has the ability to specify that jobs be run in parallel processes. Necessary parameters for the makefiles were calculated in the controlling Tcl script and passed from there to make. Processing of files for each orientation required about a week of continual processing on a 4-CPU desktop computer.

Tcl was an essential tool in this undertaking, because the huge amounts of data which required processing stressed the available hardware resources and software tools to their limit and beyond. Tcl was the only tool used whose mean time between failures was not less than the total time required to complete each task. The exceptional reliability and stability of Tcl, in addition to the ease with which external processes can be called, controlled and errors trapped, made Tcl indispensable.

5 The Anatomical Name Server

The zMap database contains over nine Gigabytes of colormap data that must be accessible in real time from the AnatLab client. The database is designed to be a fast read-only database of large amounts of multi-dimensional information. It takes the form of a set of colormap TIFF images. The zMap concept takes advantage of the fact that in a TIFF image, small sets of raster lines can be compressed individually. The header of a given TIFF file contains offsets to the location in the file of the start point of each set of separately-compressed lines, thus an individual pixel can be read quickly by accessing a small subset of the file directly.

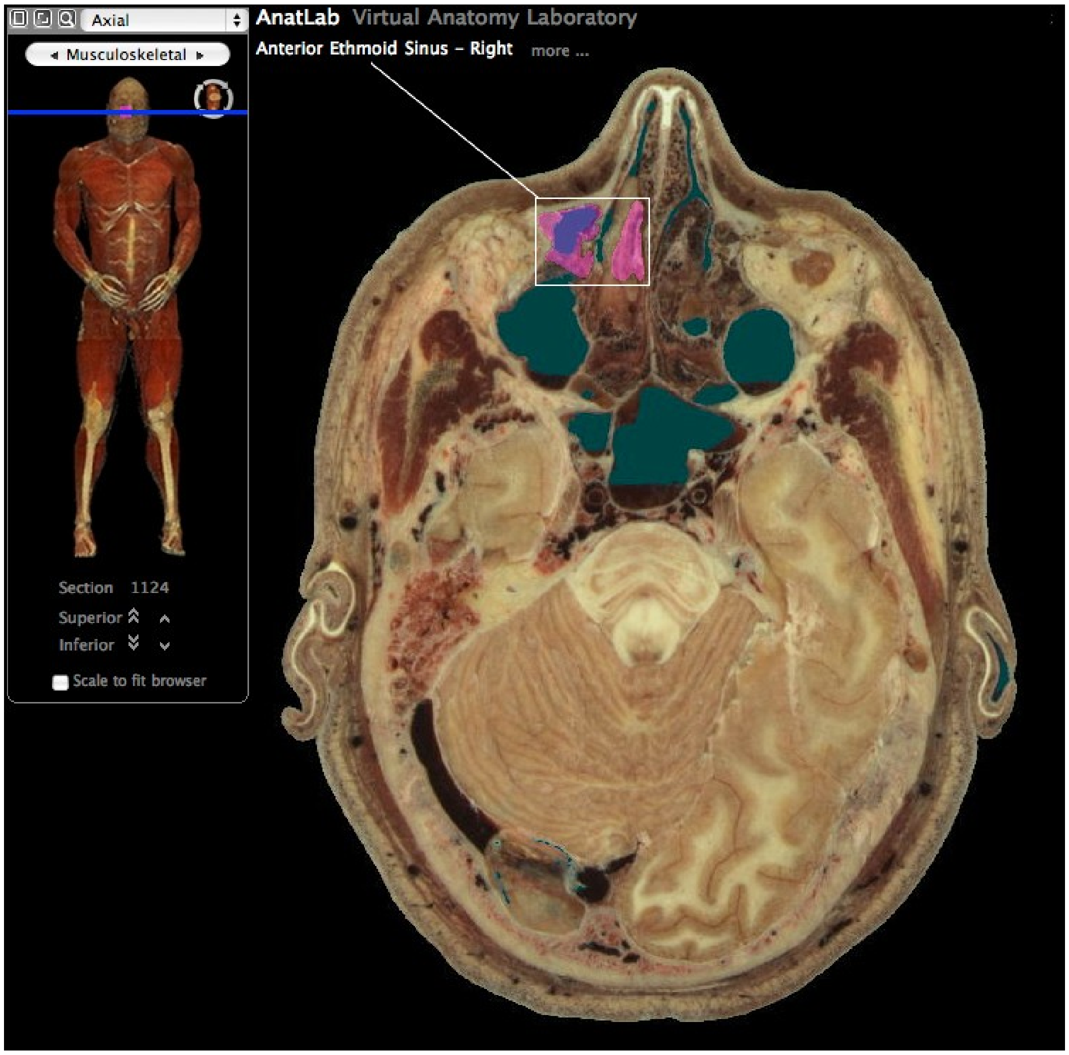
This zMapping functionality is handled by a separate process in the AnatLab Server, partly for historical and partly for performance and scaleability reasons. This process is called the Anatomical Name Server. The Anatlab Web Server communicates with the Name Server on a socket via a standard Tcl three-line event-driven server procedure.

When a user selects a feature of interest via the Web browser interface, a client-side image-map handler passes the coordinates to the AnatLab Web server via a RESTful URL [19]. The Web server passes the request to the name server, which reads the zMap database to access the associated structure's unique 24-bit index. This index is used to access the structure's name from a memory-resident array, and the information is returned to the main AnatLab Web server.

A previous incarnation of the AnatLab product used custom Java code to open, decompress and retrieve the needed information from the zMap TIFF files. A separate JVM was started for each retrieval. In order to improve speed and scalability, the Java code was scrapped and replaced by pure Tcl code. Some TIFF-reading code was adapted from the Tclers' Wiki [20]. The Tcl code is smaller, faster, more scalable, easier to configure and run, and it has proven easy to refine and add features to it.

6 Generation of Overlay Images

The initial AnatLab user interface highlighted selected structures by re-displaying a new version of the section image that had the specific structure highlighted. This had two downsides: the storage space required for each image, and the consequent update time when a user clicks on a structure. To address this, a new version of the AnatLab Web client was implement that overlays the section image with a translucent, colored image that matches the shape of the structure – as shown in the following screenshot.



There is one overlay image corresponding to each annotation, a total of over 700,000 overlay images across the three planes of view (axial, sagittal and coronal). The size of each overlay is about an order-of-magnitude smaller than a full section image, so one can see the savings are significant.

The color overlays are generated following post-processing of the annotations – one per anatomical structure per section.

Generating overlays is an arduous and resource-intensive process, similar to deriving the coronal and sagittal images from the axial sections. But unlike the section image generation, the overlay generation must be done repeatedly, on an ongoing basis as annotations are made, so there is an incentive to optimize the process. At the outset, a single overlay generation run took weeks, but refinement of the code, combined with migration of the process to the Amazon Web Services [21] Cloud Computing environment (and utilization of the largest virtual servers AWS has to offer), has reduced the time frame to days.

Overlay generation is accomplished by a combination of Netpbm utilities and makefiles controlled by Tcl scripts, similar to the process of generating the coronal and sagittal section images. For each colormap file in the zMap database, a list of unique feature indexes (represented in the file as 24-bit colors) is extracted. A color mask for each feature is generated, then the color mask is overlaid onto the corresponding section image file, resulting in a new file containing only the image area of the feature. The overlay image is tinted to provide a highlight effect when displayed over the section image in a Web browser.

Once completed, overlay files are uploaded to Amazon's S3 [22] static file server, which provides fast, scalable access to the users worldwide who use AnatLab.

7 Introspection of Section and Overlay Images

The AnatLab user interface needs to be able to obtain the size and position of a structure on a section. This information is used in a number of ways:

- ensuring the entire section is visible when changing sections
- positioning the section so a structure is visible when highlighted or on return from the search feature
- drawing a box around the highlighted structure (using a transparent HTML canvas, as shown in the above screenshot)

The overlay offsets (i.e. the position and size of each structure on each section) are calculated by post-processing the overlay images. This is done using a Tcl program that calls the ImageMagick [23] image processing system and extracts the size and position values from the output:

```
convert $image -trim info:-
image PNG 144x48 1760x1024+804+205 DirectClass 16-bit
```

The section offsets are calculated in a similar fashion, deriving the position and extent of the non-black parts of the section images, again using ImageMagick:

```
    convert $image -transparent black png:- | convert - -trim info:-
```

The 700,000 plus overlay and section offset records are needed to be stored and retrieved efficiently when the Web server responds to structure identity requests from the AnatLab Web client. An issue was how to do this without the need to create and maintain indices, especially during the development stage. The use of arrays was considered (since these provide hashed access to data) but the time taken to load the array on Web server startup is too long. A persistent solution was needed. In the end, the Metakit database system's [24] hashed views were use to provide fast access without the need for indices. All key values (view, section, structure) were coerced to integers, the optimal approach with Metakit's use of memory mapping and column-wise storage. The following timing tests shows performance via the Mk4tcl[25] interface, and the more object-oriented Mk4too[26] interface – with and without hashing.

```
    Mk4tcl
        first = 111.8745 microseconds per iteration = row 0
        mid = 2032466.1815 microseconds per iteration = row 366066
        end = 3589616.7145 microseconds per iteration = row 732012

    Mk4too
        first = 45.2705 microseconds per iteration = row 0
        mid = 229994.4485 microseconds per iteration = row 366066
        end = 393985.3045 microseconds per iteration = row 732012

    Mk4too with hash
        hash = 118 microseconds per iteration
        first = 58.0275 microseconds per iteration = row 0
        mid = 42.0225 microseconds per iteration = row 366066
        end = 50.6565 microseconds per iteration = row 732012
```

It can be seen that hashing is an extremely efficient way of accessing integer-keyed values. The "hash" time is the one-off cost of creating the hash when the AnatLab server starts.

Creating a hashed view is straightforward.

```
      # create section hash
      mk::view layout offset.sect { view:I sect:I x:I y:I w:I h:I }
      mk::view layout offset.sect_hash { _H:I _R:I }
      set raw [mk::view open offset.sect]
      set hash [mk::view open offset.sect_hash]
      set sect_offset [$raw view hash $hash 2]
```

The first line specifies a view (table) that will hold the data, the second line a hash view to access it. The third line opens the raw (unhashed) view, the next one opens the hash and the last line creates the hashed view of the data. From there on, all operations are via the sect_offset handle and result in the performance shown in the output.

8 The AnatLab Web Server

The AnatLab Web server is based on the TclHttpd [27] Tcl-based Web and application server. It uses typical TclHttpd features, e.g. Direct Domains to map URLs to Tcl procedures within the server. Where the AnatLab Server differs from many other TclHttpd installations is its support for Ajax in the client, necessitating non-blocking operations in the server.

Threads might seem the obvious way of addressing this – each Web server request would be handled by a separate thread and, if it blocked, then other threads would continue to service requests. But threads introduce complexity and can make debugging more difficult. As Donal Fellows has said, “When I deal with threading, I work on the principle that the hardware and OS are my enemies and hate me, and so I code accordingly. It probably doesn't make for a maximally-efficient program, but it does reduce debugging pain." [28]

So an alternative approach is used in the AnatLab Server: callbacks and the mostly undocumented but nevertheless very useful HTTPD_SUSPEND operation. HTTPD_SUSPEND tells TclHttpd to suspend the current operation, allowing it to service other requests. This is done by a Direct_Url command returning the HTTPD_SUSPEND error code:

```
return -code error -errorcode HTTPD_SUSPEND
```

At a later point, the operation can be resumed using the Httpd_Resume command.

This has been used in a number of places within the AnatLab server where it needs to query information from other Websites using http::geturl. For example, when identifying a structure using the AnatLab Anatomical Name Server (which runs as a separate process), code similar to the following is run:

```
proc identify {v s x y} {
      ...
   set sock [Httpd_CurrentSocket]
   http::geturl $url -command [list AnatLab::finish $sock $v $s]
   return -code error -errorcode HTTPD_SUSPEND
}
```

And later, when the http::geturl request finishes and calls the AnatLab::finish command

```
proc finish {sock vn se token} {
    Httpd_Resume $sock
    http::cleanup $token
      ...
    Httpd_ReturnData $sock application/json $result
}
```

The use of callbacks and a separate Anatomical Nameserver process has allowed AnatLab to show excellent performance and scaleability – and there is still the potential to use threads if that should ever be necessary.

9 3D VR Rendering

One of the consequences of the zMap approach is that it is straightforward to extract a structure's image from a colormap using ImageMagick, by supplying the RGB for the structure.

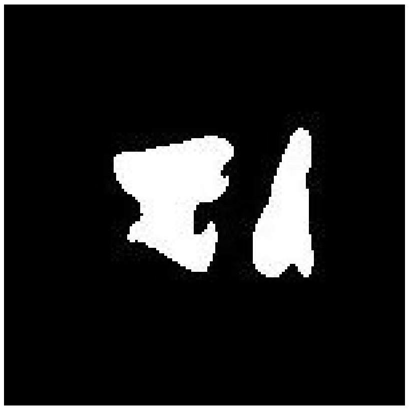
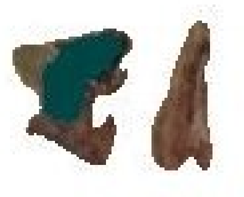
Further, a mask can be produced that, when applied to the section image, will show just that structure. The ImageMagick commands to create the mask are:

```
      convert $imagemap -fill '#ffffff' -opaque '#$RGB' \
                  -black-threshold 99% +matte $mask
```

and the commands to apply the mask to the section image, generating a transparent PNG are:

```
      composite -compose CopyOpacity $mask.jpg $section \
                -transparent-color black $result.png
```

For example, doing this for the Anterior Ethmoid Sinus - Right shown on the previous screenshots gives the following results: the left image is the mask and the right shows the mask applied to the section.

A Tcl program automates the process of doing this for all sections for a given structure or structures. It accepts a spreadsheet containing the name of one or more anatomical structures, and generates a series of masked images corresponding to the sections where these structures are found.

The resulting set of masked section images is then input to the Osirix image processing system [29] and a set of 3D Virtualization models rendered.

For example, the gastrointestinal system has been rendered in several ways, with and without the associated vascular system. The image on the left below shows the entire gastrointestinal tract, the image on the right shows the GI vascular system and the spleen.

These 3D renderings can be exported as QuickTime VR models [30] for viewing via a Web browser.

10 Conclusion

The Iomas AnatLab Virtual Anatomy Laboratory makes extensive use of Tcl/Tk and related technologies in its back-end infrastructure. Uses vary from the development of graphical interfaces, to image processing, to the automation of tasks.

In addition, the resources of the Tclers Wiki and the Tcl Community have proven invaluable.

As this project continues, the AnatLab development plans include using the Stargus Project's Cloud-Computing based Shared Storage facilities to create a shared annotator. It is our hope that the AnatLab project's extensive use of Tcl will help to demonstrate both the power of the Tcl platform and its versatility, as well as its suitability for the development and deployment of extremely high-performance Web applications that exploit cutting-edge technologies to deliver a highly satisfying user experience.

References

[1] AnatLab at Tcl2008 – http://wiki.tcl.tk/21794
[2] The National Library of Medicine's Visible Human Project - http://www.nlm.nih.gov/research/visible/visible_human.html
[3] http://en.wikipedia.org/wiki/Transverse_plane
[4] Coronal plane - http://en.wikipedia.org/wiki/Coronal_plane
[5] Sagittal plane - http://en.wikipedia.org/wiki/Sagittal_plane
[6] Ajax - Asynchronous JavaScript and XML – http://en.wikipedia.org/wiki/Ajax_(programming)
[7] The HTML Canvas – http://en.wikipedia.org/wiki/Canvas_(HTML_element)
[8] Canvas item selections - http://wiki.tcl.tk/8330
[9] Simple zooming and scaling in a canvas – http://wiki.tcl.tk/10381
[10] Image scaling – http://wiki/tcl/tk/8448
[11] Drawing and editing polygons – http://wiki.tcl.tk/3179
[12] Canvas Zooming – http://wiki.tcl.tk/4844
[13] Colors with Names – http://wiki.tcl.tk/16166
[14] Maze Solver – http://wiki.tcl.tk/20634
[15] Canvas: Polygons from lines - http://wiki.tcl.tk/1415
[16] Getting the Canvas View Area in Pixels – http://wiki.tcl.tk/2451
[17] SQLite – http://www.sqlite.org/
[18] Netpbm – http://netpbm.sourceforge.net
[19] REST – Representative State Transfer http://en.wikipedia.org/wiki/Representational_State_Transfer
[20] Tiff package (in pure tcl) – http://wiki.tcl.tk/9245
[21] Amazon Web Services - http://aws.amazon.com/
[22] Amazon S3 - http://aws.amazon.com/s3
[23] ImageMagick – http://www.imagemagick.org/script/index.php
[24] Metakit – http://www.equi4.com/metakit/
[25] Mk4tcl - http://www.equi4.com/metakit/tcl.html
[26] Mk4too – http://wiki.tcl.tk/8866
[27] TclHttpd – http://wiki.tcl.tk/2085
[28] Donal Fellows, quoted in Threads – http://wiki.tcl.tk/3447
[29] Osirix – http://www.osirix-viewer.com/
[30] QuickTime VR - http://www.apple.com/quicktime/technologies/qtvr/