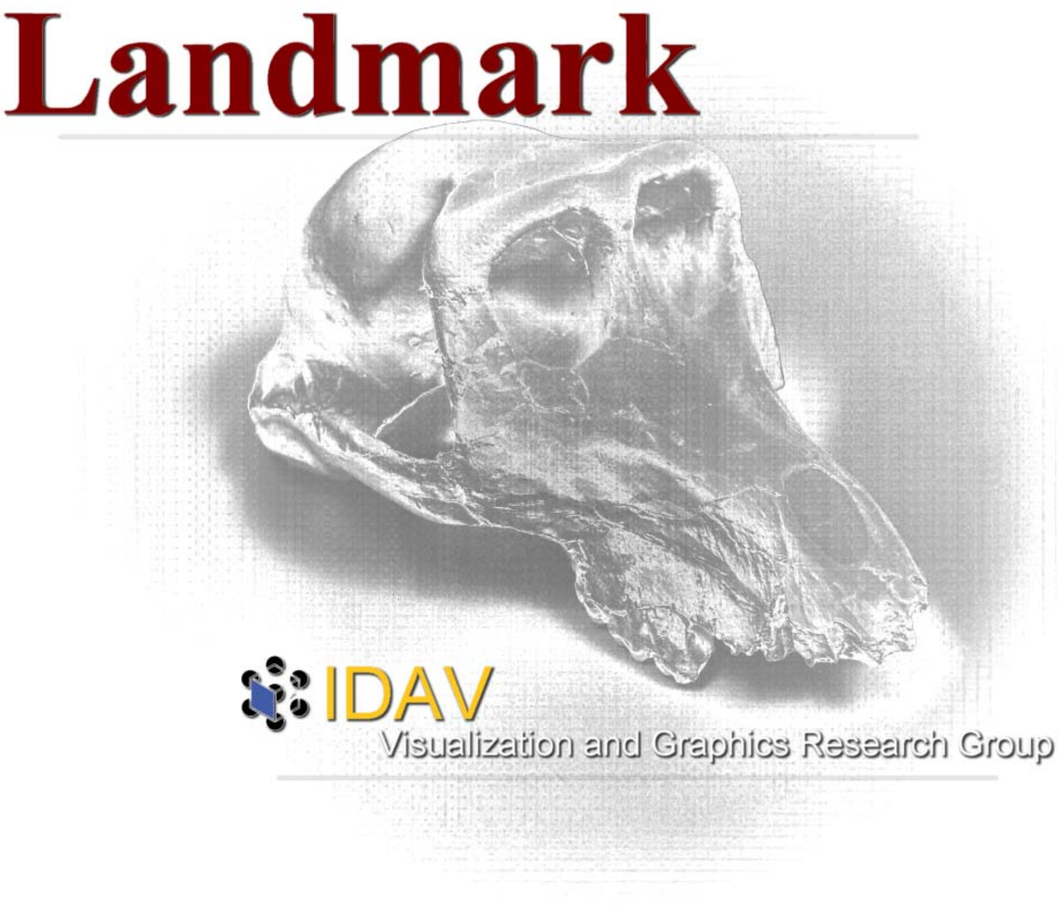
User Guide 3.6



November 2007

4. You may copy and distribute the Program (or a work based on it, under Section 3) in object code or executable form under the terms of Sections 2 and 3 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 2 and 3 above on a medium customarily used for software interchange; or,

   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 2 and 3 above on a medium customarily used for software interchange; or,

   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application

Landmark 3.6

Contents

Contact information: Dr. David F. Wiley, wiley@cs.ucdavis.edu, produced by the Institute of Data Analysis and Visualization (IDAV) group at the University of California, Davis.

1 Overview

The goal of Landmark is to put landmark points on surfaces and to match landmark points on different surfaces to each other. There are four types of landmark primitives available

1. **Single point.** This landmark consists of a single point—the conventional type of landmark placed on surfaces.
2. **Curve.** Place three points on a surface to form a curve. Several *semi-landmark* points are automatically generated along that curve to quickly and accurately lay down points.
3. **Patch.** Place a patch of points. Defined by a 3×3 grid of control points you can manipulate to change the shape and coverage of the patch. Several semi-landmarks are generated automatically across the patch. This primitive is useful for quickly laying down several points across featureless regions.
4. **Flexible Patch.** Similar to a *Patch* in that it has nine primary controls points, however, a flexible patch can additionally be manipulated by selecting any one of the intermediate points generated across the patch allowing greater flexibility.
5. **Dimension.** Specify two points to find the linear distance between those points. This primitive is also used to specify retrodeformation pairs across the sagittal plane.

Assuming you have many surfaces you want to place landmarks on, follow these steps to most quickly and easily place landmarks on all surfaces:

1. Choose one surface to be the *atlas*, to which all others will be corresponded (homologous).
2. Carefully place all landmark primitives on this surface and annotate where desired.
3. Iteratively load other surfaces and semi-automatically apply all landmark information from the atlas surface onto this new surface.
4. Adjust semi-automatic application of landmarks to fix any errors.
5. Repeat process for additional surfaces.

When finished, all of your surfaces are mapped to the atlas surface. Since the knowledge of this atlas mapping is stored, the implied mapping between all of the surfaces is readily available and is used within Landmark. Thus, if you have four surfaces S_1, S_2, S_3, and S_{Atlas} and you designate S_{Atlas} as the atlas by mapping S_1, S_2, and S_3 to it, then you get the mapping between S_1 and S_2 for free since Landmark can extrapolate the relationship between S_1 and S_2 from their relationships with S_{Atlas}, see Figure 1.

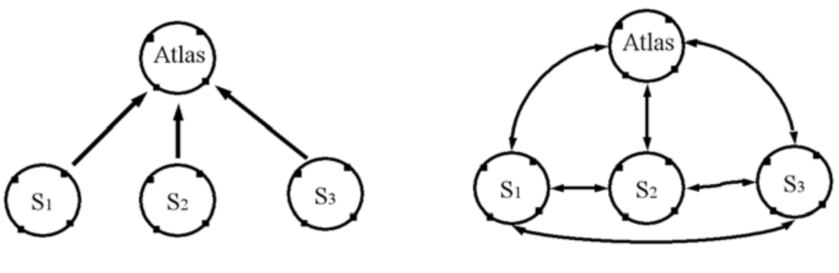


Figure 1: Corresponding surfaces to an atlas (left) implies many relationships defined by the surfaces' common relationship to the atlas (right).

The reason for the atlas surface is to coordinate the landmark primitives and to significantly reduce human error. Consider the problem of placing landmarks on several hundred surfaces. Several hundred points are required to accurately represent the features, thus, several hundred points need to be placed manually. The naive method is to *not* have an atlas and to just assume the user will place all the points on all the surfaces in the same order. This is unlikely in such a large system and by defining the mapping to the atlas, this ordering is always preserved.

Note: There is no tangible designation for the selection of an atlas other than simply choosing to map all surfaces to one in particular.

2 System Requirements

The minimum recommended system is:

- Windows® XP / Windows 2000 Professional (service pack 2 or higher)
- Intel® Pentium® III or higher or AMD Athlon™ processor
- 512 Megabytes of RAM
- Hardware-accelerated OpenGL® graphics hardware. OpenGL® implementation version 1.5 and higher yield a significant improvement in rendering speed.
- 20 Megabytes of hard disk space. (More for your data.)

Some features require more RAM:

- Species average. Depends on the chosen resolution. You should have more than 1GB of main memory if you plan to compute species averages with a resolution higher than “medium.”
- Evolutionary tree processing. Similar issues to computing a species average.

Some features require higher quality video hardware:

- Rapid evolutionary tree previewing.
- High-performance rendering of large (over 1M triangles) models.

Some features benefit from higher-performance CPUs:

- Overall performance improvement.
- Multi-core processors. Landmark takes advantage of multi-core processors when available by using multi-threading techniques. Thus, an investment in this area will be utilized by Landmark.
- Species average. RAM plays a more important role than does CPU speed in this case. Make sure the main memory requirements are met prior to considering a CPU upgrade.
- Evolutionary tree processing.
- Automatic landmark transferring.

High-performance storage (hard disks) can improve performance as well. If you are working with large numbers of specimen (over 50) or large model meshes (over 1 million triangles), then performance could be improved by upgrading your hard disk that your data (and .land) files reside on. You will be able to diagnose this if you notice significant slowdowns during importing-into and loading-from .land files.

Laptop hard disks typically have a rotational speed of 4200 RPM while standard desktop drives are 7200 RPM (which could account for why your desktop runs Landmark better when importing and loading). There are high-performance 2.5in 7200 RPM drives available for laptop computers that improve overall system performance. Additionally, one can purchase an external 3.5in 7200 RPM (EIDE or SATA, a standard desktop hard disk) USB 2.0 or firewire drive and connect it to the laptop to gain in performance.

Desktop computers can also use higher-performance 10000 RPM EIDE or SATA drives to gain in performance. The best choice is a high-performance 10000 RPM SCSI hard disk. These, however, tend to cost two to three times more than a comparable EIDE or SATA drive, but have two to three times the performance improvement.

Finally, one can consider building a RAID 0 system using high-performance EIDE, SATA, or SCSI drives that would respectively improve performance.

3 Getting Started

3.1 Creating and using a Landmark database ".land" file

In order to organize and keep track of surfaces, their landmark points and correspondences, and other information and files, Landmark groups everything into a single database that you can easily access within the program.

Select `File | New` from the menu to create a new Landmark database file. Once created, a window having three panes labelled A, B, and C and a fourth pane on the left side is shown, see Figure 2.

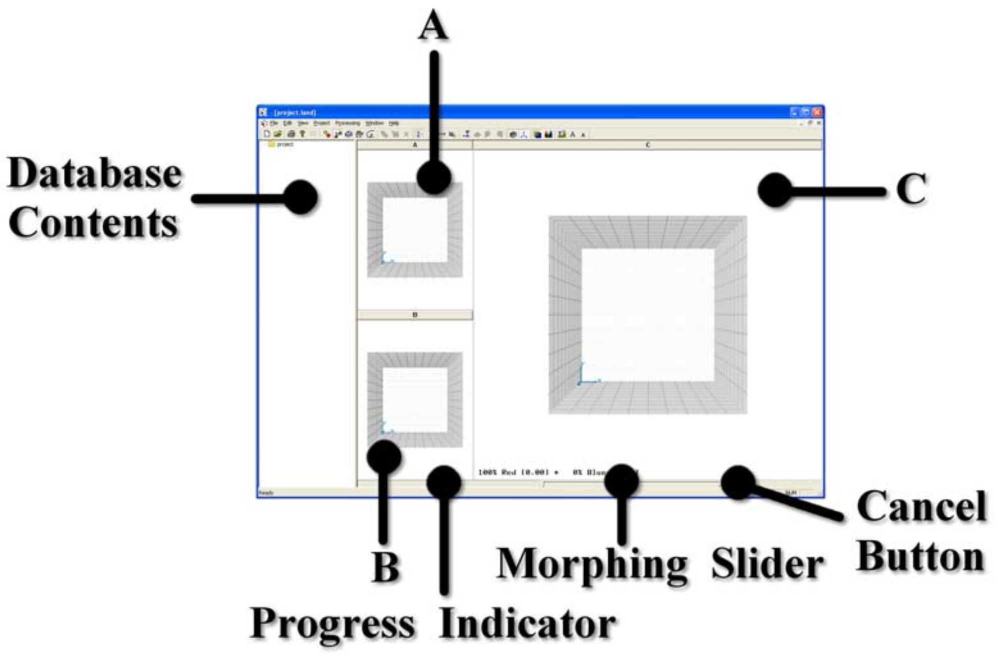


Figure 2: The main screen components of the Landmark editor.

Screen components:

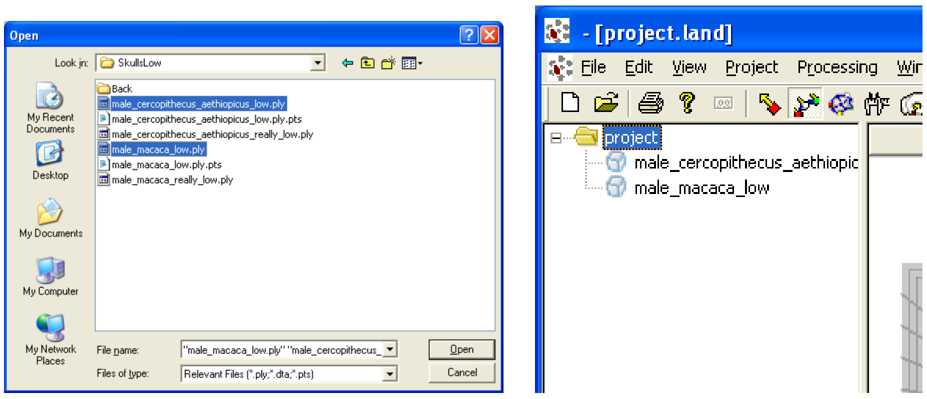
- **Database Contents:** shows the surface models, points, correspondences, and other files contained within the Landmark database
- **A and B:** load surfaces (or points) into the A and B panes for the purpose of placing and manipulating landmark primitives (points, curves, and patches). Typically, the atlas is loaded into A and surfaces being mapped to the atlas are loaded into B.
- **C:** this pane is used as the "result" window. The C pane displays the morph when morphing between surfaces loaded in the A and B panes and it shows the retrodeformed mesh when retrodeforming a surface.
- **Morphing Slider:** If two meshes are loaded into the A and B panes and they are corresponded to one another, selecting a position on this slider computes the morph between those two surfaces.
- **Progress Indicator:** whenever Landmark performs a task that requires an extended period of time, the progress of the task is displayed here.
- **Cancel Button:** cancel task processing by pressing this button.

Above each pane is a button labelled with the pane identifier (A, B, or C) and the name of the PLY loaded into it. Press this button to move the pane to the larger window on the right.

3.1.1 Importing files

Import files into the Landmark database by selecting `Project | Import...`, see Figure 3. Any file can be imported into the database. All files are compressed to save space. Special treatment is given to files of these types:

- **.ply** (and **.stl**) define a surface model that can be loaded into the A and B panes allowing placement of landmark primitives.
- **.dta** defines a previously collected set of landmark points that can be loaded on their own or can accompany a `.ply` (or `.stl`) surface model.
- **.pts** defines a single set of points that can be loaded on their own or can accompany a `.ply` (or `.stl`) surface model.


Open
Look in: SkullsLow
Back
male_cercopithecus_aethiopicus_low.ply
male_cercopithecus_aethiopicus_low.ply.pts
male_cercopithecus_aethiopicus_really_low.ply
male_macaca_low.ply
male_macaca_low.ply.pts
male_macaca_really_low.ply
My Recent Documents
Desktop
My Documents
My Computer
My Network Places
File name: "male_macaca_low.ply" "male_cercopithecus_
Files of type: Relevant Files (*.ply;*.dta;*.pts)
Open
Cancel
- [project.land]
File Edit View Project Processing Wir
project
male_cercopithecus_aethiopic
male_macaca_low

I II

Figure 3: Image I shows two ply models being imported into the database. If the ply model already has a correspondence file (from an older version of Landmark) than it is automatically imported, shown in Image II, if not, a new one is created.

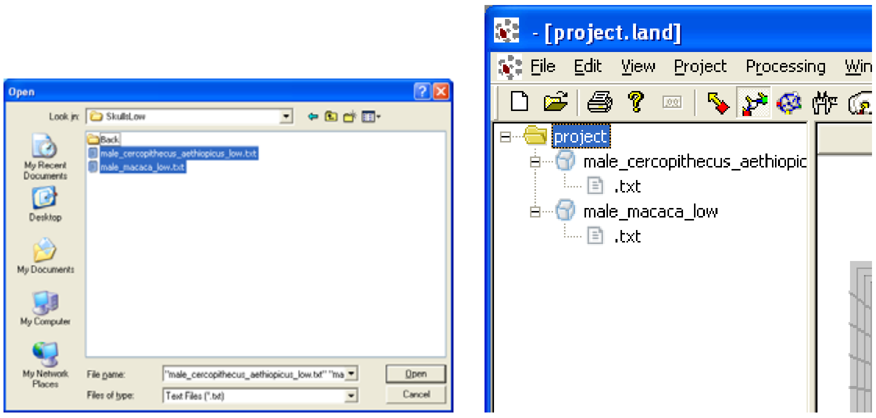
When importing, Landmark attempts to match imported files to objects already contained in the database based upon their file names. For example, suppose you have files named `cat.ply, dog.ply, and fish.ply` and have imported them into the database. In the database, their names will be cat, dog, and fish, respectively. Landmark will automatically match files to these if they are named `cat.nnn, dog.nnn, and fish.nnn` where `nnn` can be any file extension.

As another examples, suppose you have already imported several surface models named `ap1.ply, ap2.ply, ..., ap99.ply` into the database and also have an NTSys `.dta` file containing several groups of landmark points you've previously collected. If you import the `.dta` file and the groups of landmark points are named `ap1, ap2, ..., ap99`, within the `.dta` file, then Landmark will be able to figure out the relationship to the `.ply` models and will either replace or merge these points with the landmark primitives associated with the previously loaded `.ply` files of the same name.

It may be advantages to import intermediate files (having extensions other than .ply, .dta, .corr, or .pts) that you generate in your research process into the Landmark database. They are then associated with each model and categorized appropriately, see Figure 4. This provides a mechanism for data management. You can view imported files by selecting `Project | Open Item...`, see Figure 5. If you need to get the files out of the database then you can simply export them. Additionally, a plugin can be written to open these files, see Section 11 for how to do this.

3.1.2 Exporting files

Data and files are exported from the Landmark database by selecting `Project | Export...`. The items that are currently selected in the database view and their child items is what will be exported.



I II

Figure 4: Image I shows two `.txt` files being imported into the database. The names of the `.txt` files are matched with existing models in the database and added as a child item of that model, shown in Image II.

The dialog that appears gives you several options for exporting the available data, see Figure 6. Each are described below:

- **(*.land) Landmark database:** Export selected items as a subset into a new Landmark database
- **(*.ply) PLY models:** Export selected items as PLY models
- **(*.stl) STL models:** Export selected items as STL models. Note that color information is not captured well in STL files (6 bits per RGB channel, as opposed to 8), thus, expect a degradation in color quality if you use STL files.
- **(*.corr) Correspondences:** Export landmark data for models into .corr files (accompanies .ply files)
- **(*.dta) NTSys landmark points:** Exports landmark points into NTSys format
- **(*.pts) Raw landmark points:** Exports points into a text file
- **(*.???) ??? File:** Other file extensions appear at the end of this list if they are present in the database and can be exported by selecting these items

The `.land` and `.dta` options require a filename that is specified to the right of the list. Any item that exports points requires an atlas to be specified in order to enforce landmark point sequence numbering.

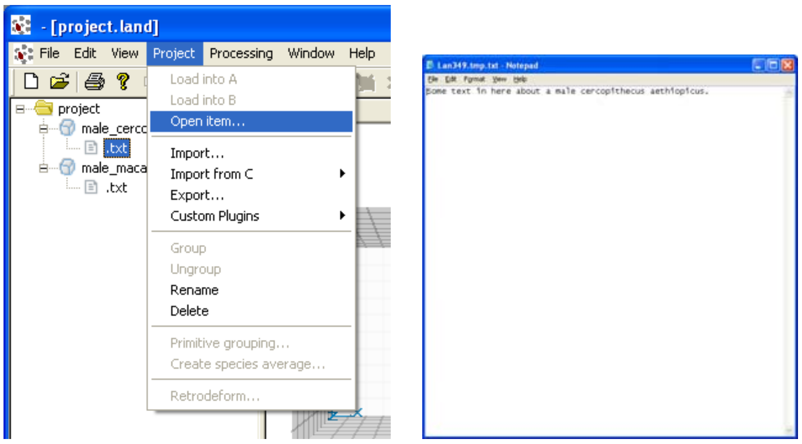
3.2 Loading surface files

To load surface files from the database, select the model you want to load and then select `Project | Load into A...` or `Project | Load into B...` to load that model into the A and B panes, respectively, see Figure 7.

3.3 Moving surfaces around

Once a surface is loaded you will need to move it around to be able to place landmarks on it. Use the following to move the surfaces around

- **Left mouse button.** Rotates the surface.
- **Middle mouse button.** Rotates on the plane of the screen.

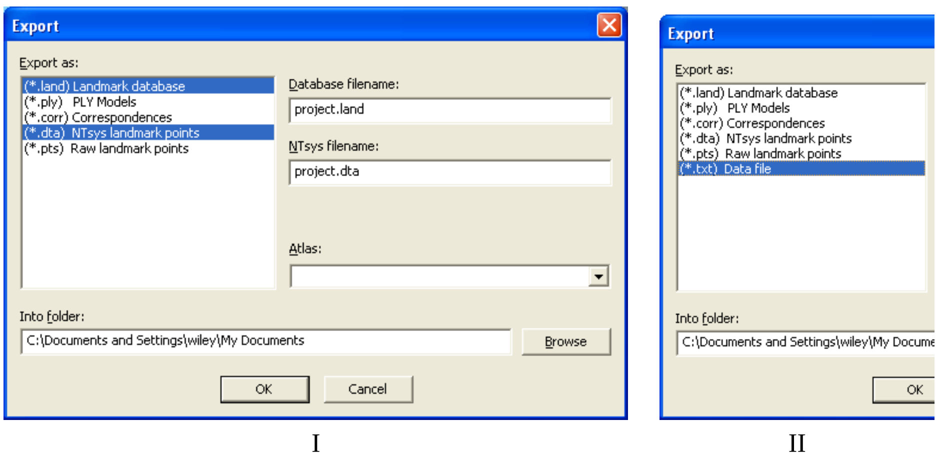


```
Some text in here about a male cercopithecus aethiopicus.
```

I　　II

Figure 5: Image I shows a `.txt` file being viewed directly from the database. Notepad is associated with the `.txt` extension on this particular machine, thus Notepad is used to view the file, shown in Image II. *Changes to files opened in this manner are not saved in the database.*

- **Right mouse button.** Pans across the plane of the screen.
- **Mousewheel.** Zoom in and out. If a mousewheel is not present, use `CTRL-left-mouse-button` and move the mouse up-down or left-right.



Database filename: project.land

NTsys filename: project.dta

Into folder: C:\Documents and Settings\wiley\My Documents

Into folder: C:\Documents and Settings\wiley\My Docume

Figure 6: Image I shows the file exporting options. If database items have files having different extensions than those shown above, they will be added to the end of the list so that you can export them, shown in Image II.

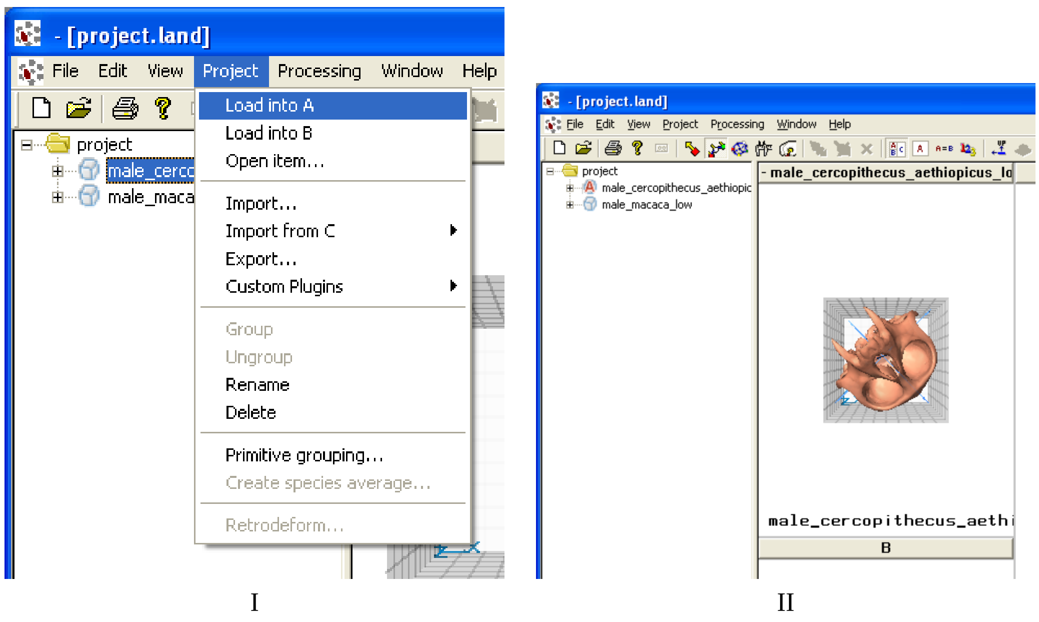


Figure 7: Image I shows which ply model is being loaded. Image II shows the loaded model in pane A. The model loaded in pane A is indicated by a red A in the database view.

4 Using the Landmark Editor

The following sections describe the various tools that are available for manipulating landmarks while in the three-pane landmark editor.

4.1 Axes and bounding box

There are many rendering options that you can access through the `View` menu to aid in placing landmarks. Transparent surface rendering often helps when placing landmark primitives but may render too slowly on low-end graphics hardware. In this case, you would want to turn this option off. The axes and bounding box, for example, can be turned off by `View | Data bounds | Show`, see Figure 8.

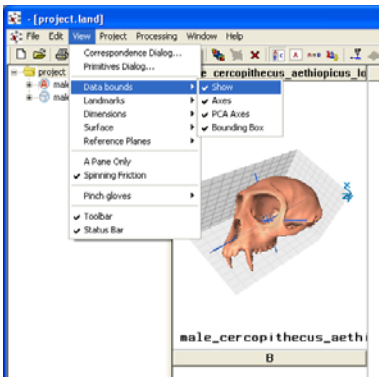


Figure 8: Hide the axes and bounding box.

4.2 Placing landmark primitives

Select the primitive type from the `Edit | Add...` menu to choose Single point, curve, patch, flexible patch, or dimension primitives.

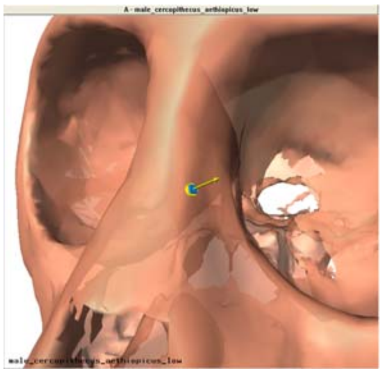
- Single points are placed by holding down the shift key while clicking the left mouse button on the surface at the location you want to place the landmark. Figure 9 shows an example of placing a single landmark point.

- Curves are placed by holding down the shift key while clicking the left mouse button at three different locations on the surface to designate the three *control points* of the curve. Figure 10 shows an example of placing a curve.

- Patches are placed by holding down the shift key while clicking the left mouse button (dragging moves the patch around before fixing its position). The program tries to find a reasonable patch to start with by automatically placing the nine control points in the region near where you clicked. You should then move the automatically placed points around. Figure 11 shows an example of placing a patch.

- Flexible patches provide more control over the shape of a patch. Flexible patches are placed by holding down the shift key while clicking the left mouse button (dragging moves the patch around before fixing its position). The program tries to find a reasonable patch to start with by automatically placing the control points in the region near where you clicked. You should then move the automatically placed points around. Figure 12 shows an example of placing a flexible patch.

- Dimensions are specified by holding down the shift key while clicking the left mouse button at two locations. This primitive is used to measure linear distance between the two points and also to specify point pairs during retrodeformation. Additionally, dimension primitives can be used to calibrate the model by opening `View | Edit primitives...` and entering the actual measured distance for the selected dimension primitive. This value is then used to properly scale the model.


A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
Figure 9: Placing a single landmark point.

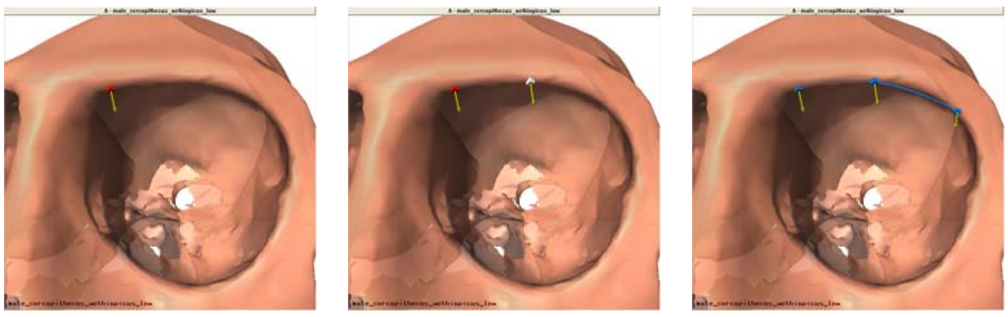

I II III

Figure 10: Placing a curve. Images I through III show the addition of the three points that construct a curve primitive.

4.3 Manipulating landmarks

Choose `Edit | Select points` and select points by holding down the shift key while clicking the left mouse button. Selected points are highlighted yellow when selected. There are a number of operations you can perform on selected points:

1. Drag them around while still pressing the left mouse button and holding down the shift key. (Points can only be placed on surfaces, thus points cannot be dragged off of a surface.)
2. Pressing the space bar removes selection from all points.
3. Pressing the delete or backspace key when any points are selected deletes those points and any landmark primitives connected to them.

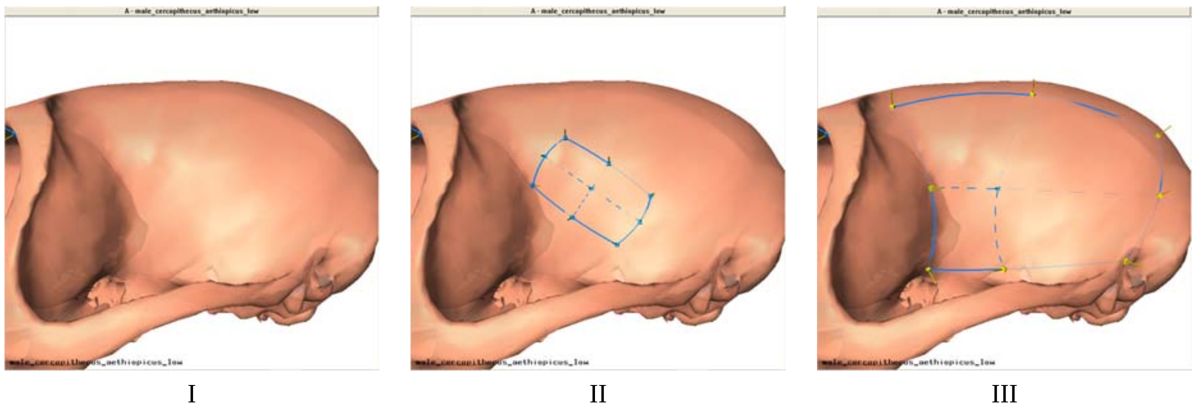


Figure 11: Placing a patch. Image I shows the surface before placing the patch. Image II shows the surface after the patch has been added by left-clicking while holding down shift. Image III shows the patch after moving the points around to their appropriate locations.

4. Join points together by selecting more than one point and choosing `Edit | Join points` from the menu. Points are joined to the first point selected.

5. Split points by choosing `Edit | Split points` from the menu. Two points will then be created on top of each selected point. Select and move one point aside to access them both.

6. Add annotations by selecting `View | Edit primitives...`, select the primitive in question, and type the annotation in the available edit box. Remove the annotation by setting the text to nothing. Figure 13 shows an example of adding an annotation.

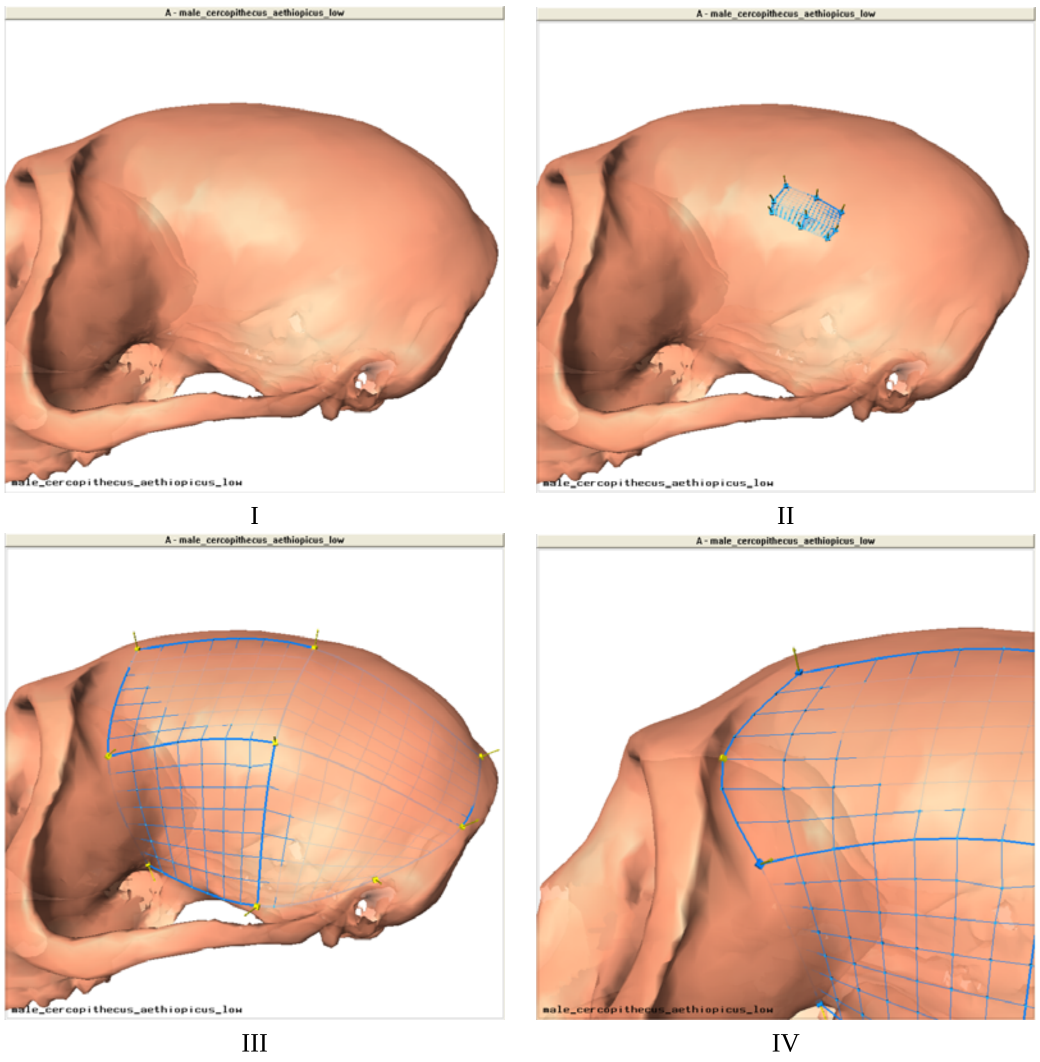
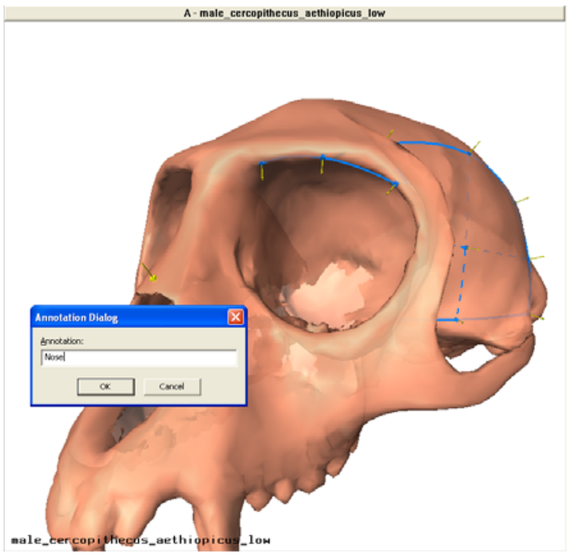
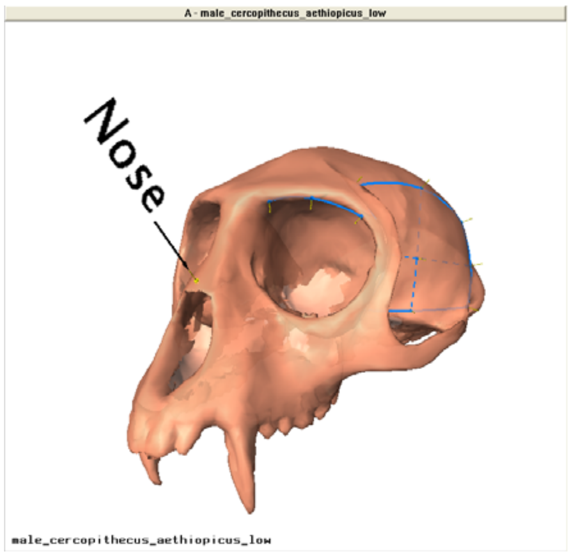

A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
I
A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
II
A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
III
A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
IV

Figure 12: Placing a flexible patch. Image I shows the surface before placing the patch. Image II shows the surface after the patch has been added by left-clicking while holding down shift. Image III shows the patch after moving the major control points around to their appropriate locations. Image IV shows the patch after moving a minor control point to fit one region of the mesh better. Left-clicking while holding down shift selects minor points. Dragging fixes their position. Double-clicking while holding down shift "un-fixes" its position and allows the point to roam freely.


A - male_cercopithecus_aethiopicus_low
Annotation Dialog
OK
Cancel
male_cercopithecus_aethiopicus_low

Annotation:
Nose

I


A - male_cercopithecus_aethiopicus_low
Nose
male_cercopithecus_aethiopicus_low

II

Figure 13: Example of adding an annotation to a single landmark point. Double clicking on the single landmark point on the nose opens the annotation edit dialog shown in Image I. Image II shows how the annotation is rendered. Remove an annotation by setting the annotation text to be empty.

4.4 Semi-automatic landmark application

Landmark semi-automatically applies all of the landmark primitive information from one surface to another. As an example, I've gone ahead and added some more landmarks, curves, and patches to the "red" surface in the previous examples. Specifically, I've added single landmark points on the tips of the fangs, a point at the back of the skull and moved a point to the brow. Figure 14 shows the red surface and its landmarks.

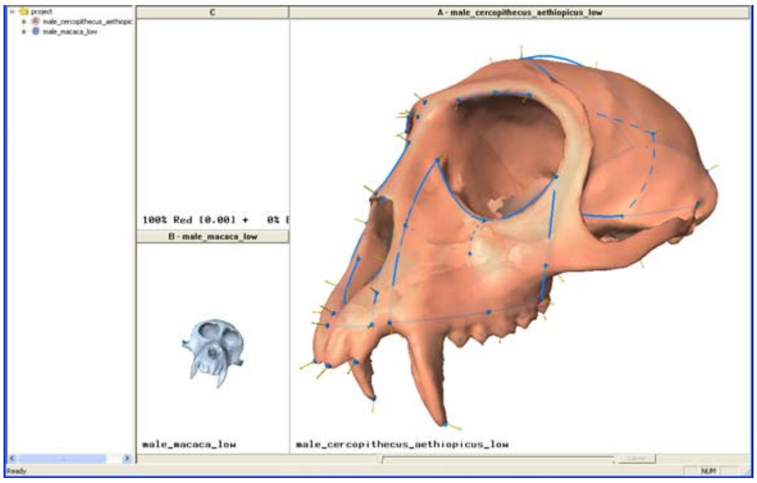

C
A - male_cercopithecus_aethiopicus_low
100% Red [0.00] + 0% I
B - male_macaca_low
male_macaca_low
male_cercopithecus_aethiopicus_low
Ready
NUM

Figure 14: Landmark primitives on the "red" surface.

To use the auto application feature, we must identify at least four landmark primitive that the two surfaces have in common. Thus, I will add the four single landmark points I described above to the "blue" surface. Figure 15 shows the blue surface after adding the four points.

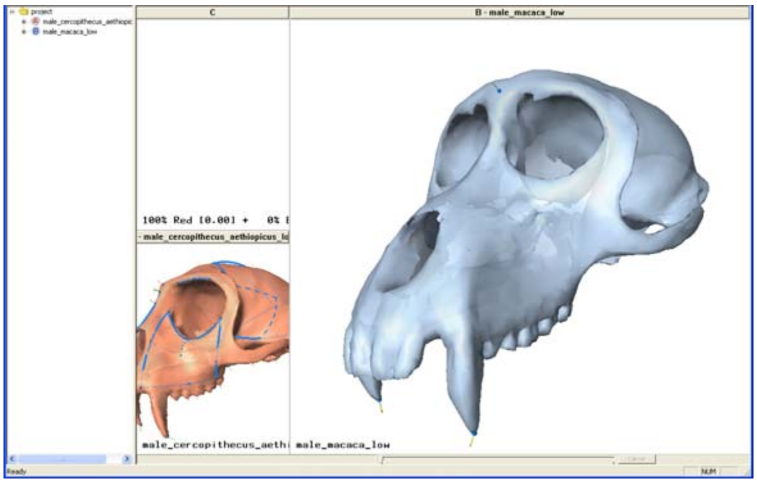


Figure 15: Landmark primitives on the "blue" surface.

Next, to establish a relationship between the points on the two surfaces, open the Correspondence dialog by choosing `View | Correspondence Dialog...` from the menu. Establish a correspondence between each pair of points by selecting each point primitive in the A and B columns, verify that the point selected is the one intended by moving the surfaces around to locate the highlighted points, and select the >> button to confirm the relationship. Figure 16 shows the correspondence dialog for this example.

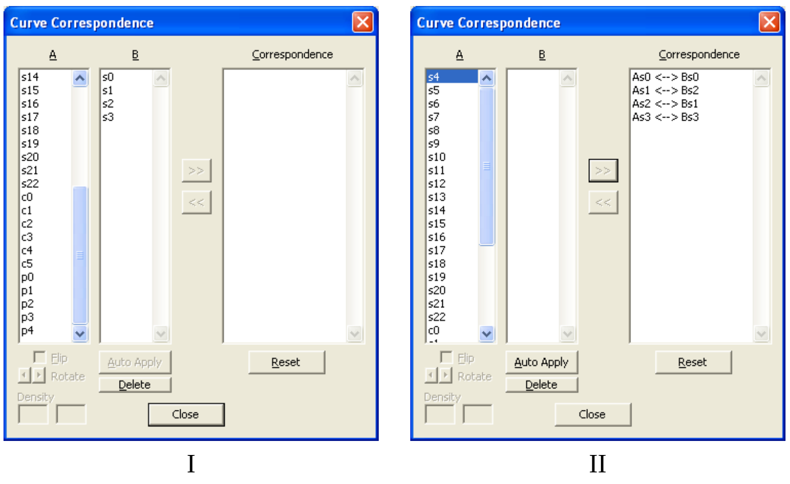


Figure 16: Correspondence dialog. In both images, single landmark points are denoted by an s, curves by a c, and patches by a p. The number next to the letter denotes the index for the primitive. In image II, note that the point indices are matched to different numbers in the right-most column, this is because I added the points in a different order for each model.

Since four primitives have been corresponded to each other, the `Auto Apply` button (underneath the B column) is enabled. Press this button to transfer the rest of the primitives from A to B. Figure 17 shows the what happens after pressing the `Auto Apply` button.

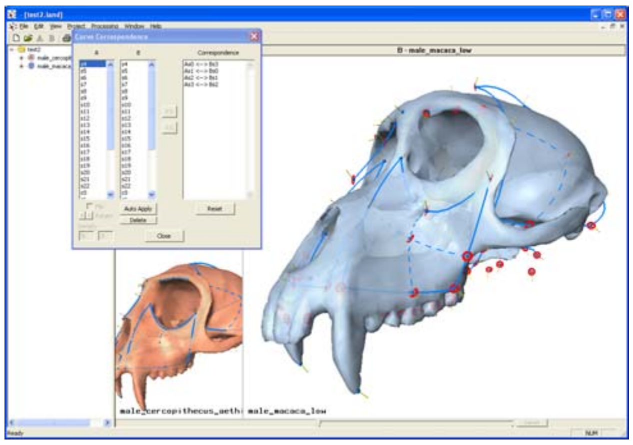


Figure 17: Result after pressing `Auto Apply` button. Note that new primitives have been added to column B in the correspondence dialog. Also note that these primitives have been added to the blue surface and their control points have been highlighted with rotating red disks.

This landmark transference is semi-automatic since you must specify at least four points to begin the transfer and also that, as you probably noticed, the transfer isn't perfect. Thus, you must manually move the points around to their appropriate locations. Rotating red disks indicate the landmark control points that need to be adjusted by the user. Once moved, the red disk disappears. Moving these points improves the overall placement of the control points making it easier to quickly adjust the automatically transferred points.

Adjustments to the landmark primitives on the source surface invalidates the correctness of the landmarks and their placement, thus one should re-transfer the landmarks after adding to or removing landmark primitives from the source surface.

4.5 3D morphing

Once the relationships between landmark primitives have been established, you can create morphs between the surfaces. Select the horizontal slider at the bottom of the landmark editor window (below the big pane) to create a morph of the two surfaces. Figure 18 shows an example of morphing the two surfaces.

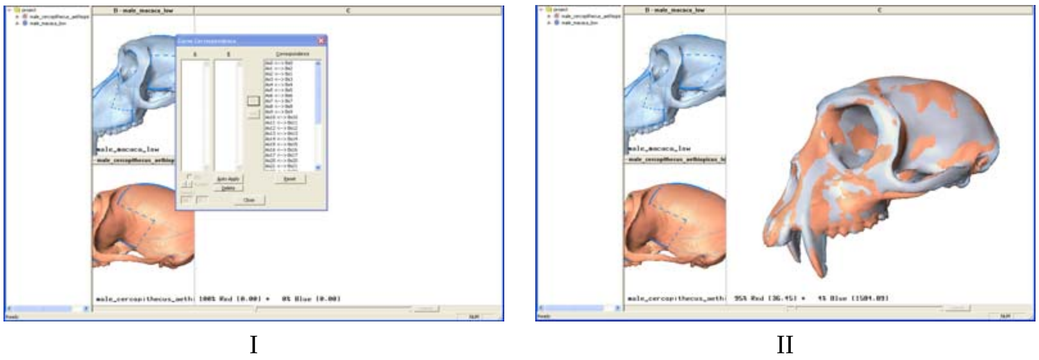


Figure 18: Example of a morph between the two surfaces. Image I shows that the correspondences have been established. Image II shows the two surfaces morphed into the same space as specified by the slider at the bottom of the window.

You can save the morphed surfaces back into the landmark database by selecting `Project | Import from C...`. You can save the red and blue surfaces individually or both of them together as one mesh. A new item representing the chosen surface is added to the selected group in the database view.

If there are several landmark primitives (more than a few hundred), the morphing computation can take a very long time (several minutes, depending upon the machine). To pre-compute the morph for all positions across the slider, select `Processing | Precompute TPS data...` from the menu. You can follow the progress of the operation by the progress bar at the bottom-left side of the window. Figure 19 shows the data item that's created for this example in the project database.

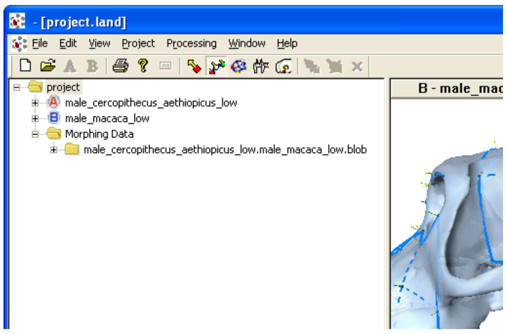

- [project.land]
File Edit View Project Processing Window Help
project
male_cercopithecus_aethiopicus_low
male_macaca_low
Morphing Data
male_cercopithecus_aethiopicus_low.male_macaca_low.blob
B - male_mac

Figure 19: A folder under "Morphing Data" called "male_cercopithecus_aethiopicus_low.male_macaca_low.blob" is created for the pre-computed morph data.

If this data is present when the slider is selected with these two input files loaded into their respective panes, the data will be loaded and accessed instead of recomputing the data. If you add new landmark primitives, you will want to delete this data and re-compute it. Several of these data items can be created and stored in the project database for different combinations of surfaces.

4.6 Grouping landmark primitives

Landmark primitives can be grouped together to allow processing based upon these groups. For example, exporting landmark points only in certain regions as opposed to landmark points across the whole surface. Grouping specification is only required on the atlas. Select the atlas in the database view then select `Project | Primitive grouping...` from the menu to open the grouping specification dialog, see Figure 20.

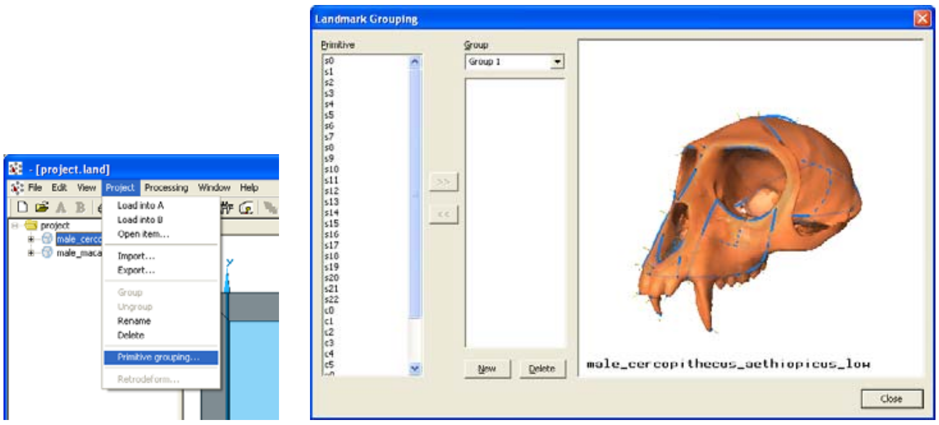


Figure 20: Grouping of landmark primitives. Image I shows how to open the grouping specification dialog for a selected atlas. Image II shows the ungrouped primitives in the left-most column and the surface on the right-hand side.

Group landmark primitives by selecting a subset of primitives in the left-most column then move them to the group selected beside it by pressing >>, see Figure 21.

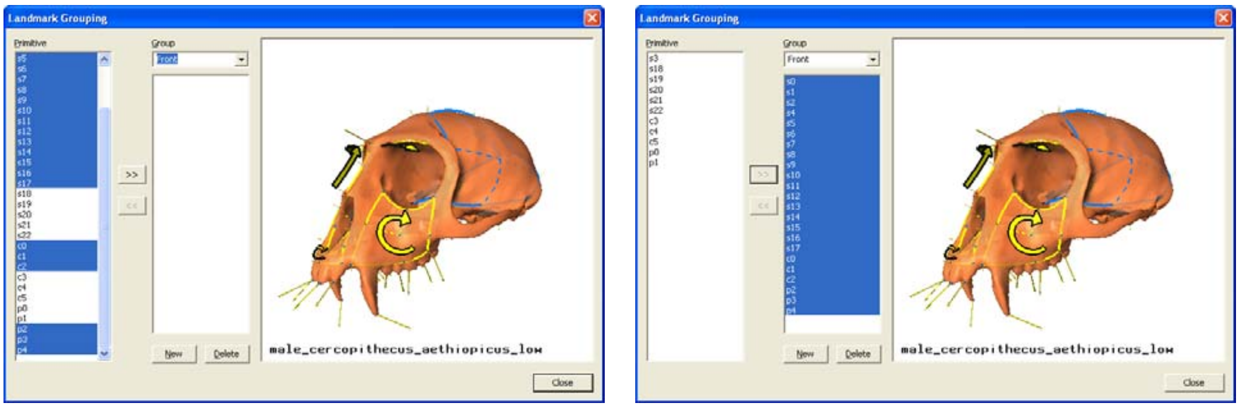
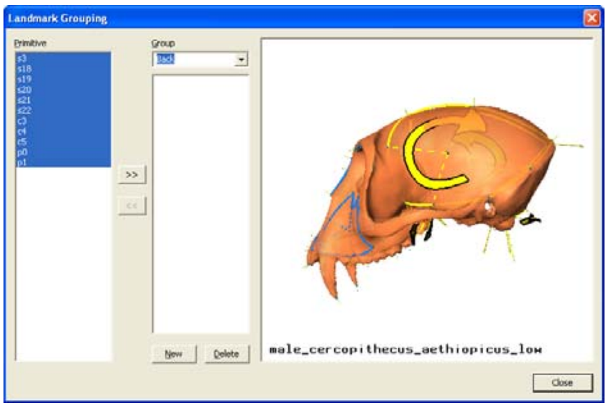
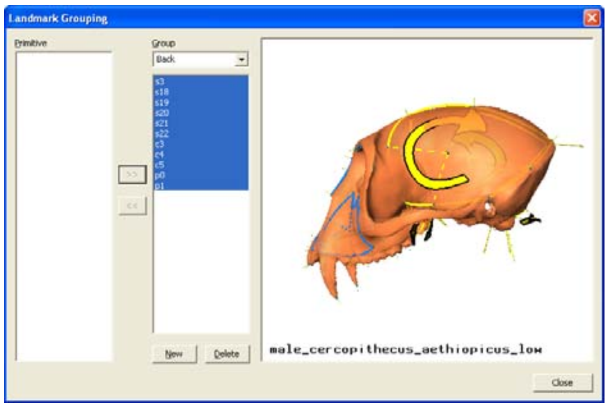


Figure 21: Image I shows primitives selected for grouping as indicated in the preview window to the right. Image II shows the grouped primitives after pressing >>. The group name can be changed in the edit box above the group list.

Repeat the grouping process to define subsets of the landmark primitives by creating new groups by pressing `New` and moving primitives into them, see Figure 22. Remove primitives from a group by selecting them in the group list and pressing << to move them back into the left-most column. Delete groups by pressing `Delete`; Primitives in a deleted group get moved back to the left-most column.



I

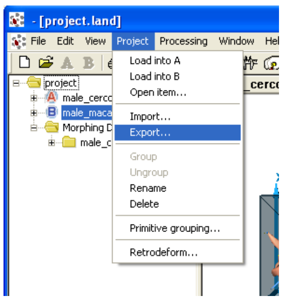


II

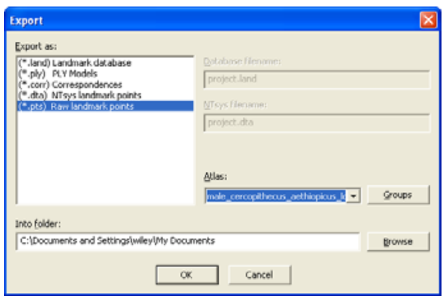
Figure 22: Image I shows primitives selected for grouping as indicated in the preview window to the right. Image II shows the grouped primitives after pressing >>.

4.7 Exporting landmark points

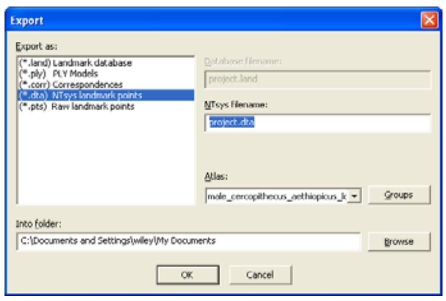
The automatically generated landmark points (single points and those across curves and patches) can be exported to a text file for use in other programs. This is done by first selecting the meshes you want to export the points from and then choosing `Project | Export...` from the menu, see Figure 23. Details of the `.pts` and `.dta` file formats are described in Appendices 9 and 10, respectively.



I


Export
Export as:
(*.land) Landmark database
(*.ply) PLY Models
(*.corr) Correspondences
(*.dta) NTsys landmark points
(*.pts) Raw landmark points
Database filename:
project.land
NTsys filename:
project.dta
Atlas:
male_cercopithecus_aethiopicus_l
Groups
Into folder:
C:\Documents and Settings\wiley\My Documents
Browse
OK
Cancel

II


Export
Export as:
(*.land) Landmark database
(*.ply) PLY Models
(*.corr) Correspondences
(*.dta) NTsys landmark points
(*.pts) Raw landmark points
Database filename:
project.land
NTsys filename:
project.dta
Atlas:
male_cercopithecus_aethiopicus_l
Groups
Into folder:
C:\Documents and Settings\wiley\My Documents
Browse
OK
Cancel

III

Figure 23: Exporting landmark points. Image I shows the item from which the landmark points will be exported from (more than one item can be selected). In the export dialog, an atlas must be selected so that the exported landmark points are generated in a sequence consistent with that atlas, see Images II and III. For the `.pts` format, files beginning with the name of the exported items and ending in `.pts` are created in the specified folder, see Image II. For the `.dta` format, a file name must be provided, see Image III.

Subsets of landmark points, as specified in Section 4.6, can be selected by choosing the `Groups` button on the export dialog. A group selection dialog is shown allowing you to choose which groups of landmarks to export, see Figure 24.

Advanced options allow further refinement of the exported data. `Allow duplicates` exports the same x,y, and z coordinate landmark and semi-landmark points in the case when primitives have been joined together. For example, single-point landmarks that have been joined to the curve primitive control points. In this case, landmark points would be generated for the three single points and the semi-landmarks generated for the curve would duplicate those generated for the single point primitive.

An odd density value for curves or patches produce semi-landmark points at the control point locations. Thus, if

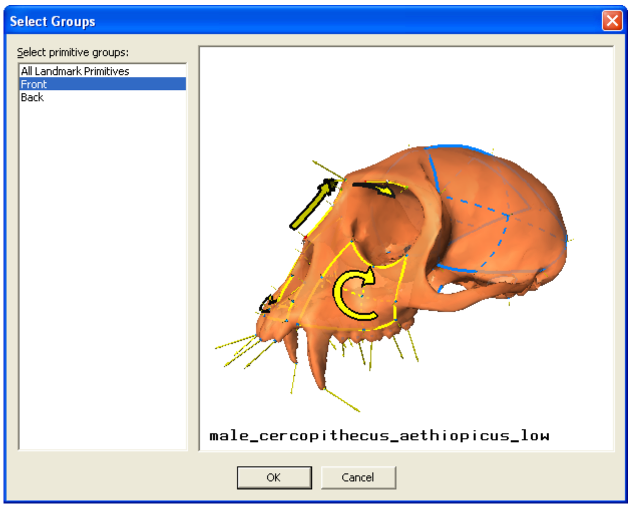


Figure 24: Subsets of landmark points that have been grouped can be selected in order to export certain regions of points. The column on the left shows the available groups. The window on the right shows which primitives are represented by the selected groups. *Note that the shown surface is that of the atlas.*

another primitive is joined at that control point, the x,y, and z coordinates will be duplicated for those points if the `Allow duplicates` option is selected.

Using calibrated coordinates scales the model and landmarks based on calibration measurements set for dimension primitives. To establish correct scaling and coordinates for your specimen, first place a dimension primitive on your specimen using two single landmarks. Then, select `View | Edit primitives...`, choose to view dimension primitives, and then enter in the actual measured linear distance (using calipers, for example) between the two landmark points specifying the dimension primitive. This value is then used to scale the model to match the value entered. If you specify multiple measurements, the average scaling factor computed for each dimension primitive is used.

Selecting `Single points`, `Curves`, `Patches`, and `Flexible patches` determine if those particular primitive types are exported.

4.8 Viewing exported landmark points

Select the PLY surface in the database view then select `Project | Open item...` to view the surface. Then, choose `File | Import | Landmark points...` to load the `.pts` landmark points file. This is an important step for verifying that your landmark points are ordered appropriately if you seem to be having problems. Figure 25 shows the exported landmark points exported in Section 4.7.

4.9 Editing Landmark Primitives

Landmark primitives can be edited using the Edit Landmark Primitives dialog. General operations such as re-ordering, deleting, annotating, flipping, and rotation are supported for appropriate primitives. Invoke this dialog by selecting `View | Edit Primitives...`, see Figure 26.

- Single point primitives can only be re-ordered, deleted, or annotated.
- Curves can be flipped in addition to re-ordering, deleting, and annotating.
- Patches can be flipped and rotated in addition to re-ordering, deleting, and annotating.
- Dimensions can be specified for a dimension primitive in order to calibrate model units to real-world measurements.

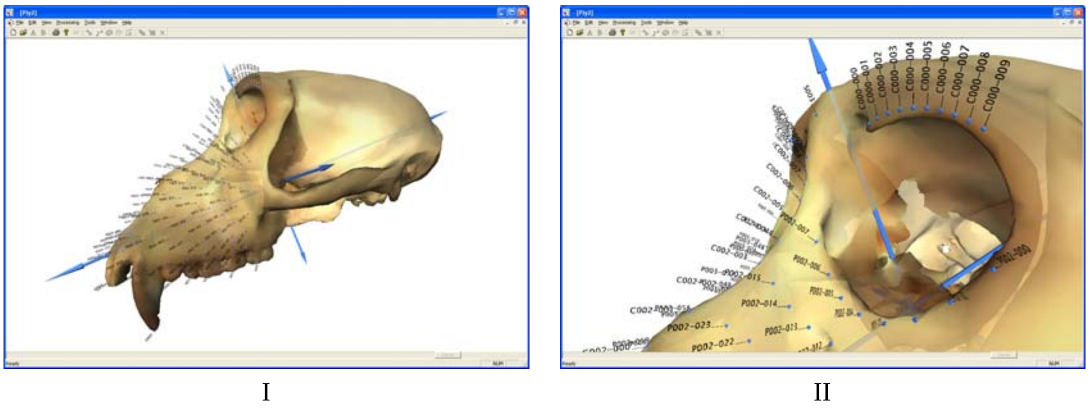
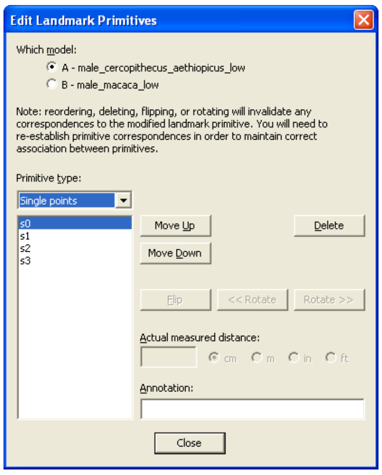


Figure 25: Viewing the exported landmark points.

- The transverse plane can be flipped or rotated.


Edit Landmark Primitives

Which model:

[✓ A - male_cercopithecus_aethiopicus_low]

B - male_macaca_low

Note: reordering, deleting, flipping, or rotating will invalidate any correspondences to the modified landmark primitive. You will need to re-establish primitive correspondences in order to maintain correct association between primitives.

Primitive type: Single points

s0
s1
s2
s3

Actual measured distance: [✓ cm] m in ft

Annotation:

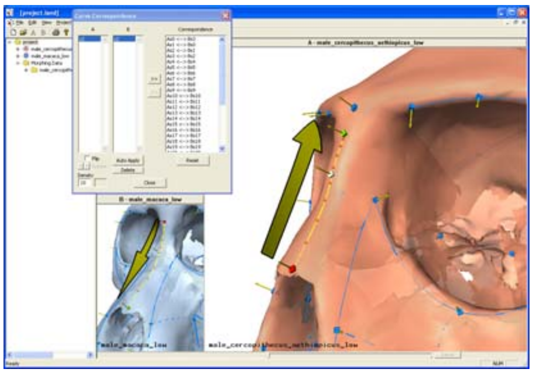
Figure 26: Edit Landmark Primitives dialog. Invoked by selecting `View | Edit primitives...`. Primitives can be re-ordered, deleted, annotated, rotated, and flipped.

Note that re-ordering and deleting a primitive will destroy any correspondences to affected primitives.

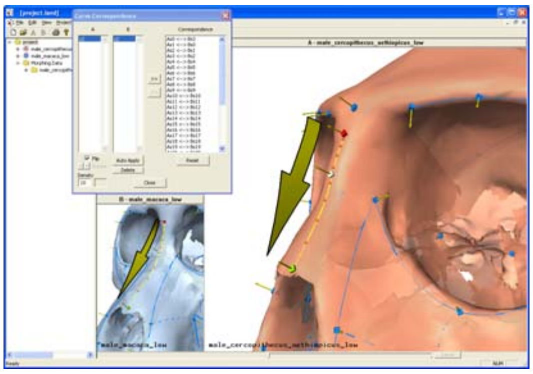
4.10 Alignment of curves and patches

If the `Auto Apply` feature for transferring landmark primitives is not used, it is likely that curves and patches will be placed such that extra care must be taken to ensure correct correspondence of the primitives. Figure 27 shows an example of the problem as it relates to curves. Alternatively, you can use the Edit Landmark Primitives dialog to change the orientation of curves and patches rather than having the correspondence itself manage the inconsistency, see Section 4.9.

Patches have more variation and require a bit more attention. Figure 28 shows an example of the problem as it relates to patches. Here, the goal is to match up the swoosh (the arrow) over the patches.

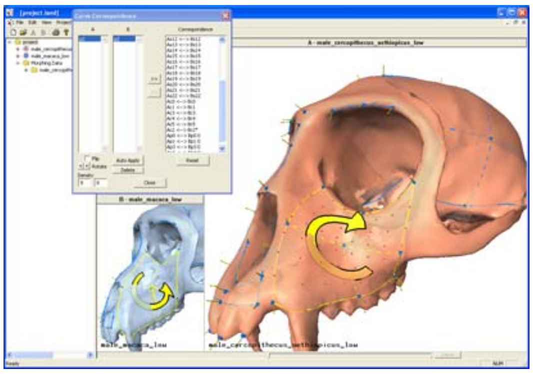


I

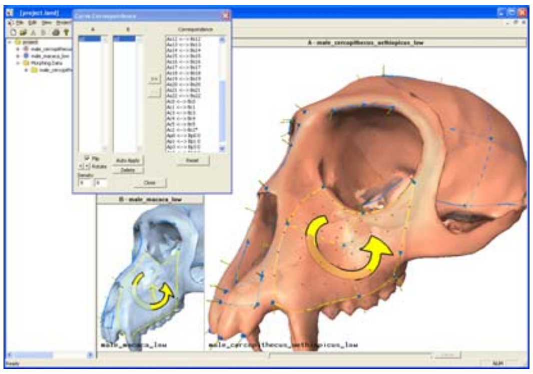


II

Figure 27: Example of orientation problem caused by placing curve points in a different order across bridge of nose. Image I shows the orientation of the selected curves is different since the red and green nodes are flipped. Image II shows the corrected correspondence achieved by selecting the `Flip` check box in the correspondence dialog.



I



II

Figure 28: Example of orientation problem caused by placing patch points in a different order across cheek. Image I shows the orientation of the selected patches is different since the swooshes do not match up. Image II shows the corrected correspondence achieved by selecting the `Flip` check box and by rotating the swoosh in the correspondence dialog.

4.11 Semi-landmark density

The density of the automatically generated semi-landmark is controlled from within the correspondence dialog. When a matching pair of curves or patches is selected from the A and B columns, the `Density` edit boxes become enabled and you should be able to see orange dots on the surfaces across the curve or patch. These orange dots are the automatically generated semi-landmark points.

There is a minimum and maximum number of automatically generated points. Curves can have between 3 and 41 automatically generated points. Patches can have a minimum grid size of 3×3 (9 points) and a maximum grid size of 33×33 (1089 points). Select on odd number for the density in order to have them fall on the manually placed control points.

4.12 Finding Maximal Landmark Locations

Placing landmark points at extremal surface locations is difficult without some sort of guidance. Landmark provides a way to denote the transverse or coronal plane of an object which implies set of axes defined by this plane. The user can then sweep transparent planes aligned to the axes in order to better show extremal locations of points.

Define the transverse plane by first placing four single-point landmarks on the model that lie on this plane. In the case of the monkey crania in our examples, these four points lie at the bottom of the eye sockets and just above the ear holes, defining the Frankfurt orientation for primate crania, see Figure 29.

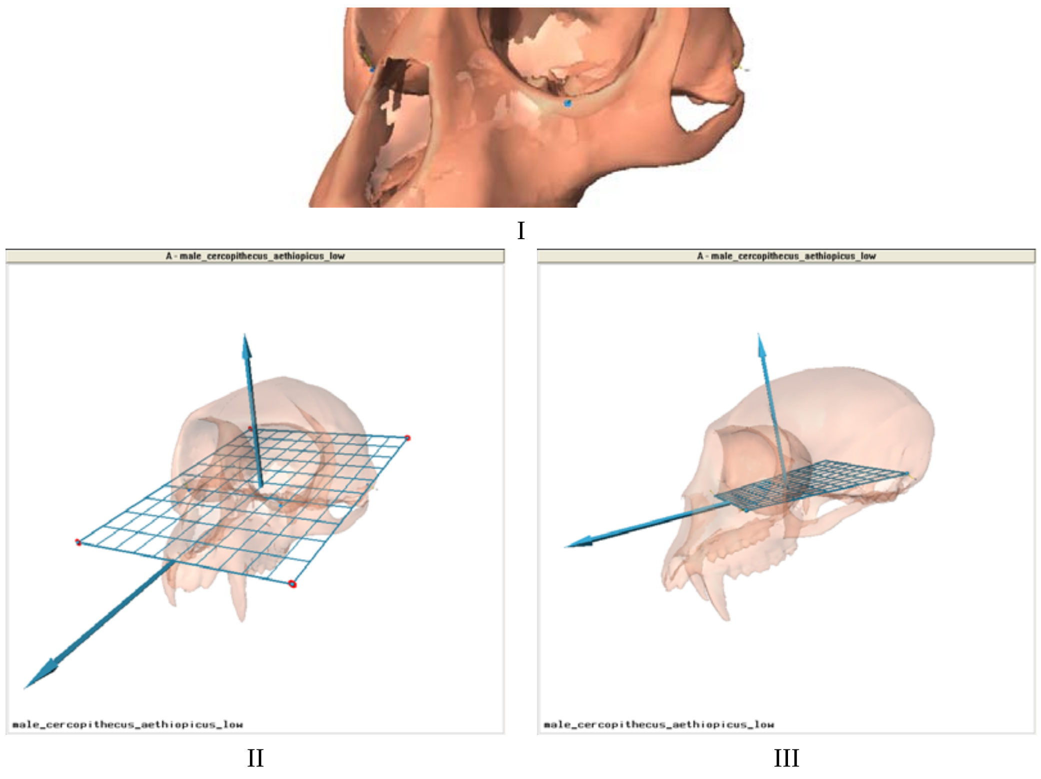

I
A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
II
A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
III

Figure 29: Image I shows placement of single-point landmarks at the bottom of both eye sockets and also just above the ear holes. Image II shows the how to define the transverse plane by dragging the red-circled dots onto the four previously placed landmarks. Large blue arrows indicate “up” and “front” directions. Image III shows the final placement of the four points defining the transverse plane.

Once the transverse plane is defined, transparent planes rendered along the implied axes can be used to find maximal landmark locations. Select `View | Reference Planes | Show ???`, where ??? is axial (or transverse), sagittal, or coronal, to display a corresponding transparent plane, see Figure 30. You can move a plane along the perpendicular axes by left-clicking and dragging while holding down shift on the sphere at the intersection of the dark grey circle and cross-hatched lines.

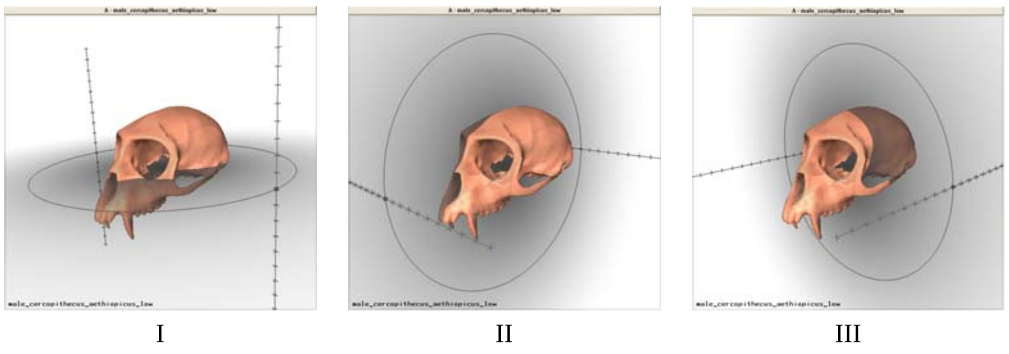

A - male_cercopithecus_aethiopicus_low
male_cercopithecus_aethiopicus_low
I
II
III

Figure 30: Transparent planes rendered along the axes implied by the transverse plane. Image I shows the transverse plane. Image II shows the sagittal plane. Image III shows the coronal plane. You can move a plane along the perpendicular axes by left-clicking and dragging while holding down shift on the sphere at the intersection of the dark grey circle and cross-hatched lines.

5 Working With Fossils

There are a number of problems that fossils present when it comes to placing landmarks on their surfaces. The following tools are provided in order to adjust deformed and fractured meshes so that they can be used.

5.1 Retrodeformation

Retrodeformation is used to un-deform a mesh based upon pairs of points on opposite sides of the (sagittal) plane of symmetry for the fossil. This retrodeformation process operates under the assumption that these pairs should be symmetric across this plane and this operations attempts to optimize the symmetry of selected pairs of points. Pairs of points are specified using the dimension primitive , see Figure 31.

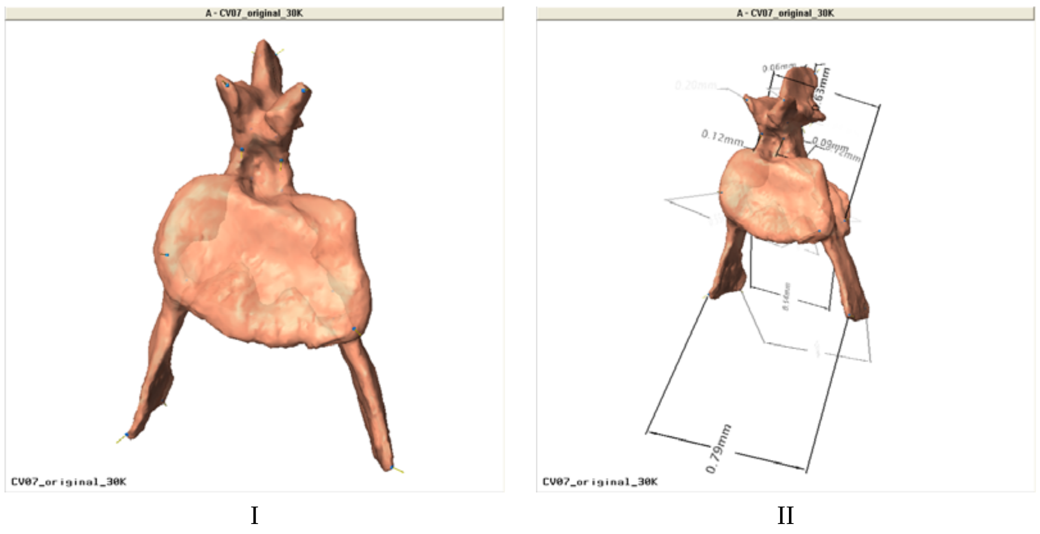
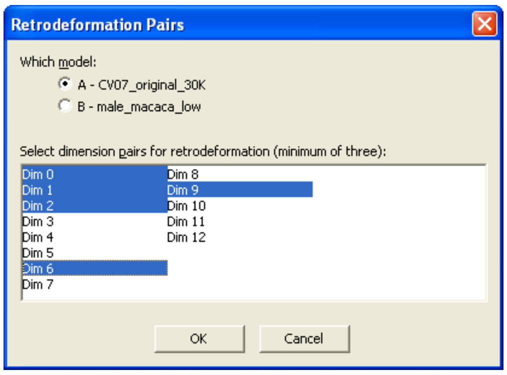


Figure 31: Image I shows original fossil surface. Image II shows placement of dimension primitives having paired points across the plane of symmetry.

Begin the retrodeformation process by selecting `Project | Retrodeform...` from the menu. Select which surface to retrodeform (A or B if more than one surface is loaded). Choose which dimension primitives to use for the retrodeformation process. A minimum of three dimension primitives (three pairs of points) is required in order to define a proper retrodeformation. More than three can be used to further constrain the process, see Figure 32.


Retrodeformation Pairs

Which model:

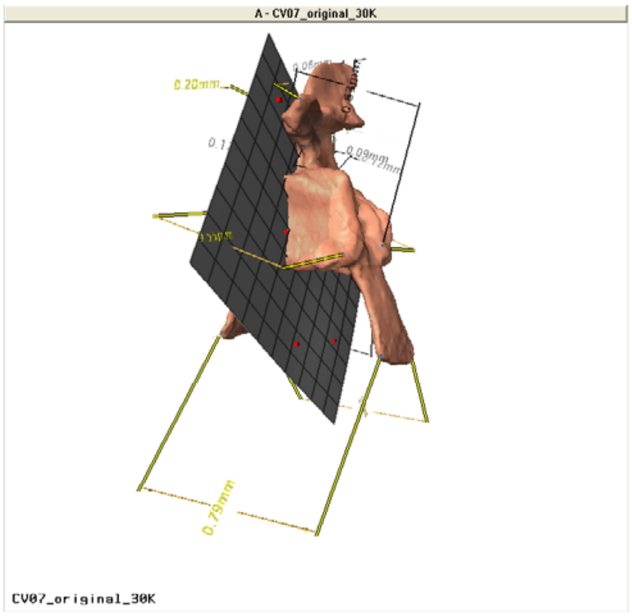
(●) A - CV07_original_30K

( ) B - male_macaca_low

Select dimension pairs for retrodeformation (minimum of three):

[✓ Dim 0] [✓ Dim 1] [✓ Dim 2] Dim 3 Dim 4 Dim 5 [✓ Dim 6] Dim 7 Dim 8 [✓ Dim 9] Dim 10 Dim 11 Dim 12

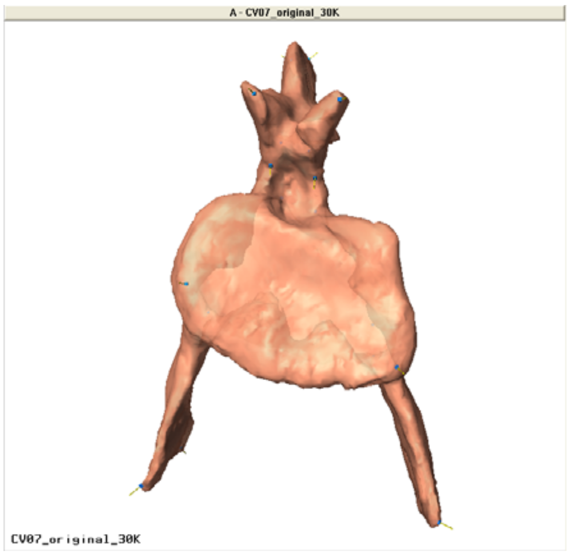
I



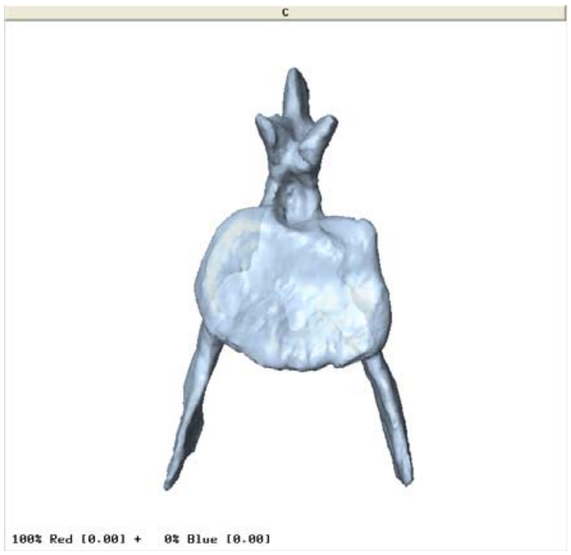
II

Figure 32: Image I shows the retrodeformation dialog; Selected dimension primitives are are those that contribute to the retrodeformation process. Image II shows which dimension primitives have been selected as highlighted in yellow. A deformed plane of symmetry is shown in gray and can be used to estimate the amount of deformation that took place. This plane comes from fitting a smooth surface through the midpoints of the point pairs (small red boxes) used to establish the retrodeformation procedure.

Once three or more dimension primitives have been selected, select `OK` to retrodeform the surface. The result is placed in pane C of the editor, see Figure 33. You can save the retrodeformed surface in the landmark database by selecting `Project | Import from C | Blue surface`. A new item representing the chosen surface is added to the selected group in the database view.



I

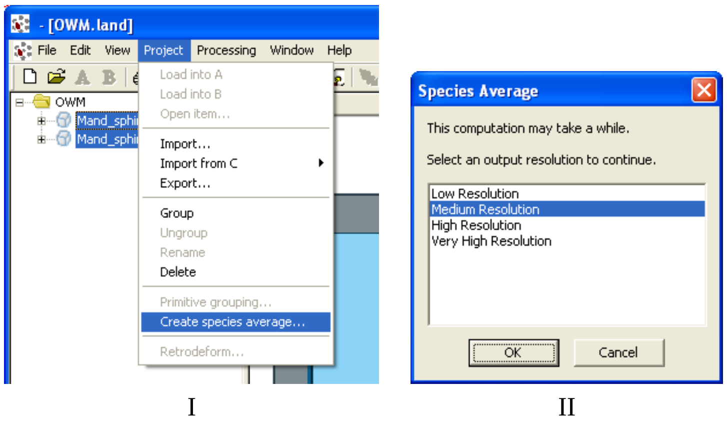


II

Figure 33: Comparison between original fossil, shown in Image I, and retrodeformed surface shown in Image II. Save the retrodeformed surface in the landmark database by selecting `Project | Import from C | Blue surface`.

6 How to Create a Species Average

Select the models in a database that will contribute to the species average. You must select more than one model and they must all have landmarks placed and corresponded on them for this process to work. Select `Project | Compute species average...` to compute an average of all selected surfaces, see Figure 34. You must choose the desired output resolution of the species average. The resolution level and the number of triangles in the surface meshes is directly proportional to the amount of time and memory this process uses. Scaling is removed when computing this average in order to capture only shape changes rather than scale differences. The resulting average model is displayed in pane C and can be saved into the database by selecting `Project | Import from C | Blue surface`. After the blending process has finished, the resulting species average is shown in pane C, see Figure 35.



This computation may take a while.

Select an output resolution to continue.

Figure 34: Image I shows how to begin the species average process. Image II shows the selected output resolution of the average. The higher the resolution the longer the process takes.

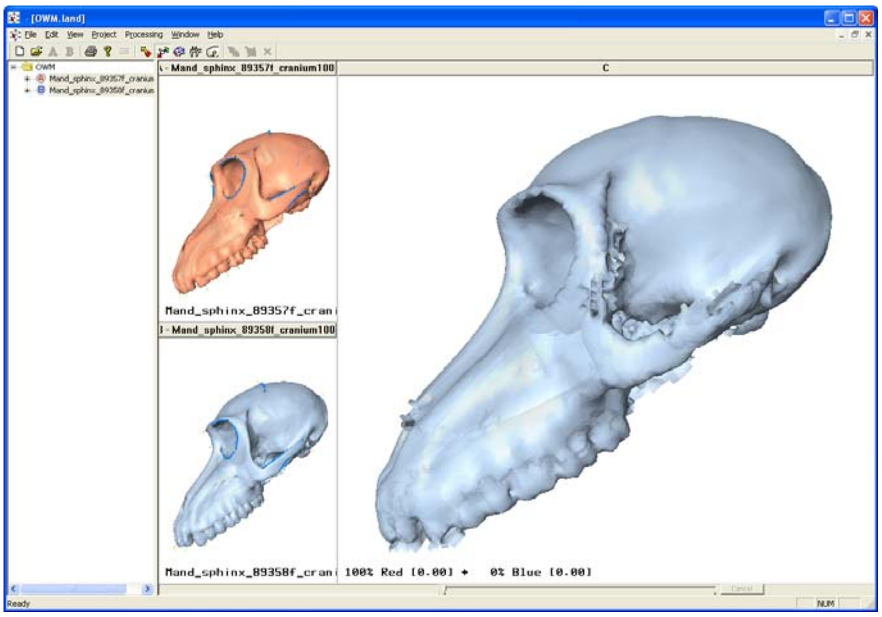

[OWM.land]
File Edit View Project Processing Window Help
OWM
Mand_sphinx_89357f_cranium
Mand_sphinx_89358f_cranium
A - Mand_sphinx_89357f_cranium100
C
Mand_sphinx_89357f_crani
B - Mand_sphinx_89358f_cranium100
Mand_sphinx_89358f_crani
100% Red [0.00] ♦ 0% Blue [0.00]
Ready
NUM

Figure 35: Species average. Save the surface into the landmark database by selecting `Project | Import from C | Blue surface`.

7 Working with Individual PLY Surfaces

7.1 Loading a single PLY file

In terms of surface data, Landmark can only load PLY files. You can open PLY files for the purpose of viewing the surface or for working with landmarks. To view a PLY file, select `File | Open...` and choose the PLY file you want to open. You can also drag the file into the workspace to open it automatically. Figure 36 shows an example PLY surface loaded in Landmark.

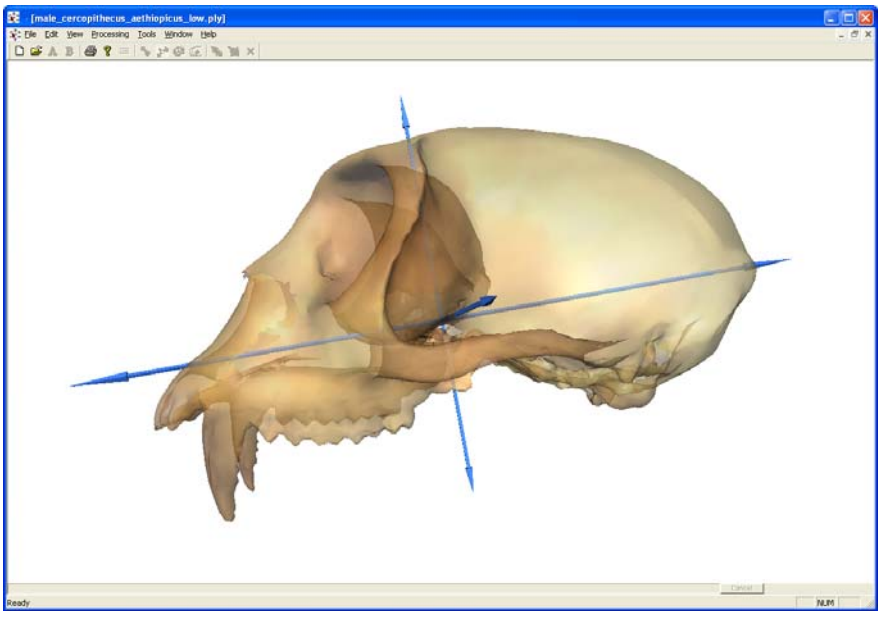

[male_cercopithecus_aethiopicus_low.ply]
File Edit View Processing Tools Window Help
Ready
NUM

Figure 36: A PLY surface within Landmark. The PLY viewer (does not have three panes) is only for viewing PLY surface. Landmarks can only be placed on surfaces loaded from a `.land` database.

8 2D Image Morphing

A very basic two-dimensional image warping process is also supported in Landmark. This is accomplished using the following steps.

1. Load two images that you want to morph from one to another, see Figure 37.
2. Place single point landmarks on them by left-clicking the mouse while holding down the shift key, see Figure 38. The order in which you places points on each image is crucial.
3. Select the image you want to warp from then select `Tools | Warp...` from the menu to select the image you want to warp to, see Figure 39.

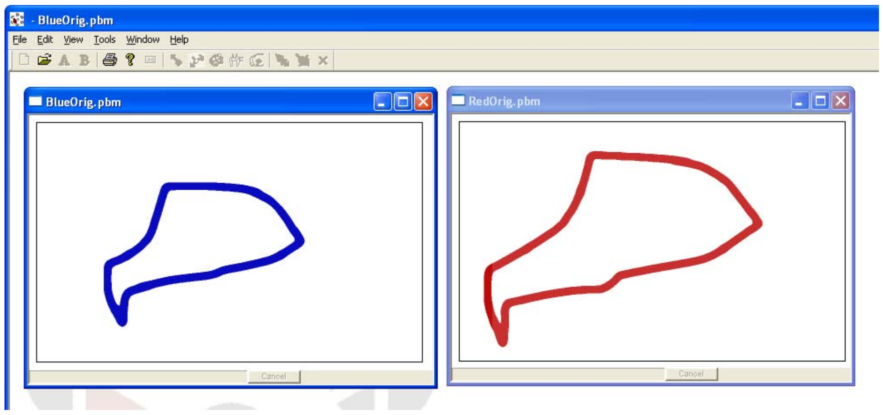
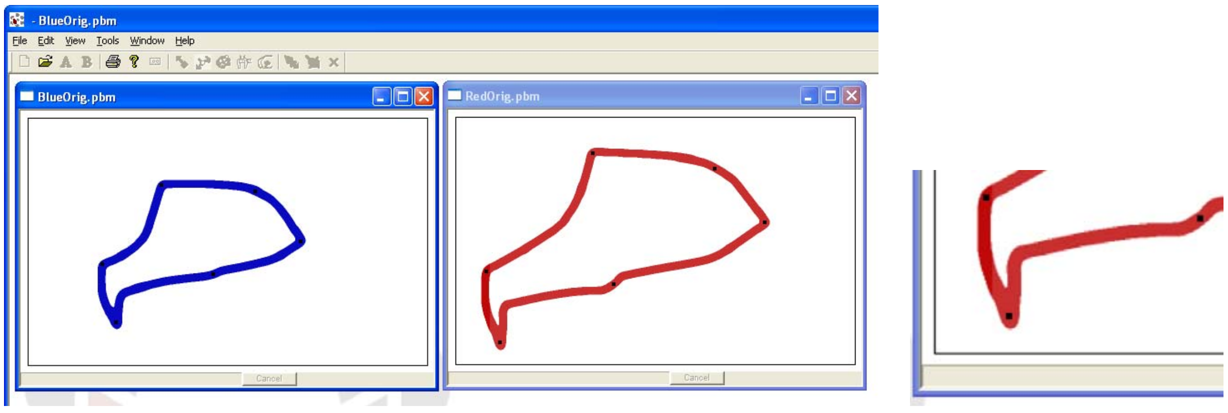


Figure 37: Load two images to warp one to another. Landmark only supports a few image file formats.



I II

Figure 38: Image I shows six landmark points (black dots) placed on each image. Image II shows a close up of the red surface.

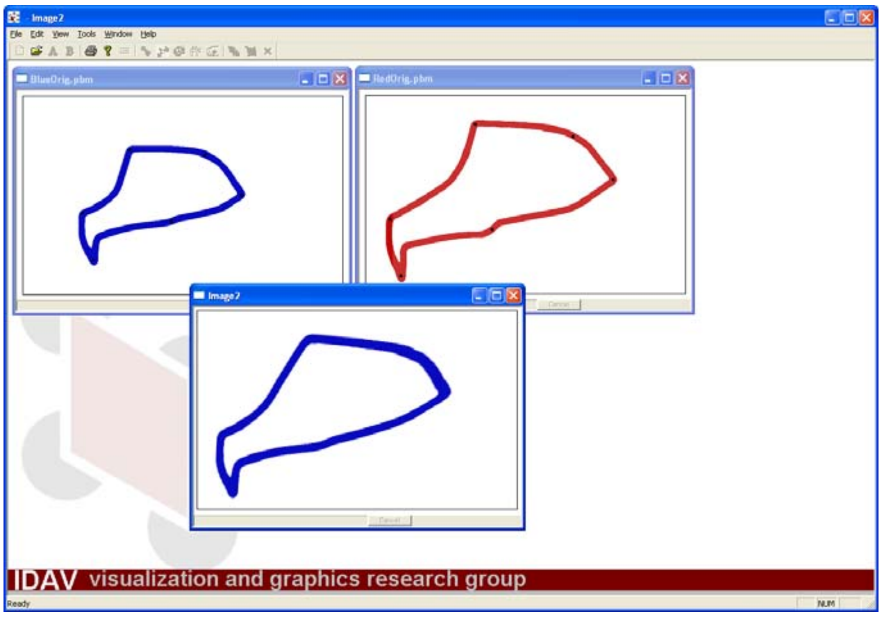


Image2
File Edit View Tools Window Help
BlueOrig.pbm
RedOrig.pbm
Image2
IDAV visualization and graphics research group
Ready
NUM

Figure 39: Warped image from `BlueOrig.pbm` to `RedOrig.pbm` using six landmark points. More points will produce better warps.

9 The `.pts` File Format

Landmark reads two types of text-based `.pts` files. Their file formats are described below. Missing data is identified by the value set in `Tools | Options...` If any of the point coordinates are this value, then the point is considered “missing.”

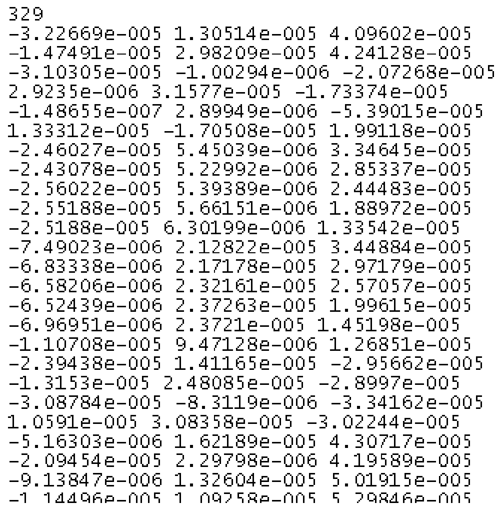
9.1 Version 0

Version zero simply has a the number of points *N* on the first line followed by *N* x,y,z floating-point triples, see Figure 40.

```
N
x1   y1   z1
x2   y2   z2
x3   y3   z3
     ...
xN   yN   zN
```

Figure 40: Version 0 text-based file format for a `.pts` file. First line indicates the number of points *N* followed *N* x,y,z floating-point triples.

Figure 41 shows an example .pts file having 329 landmark points in it.



```
329
-3.22669e-005 1.30514e-005 4.09602e-005
-1.47491e-005 2.98209e-005 4.24128e-005
-3.10305e-005 -1.00294e-006 -2.07268e-005
2.9235e-006 3.1577e-005 -1.73374e-005
-1.48655e-007 2.89949e-006 -5.39015e-005
1.33312e-005 -1.70508e-005 1.99118e-005
-2.46027e-005 5.45039e-006 3.34645e-005
-2.43078e-005 5.22992e-006 2.85337e-005
-2.56022e-005 5.39389e-006 2.44483e-005
-2.55188e-005 5.66151e-006 1.88972e-005
-2.5188e-005 6.30199e-006 1.33542e-005
-7.49023e-006 2.12822e-005 3.44884e-005
-6.83338e-006 2.17178e-005 2.97179e-005
-6.58206e-006 2.32161e-005 2.57057e-005
-6.52439e-006 2.37263e-005 1.99615e-005
-6.96951e-006 2.3721e-005 1.45198e-005
-1.10708e-005 9.47128e-006 1.26851e-005
-2.39438e-005 1.41165e-005 -2.95662e-005
-1.3153e-005 2.48085e-005 -2.8997e-005
-3.08784e-005 -8.3119e-006 -3.34162e-005
1.0591e-005 3.08358e-005 -3.02244e-005
-5.16303e-006 1.62189e-005 4.30717e-005
-2.09454e-005 2.29798e-006 4.19589e-005
-9.13847e-006 1.32604e-005 5.01915e-005
-1.14496e-005 1.09258e-005 5.29846e-005
```

⋮

Figure 41: Example of version zero `.pts` file format.

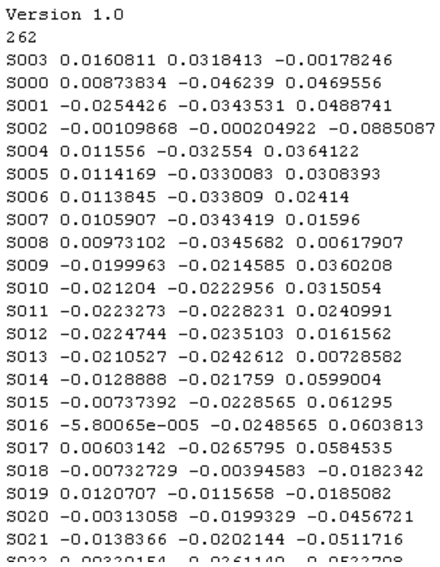
9.2 Version 1.0

Version one has the added features of version control and annotations, see Figure 42.

```
Version 1.0
N
a1   x1   y1   z1
a2   x2   y2   z2
a3   x3   y3   z3
     ...
aN   xN   yN   zN
```

Figure 42: Version 1 text-based file format for a `.pts` file. First line indicates file version. Second line indicates the number of points *N* followed *N* a,x,y,z where a is a single word annotation and x,y,z are the floating-point triple of the point.

Figure 43 shows an example `.pts` file having 262 landmark points in it.



```
Version 1.0
262
S003 0.0160811 0.0318413 -0.00178246
S000 0.00873834 -0.046239 0.0469556
S001 -0.0254426 -0.0343531 0.0488741
S002 -0.00109868 -0.000204922 -0.0885087
S004 0.011556 -0.032554 0.0364122
S005 0.0114169 -0.0330083 0.0308393
S006 0.0113845 -0.033809 0.02414
S007 0.0105907 -0.0343419 0.01596
S008 0.00973102 -0.0345682 0.00617907
S009 -0.0199963 -0.0214585 0.0360208
S010 -0.021204 -0.0222956 0.0315054
S011 -0.0223273 -0.0228231 0.0240991
S012 -0.0224744 -0.0235103 0.0161562
S013 -0.0210527 -0.0242612 0.00728582
S014 -0.0128888 -0.021759 0.0599004
S015 -0.00737392 -0.0228565 0.061295
S016 -5.80065e-005 -0.0248565 0.0603813
S017 0.00603142 -0.0265795 0.0584535
S018 -0.00732729 -0.00394583 -0.0182342
S019 0.0120707 -0.0115658 -0.0185082
S020 -0.00313058 -0.0199329 -0.0456721
S021 -0.0138366 -0.0202144 -0.0511716
```

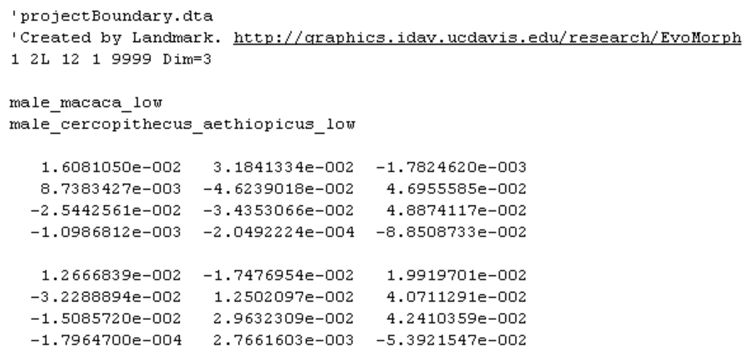
⋮

Figure 43: Example of version one `.pts` file format.

10 The `.dta` NTSys File Format

For exporting, the value used to denote missing data set in `Tools | Options...` For importing, the value for missing data is described within `.dta` files and that value is used instead of that set in `Tools | Options...`

Do not consider the following description to be the definitive definition of the NTSys file format. I obtained a brief description of it from Steve Frost, which is summarized below. Figure 44 shows an example `.dta` file having four landmark points defined on two surfaces contained within it. The first two lines are comments; there can be an



```
'projectBoundary.dta
'Created by Landmark. http://graphics.idav.ucdavis.edu/research/EvoMorph
1 2L 12 1 9999 Dim=3

male_macaca_low
male_cercopithecus_aethiopicus_low

   1.6081050e-002   3.1841334e-002  -1.7824620e-003
   8.7383427e-003  -4.6239018e-002   4.6955585e-002
  -2.5442561e-002  -3.4353066e-002   4.8874117e-002
  -1.0986812e-003  -2.0492224e-004  -8.8508733e-002

   1.2666839e-002  -1.7476954e-002   1.9919701e-002
  -3.2288894e-002   1.2502097e-002   4.0711291e-002
  -1.5085720e-002   2.9632309e-002   4.2410359e-002
  -1.7964700e-004   2.7661603e-003  -5.3921547e-002
```

Figure 44: Example of a `.dta` file format.

unlimited number of these. The header “1 2L 12 1 9999 Dim=3” describes the following items

- The first “1” indicates that a data matrix follows.
- The “2L” means there are two objects with labels.
- The “12” states that there are 12 coordinates (x, y, and/or z) per object (i.e., 4 landmarks per object).
- The second “1” indicates that there may be missing data (a “0” here would mean no missing data).
- The “9999” denotes the value used to indicate missing data.
- The “Dim=3” specifies the dimension of the data.

The names of the objects are on the two lines after the header. Then the coordinates of the landmark points are listed.

Landmark is guaranteed to be able to read and write NTSys files in the above described format and has not been tested on variations of this format. Landmark may not correctly import NTSys files having missing data. If you have a variation of this format and would like it added to Landmark please send an email to David F. Wiley at wiley@cs.ucdavis.edu.

11 Landmark Plugins

A plugin SDK is provided with Landmark for allowing users to perform custom operations on landmark points and surfaces. Landmark passes corresponded landmarks (and semi-landmarks) on selected meshes to the chosen plugin. Within the plugin, you can perform operations on the landmarks and surface meshes if needed. Results can then be passed back to Landmark in the form of displaying a message or adding files into the Landmark database, see Section 11.4 for a complete description of the Landmark Plugin Interface. The plugin SDK is located in the `PluginSDK` sub-folder in the Landmark installation folder.

11.1 Creating a Landmark Plugin Project

Select `Tools | Plugin Manager...`. Type the name of the new plugin in `Plugin Name`, select a project folder, select a project type, and then press `Create` to create the project files.

11.2 Using the Landmark Plugin SDK with Visual Studio

11.2.1 Setup your plugin for debugging with Landmark

In order to setup your project for debugging and testing with Landmark, follow these steps.

1. To make debugging and testing the plugin easier, select `Project | Properties...` for the new plugin project.
2. Select the `Debug` build
3. Select `Debugging` under `Configuration Properties`
4. Set the `Command` to execute as `LandmarkDebug.exe` modifying the path to reflect your plugin project folder. This executable is a debug version of Landmark and is required to test debug builds of Landmark plugins.
5. Set the `Command Arguments` as:

   ```
   -debugplugin "$(TargetPath)"
   ```

   or

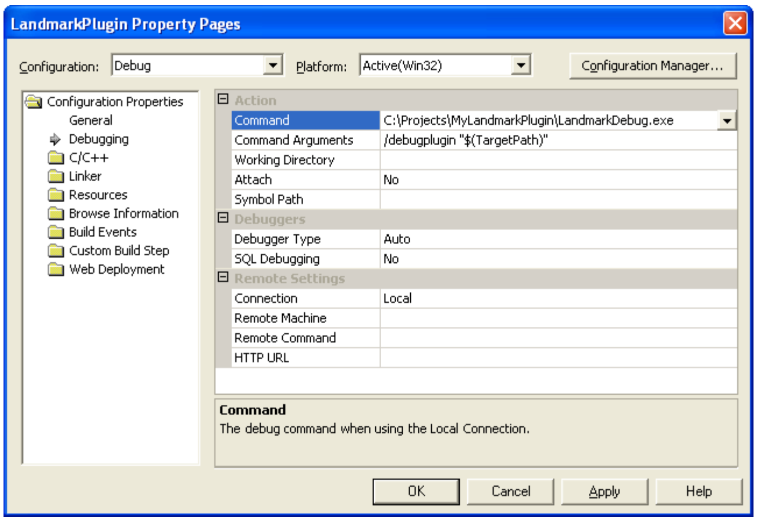
   ```
   /debugplugin "$(TargetPath)"
   ```

   if the first one doesn't work. Note that

   ```
   "$(TargetPath)"
   ```

   needs to be in quotes, see Figure 45.
6. Repeat steps 10 and 11 for the Release build, changing the Command to LandmarkRelease.exe instead.

You should be able to compile, run, and test your plugin at this point.


LandmarkPlugin Property Pages

Configuration: Debug   Platform: Active(Win32)   Configuration Manager...

Configuration Properties
- General
- Debugging
- C/C++
- Linker
- Resources
- Browse Information
- Build Events
- Custom Build Step
- Web Deployment

Action	
Command	C:\Projects\MyLandmarkPlugin\LandmarkDebug.exe
Command Arguments	/debugplugin "$(TargetPath)"
Working Directory	
Attach	No
Symbol Path	
Debuggers	
Debugger Type	Auto
SQL Debugging	No
Remote Settings	
Connection	Local
Remote Machine	
Remote Command	
HTTP URL	

Command
The debug command when using the Local Connection.

Figure 45: Setting up the plugin project properties for debugging.

11.2.2 Compiling, running, and testing your plugin

To compile, run, and test your plugin, follow these steps.

1. Build your plugin, Release and/or Debug builds (assuming you followed the steps above)
2. Run your plugin with the debugger, this should start `LandmarkDebug.exe` or `LandmarkRelease.exe` within the SDK folder.
3. Open a .land file that you want to test your plugin with.
4. Select items in the Database view and then choose `Project | Custom Plugins`. The name of your plugin should appear, something similar to `MyPlugin (debug)`. The `(debug)` indicates that the plugin was started by Landmark with the `/debugplugin` command argument, see Figure 46.

Proceed to Section 11.4 for instructions on how to customize your newly created plugin.

11.3 If you don't have Visual Studio

When creating the the plugin project, select `Non-visual Studio` for the project type. You will need to create a new project in whatever development environment you are using and add the plugin files to the new project.

At this point, you should be able to compile the project. Next, follow the instructions in Sections 11.2.1 and 11.2.2 to setup the plugin for debugging and testing within Landmark. Then, proceed to Section 11.4 for instructions on how to customize your newly created plugin.

11.3.1 Building a Landmark Plugin in a non-Microsoft Development Environment

You should be able to create a valid Landmark plugin using a non-Microsoft development tool. The resulting Landmark plugin is a standard Windows DLL. If you can create a DLL that adheres to the Landmark plugin interface defined in `Plugin.h`, then everything should work fine. Follow these steps to create a non-Microsoft Landmark plugin.

1. Create a DLL project using your development environment.
2. Define `BUILD_PLUGIN_DLL` in your project settings to indicate that you are building the plugin DLL.

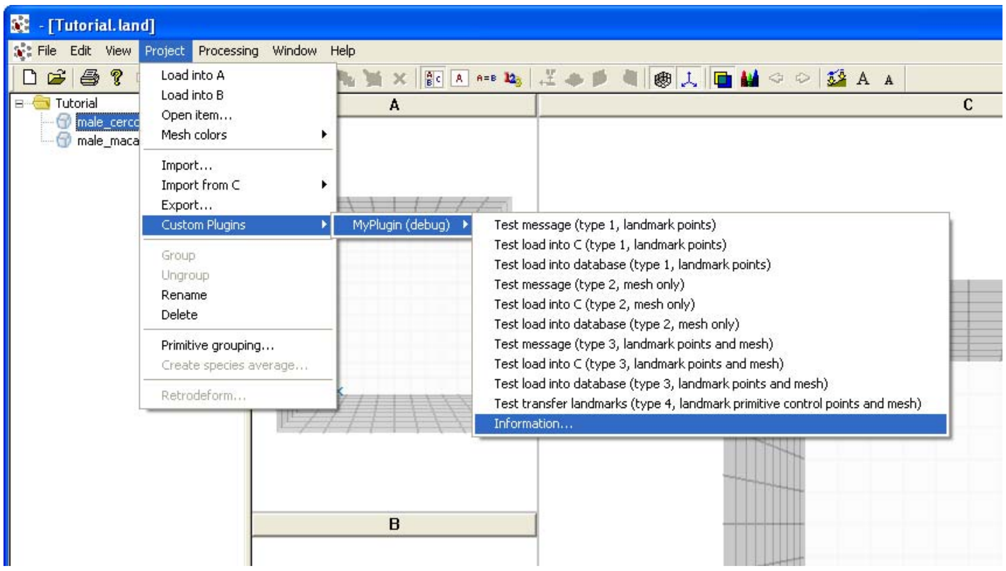

- [Tutorial.land]
File
Edit
View
Project
Processing
Window
Help
Tutorial
male_cerco
male_maca
A
B
C
Load into A
Load into B
Open item...
Mesh colors
Import...
Import from C
Export...
Custom Plugins
Group
Ungroup
Rename
Delete
Primitive grouping...
Create species average...
Retrodeform...
MyPlugin (debug)
Test message (type 1, landmark points)
Test load into C (type 1, landmark points)
Test load into database (type 1, landmark points)
Test message (type 2, mesh only)
Test load into C (type 2, mesh only)
Test load into database (type 2, mesh only)
Test message (type 3, landmark points and mesh)
Test load into C (type 3, landmark points and mesh)
Test load into database (type 3, landmark points and mesh)
Test transfer landmarks (type 4, landmark primitive control points and mesh)
Information...

Figure 46: Accessing your plugin from within Landmark.

3. Change `LMPLUGINAPI` in `Plugin.h` to reflect the correct exporting of DLL methods.

At this point, you should be able to compile the plugin project in your development environment.

11.4 The Landmark Plugin Interface and How to Use it

This section contains a detailed description of the Landmark Landmark Plugin Interface and how Landmark communicates to the plugin.

11.4.1 Interface Objects

A brief overview of the interface is shown in Tables 1 and 2.

Landmark loads the plugin DLL and then queries the interface to see what type of plugin it has loaded. Landmark calls `GetPluginVersion()` just after loading each plugin to determine what version of the interface the plugin was built with. Landmark is guaranteed to work correctly with plugins that are built using a older versions of the Landmark Plugin Interface. If a newer version of Landmark accesses an older plugin, it will prepare the data fed into the plugin as it was in the older version and also process data returned from the plugin as it should.

Landmark next calls `GetPluginName()` to obtain the plugin name. This is used to identify the plugin in Landmark's menus. `GetPluginInformation()` should be used to identify who wrote the plugin.

Each plugin can support several different landmark operations, the number of which Landmark queries using `GetNumberOfOperations()`. Landmark obtains a detailed description of each operation calling the plugin method `GetOperationInformation(...)`. The information obtained describes the operation type and various requirements on the number of models that the operation can handle instructing Landmark on how to invoke each operation.

Depending upon the type of operation, Landmark calls one of the four `InvokeOperationTypeX(...)` methods. Methods 1 through 3 are similar in that either (semi-)landmark points and/or model meshes are passed. The landmark points that are passed represent those points that are automatically generated from the Landmark primitives within Landmark. The reason these are separated into four different methods is that it takes time to compute the (semi-)landmarks, thus, if a plugin only requires meshes, there's no need to compute (semi-)landmark points (or to prepare model meshes for transfer).

`InvokeOperationType4(...)` is intended for automatically transferring landmark primitives from an atlas model to other models. Thus, the "landmarks" that are passed into this method are the control points for the Landmark primitives, rather than the (semi-)landmark points themselves. The complexity of curves, patches, and such is hidden from the plugin. You can determine corresponding landmark control points by comparing their labels with the atlas, always the 0th item in `vLand` and `vMesh` (only for type 4 is it always the 0th item). Duplicates are never passed for this operation.

Plugins send information back to Landmark using the `ILandmarkApp` interface. Calls to methods in this object go directly to Landmark, please refer to the following section for details on this object's methods.

11.4.2 ILandmarkApp

The `ILandmarkApp` interface object allows the plugin to communicate with Landmark. A brief description of each method is described in Table 3.

The file `Plugin.cpp` is intended as an instructional tools as well as a starting location for building your own plugin. The original code in this file covers all aspects of the interface and how to properly communicate back and forth from the plugin and Landmark.

11.5 Deployment of Landmark Plugins

Once you have finished debugging and testing of your plugin, you can "deploy" your plugin by copying it into the `Plugins` sub-folder in the Landmark installation folder. It will now be automatically loaded by Landmark every time Landmark is started. You should also copy any DLL dependencies your plugin may have into this folder or insure that the required DLLs are available in the system path. Landmark will not be able to load and use your plugin if dependencies are missing.

Interface Item	**Description**
BUILD_PLUGIN_DLL	This macro definition is used to tell the plugin interface whether it is being used to build the plugin DLL or not. All plugin projects should have this symbol defined in the project settings.
LMPLUGINAPI	This defines the appropriate importing or exporting of plugin methods to or from the DLL. Plugin projects should be exporting method interfaces.
namespace LandmarkPlugin	Plugin data structures are wrapped in this namespace.
HMODEL	This is a handle to uniquely identify models passed into a plugin.
MODEL	This structure groups a model handle and the model name together.
template< > Vector	Used to group together a number of objects. This is instead of std::vector< > since the C runtime library linkage can cause problems when different builds (debug or release) of Landmark are used to run the plugin.
String	Represents a character string. See Vector for why std::string is not used.
POINT	Landmark points passed into plugins have x, y, and z coordinates as well as the normal vector associated with the landmark (nx, ny, and nz). The label can be used to uniquely identify an individual point. Corresponding points on different models will share the same label.
LANDMARKPOINTS	This structure groups together a set of landmark points for the given model.
MESHPOINT	This is a point used in a triangle mesh of a model. It has x, y, and z coordinates, associated normal vector (nx, ny, and nz), and red, green, and blue (r, g, and b, respectively) color components.
TRIANGLE	Holds the indices to MESHPOINTS to define model triangles.
MESH	Holds a vector of MESHPOINTS and a vector of TRIANGLES that define the model's triangle mesh.
ILandmarkApp	This is Landmark's representative interface object that the plugin can use to send data back to Landmark, for example, see Section 11.4.2.
OPTYPE	Defines the four basic operation types that plugins can perform.
OPERATION	Detailed information about a particular plugin operation.
VERSION	Versioning for the plugin interface.

Table 1: A brief description of the Landmark Plugin Interface objects defined in `Plugin.h`.

Interface Method	**Description**
LandmarkPlugin::VERSION **GetPluginVersion**()	Landmark calls this method to determine the version of the plugin interface used. You should not change the value returned by this method.
const char* **GetPluginName**()	Returns the name of your plugin. Default is “MyPlugin”, you should change this value.
const char* **GetPluginInformation**()	Returns a string describing the authors of the plugin.
bool **CanOpenItem**(const LandmarkPlugin::String &strFileExtension)	Returns whether or not the plugin can open a file with the passed extension. The dot ’.’ is included, for example, “.txt”.
bool **OpenItem**(const LandmarkPlugin::String &strFilename, HWND hParentWnd, LandmarkPlugin::String &strTitle, LandmarkPlugin::ILandmarkApp *pApp)	The plugin should open the file passed. The file is a temporary file and writes to this file will not be recorded unless the plugin call `AppendResult()` to re-import the file into the Landark database. hParentWnd is the window in which the plugin should create a “viewer” if appropriate. strTitle can be used to set the title for the opened item.
int **GetNumberOfOperations**()	Returns the number of operations this plugin supports.
bool **GetOperationInformation**(int nOp, LandmarkPlugin::OPERATION *pInfo)	Retrieves information for a given operation.
bool **InvokeOperationType1**(int nOp, const LandmarkPlugin::Vector< LandmarkPlugin::LANDMARKPOINTS > &vLand, LandmarkPlugin::ILandmarkApp *pApp)	Passed only (semi-)landmark points.
bool **InvokeOperationType2**(int nOp, const LandmarkPlugin::Vector < LandmarkPlugin::MESH > &vMesh, LandmarkPlugin::ILandmarkApp *pApp)	Passed only model meshes.
bool **InvokeOperationType3**(int nOp, const LandmarkPlugin::Vector< LandmarkPlugin::MESH > &vMesh, const LandmarkPlugin::Vector< LandmarkPlugin::LANDMARKPOINTS > &vLand, LandmarkPlugin::ILandmarkApp *pApp)	Passed both (semi-)landmark points and model meshes.
bool **InvokeOperationType4**(int nOp, const LandmarkPlugin::Vector< LandmarkPlugin::MESH > &vMesh, const LandmarkPlugin::Vector< LandmarkPlugin::LANDMARKPOINTS > &vLand, LandmarkPlugin::ILandmarkApp *pApp)	Passed control points for Landmark primitives (NOT landmarks or semi-landmarks) along with model meshes for the purpose of automatically transferring Landmark primitives from the atlas (0th item in vLand and vMesh) to the remaining models.

Table 2: A brief description of the Landmark Plugin Interface methods defined in `Plugin.h`.

Interface Method	**Description**
HWND **GetMainLandmarkWindow**()const	Call this method to get the window handle of Landmark's main window. This can be used as the parent window of dialogs you may want to originate from the plugin.
bool **Progress**(long lPos, long lMax)	Use this method to notify Landmark of operation progress. If your operation consists of three steps, then you would first call Progress(0,3) to get things started, then (1,3), (2,3), and finally (3,3) to finish. This method controls the progress bar at the bottom of Landmark. A return value of false means the user pressed the cancel button during the operation.
bool **Status**(const String &strStatus)	To be used in conjunction with Progress(...). Currently not used anywhere within Landmark. A return value of false means the user pressed the cancel button during the operation.
void **DisplayMessage**(const String &strMsg)	Pops-up a message box within Landmark to display some text.
bool **LoadIntoC**(const MESH &mesh, const LANDMARKPOINTS &land, bool bResetView)	Instructs landmark to load the passed mesh and/or landmark points into the C window pane. The parameter `bResetView` controls whether or not the viewing parameters are reset each time this method is called. You can call this method repeatedly.
bool **AppendResult**(const MESH &mesh, const LANDMARKPOINTS &land, const std::string &strName)	Similar to LoadIntoC, but this method appends the passed information into the Landmark database.
bool **ImportFile**(const String &strFilename)	This imports any file into the database, for example, any file you may make in the course of running your plugin. The file is compressed internally to conserve space.
bool **SetModelLandmarks**(HMODEL hModel, const LANDMARKPOINTS &land)	Intended to be used with automatic landmark transferring operations of type 4, calling this method destroys any existing landmarks for the specified model and replaces them with those passed.

Table 3: A brief description of the `ILandmarkApp` interface object defined in `Plugin.h`.

12 Acknowledgements

Send comments, suggests, or otherwise regarding Landmark to David F. Wiley at `wiley@cs.ucdavis.edu`. A complete list of the team members involved in building Landmark are

- David F. Wiley[1,2]
- Nina Amenta[1,2]
- Eric Delson[4,5,6,7,8]
- F. James Rohlf[3,4,5,6]
- Bernd Hamann[1,2]
- Katherine St. John[5,6,9]
- Dan Anthony Alcantara[1,2]
- Yong J. Kil[1,2]
- Deboshmita Ghosh[1,2]
- Shengyin Gu[1,2]
- Will Harcourt-Smith[6,8]
- Steven Frost[6,10]
- Alfred L. Rosenberger[4,5,6,11]
- Lissa Tallman[4,5,6,7]
- Todd Disotell[5,12]
- Ryosuke Motani[13]

1. Institute for Data Analysis and Visualization, University of California, Davis
2. Department of Computer Science, University of California, Davis
3. Department of Evolution and Ecology, Stony Brook University
4. Ph.D. Program in Anthropology, City University of New York
5. NYCEP (New York Consortium in Evolutionary Primatology)
6. NYCEP Morphometrics Group
7. Department of Anthropology, Lehman College/CUNY
8. Division of Paleontology, American Museum of Natural History
9. Department of Mathematics and Computer Science, Lehman College/CUNY
10. Department of Anthropology, University of Oregon
11. Department of Anthropology, Brooklyn College, CUNY
12. Department of Anthropology, New York University
13. Department of Geology, University of California, Davis

This material is based upon work supported by the National Science Foundation under Grant No. SEI(SBE)-0513894. Preliminary work of this material was supported by the National Science Foundation under Grant Nos. DEB-0121681/0121682, ACI-9982351, DEB-0212023, ACI-962034, and NIH/NSF P20 MH60975-06A2. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.