Interactive Inverse 3D Modeling

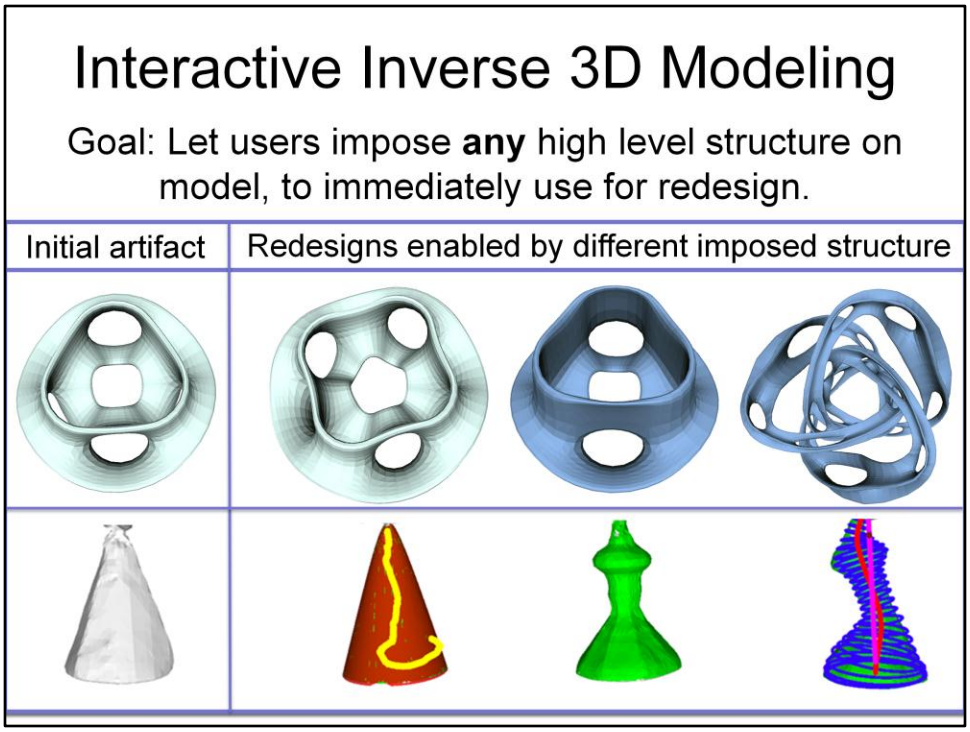
James Andrews
Hailin Jin
Carlo Séquin

Inspiration



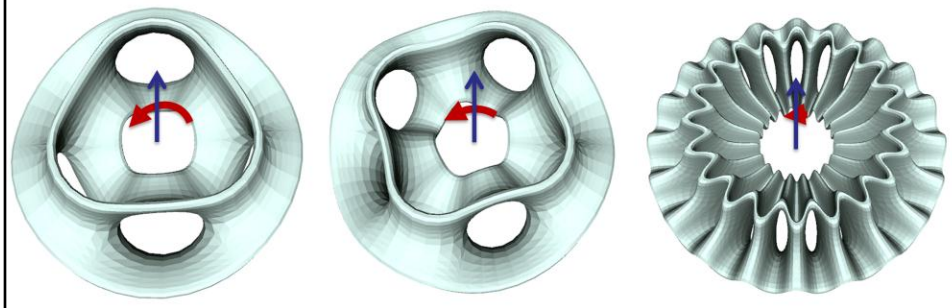
One object → generative concept →
Parameterized sculpture generator

We take inspiration from “Sculpture Generator 1,” a custom design program for the sculpture on the left, which enabled redesigns like those on the middle and right. Our goal is to generalize this idea to a broader domain – not just for nice symmetric sculpture but for arbitrary 3D objects – and to automate the fitting process as much as possible.



By letting the user impose high level, conceptual structures of their choice, our system enables diverse, dramatic shape redesigns.
For example, we show, on the top row: Editing symmetry, editing the flanges of the sculpture by treating it as a surface of revolution, and finally editing the sculpture as a sweep.
On the bottom row: Beautifying a cone by projecting it to a quadric surface, editing the profile of the cone as a surface of revolution, and editing the path of the cone as a sweep.

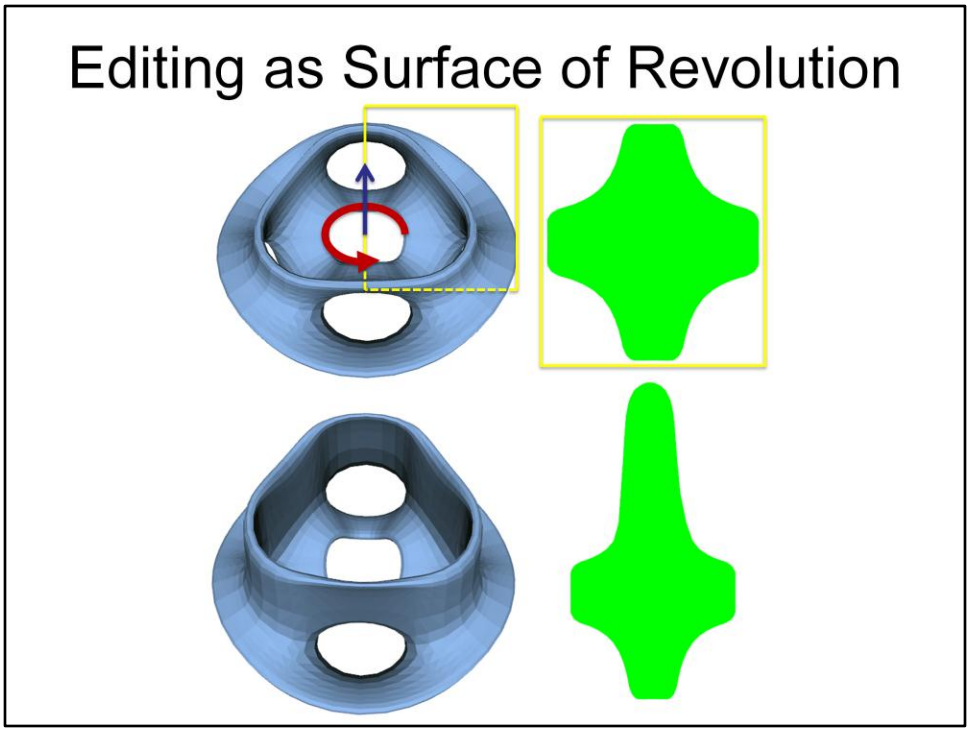
Editing Rotational Symmetry



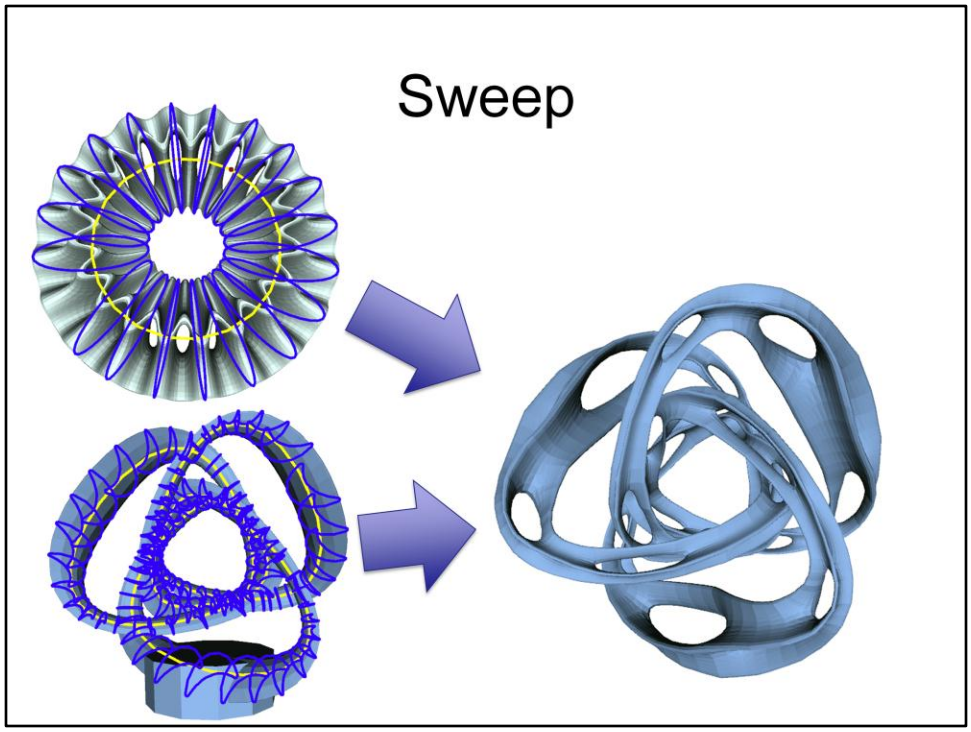
3 fold 4 fold 20 fold

Extract one sector; collapse/expand in polar coordinates.

To edit rotational symmetry, we extract a sector of the model and collapse or expand that sector in polar coordinates. We instance the sector to complete the shape.



Even though this shape is clearly not a surface of revolution, the user can impose that structure on it. The vertices of the mesh are then rotated to a shared plane, creating the “cross sectional” shape shown on the right in green. By editing these vertices in that 2D space, and projecting the results back to 3D, we can for example extend the top flange of the model as we show here.



Again, even though the shape is not a sweep, the user can impose a sweep structure on the model and then reshape the model with a sweep-like editing operation, for example by twisting it into a trefoil knot as we do here.

Our goals:

1. User-guided reverse engineering for broad applications
   - Including mechanical parts, household appliances, and freeform artistic shapes.

2. User imposes high-level structure on representation
   - Not error focused; user can impose conceptual model with desired degrees of freedom and parameters.

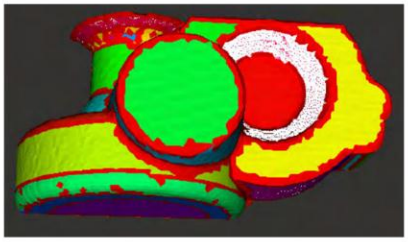
3. Fast data fitting + shape editing in same tool
   - Enable immediate, interactive re-design.

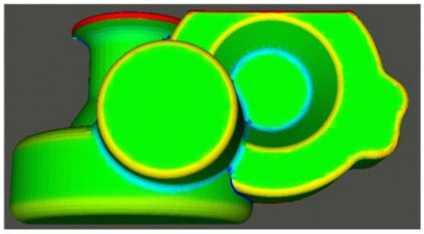
4. Freely allow interactive model re-interpretation
   - As re-design goals change, best structure changes

Related work:

Reverse Engineering

Obtaining a robust and versatile CAD model from a physical artifact or from images.





Algorithms for Reverse Engineering Boundary Representation Models

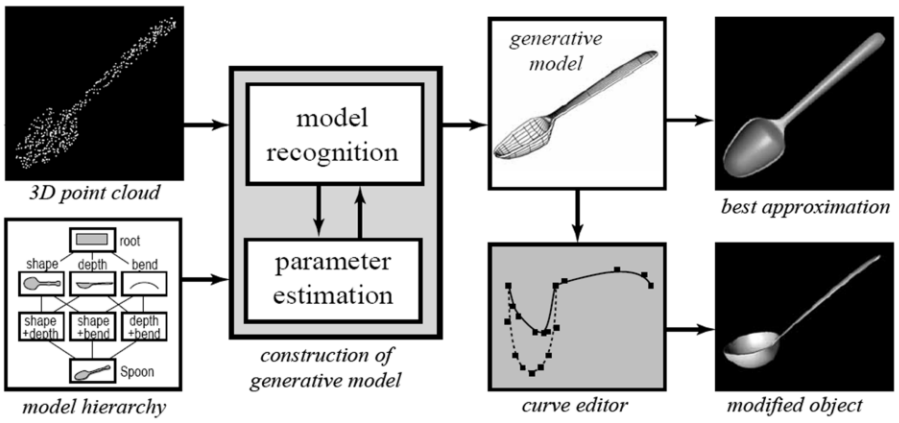
Pál Benkő, Ralph R. Martin (*), Tamás Várady

Typically focus on a single best / versatile reconstruction

There's a rich history of previous work in the field of reverse engineering, much of which focuses on finding a single "best" CAD model from a scanned or photographed physical artifact. We use a number of reverse engineering methods in our system, which will be noted as I discuss them later in the talk.

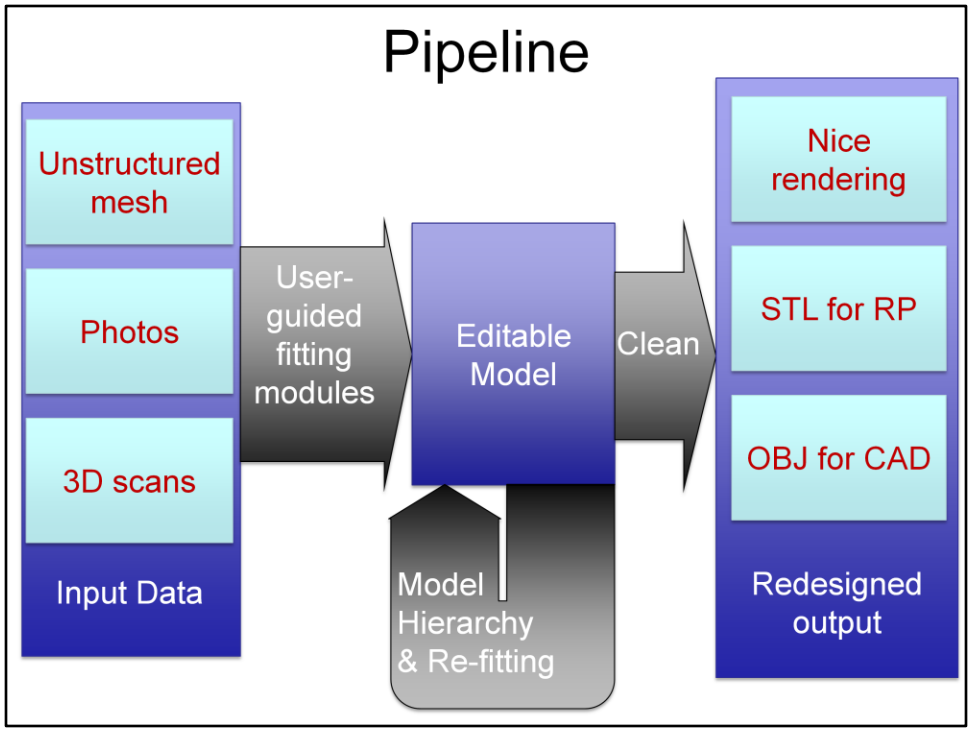
Related Work:

Optimize to find high level parameters for re-design



"Creating Generative Models from Range Images"
Ravi Ramamoorthi and James Arvo

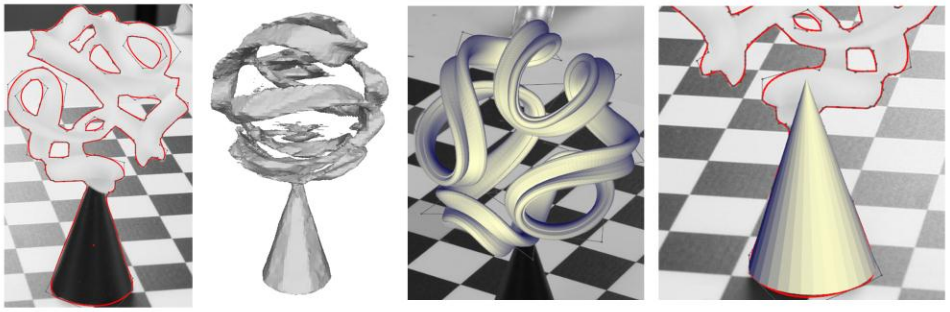
One inspiring work was a paper of Ramamoorthi and Arvo, which decomposed scanned models into parameteric representations which allowed some flexibility for later re-design.



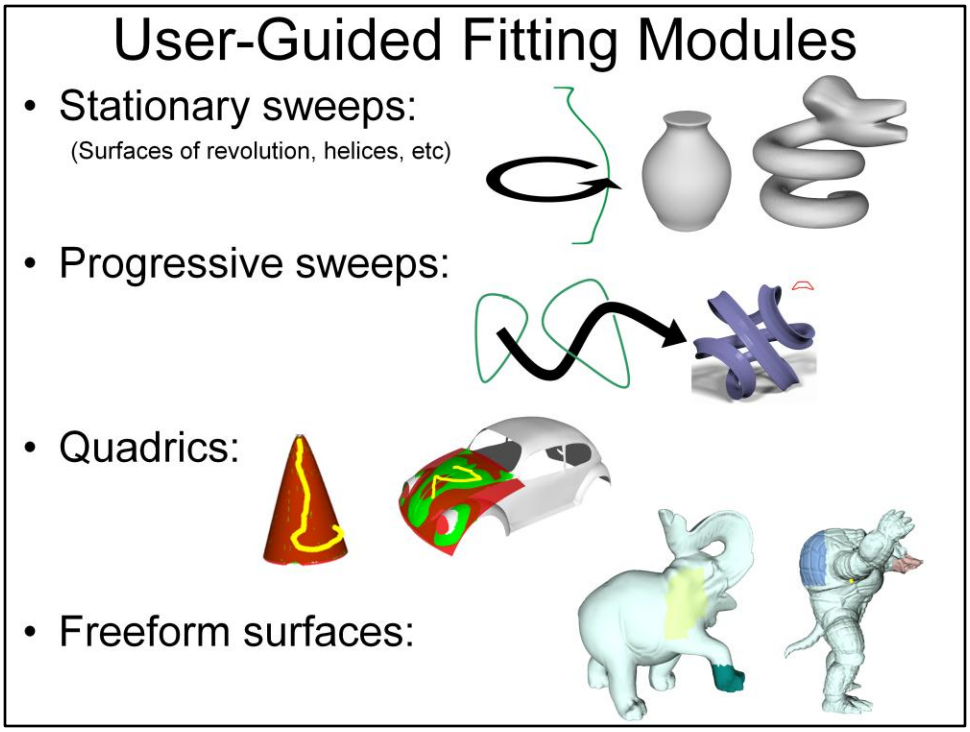
Our system will take low level input data, such a unstructured polygonal meshes or photographs, and through a user-guided fitting process generate an editable, parametric model that permits redesign. We let the users interactively re-fit shapes or impose hierarchical fits as needed. Once a satisfying redesign is created, we output a clean representation of the artifact for rendering, rapid prototyping, or further processing in a CAD system.

Input Sources

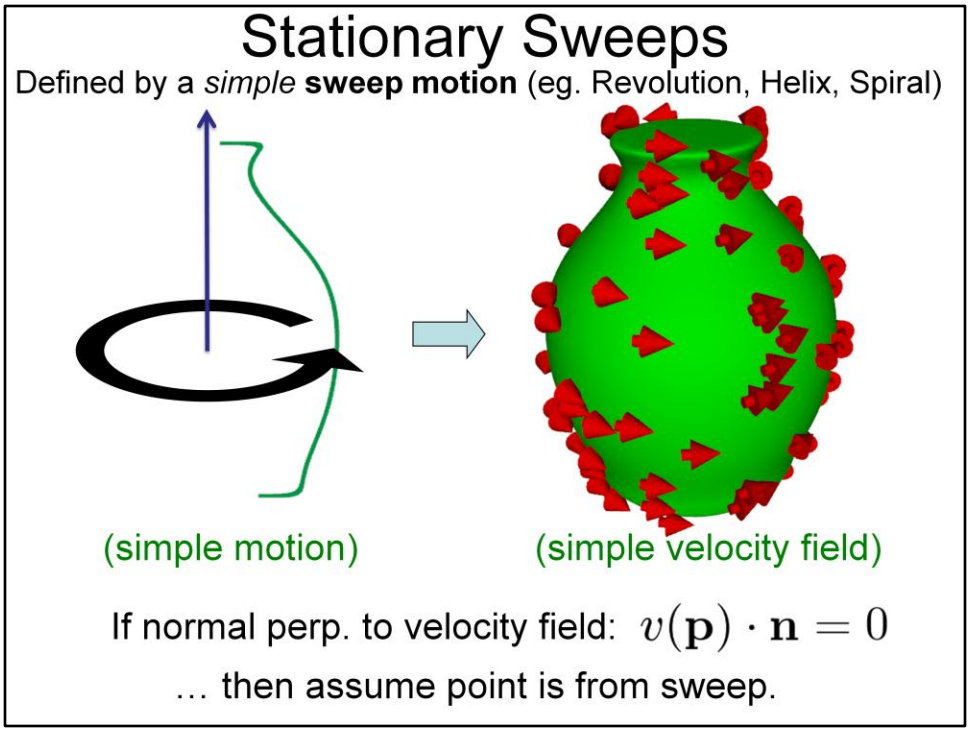
- 3D meshes
- 3D Point clouds from scanner
- Photographs



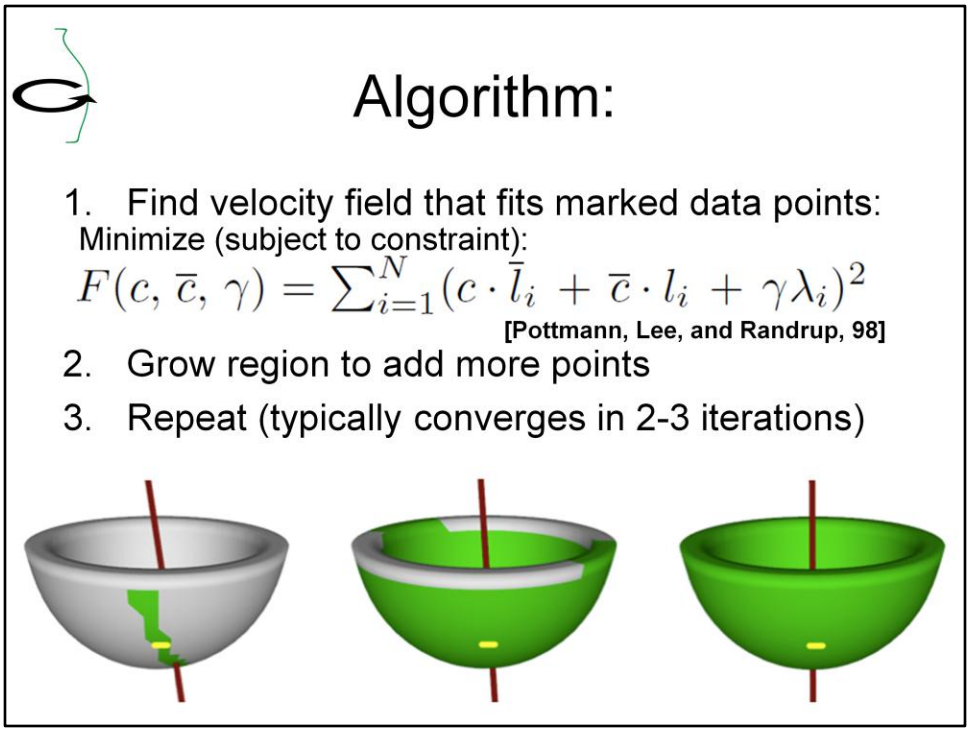
For this presentation, I'll focus on using 3D meshes as our source of input. We also plan to support input from point clouds and photographs. We have done preliminary work on supporting photographs: for example, from a collection of ten photographs of the sculpture on the left, we can extract the visual hull, (the 2nd image is an intersection of 10 silhouette cones). Then, with user hints, a sweep and cone are fit to the corresponding parts of the visual hull.



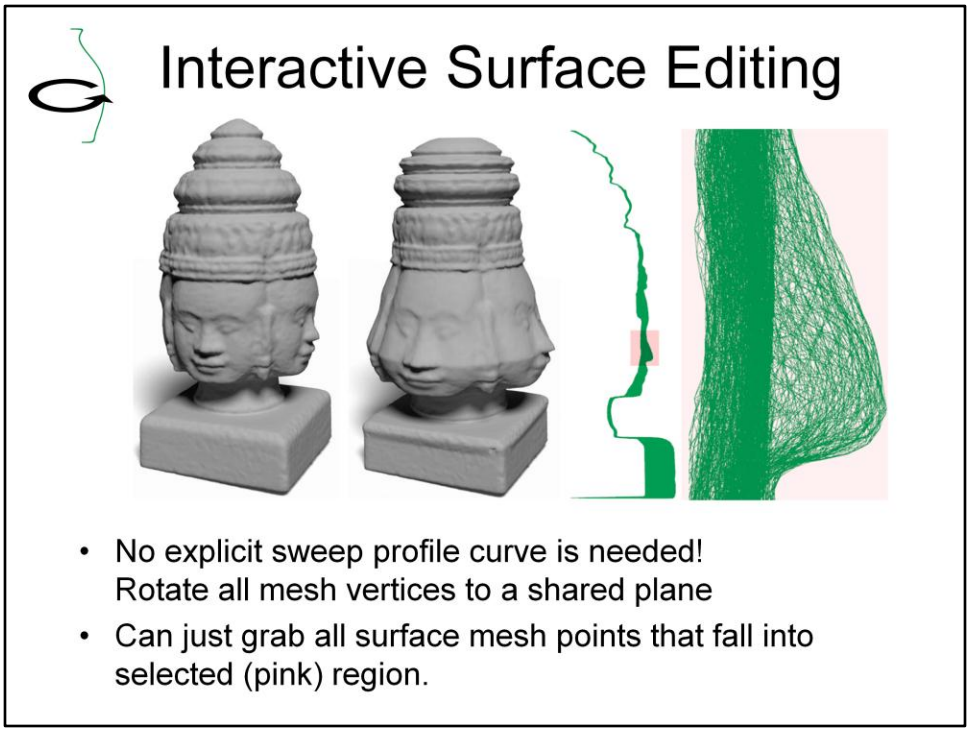
We currently have four fitting modules for different types of primitives: Two modules for different types of sweeps, a module for quadric surfaces, and a general freeform surfaces module.



Our sweep module handles what we call stationary sweeps – sweeps defined by a simple sweep motion, such as surfaces of revolution, helices or spirals. These simple sweep motions correspond to following a simple, linear, velocity field – which enables a fast fitting algorithm specialized to these sweeps.
The key idea is to find a velocity field which is consistently tangential to the surface.



Our fitting algorithm starts from a user stroke on the model (shown in yellow). We then iteratively fit a sweep model to those points, and region grow to expand the selection to those surface points that also fit within some given error tolerance. The problem of fitting a field to any given selection is solved by a small (7x7) generalized eigenvalue problem, and the whole process tends to converge in only 2-3 iterations, making it very fast in practice.



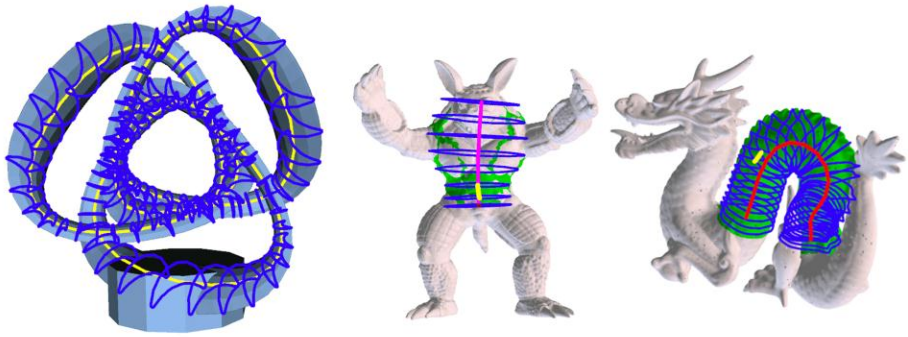
It's important to realize that we have just extracted all the surface elements that fit a particular equation, and we have not extracted an explicit profile curve for the sweep. However, "undoing" the sweep by rotating all mesh vertices into a shared profile plane can collapse the mesh into a "fuzzy" 2D cross section.
Local edits can happen in that 2D space.
The cross section may also contain extra points that fit the given velocity field, but are not parts of the desired surface of revolution. These points can often be easily de-selected in the 2D space, as we show in our video demonstration.

- [Video demo of stationary sweep edits]

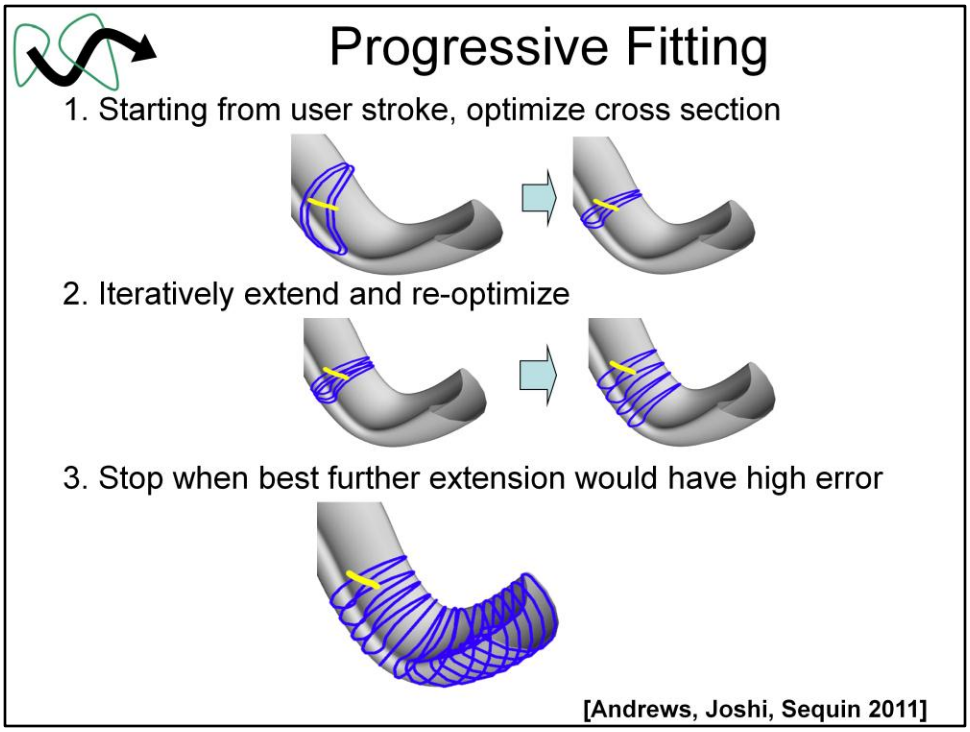
See video: stationary.wmv
Here we show a video of a user de-selecting some excessive points in the cross section of a botijo model.
We also show a video of some edits which using the fit from our stationary fitting module.

Progressive Sweeps

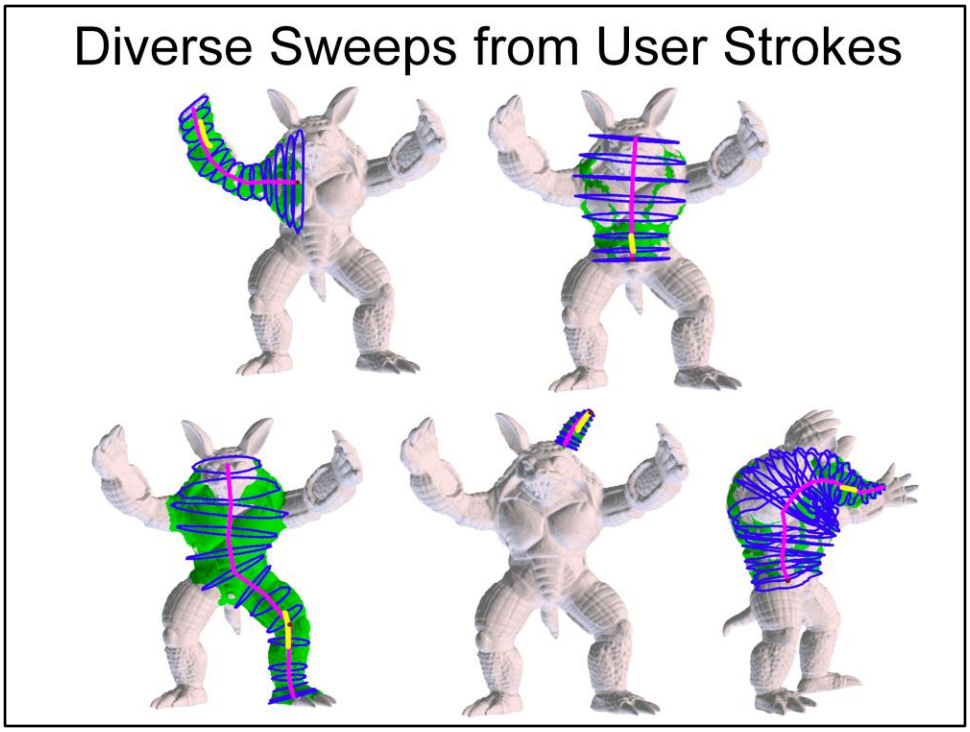
- More parameters: incremental local adjustments
- Allow more complex cross-section moves (translation, rotation, scaling)
- User stroke provides initial guess
- Fit by iteratively extending and optimizing



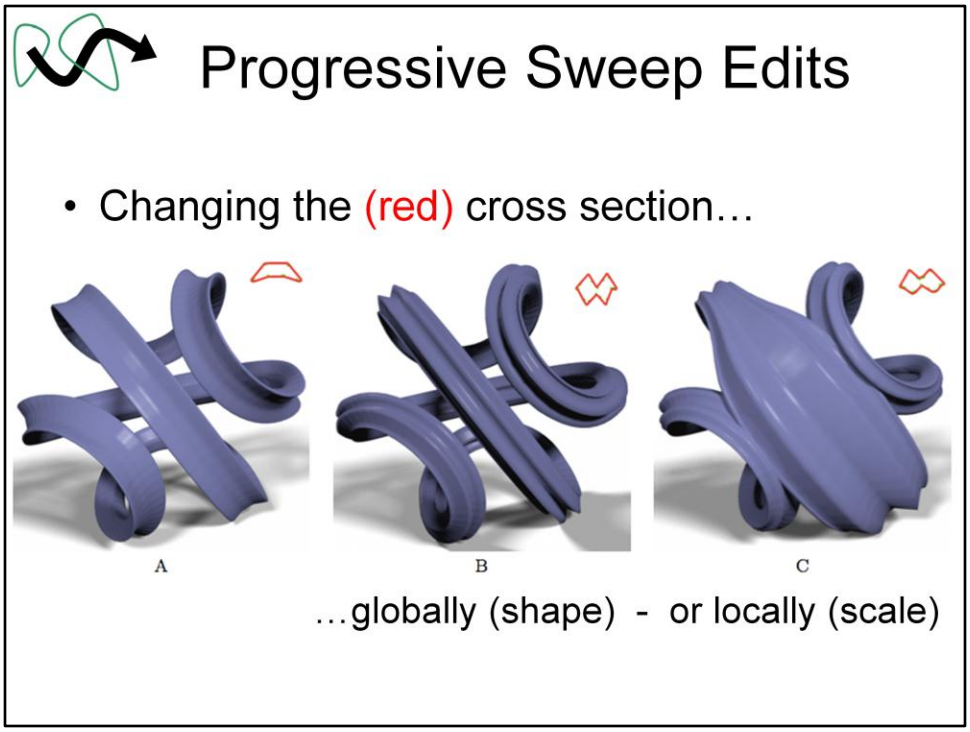
Progressive sweeps are a more general sweep primitive, good for any tubular shapes. For these primitives we actually extract an explicit cross section, and we allow that cross section to move along an arbitrarily complex smooth curve, while being smoothly scaled and slowly rotated.
Because we have so many more possible parameters, we rely on the user to show us a good isolated portion of such a tubular sweep, and to give a rough idea of the direction of the sweep path at that point.



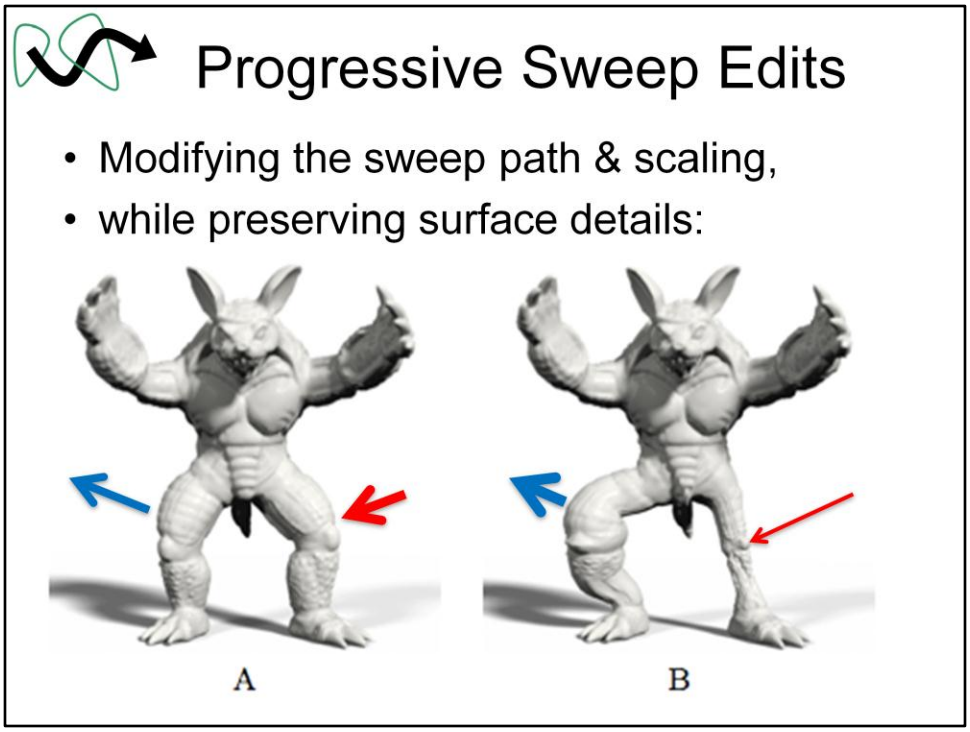
We initialize the fitting process with a sweep matching the model at the center of the user's stroke, and perpendicular to the stroke direction.
We then iteratively optimize and extend that sweep, using a non-linear Levenberg-Marquardt optimizer to minimize the distance of our sweep to the surface, while maintaining smoothness in the sweep as well. We stop this process at the point where further extension of the sweep would result in too high an error.



On a complex object like this armadillo, this fitting process can result in quite diverse sweeps, depending on where the user initially strokes – see the five examples shown.



Once a progressive sweep has been fit, for example to the model on the left, we can start re-designing the extracted shape: for example by changing the shape of the cross section (shown in middle), or by changing one of the sweep parameters, such as the scale in some portion of the sweep (shown on right).

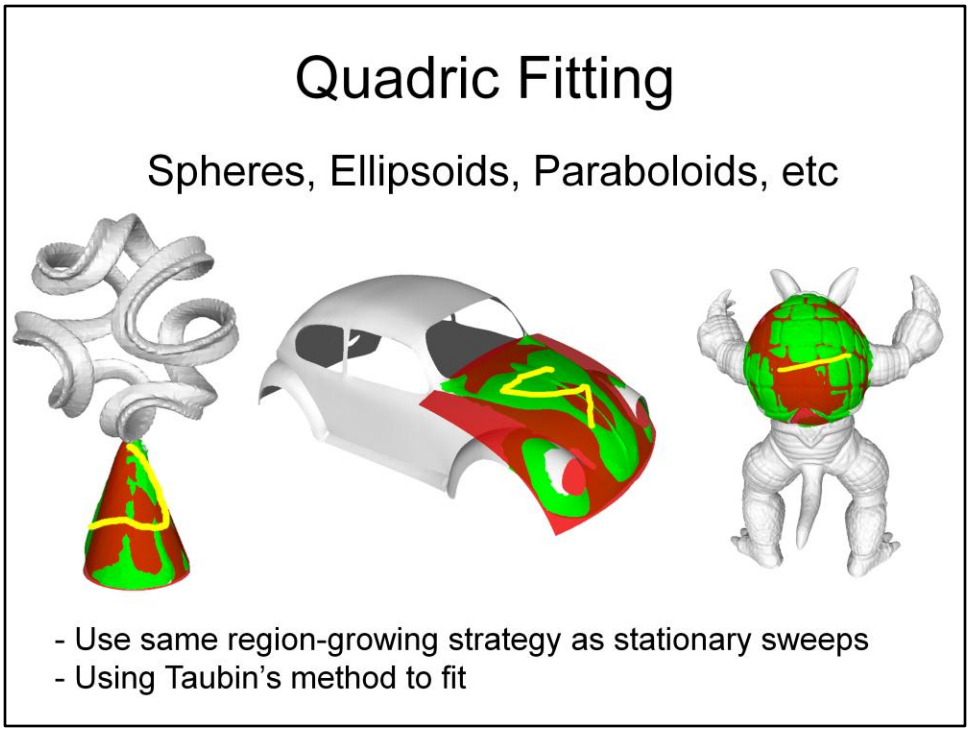


Here we edited the armadillo's legs by fitting sweeps to each leg, and then changing the sweep path and scaling: we scale up and bend the left leg (blue), and scale down and straighten the right one (red).

Note that all small scale details on the leg have been preserved, because we simply apply the transformations from our edit of the sweep structure back to the corresponding local areas of the original mesh.

- [Video demo of progressive sweep edits]

See video: progressive.wmv
Here we show a video of progressive sweep edits.



Our quadric fitting module functions similarly to our stationary sweep fitting module, starting from a user stroke and iteratively fitting and region-growing to extend the fit. To fit a quadric to a selection of points, we use Taubin's method (described next).

In these example fits, green surfaces are the original data selected by the fit, while the red surfaces are the best fitting quadric surfaces.
(Note the plates of the armadillo's armor sticking through the best fitting ellipsoid.)

Taubin's method:

minimize algebraic distance with gradient normalized to an average squared value of 1.

$$f(\mathbf{p}) = \mathbf{p}^{\mathbf{T}}\mathbf{A}\mathbf{p} + \mathbf{b}^{\mathbf{T}}\mathbf{p} + C_0 = 0$$

$$\text{with } \mathbf{A} = \begin{pmatrix} C_7 & C_4/2 & C_5/2 \\ C_4/2 & C_8 & C_6/2 \\ C_5/2 & C_6/2 & C_9 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

$$\min_{\mathbf{m}} \frac{\sum f^2(\mathbf{p}_i)}{\sum || \nabla f(\mathbf{p}_i) ||^2}$$

A small generalized Eigenvalue problem.

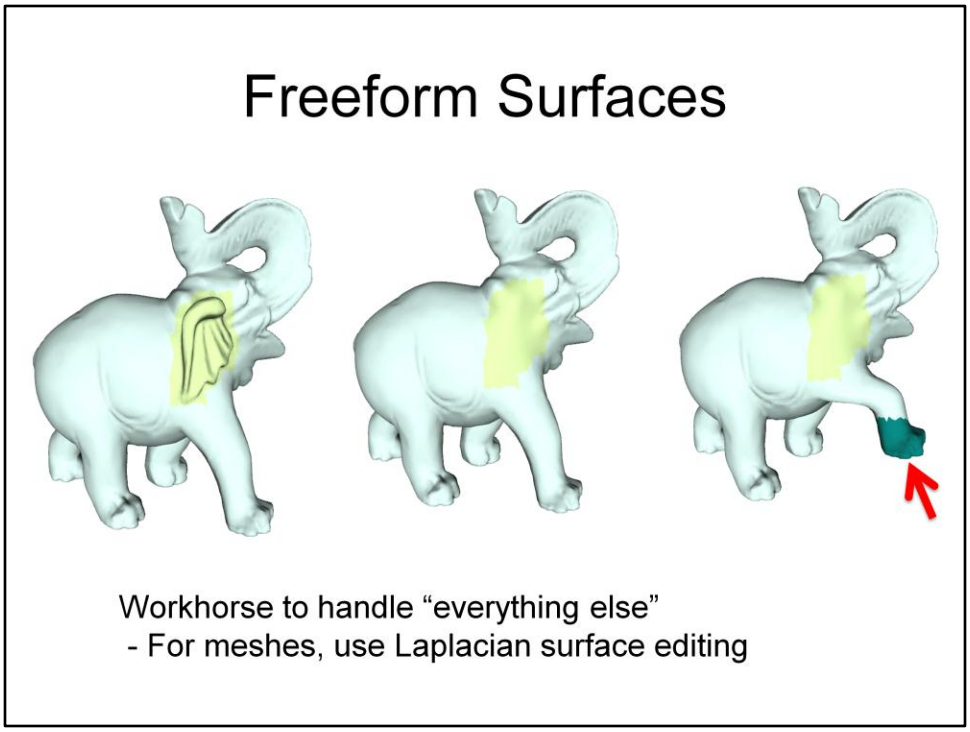
[Taubin, 91]

Taubin's method, like the stationary sweep fitting method, just requires the solution to a small generalized eigenvalue problem. This makes it very fast and easy to implement, and therefore appealing for an interactive system like ours.

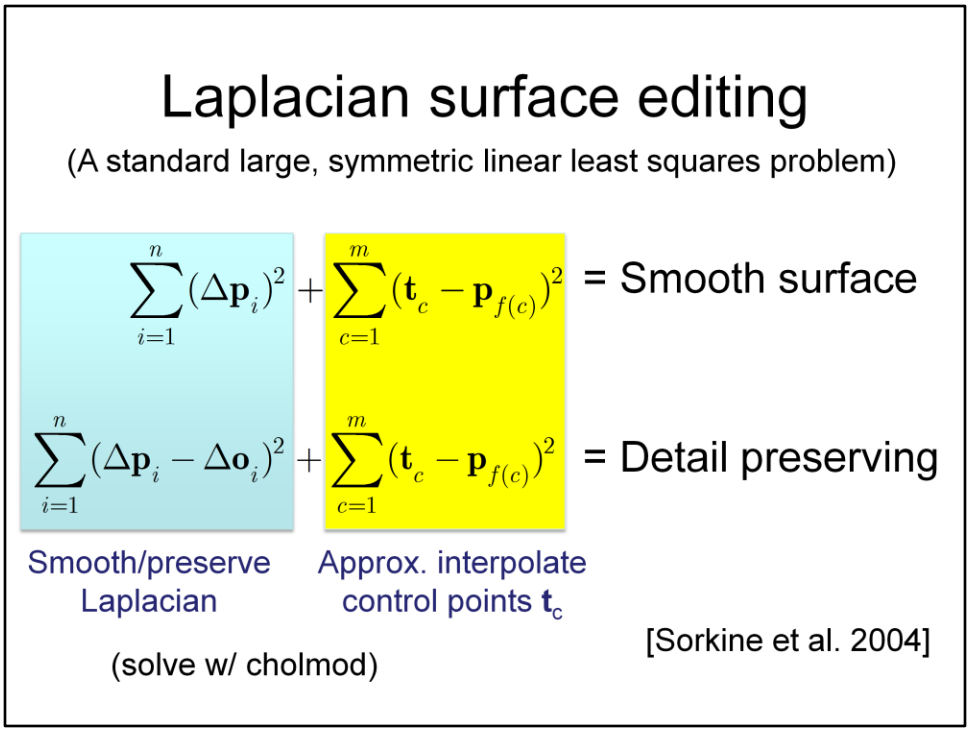
It is an algebraic fitting method, meaning that we approximate distance to the surface by the value of the implicit, algebraic function defining the quadric's isosurface, f(p). Taubin's method normalizes that distance metric by the average squared value of the gradient – avoiding a degenerate solution such as the one where the function is zero everywhere.

- [Video demo of quadric fitting]

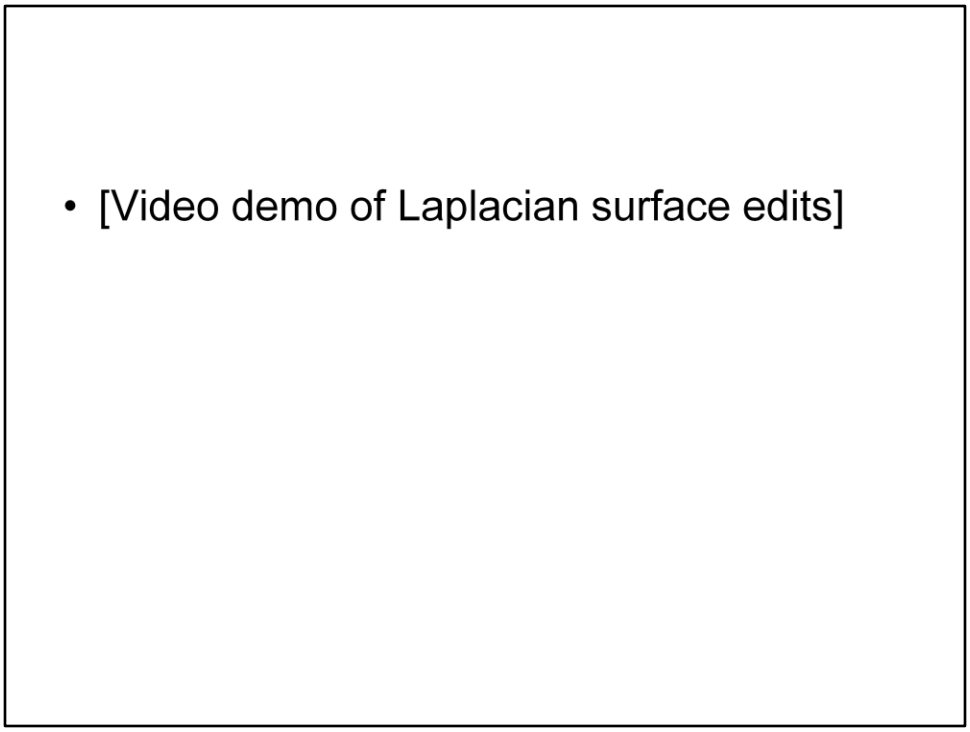
See video: quadrics.wmv
Here we show the quadric fitting process for several examples.



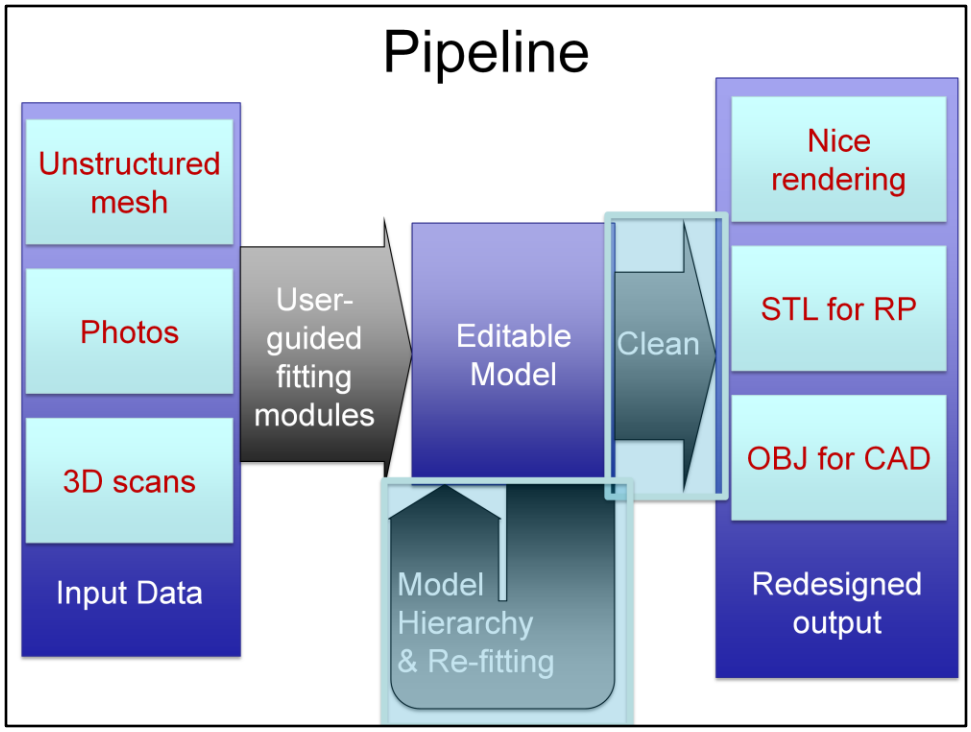
For surfaces which we do not wish to interpret as sweeps or quadrics, or which have not been marked by the user, we use a general freeform surface method: Laplacian surface editing. This will let us optionally smooth details, such as the elephant’s ear (yellow), or preserve details and maintain smoothness as we move parts of the surface around, such as the elephant’s foot (green).



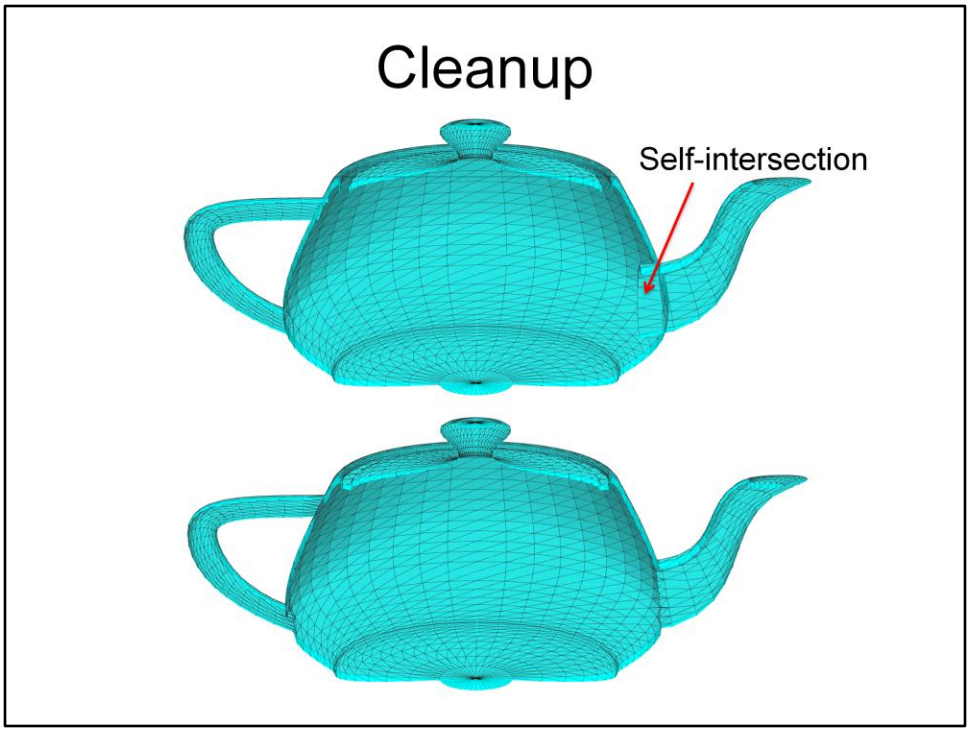
Laplacian surface editing is a smooth surface editing method that applies to arbitrary triangle meshes. By solving a large, sparse symmetric system, it can preserve or smooth surface detail by preserving or minimizing the Laplacians (left half of the equations, blue), while approximately interpolating constraints on the surface – points which the user moves manually (right half of the equations, yellow). We solve this linear least squares problem with "Cholmod": a direct sparse Cholesky solver.



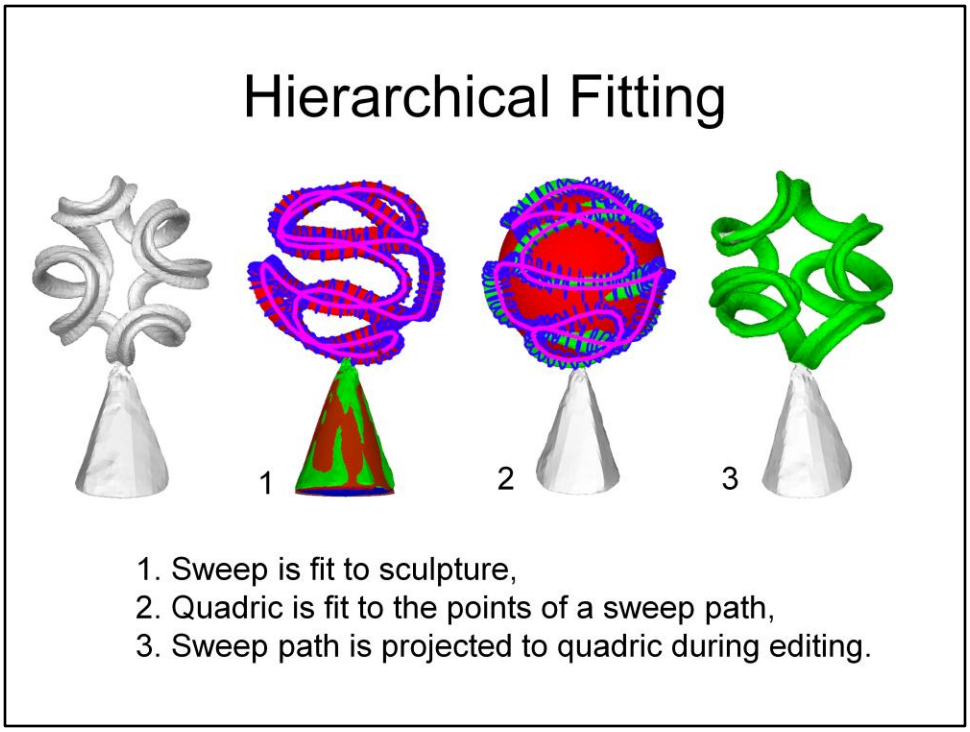
See video: freeform.wmv
Here we show a video of Laplacian-based editing. In one example, a selected portion of a model is edited as a stationary sweep, and continuity with the rest of the model is preserved automatically by Laplacian surface editing.



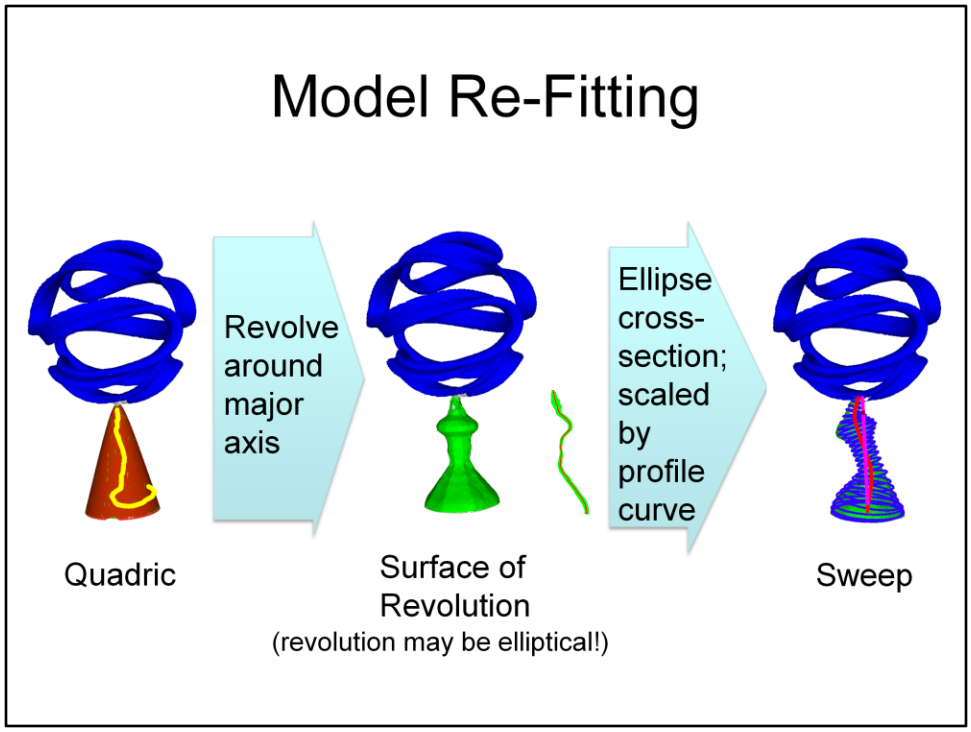
I've described our set of user guided fitting modules, and now move on to the remaining stages of our process: How we clean the model for export, and how we allow the user to impose hierarchy and change easily between differently parameterized representations.



As a user separately extracts and edits pieces of an artifact with different model types, the resulting shape description may exhibit problems such as holes in the surface, or intersecting geometry, and the surface may not blend smoothly between different parts of the shape. We've begun developing a cleanup module to address these problems, and have some promising initial results using a global, graph-cut-based approach. For example, we can fix the self intersections and holes between the separately extracted belly and spout of the Utah teapot.



In addition to fitting primitives directly to the shape, our system also permits hierarchical fitting. For example, we fit a sweep to this sculpture, and then fit a sphere (red) to the points of the sweep path (magenta). We can then enforce the spherical structure of the sweep path by projecting points on the sweep path back to the quadric surface as they are edited.



While we can of course interactively re-fit models from scratch, our system can also transform one type of fit directly into another.
For example, starting from the quadric fit on the left, we can revolve around the major or “most different” axis of the quadric to create a surface of revolution which we can the edit with arbitrary scaling of the circular cross section.
We can further convert that surface of revolution into a sweep, and reshape its straight-line path into warped figure S shape (right).

(Note that the minor axes of the quadric may have different scales, in which case we’d simply sweep an elliptical surface of revolution and have an ellipse instead of a circle for the sweep cross section.)

Future Work

We have presented a promising initial prototype of our interactive inverse 3D modeling system. We see many avenues to expand on these ideas, with better support for more sources of data, different primitive types, and more. For example, this pelican inspires two ideas:

1) A teddy-style inflation primitive, which would allow us to model structures like the pelican wings with a thin freeform surface and a varying thickness function.
2) The ability to enforce topological, structural symmetry, so that the same structure can be fit symmetrically to the pelican even though the pose is not symmetrical.

Acknowledgements

This work was supported in part by the National Science Foundation (NSF award #CMMI-1029662 (EDI))
and by Adobe Systems.

Thanks to:
The Image-based 3D Models Archive, TIcom Paris,
and to the Stanford Computer Graphics Laboratory
for some of the test data used.