# Implementation

The system has an offline processing step after which comes the interactive modeling step. The input is assumed to consist of continuous polylines, each representing a sketched curve. During the offline processing step we compute a discrete distance-transform function for each curve over a 512x512 grid. The offline processing step is performed when a sketch is first loaded and its results are saved when the user chooses the “save” option from the menu.

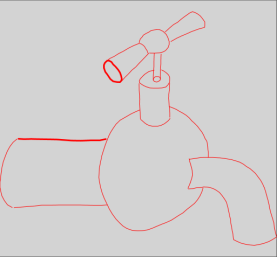


Figure 1 A sketch of a tap and distance transform images of the bold curves.

The interactive modeling process maintains a data-model that consists of sketch curves (including their associated distance-transform grids), primitives, feature curves and constraints. Primitives are parametrically defined and the fitting process computes those parameters. For example – a cylinder has center, axis vector and radius. Each primitive has associated feature curves – 3D curves that represent sharp features of the primitive. For example, a cylinder has two circles as its feature curves. Constraints are equations that constrain the same parameters that define the primitives. For example – given a cylinder and a cone, if we want their axes to be perpendicular we will constrain the inner product of those axes to be zero.



high weight

low weight

Figure 2 Sketch curves to silhouette curves matching example. The red silhouette curve has a higher value of distance transform integral than the yellow curve. Therefore it is a better match to the sketch curve that this distance transform image belongs to.

When the user drags a primitive over the sketch the system automatically assigns sketch curves to the silhouette curves of that primitive. We use the integral of the distance-transform functions to compute a matching weight, and then run the Hungarian algorithm to find the best matching. The Hungarian is an algorithm for finding maximal-weight matching in a bipartite graph. We define the graph’s vertices to be the sketch curves on one side and a primitive’s silhouette curves on the other side. This matching gives semantic meaning to the curves on the sketch. For example, a curve can be the top-circle of a cylinder, or its left silhouette.

After the matching has been computed we can fit the primitive to the assigned curves. The fitting is done by optimizing the parameters of the primitives subject to the current set of constraints. For each primitive we define an objective function and a set of intrinsic constraints. In addition, we have the external constraints that involve more than one primitive such as orthogonality or parallelism between the axis vectors of the primitives. We minimize the sum of the objective functions of all the primitives, subject to all constraints (intrinsic and external).

We perform the optimization twice. First we just fit the new primitive to the sketch using only its own intrinsic constraints. Then we attempt to infer external constraints between the newly fit primitive and the existing ones, and run the optimization again – this time with all the primitives and all the constraints. The external constraints are the ones that eventually specify the relative depth of the primitives.

## A primitive example - cylinder

As an example, we describe the objective function for one case of a Cylinder primitive. A cylinder is defined using the following parameters:

* An axis vector
* The “bottom” center point
* A length parameter
* A radius parameter

The terms “top”, “bottom”, “left” and “right” used above and from now on are relative to the direction of the axis if we imagine that points up.

A cylinder has one intrinsic constraint: and four curves – top circle, bottom circle, left silhouette and right silhouette. The following illustration depicts the parameters:

* The blue bold point at the bottom is the “Bottom center”
* The radius and length are self-explanatory.
* The unit vector from the bottom center to the top center is the axis (and not vice versa!)
* The purple line is the left silhouette
* The orange line is the right silhouette.

The bottom feature curve is the 3D circle centered at having radius and normal direction . The top feature curve is a 3D circle centered at having the radius and normal direction .

In this case all the curves of the 3D cylinder have a matching sketch curves. In this case, the system does not use the sketched silhouette lines at all and only uses the matching feature curves on the sketch.

The objective function is the average of the “projection fit” (defined below) of the top feature curve and the “projection fit” of the bottom feature curve.

Let us define the “projection fit” function for a feature curve centered at , having normal direction and radius . We assume that the feature curve is matched to a 2D sketch curve P. We uniformly sample P at points to obtain a set of 2D samples . The projection fit of the circle to the 2D curve is the average of the projection fits of all sample points.

The projection fit function for a single sample point is given by the following formula:

The above formula, which seems quire scary, was derived from the following observation:

A 2D point is a projection of a point on a 3D circle if for some Z coordinate it resides on the plane defined by the center and the normal, and its distance from the center is the radius.

The rest is just simple algebra. We also use the intrinsic constraint in order to derive the above formula. Without this constraint the formula becomes much more complicated. Remember that we don’t actually need to differentiate the above term, only express it. AutoDiff does the tedious gradient computation work for us.

## External constraints

The external constraints constrain the feature curves of the primitives. For example, in case of a cylinder, we have its two circles as the feature curves. We use the following set of constraints:

* **Parallelism**. Constrain the normal vectors of two feature curves to be parallel. Given two feature curves having normal vectors and we use the following set of equations:
* **Orthogonality**. Constrain the normal vectors of two feature curves to be orthogonal. Given two feature curves having normal vectors and we use the equation .
* **Collinear centers**. Constrain the centers of three or more feature curves to lie on the same line. Given three feature curves having centers we define and and use the same equations as in the parallelism constraint. In case we have more than three feature curves, we use the above constraint for sequential triples.
* **Cocentric**. Constrain the centers of two or more feature curves to be equal. Given two feature curve centers we use the equation . Given more than two, we use the above constraint for sequential pairs.
* **Coplanar.** Constrain two or more feature curves to lie in the same plane. Given two feature curves having centers and normal vectors we use the following two equations:  
  When we have more than two, we use sequential pairs and use the above equations for each pair.

As we said before, the system can automatically infer those constraints. For example, if two feature curves are almost orthogonal (in our implementation – angle is between 70 and 110 degrees) we constrain them to be orthogonal. This inference is done between the two optimization steps described above.

## Optimization algorithm

As our fitting problem is an optimization problem subject to equality constraints, we use an Augmented-Lagrangian method to solve the optimization problem using a sequence of unconstrained problems. Each unconstrained problem is solved using L-BFGS. Augmented Lagrangian’s main advantage over the simpler penalty method is allowing us to use simple equations for the constraints without the need of “normalizing” them.

We will use the collinarity constraint described above to explain this concept. We use the equations to say that two vectors and are parallel. If we optimize using the penalty method, we will need to add the following penalty to the objective function

If we use the above constraints “as is” for constraining three points to be collinear, the optimal solution will make the points lie closer to each other, in addition to making them collinear (as the points become closer, the vectors approach zero and the value of the penalty decreases). To avoid this problem we would need to divide the penalty by the lengths of . However this makes the objective function non-differentiable and the performance of the optimization algorithm much worse.

Switching to an Augmented Lagrangian method is what allowed the system to achieve its near real-time performance.