Geosemantic Snapping for Sketch-Based Modeling

Alex Shtof — Tel Aviv University

Alexander Agathos — West University of Timişoara

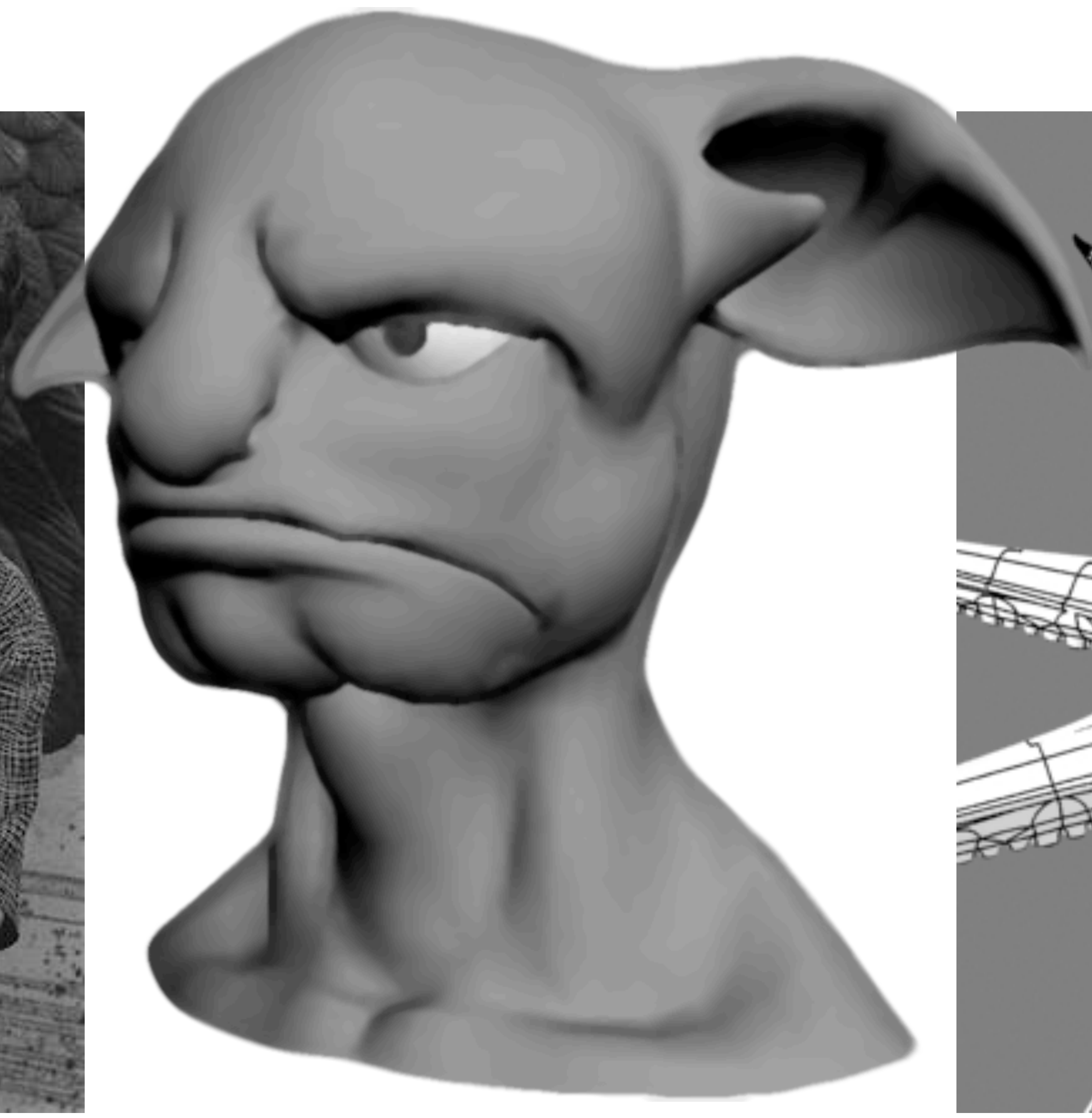
Yotam Gingold — George Mason University

Ariel Shamir — The Interdisciplinary Center, Herzliya

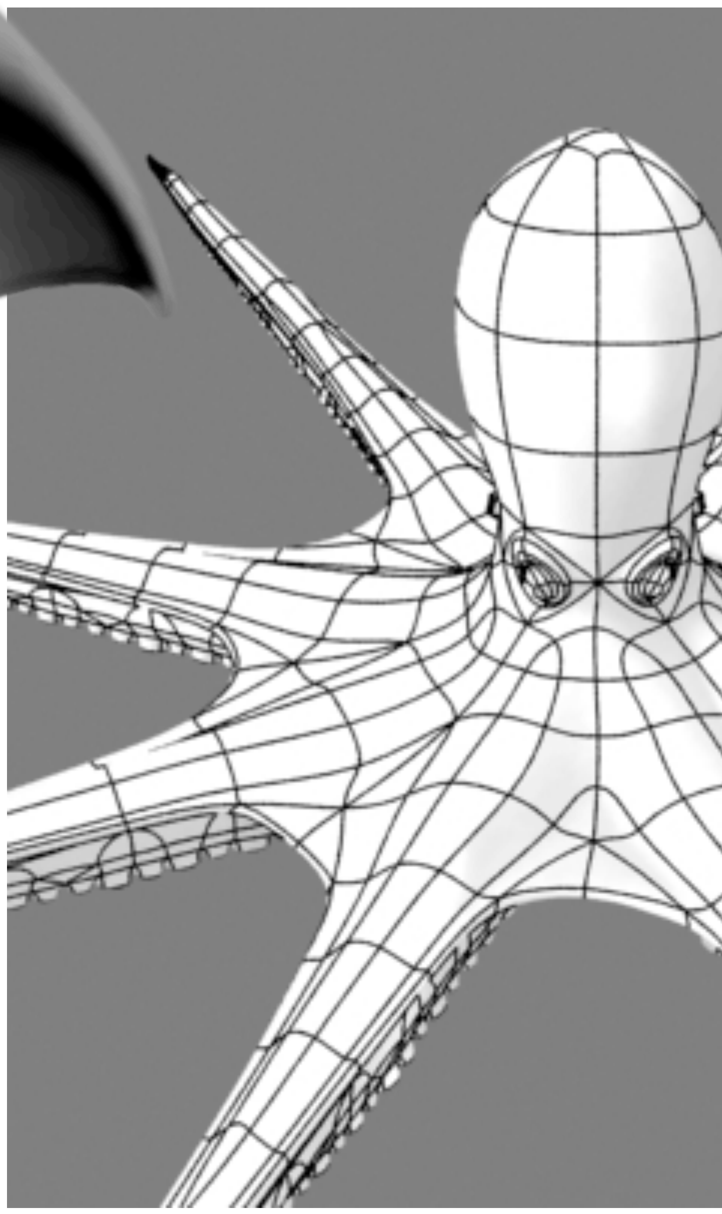
Daniel Cohen-Or — Tel Aviv University

3D Modeling

[Rip Caswell]



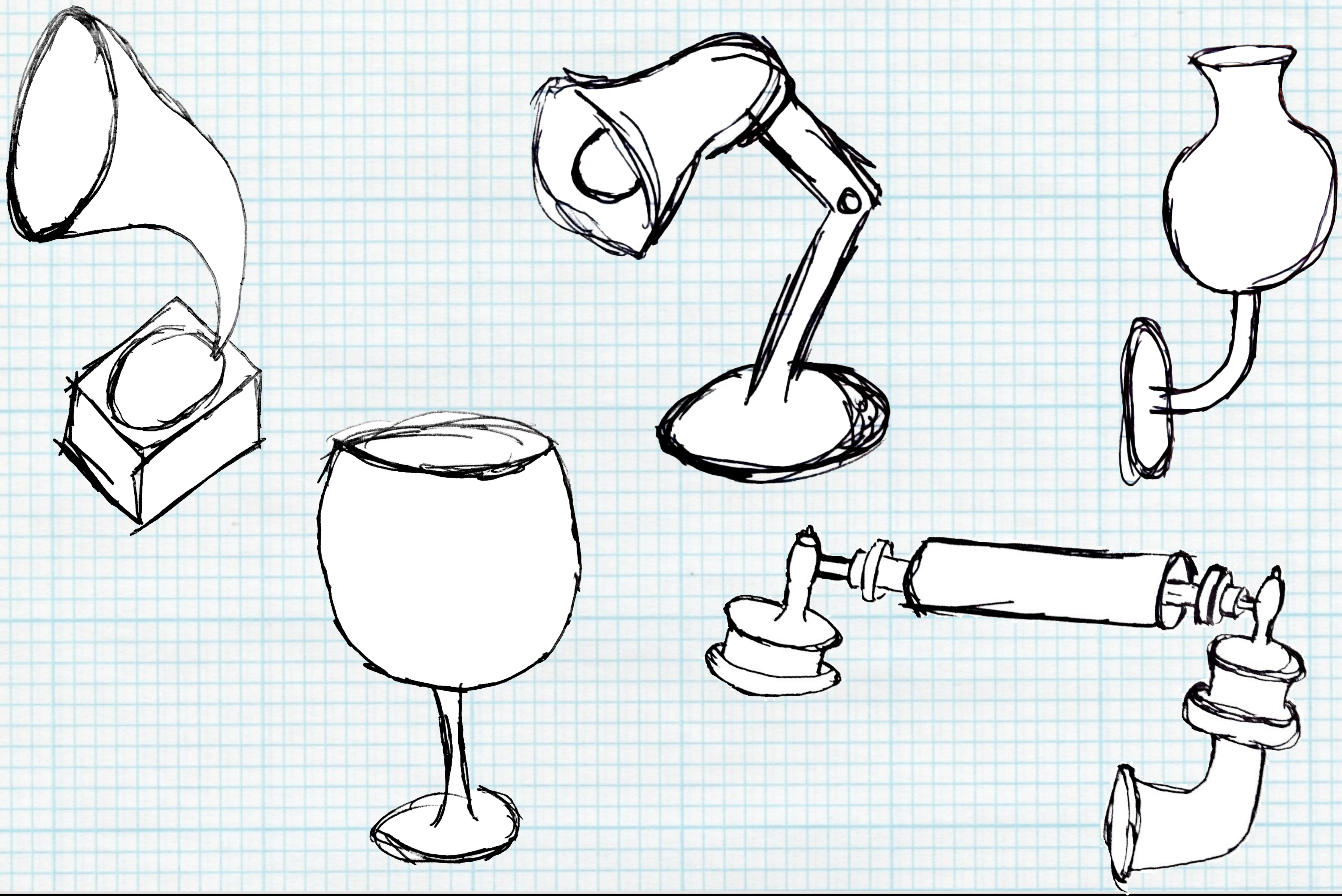
[Malanjo]



[NoneCG]

3D modeling is a challenging problem akin to digital sculpting.
Professional artists and designers create 3D models for manufacturing, movies, games, and industrial design.
Professionals use sophisticated tools such as AutoCAD and Maya, and are capable of creating amazing models.
However, these tools require a great deal of time to master, and modeling with them is a highly-skilled, time-consuming, and tedious process.

Sketches

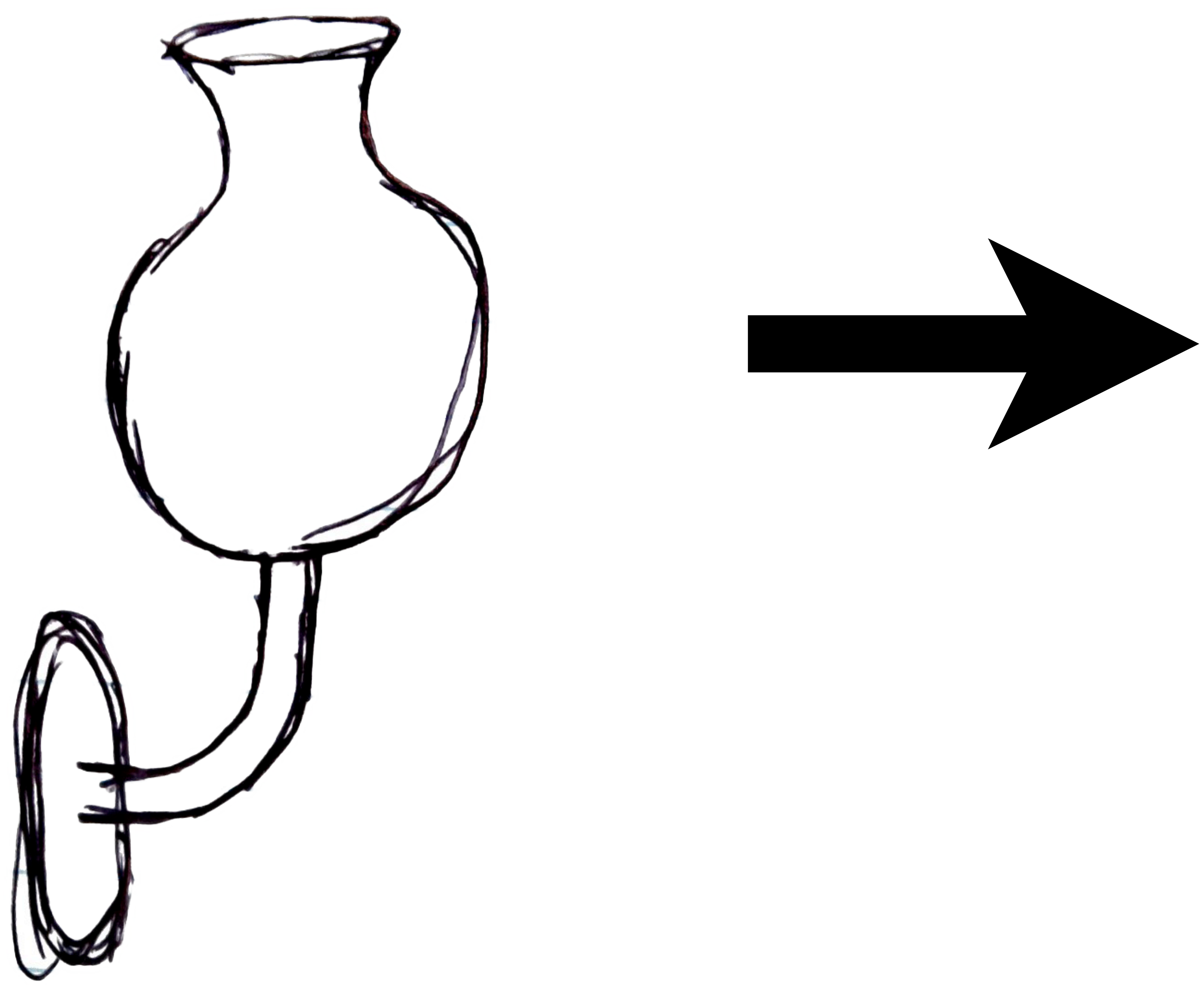


Geosemantic Snapping for Sketch-Based Modeling — Shtof, Agathos, Gingold, Shamir, Cohen-Or — Eurographics 2013

At the same time, two-dimensional sketching is far easier than 3D modeling.
Even professional 3D modelers create sketches as a “first step”.
Sketching allows artists to focus on creativity rather than technical issues in the early, exploratory stages of design.
Unfortunately, sketches are essentially “thrown away”, as there is no direct way of using them during the 3D modeling process.

For humans, sketches are easy to create and understand, whereas precise 3D modeling requires a high level of skill.
For computers, precise, repetitive tasks are easy, but interpreting sketches is challenging.

Challenges

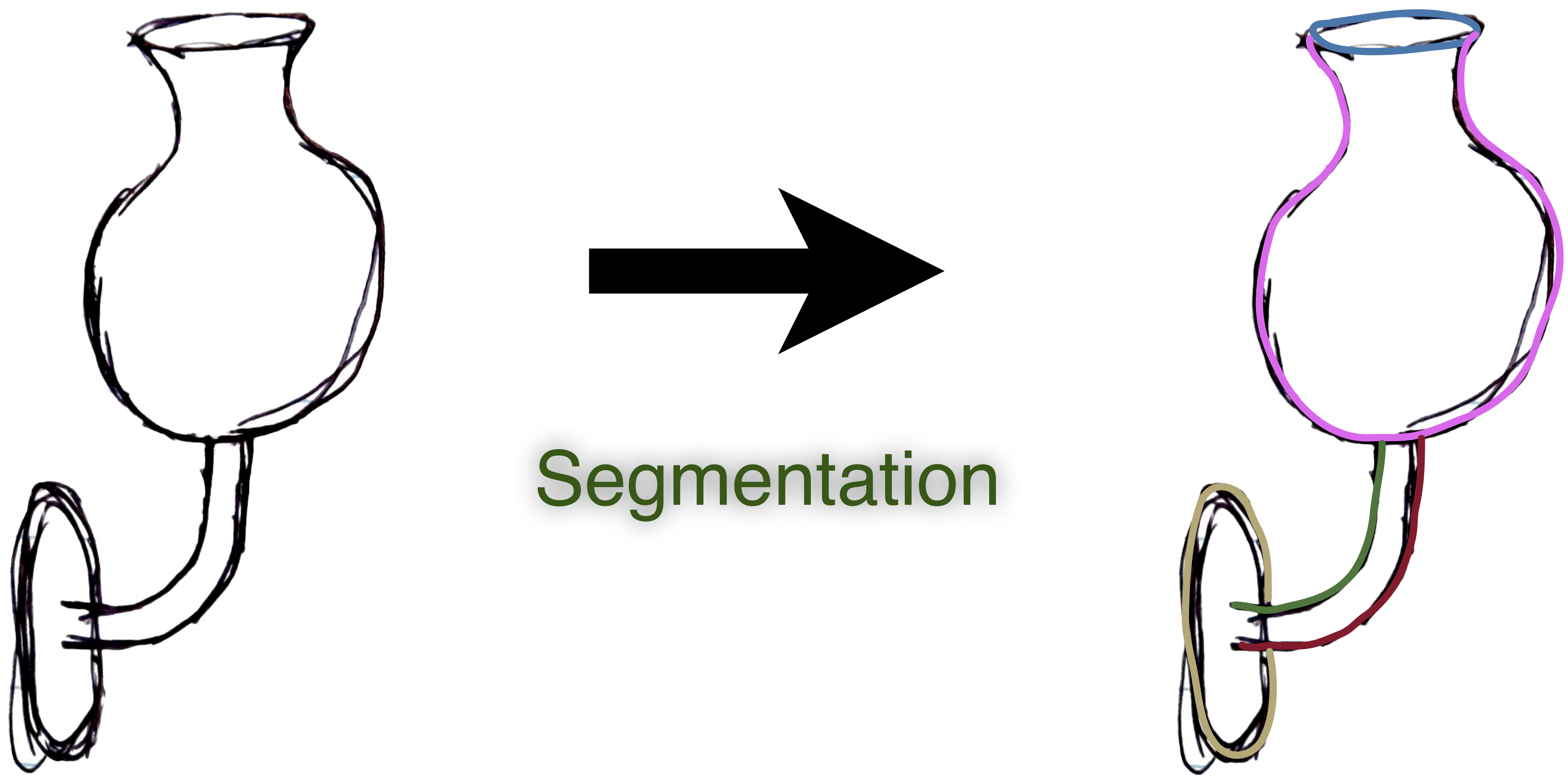


This may look easy, but it is extremely difficult. Why? There are many ambiguities. We don't know WHAT goes WHERE.

One challenge is Segmentation:
  Which pixels are connected as curves?
  What curves make up the parts?
Another challenge is Recognition:
  What kinds of parts are in the sketch?
  How do shapes fit the curves?
The final challenge is Positioning:
  How are shapes connected to each other?

Because of the segmentation, recognition, and positioning ambiguities, an automatic solution...

Challenges



This may look easy, but it is extremely difficult. Why? There are many ambiguities. We don't know WHAT goes WHERE.

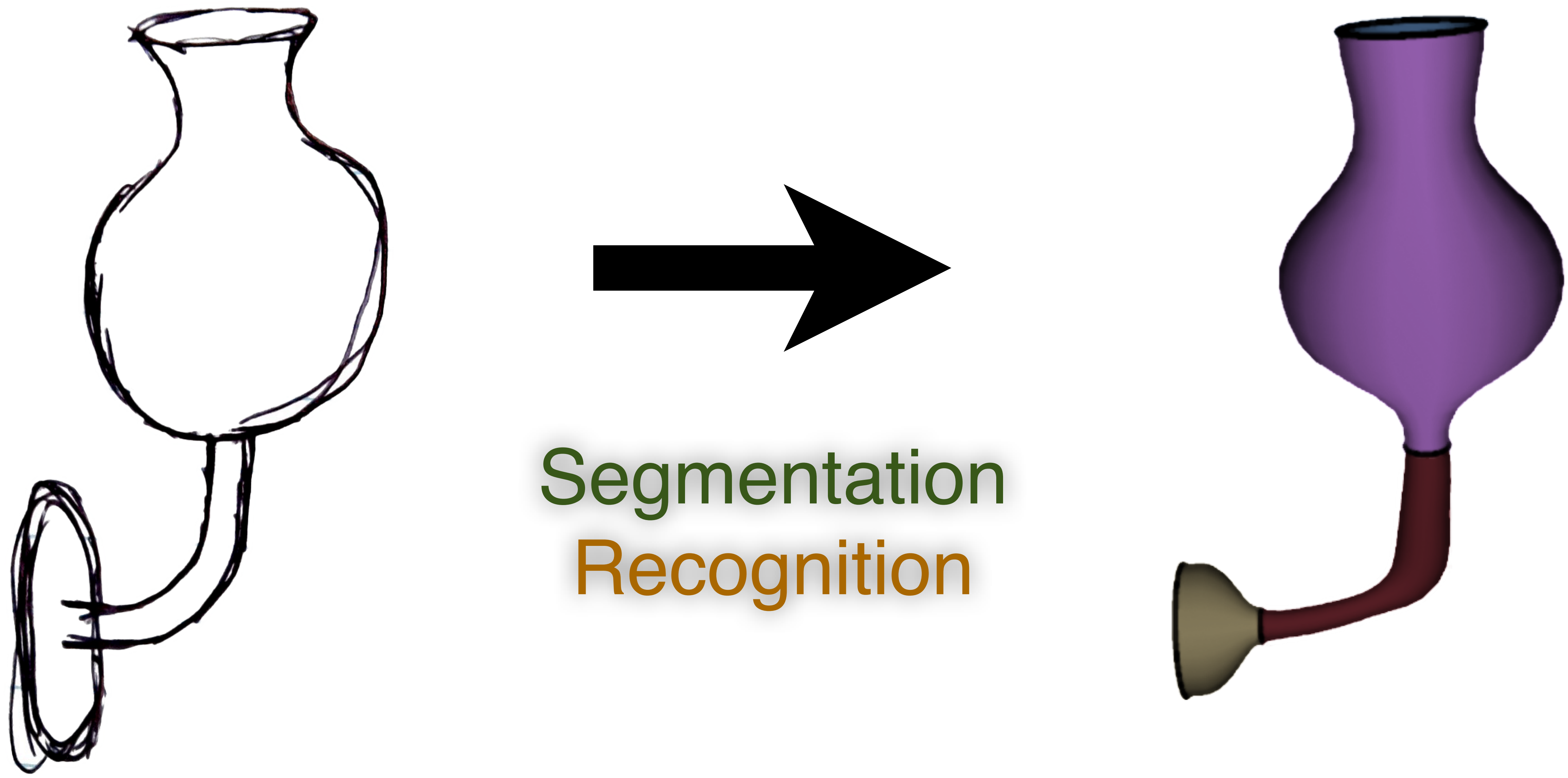
One challenge is Segmentation:
- Which pixels are connected as curves?
- What curves make up the parts?

Another challenge is Recognition:
- What kinds of parts are in the sketch?
- How do shapes fit the curves?

The final challenge is Positioning:
- How are shapes connected to each other?

Because of the segmentation, recognition, and positioning ambiguities, an automatic solution...

Challenges



This may look easy, but it is extremely difficult. Why? There are many ambiguities. We don't know WHAT goes WHERE.

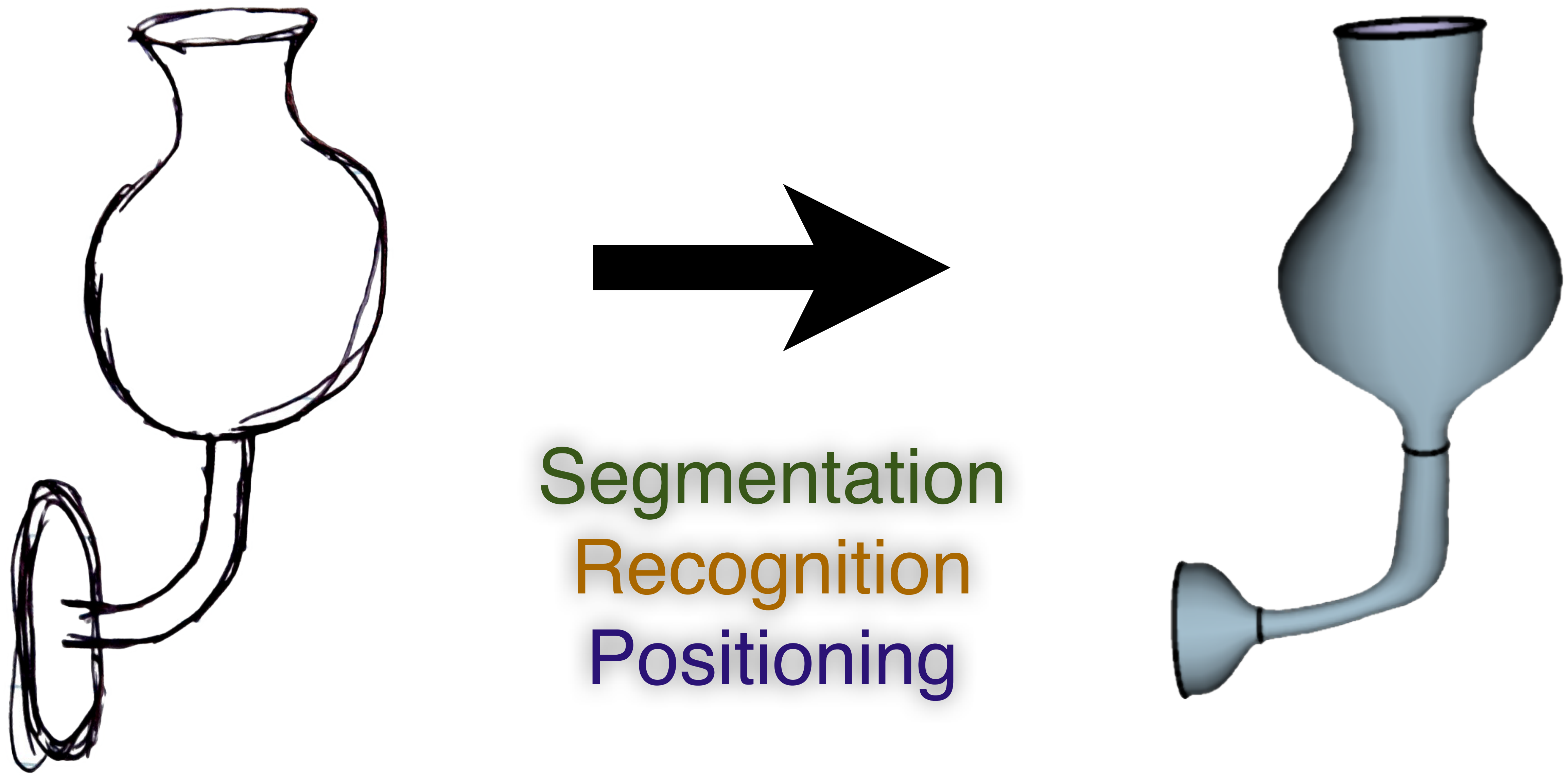
One challenge is Segmentation:
- Which pixels are connected as curves?
- What curves make up the parts?

Another challenge is Recognition:
- What kinds of parts are in the sketch?
- How do shapes fit the curves?

The final challenge is Positioning:
- How are shapes connected to each other?

Because of the segmentation, recognition, and positioning ambiguities, an automatic solution...

Challenges



This may look easy, but it is extremely difficult. Why? There are many ambiguities. We don't know WHAT goes WHERE.

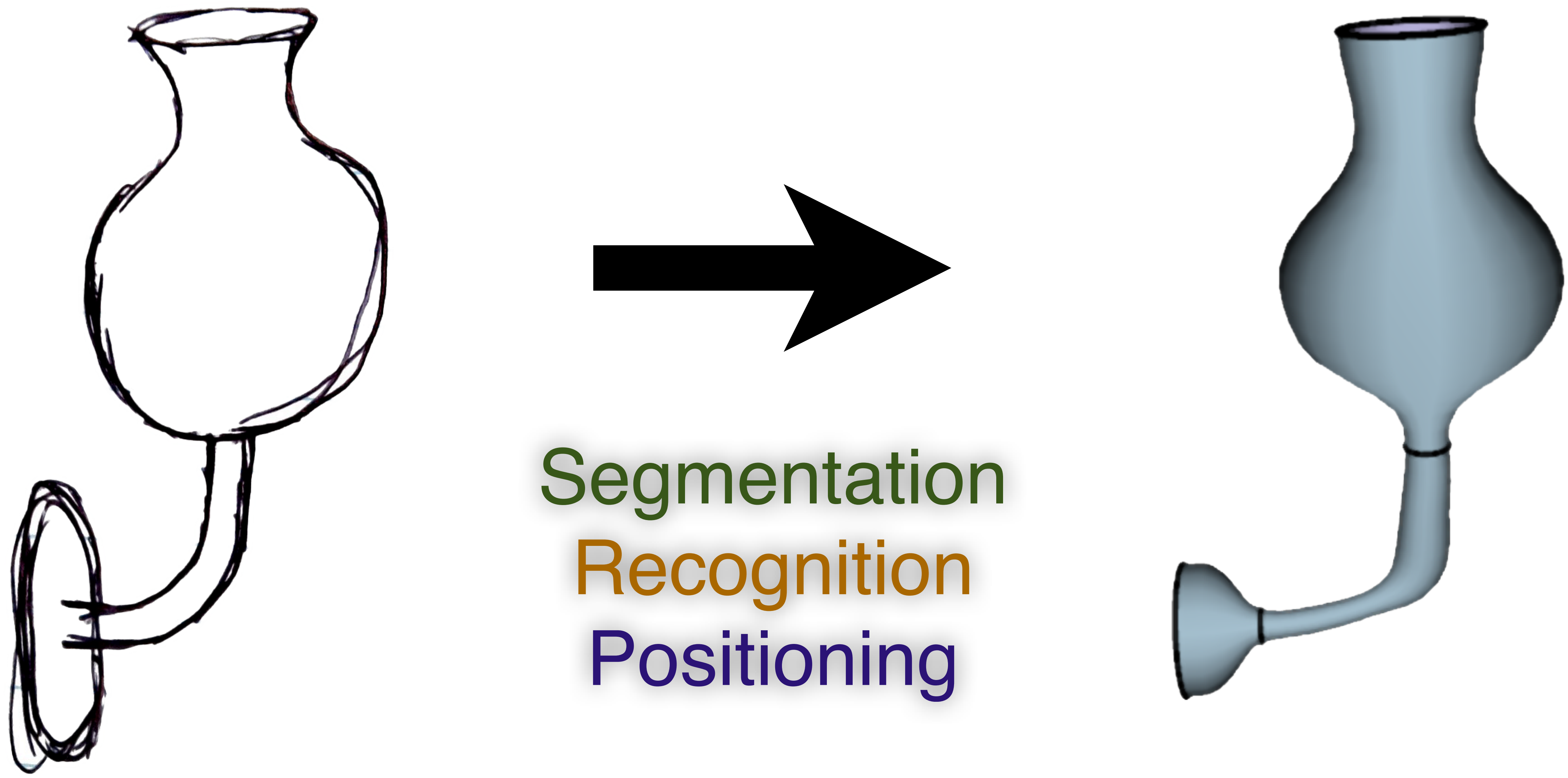
One challenge is Segmentation:
- Which pixels are connected as curves?
- What curves make up the parts?

Another challenge is Recognition:
- What kinds of parts are in the sketch?
- How do shapes fit the curves?

The final challenge is Positioning:
- How are shapes connected to each other?

Because of the segmentation, recognition, and positioning ambiguities, an automatic solution...

Challenges



An automatic solution entails solving a **complex, non-convex** optimization problem with **many local minima**.

This may look easy, but it is extremely difficult. Why? There are many ambiguities. We don't know WHAT goes WHERE.

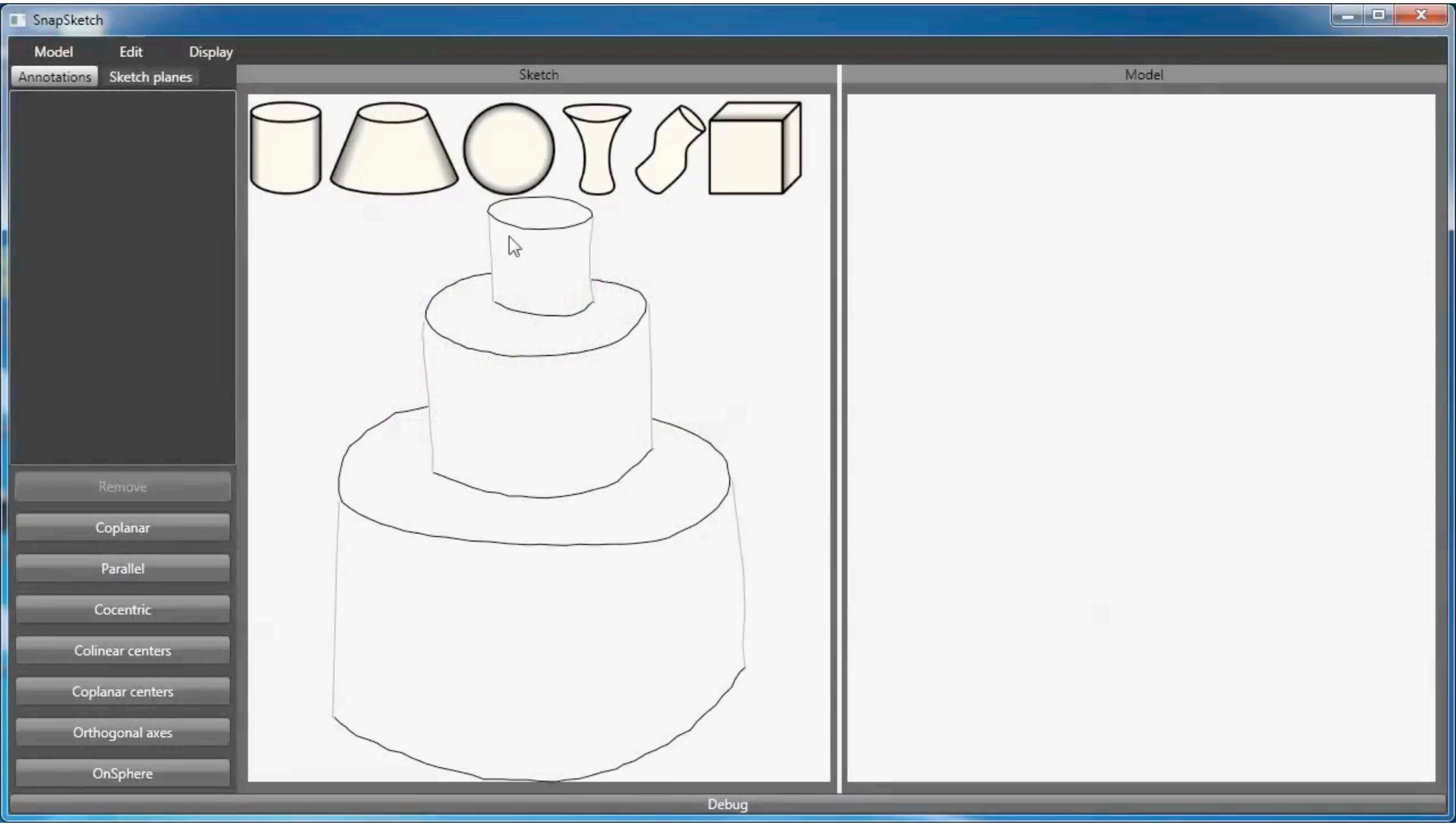
One challenge is Segmentation:
- Which pixels are connected as curves?
- What curves make up the parts?

Another challenge is Recognition:
- What kinds of parts are in the sketch?
- How do shapes fit the curves?

The final challenge is Positioning:
- How are shapes connected to each other?

Because of the segmentation, recognition, and positioning ambiguities, an automatic solution...

Interactive Approach



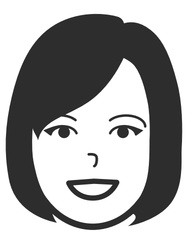
Geosemantic Snapping for Sketch-Based Modeling — Shtof, Agathos, Gingold, Shamir, Cohen-Or — Eurographics 2013

We solve the problem with an interactive approach, which is basically **drag-and-drop**. The user, by dragging an appropriate primitive over the sketch, simultaneously solves the segmentation, recognition, and positioning problem.

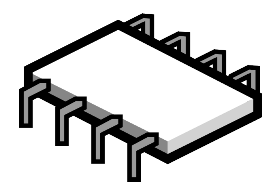
The computer performs meticulous and precise placement:
Fits primitives via optimization, given the initial position.
Infers geosemantic relationships that attach and orient primitives.

Our Solution

Separate the problem into **semantic** and **geometric** tasks.



semantic: interpreting the sketch's individual strokes and parts



geometric: fitting and reconstructing precise geometry

Our solution separates the problem into semantic and geometric tasks.

The semantic part entails interpreting the sketch's
individual strokes
and parts
This is easy for humans and hard for computers.

The geometric part involves fitting and reconstructing precise geometry.
This is easy for computers and difficult for humans.

Related Work

Image-Based Modeling

- [Gingold et al. 2009]
- [Lau et al. 2010]
- [Tsang et al. 2004]
- [Xu et al. 2011]
- [Debevec et al. 1996]
- [Sinha et al. 2008]
- [van den Hengel 2007]

Primitives and Constraints

- [Sutherland 1963]
- [Igarashi 1998]
- [Li et al. 2011]
- [Zeleznik et al. 1996]
- [Pereira et al. 2003]

Sketch-Based Modeling

- [Igarashi et al. 1999]
- [Pugh 1991]
- [Eggli et al. 1997]
- [Bae et al. 2008]
- [Schmidt et al. 2009]
- [Lipson and Shpitalni 1996]
- [Shesh and Chen 2004]
- [Chen et al. 2008]
- [Ouyang and David 2011]
- [LaViola and Zeleznik 2004]

Before we get into the details, I want to briefly situate our work in the context of others'.

Related Work

Image-Based Modeling

[Sinha et al. 2008]

Primitives and Constraints

- [Sutherland 1963]
- [Igarashi 1998]
- [Li et al. 2011]
- [Zeleznik et al. 1996]
- [Pereira et al. 2003]

Sketch-Based Modeling

- [Igarashi et al. 1999]
- [Pugh 1991]
- [Eggli et al. 1997]
- [Bae et al. 2008]
- [Schmidt et al. 2009]
- [Lipson and Shpitalni 1996]
- [Shesh and Chen 2004]
- [Chen et al. 2008]
- [Ouyang and David 2011]
- [LaViola and Zeleznik 2004]

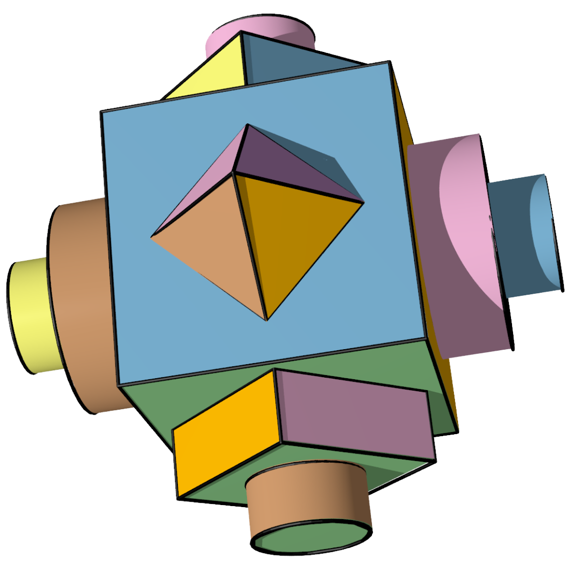
Unlike similarly motivated modeling-from-sketch or image work, in our approach the user is only concerned with high-level, semantic operations.

Related Work

Image-Based Modeling

[Sinha et al. 2008]

Primitives and Constraints



[Li et al. 2011]

Sketch-Based Modeling

- [Igarashi et al. 1999]
- [Pugh 1991]
- [Eggli et al. 1997]
- [Bae et al. 2008]
- [Schmidt et al. 2009]
- [Lipson and Shpitalni 1996]
- [Shesh and Chen 2004]
- [Chen et al. 2008]
- [Ouyang and David 2011]
- [LaViola and Zeleznik 2004]

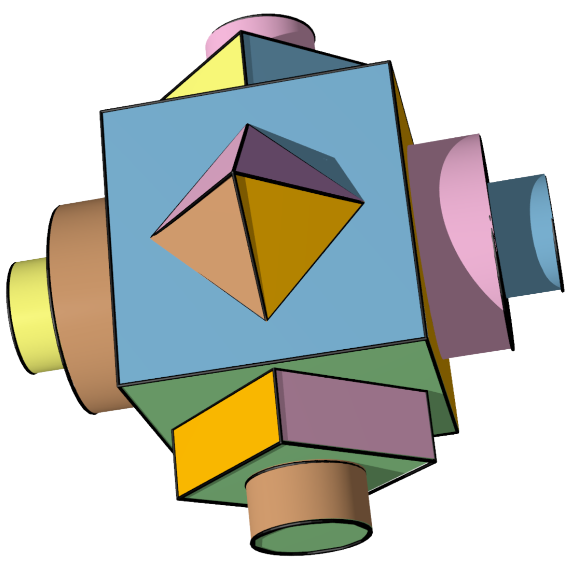
Constraints have been part of CAD systems since the beginning; our system is similar to approaches which automatically infer constraints.

Related Work

Image-Based Modeling

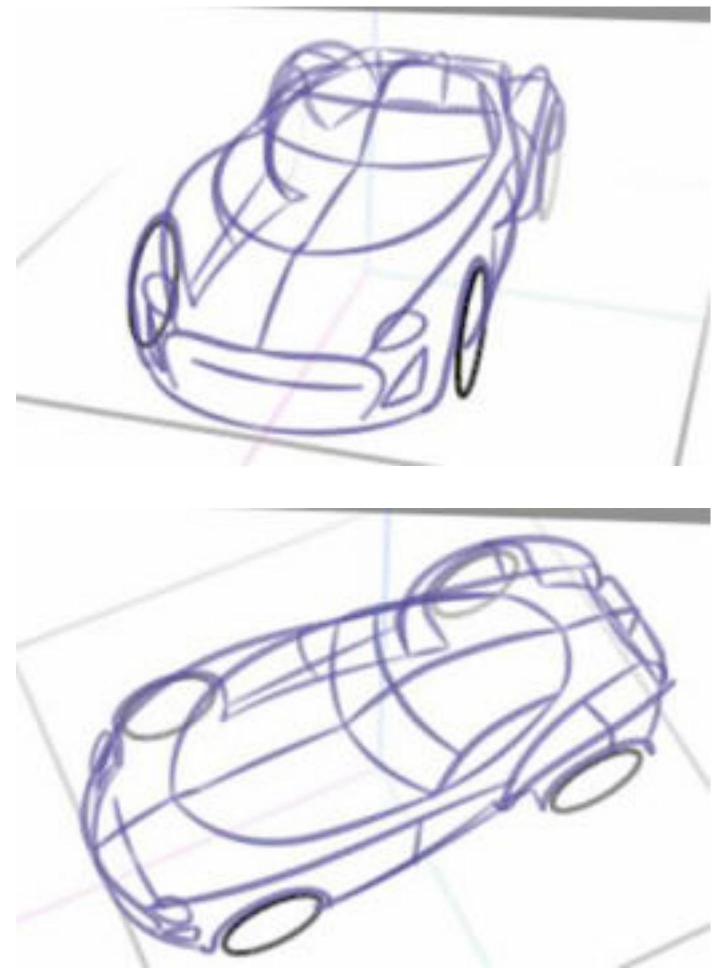
[Sinha et al. 2008]

Primitives and Constraints



[Li et al. 2011]

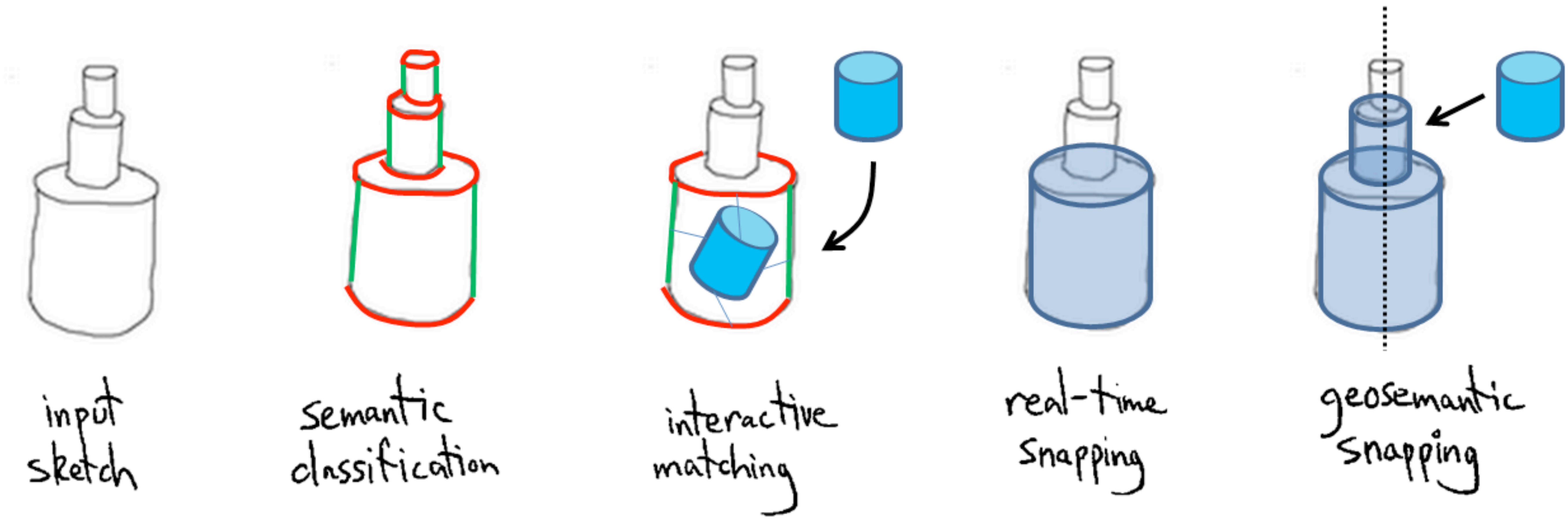
Sketch-Based Modeling



[Bae et al. 2009]

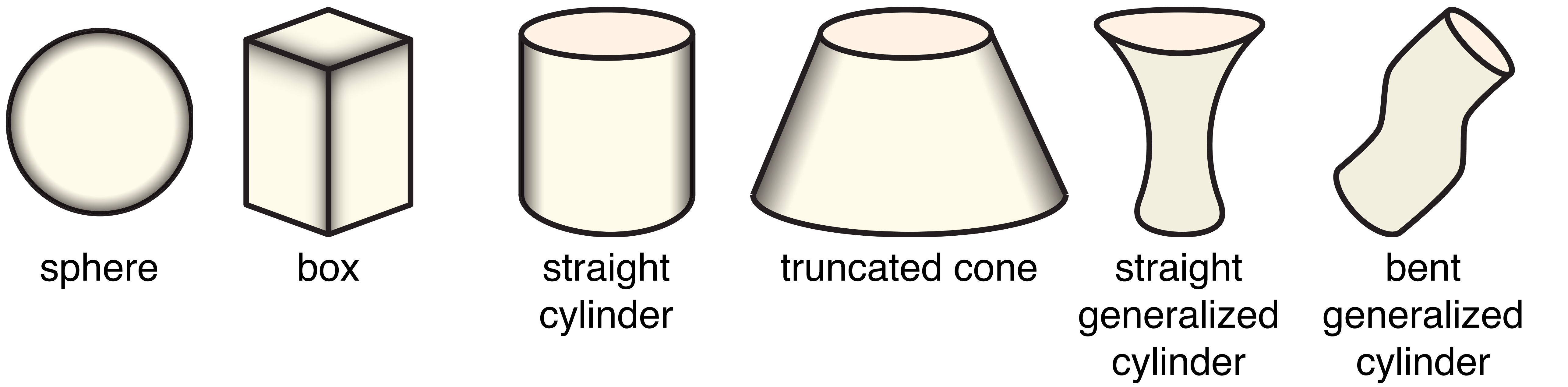
Unlike many sketch-based modeling approaches, our approach is for modeling from sketches, not modeling by sketching.

Overview



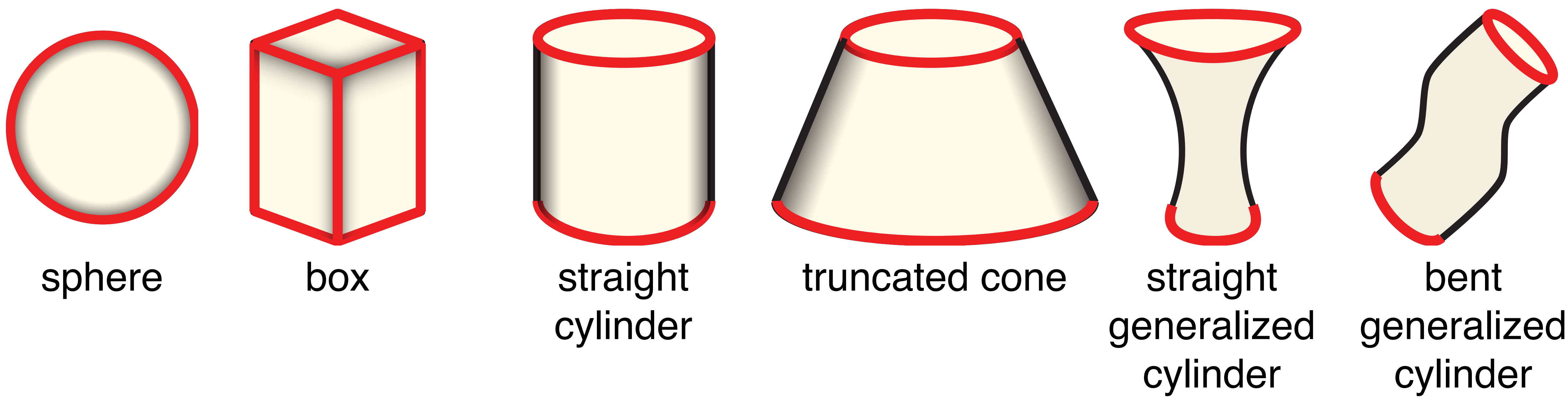
Here is an overview of our modeling process.
1. Given a vectorized input sketch,
2. The human user classifies curves in the sketch as “feature” or “silhouette”. This step is semi-automated, easy for humans, and hard for computers.
3. The user chooses appropriate shape primitives and drags them over the sketch; in real-time, the computer identifies appropriate sketch curves and
4. performs real-time snapping to fit the primitive to the curves.
5. After a primitive is dropped, the computer infers geosemantic constraints between primitives such as parallelism and coplanarity.

Primitives



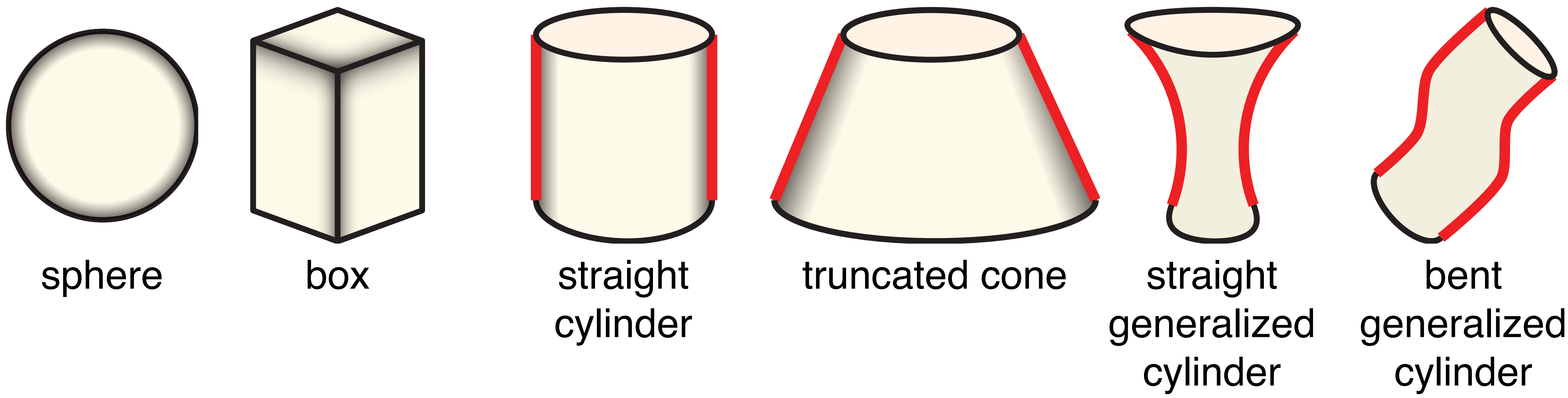
Our primitives are spheres, cuboids, and various generalizations of cylinders: cylinders with varying radii and curved spines.

Primitives: Feature Curves



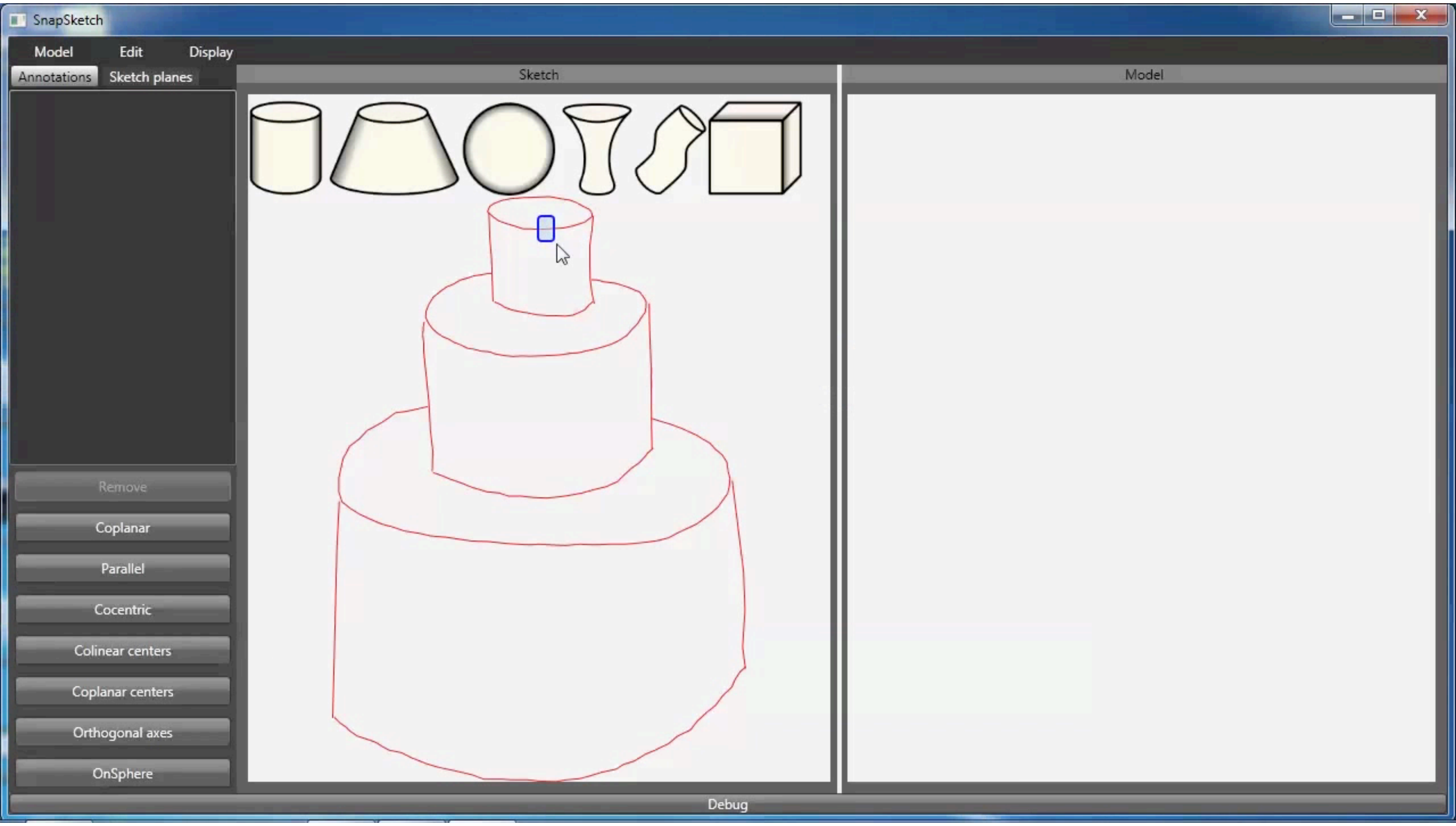
We call the non-view-dependent curves of a primitive “feature curves”. <click> These are the top and bottom circles of a cylinder, the outline of a sphere, and the edges of a box.

Primitives: Silhouette Curves



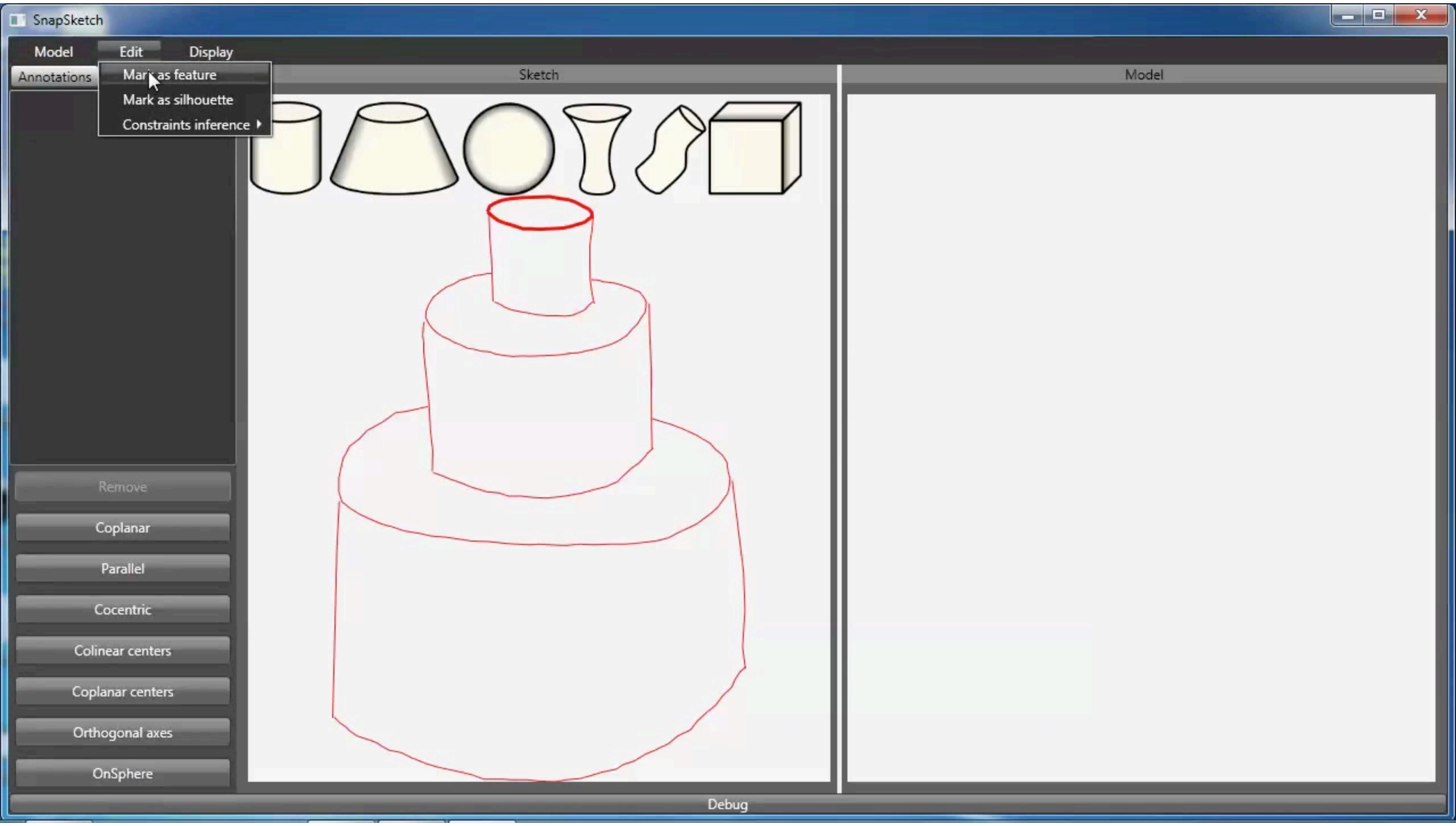
The “silhouette curves” are the others: <click> view-dependent curves corresponding to the visible boundaries of each primitive.

Tagging



To aid the real-time snapping algorithm, which I will describe shortly,
the user classifies sketch curves as either "feature" or "silhouette". This is a simple, semi-automated process.
In the example shown here, <click>
tagging just one curve <click> correctly classifies all curves. <click> The semi-automation is based on the observation that sketch curves likely belonging to a cylinder alternate between silhouette and feature, while sketch curves belonging to a box or sphere should always be classified as "feature" curves.

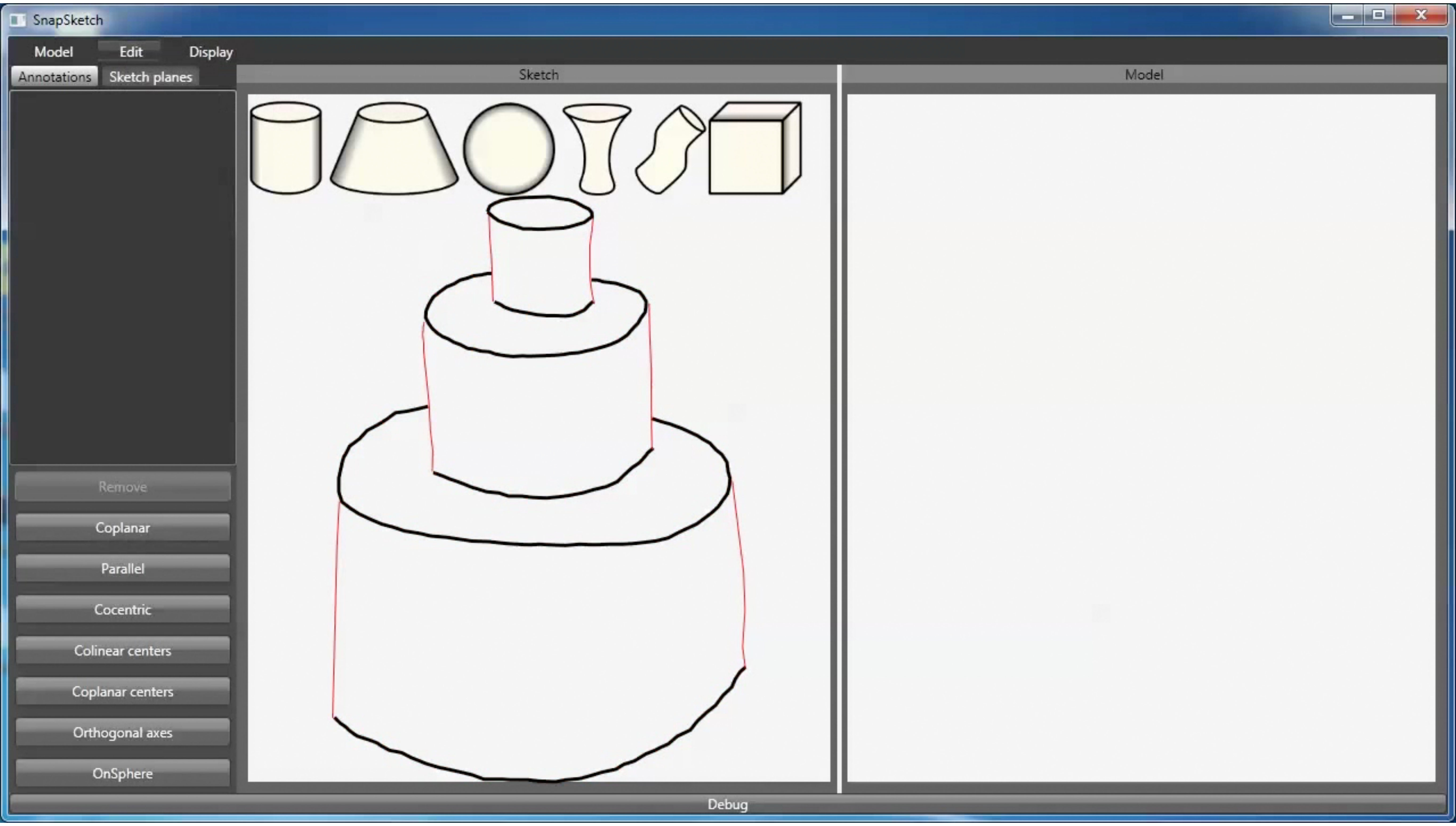
Tagging



Geosemantic Snapping for Sketch-Based Modeling — Shtof, Agathos, Gingold, Shamir, Cohen-Or — Eurographics 2013

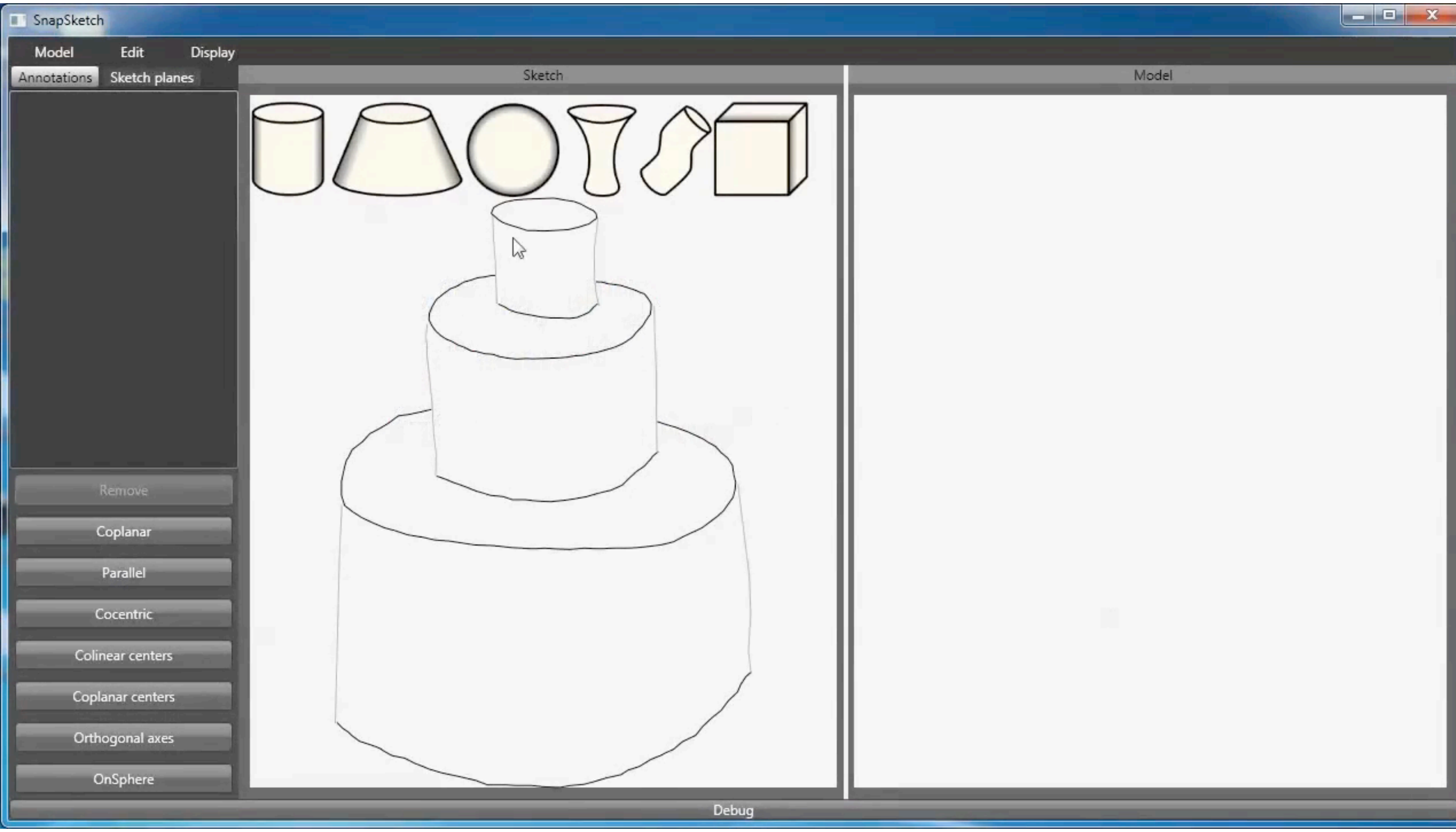
To aid the real-time snapping algorithm, which I will describe shortly,
the user classifies sketch curves as either “feature” or “silhouette”. This is a simple, semi-automated process.
In the example shown here, <click>
tagging just one curve <click> correctly classifies all curves. <click> The semi-automation is based on the observation that sketch curves likely belonging to a cylinder alternate between silhouette and feature, while sketch curves belonging to a box or sphere should always be classified as “feature” curves.

Tagging



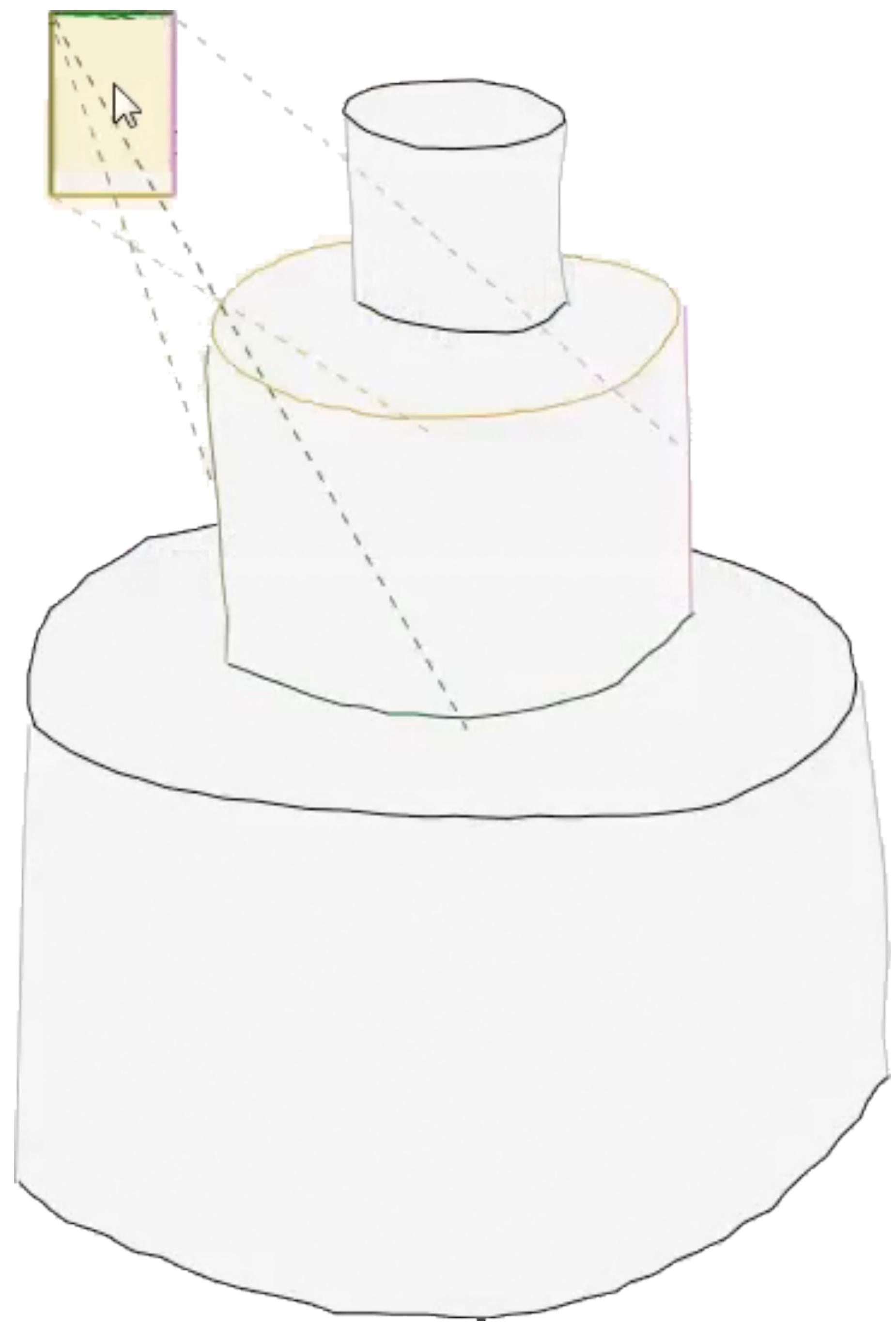
To aid the real-time snapping algorithm, which I will describe shortly,
the user classifies sketch curves as either "feature" or "silhouette". This is a simple, semi-automated process.
In the example shown here, <click>
tagging just one curve <click> correctly classifies all curves. <click> The semi-automation is based on the observation that sketch curves likely belonging to a cylinder alternate between silhouette and feature, while sketch curves belonging to a box or sphere should always be classified as "feature" curves.

Drag-and-Drop



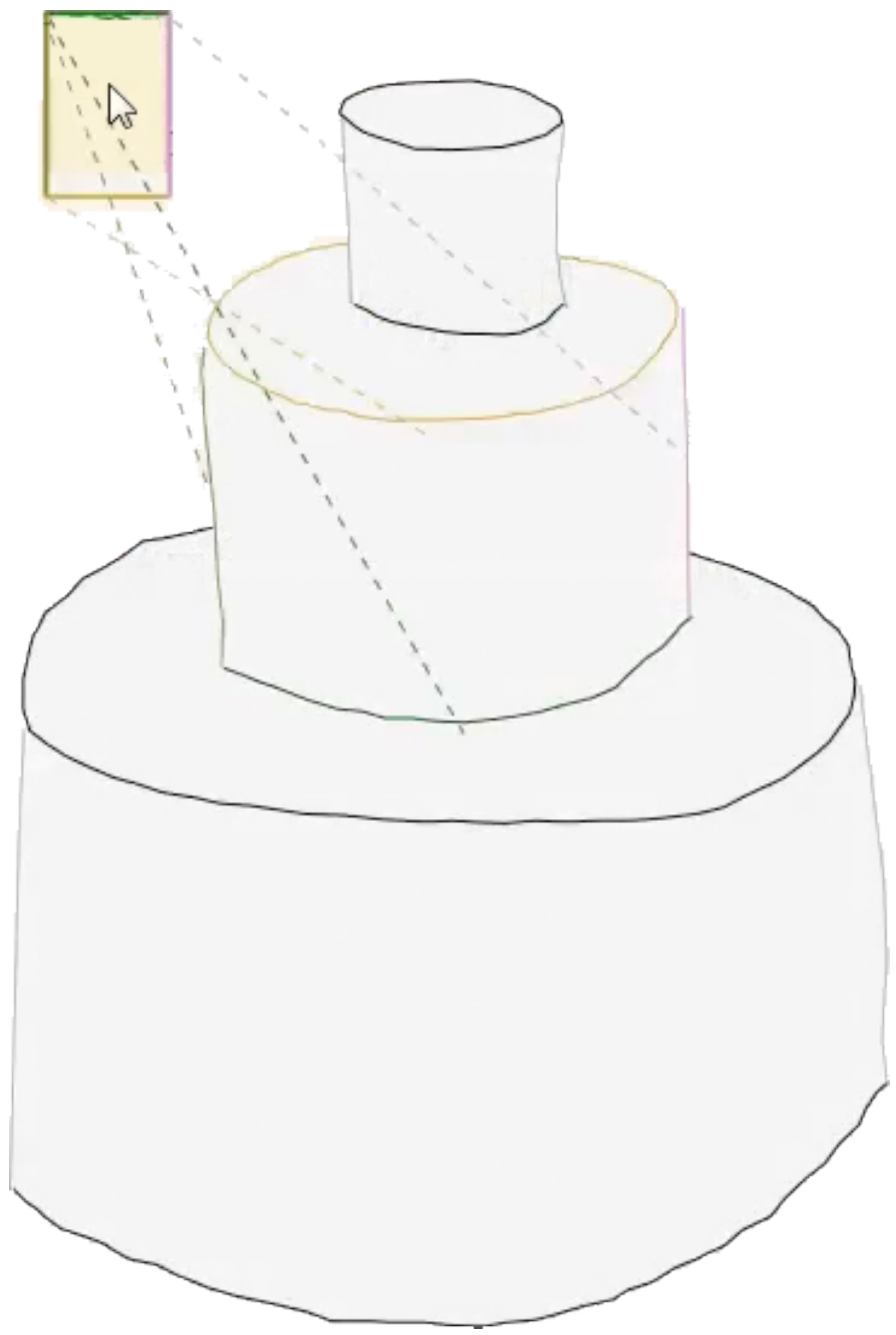
Once a sketch has been tagged, the user models by dragging and dropping primitives over it.
Note that a dragged primitive can also be rotated and scaled by the user.
Using this example, I will describe the stages of a drag-and-drop.

Anatomy of a Drag: Curve Matching



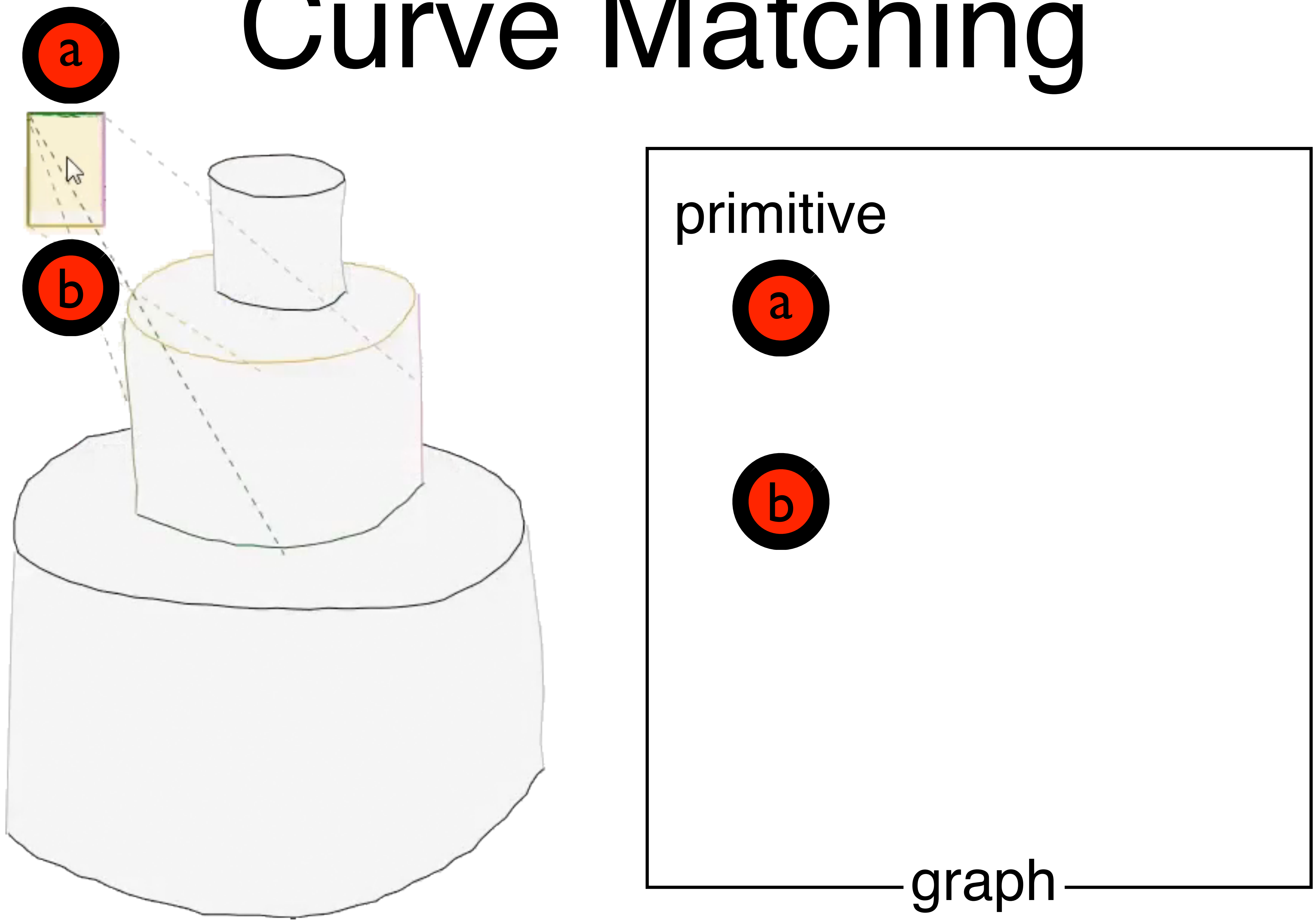
The first step of our real-time snapping algorithm is matching the feature and silhouette curves of the primitive with the feature and silhouette curves of the sketch.
This is performed by solving a bipartite graph matching problem.

Anatomy of a Drag:
Curve Matching



On one side of the bipartite graph we have the feature curves of the primitive (“a” and “b”) <click>,
and on the other side <click> the feature curves of the sketch.

Anatomy of a Drag: Curve Matching

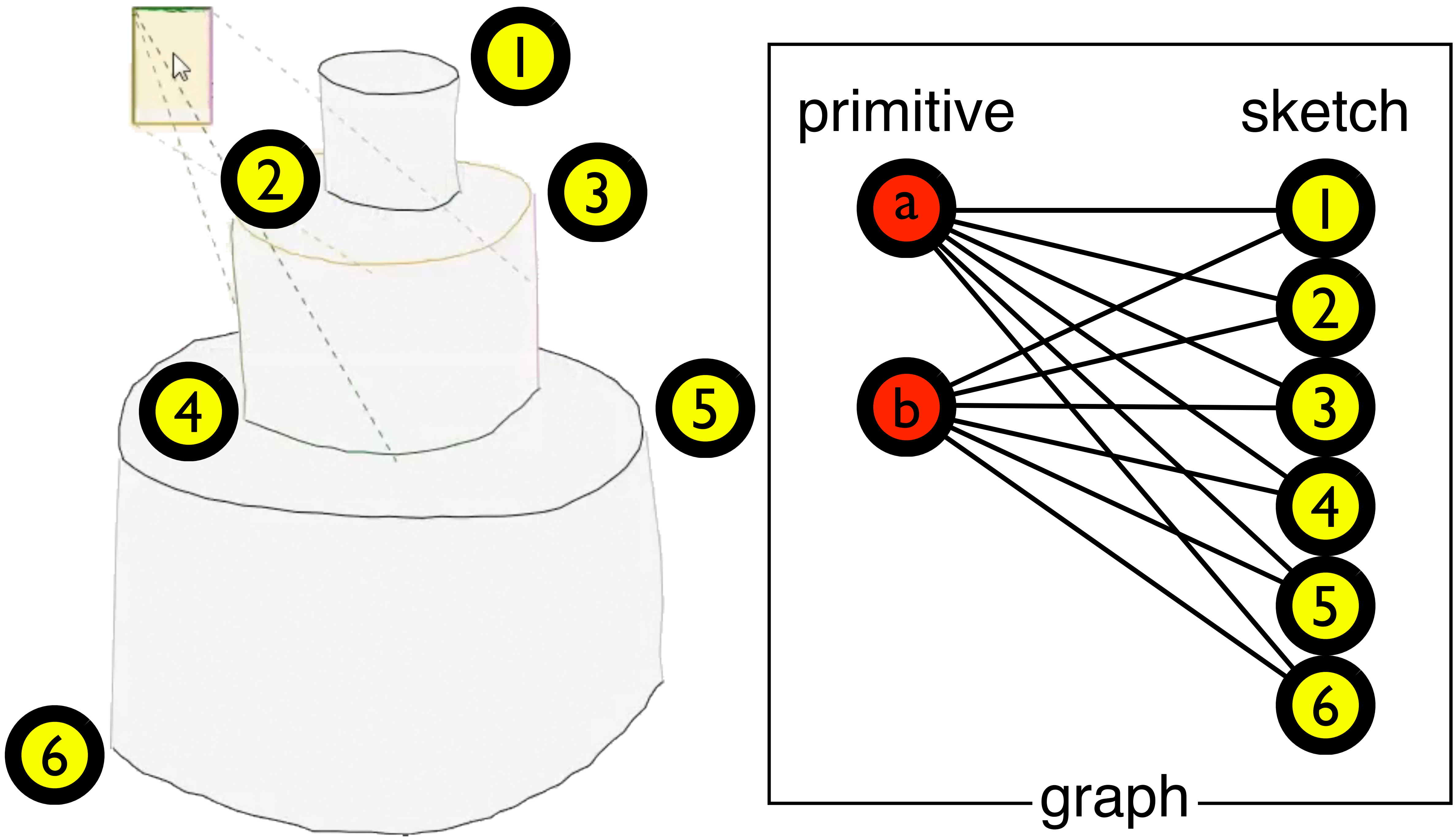


On one side of the bipartite graph we have the feature curves of the primitive (“a” and “b”) <click>,
and on the other side <click> the feature curves of the sketch.

<click>
<click>

The weight on edge (a,1)

Anatomy of a Drag: Curve Matching

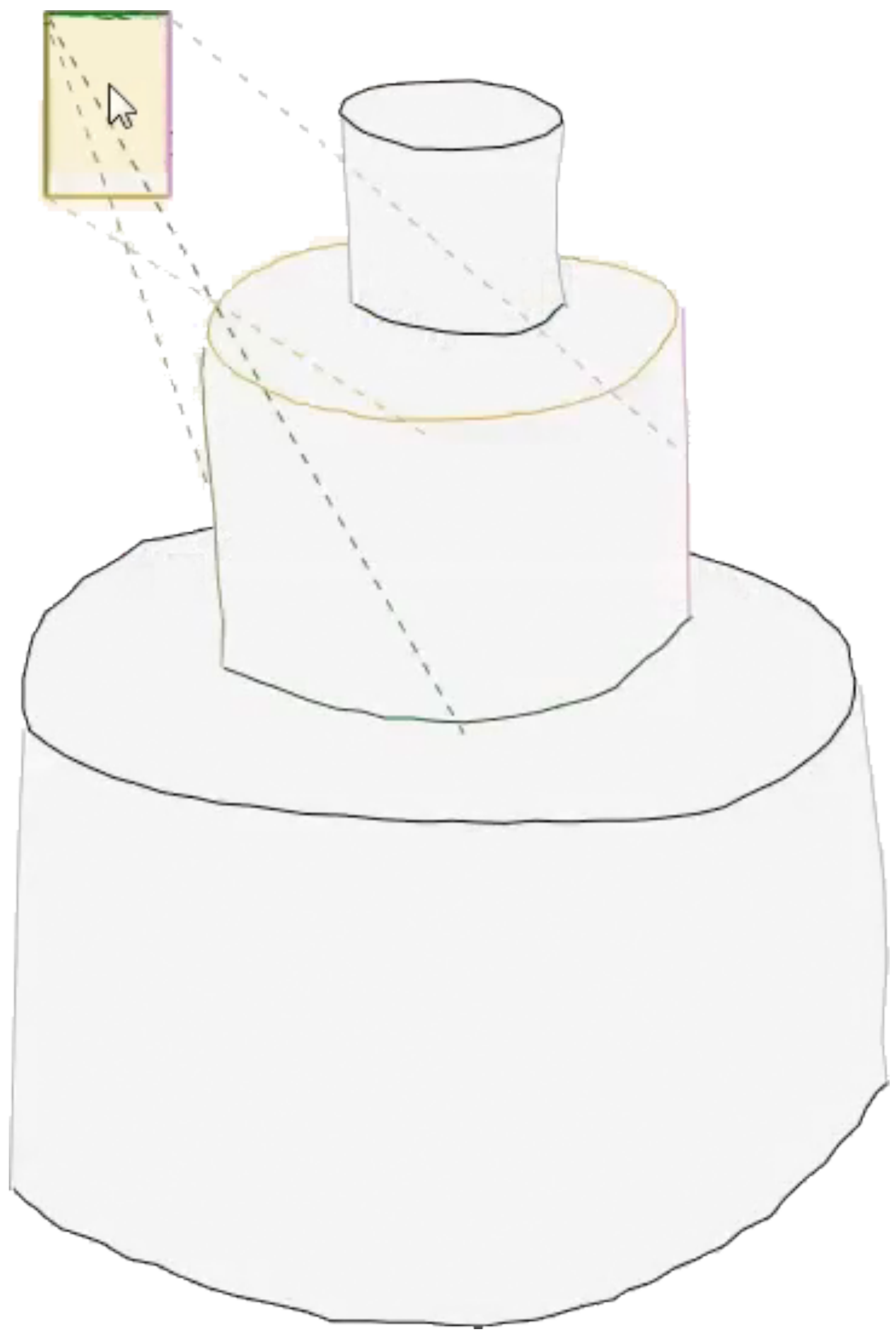


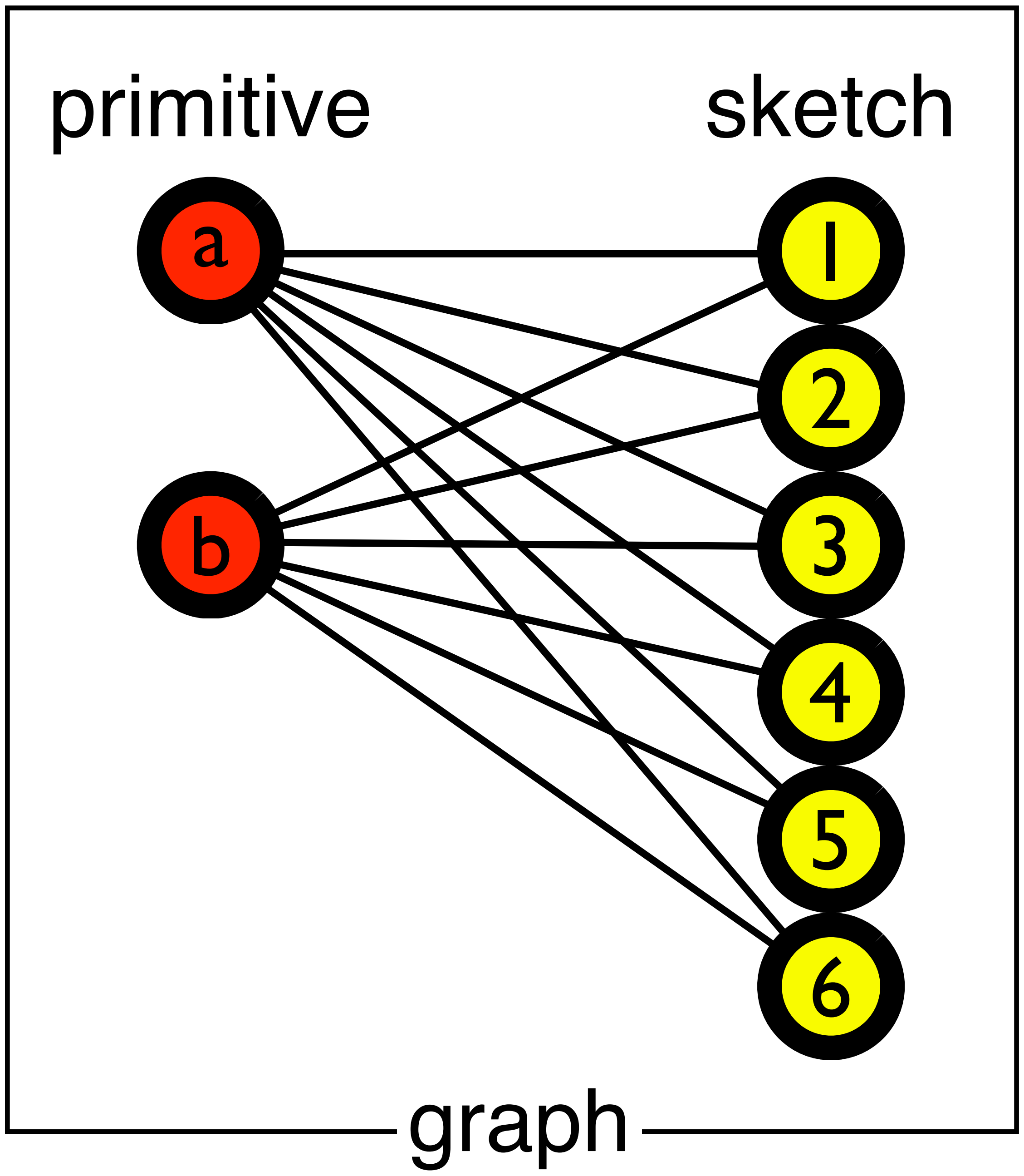
On one side of the bipartite graph we have the feature curves of the primitive (“a” and “b”) <click>,
and on the other side <click> the feature curves of the sketch.

<click>
<click>

The weight on edge (a,1)

Anatomy of a Drag: Curve Matching



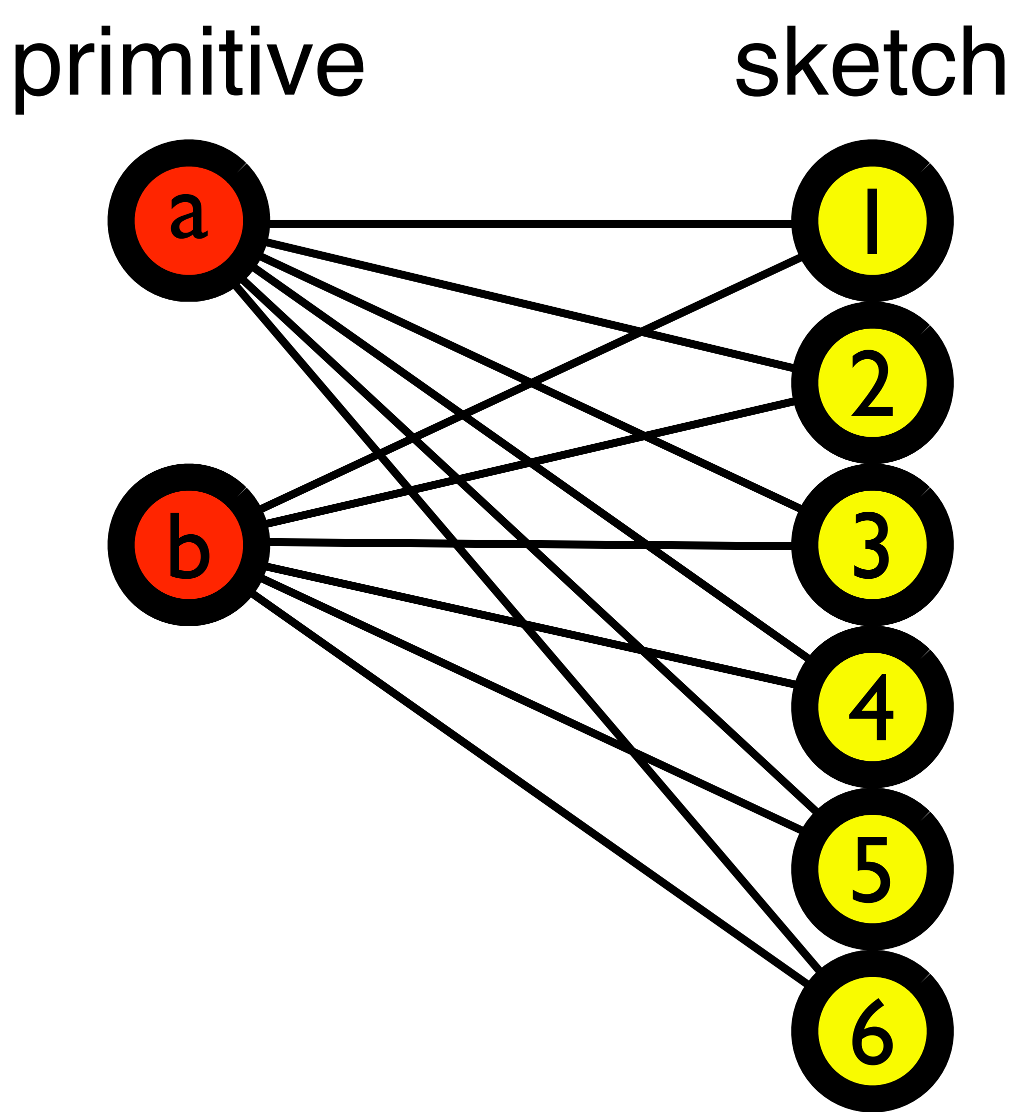


On one side of the bipartite graph we have the feature curves of the primitive (“a” and “b”) <click>,
and on the other side <click> the feature curves of the sketch.

<click>
<click>

The weight on edge (a,1)

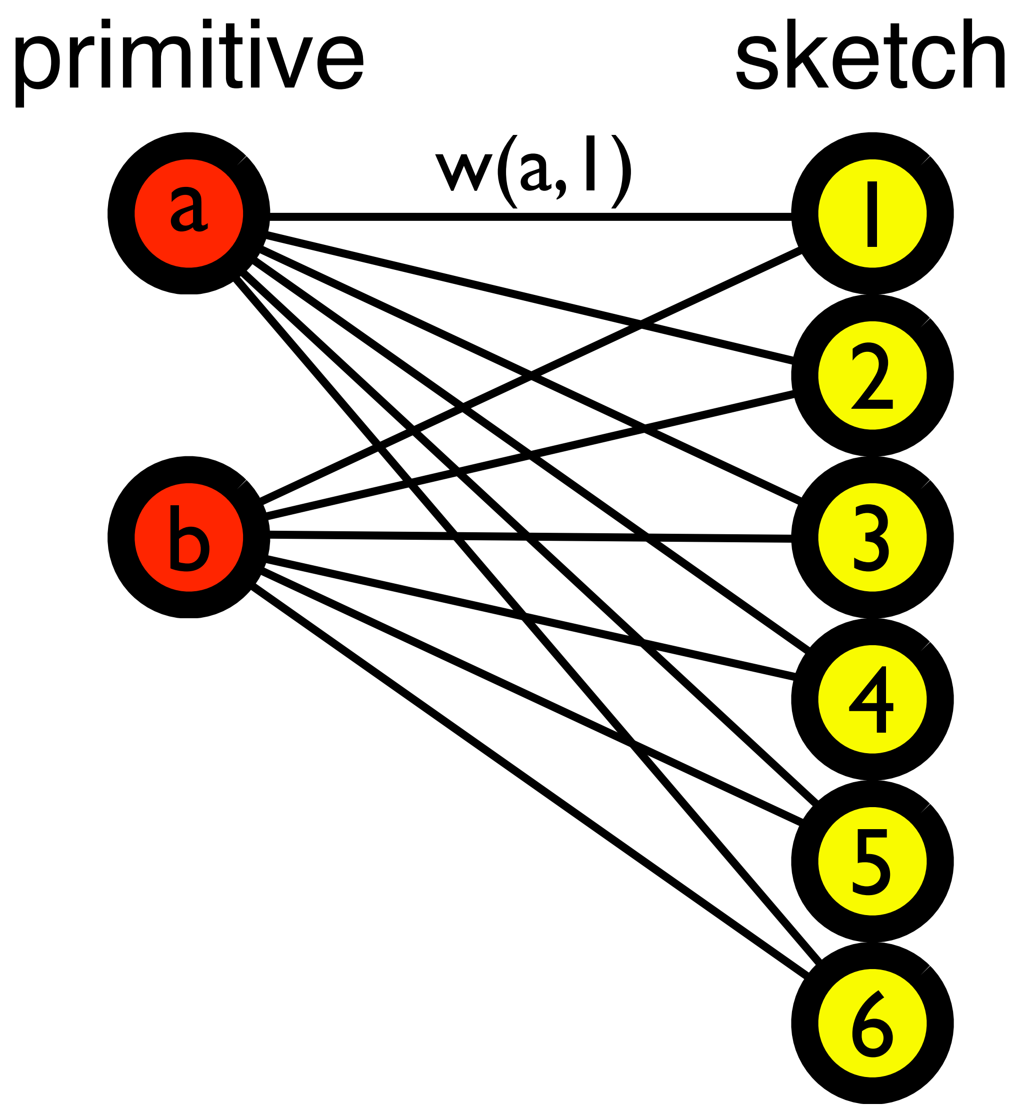
Anatomy of a Drag: Curve Matching



<click>
<click>

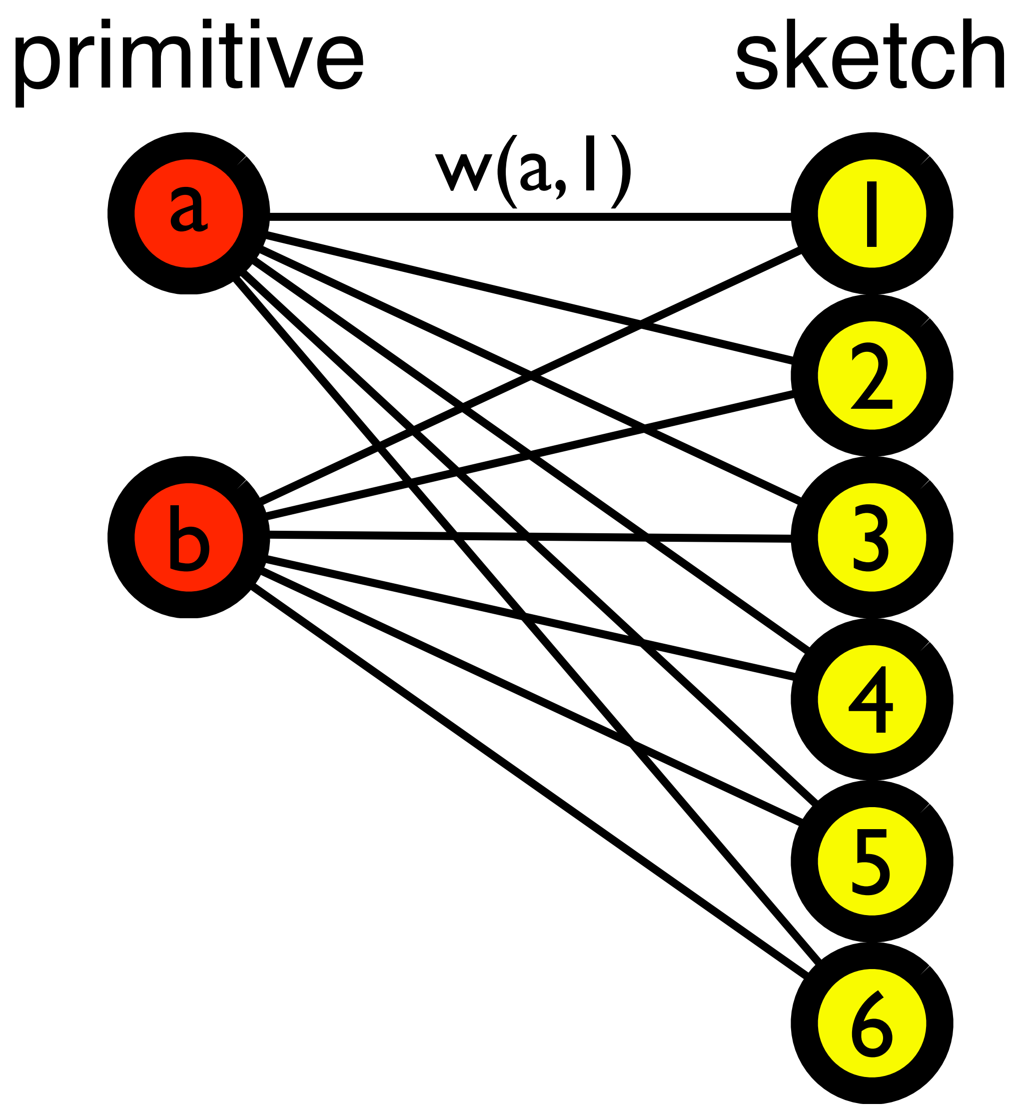
The weight on edge (a,1)

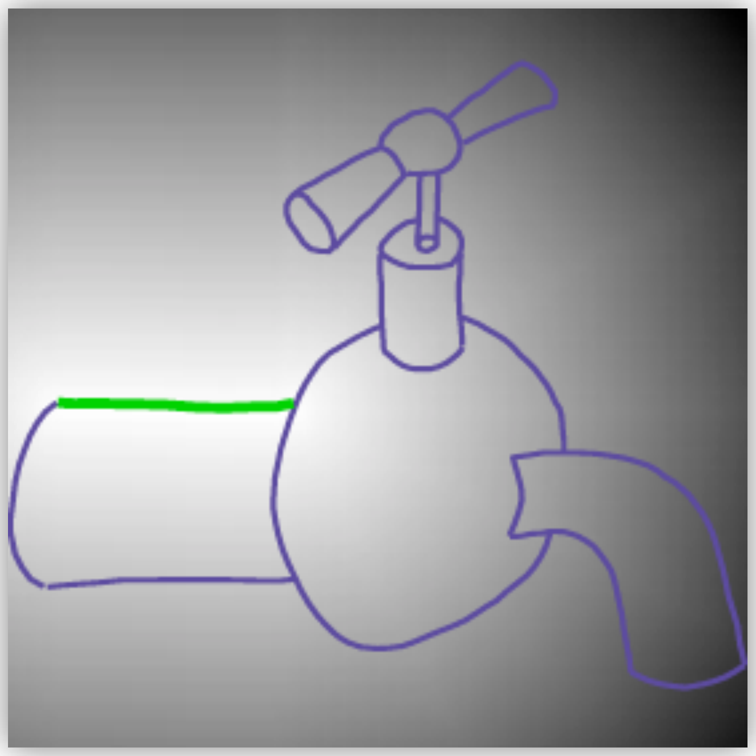
Anatomy of a Drag: Curve Matching



The weight on edge (a,1) is the integral of the per-pixel distances from the primitive curve “a” to the sketch curve “1”. This can be computed quickly by <click> precomputing the distance transform for each curve in the sketch. (This example shows the distance transform for the green curve.)
Finally, we perform additional filtering on the bipartite match to prevent improbable matches; see the paper for details.

Anatomy of a Drag: Curve Matching

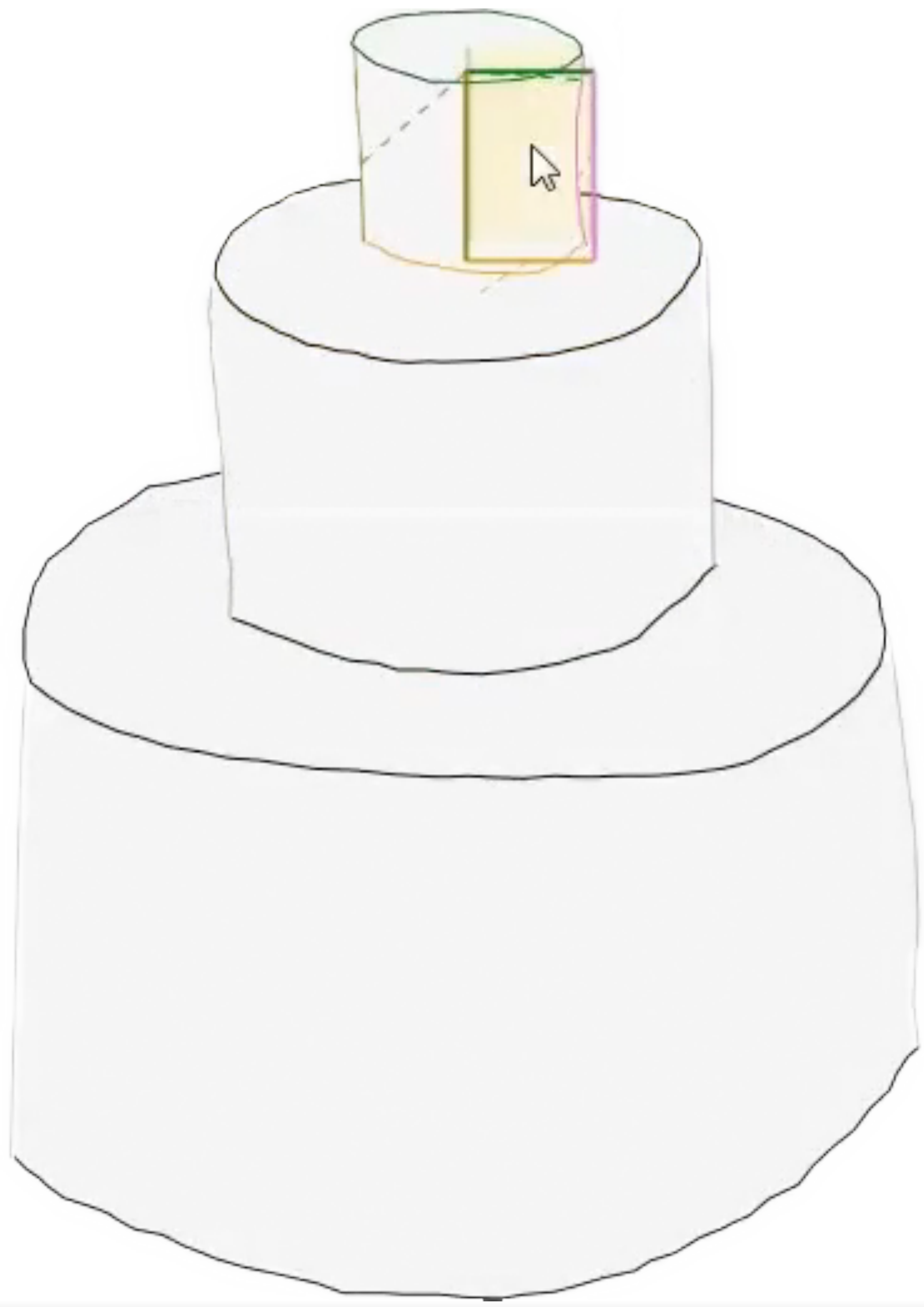




The weight on edge (a,1) is the integral of the per-pixel distances from the primitive curve “a” to the sketch curve “1”. This can be computed quickly by <click> precomputing the distance transform for each curve in the sketch. (This example shows the distance transform for the green curve.)

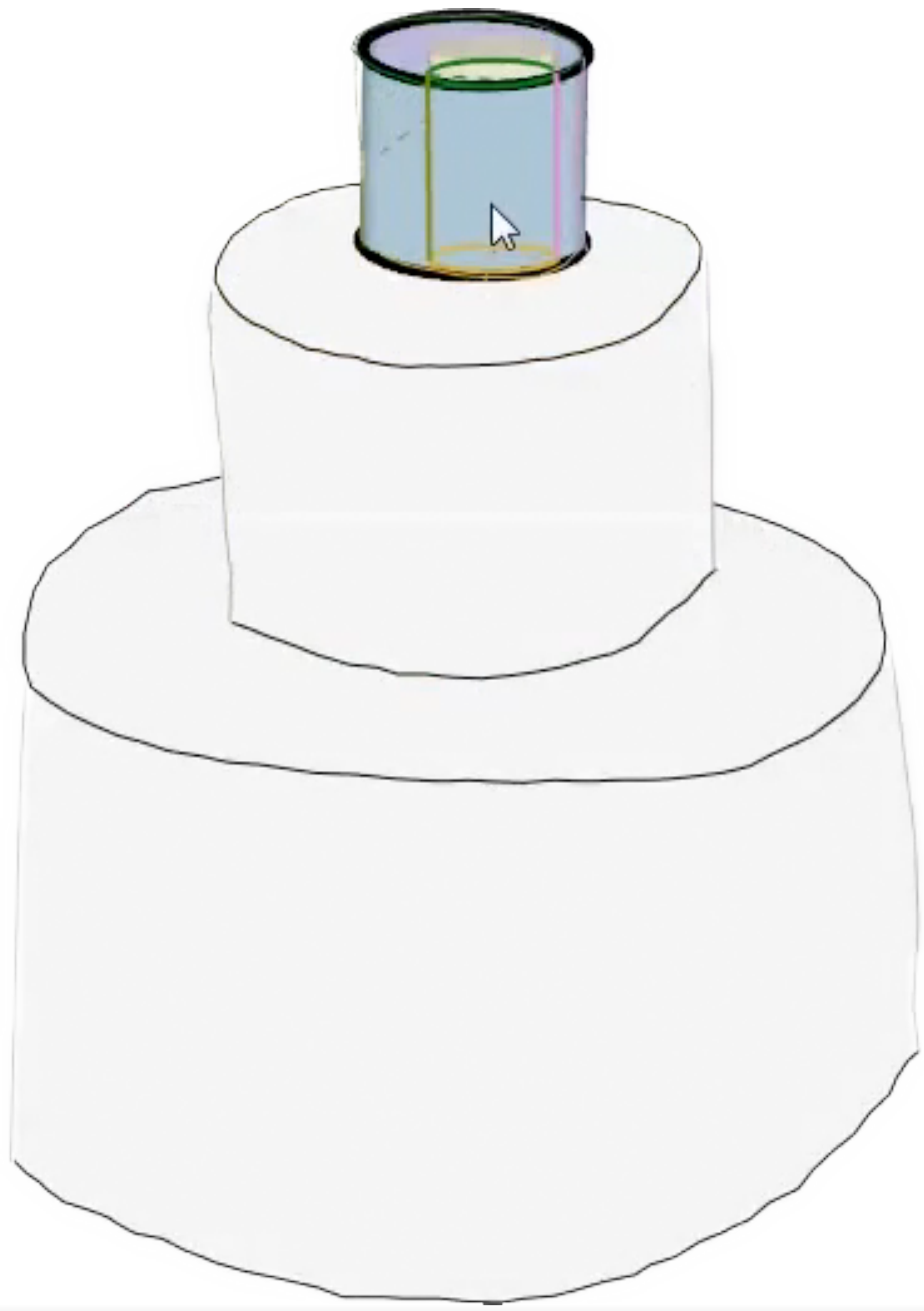
Finally, we perform additional filtering on the bipartite match to prevent improbable matches; see the paper for details.

Anatomy of a Drag: Curve Matching



Once the primitive’s feature and silhouette curves have been matched to curves in the sketch,
<click>
we perform an optimization procedure to fit the primitive to these curves. This produces an initial 3D fit at interactive rates as the user drags the primitive.

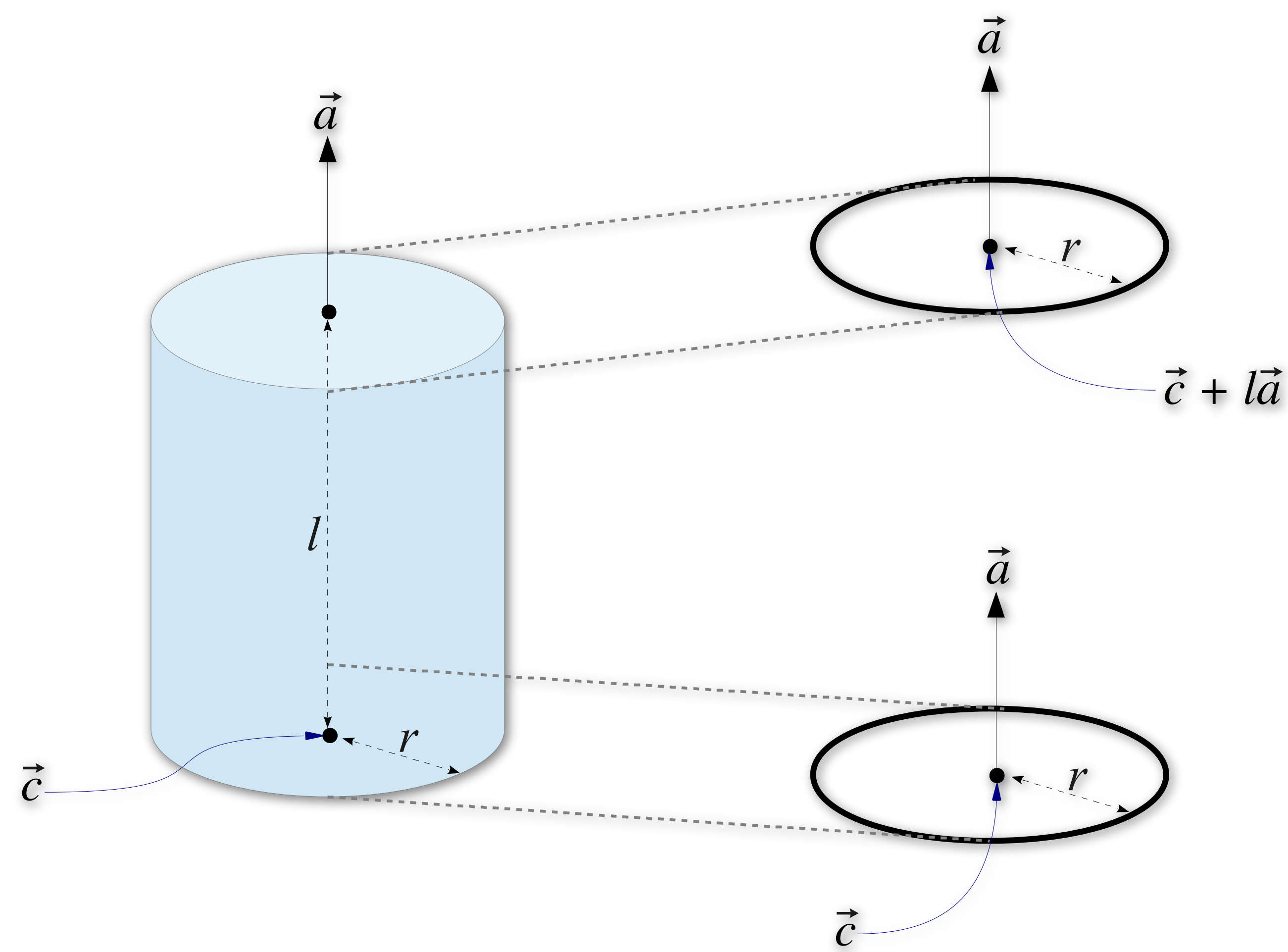
Anatomy of a Drag: Primitive Fitting



Once the primitive's feature and silhouette curves have been matched to curves in the sketch,
<click>
we perform an optimization procedure to fit the primitive to these curves. This produces an initial 3D fit at interactive rates as the user drags the primitive.

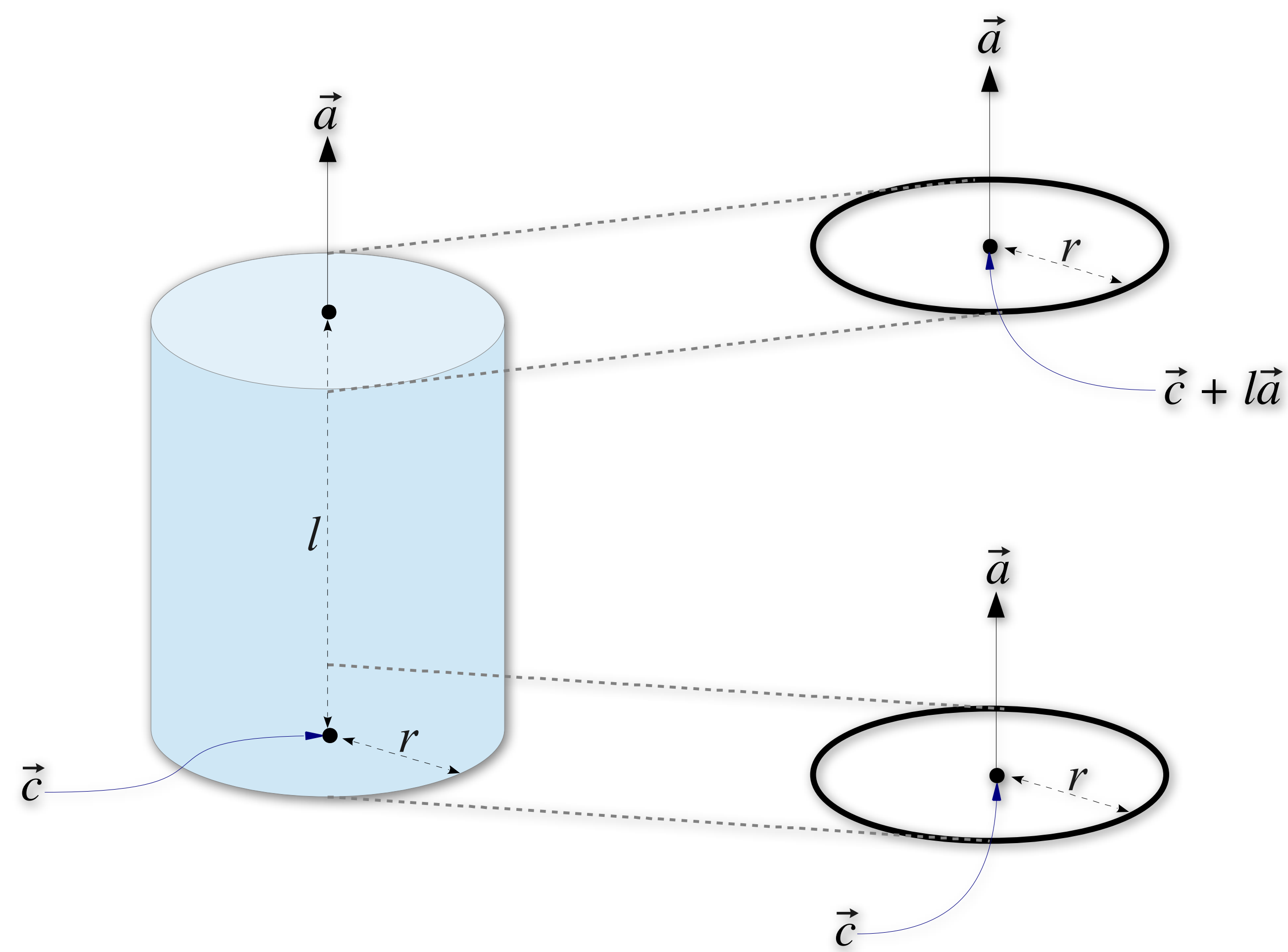
Our primitives are represented parametrically.
<click>

Anatomy of a Drag: Primitive Fitting



Our primitives are represented parametrically.
<click>
This cylinder, for example, has a bottom center point, a radius, a height, and an axis direction. To fit a primitive to the sketch,
<click>
we solve an optimization problem that minimizes the projected distance from curves on the primitive to curves in the sketch, subject to constraints maintaining the primitive's internal structure. For this cylinder, its axis direction must remain unit length.

Anatomy of a Drag: Primitive Fitting

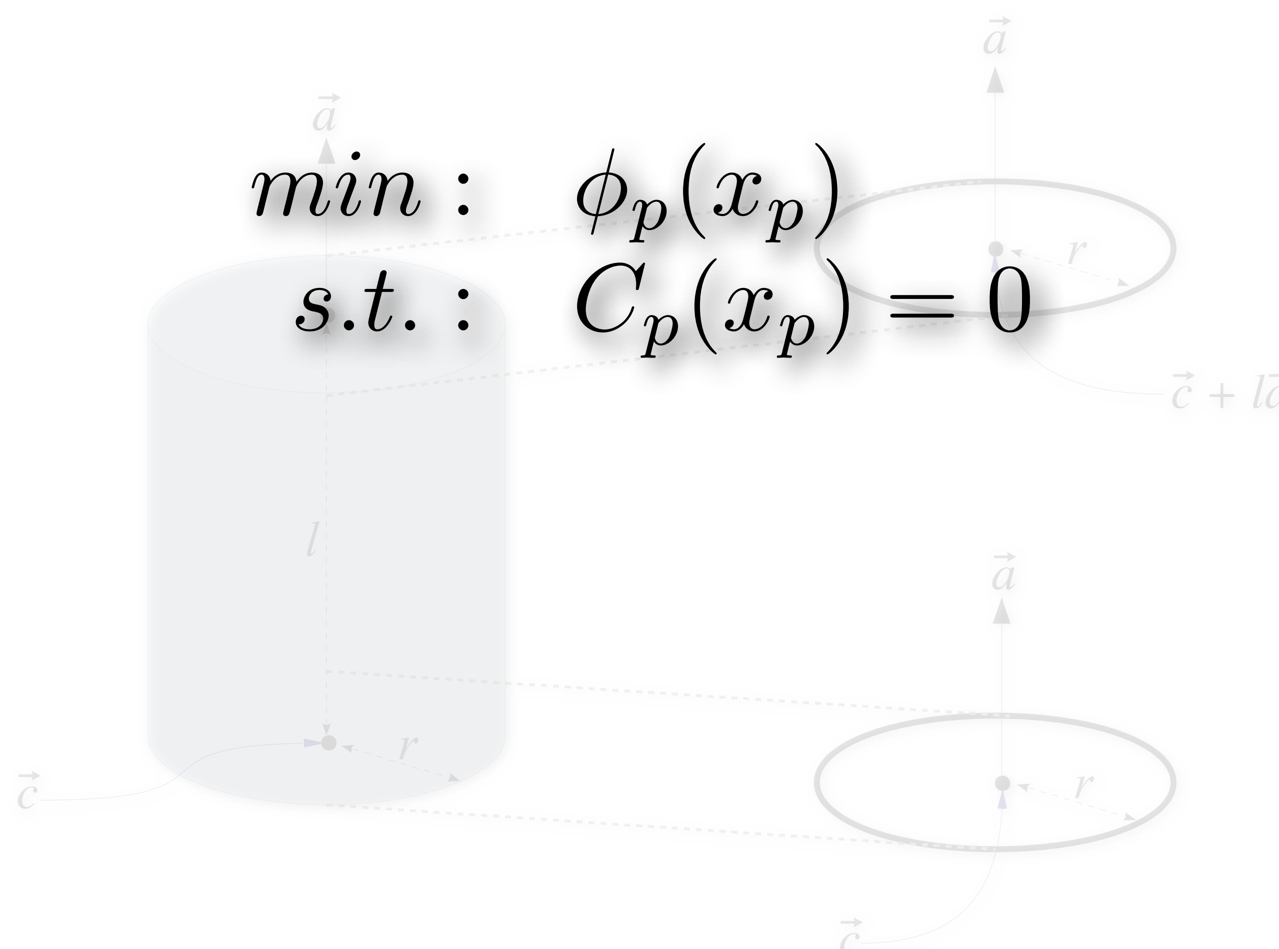


$$\begin{aligned} min: \quad & \phi_p(x_p) \\ s.t.: \quad & C_p(x_p) = 0 \end{aligned}$$

Our primitives are represented parametrically.
<click>
This cylinder, for example, has a bottom center point, a radius, a height, and an axis direction. To fit a primitive to the sketch,
<click>
we solve an optimization problem that minimizes the projected distance from curves on the primitive to curves in the sketch, subject to constraints maintaining the primitive's internal structure. For this cylinder, its axis direction must remain unit length.

Anatomy of a Drag: Primitive Fitting

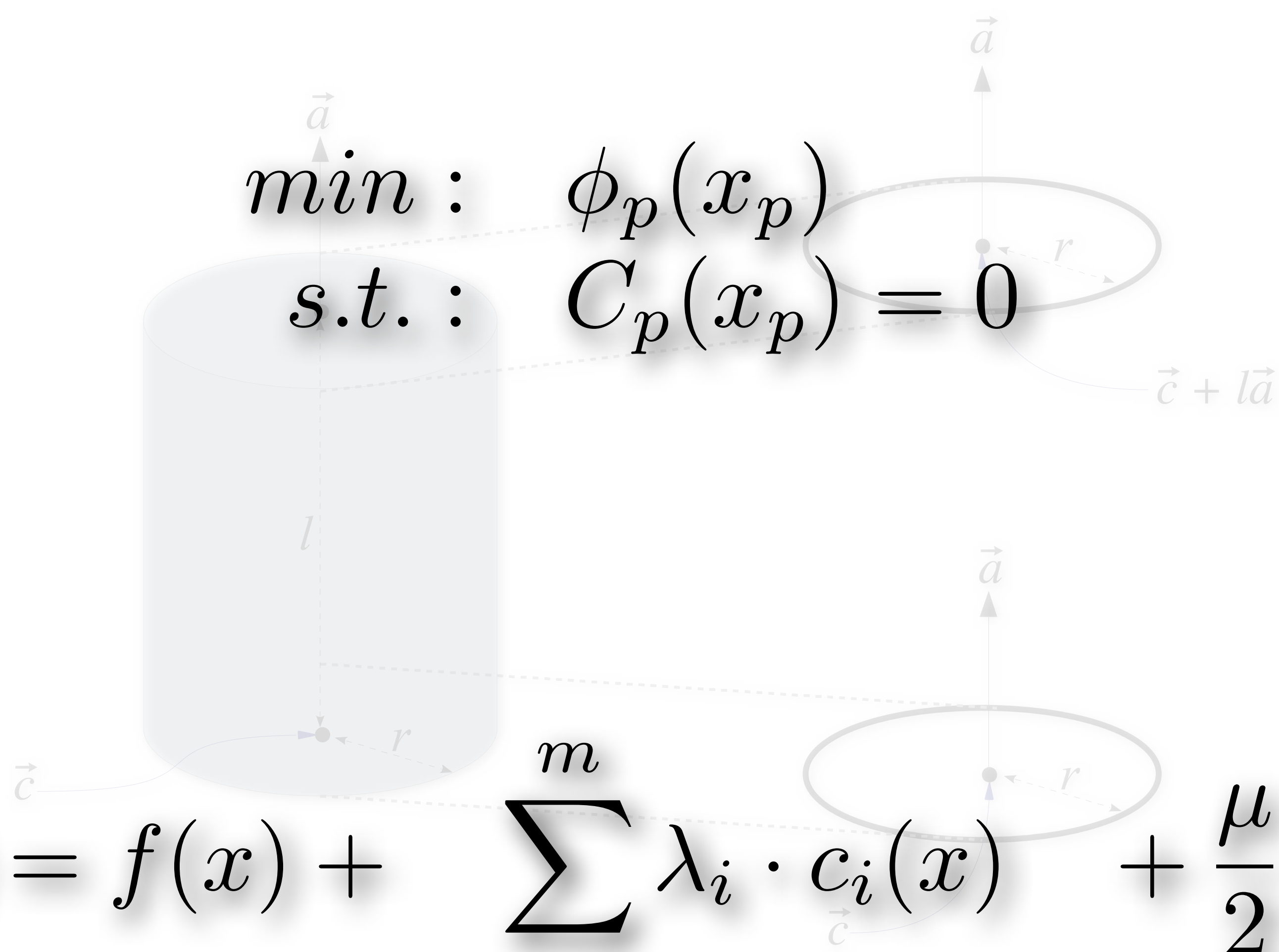
$$\begin{aligned} min: &\quad \phi_p(x_p) \\ s.t.: &\quad C_p(x_p) = 0 \end{aligned}$$



To solve the optimization problem, we use an augmented Lagrangian method.
This solves a sequence of unconstrained problems that converge to the constrained solution.
Gradients are computed via reverse-mode automatic differentiation, which is very efficient.
These two things enable the real-time optimization, for real-time feedback to the user.
See the paper and supplemental materials for the details of our objective functions and optimization algorithm.
Also, check out our automatic differentiation library, which we have released as a standalone open source package.

Our key idea is that the users's current drag provides the starting point for this difficult, non-convex optimization problem, and
<click>
feedback is displayed to the user in real-time.

Anatomy of a Drag: Primitive Fitting



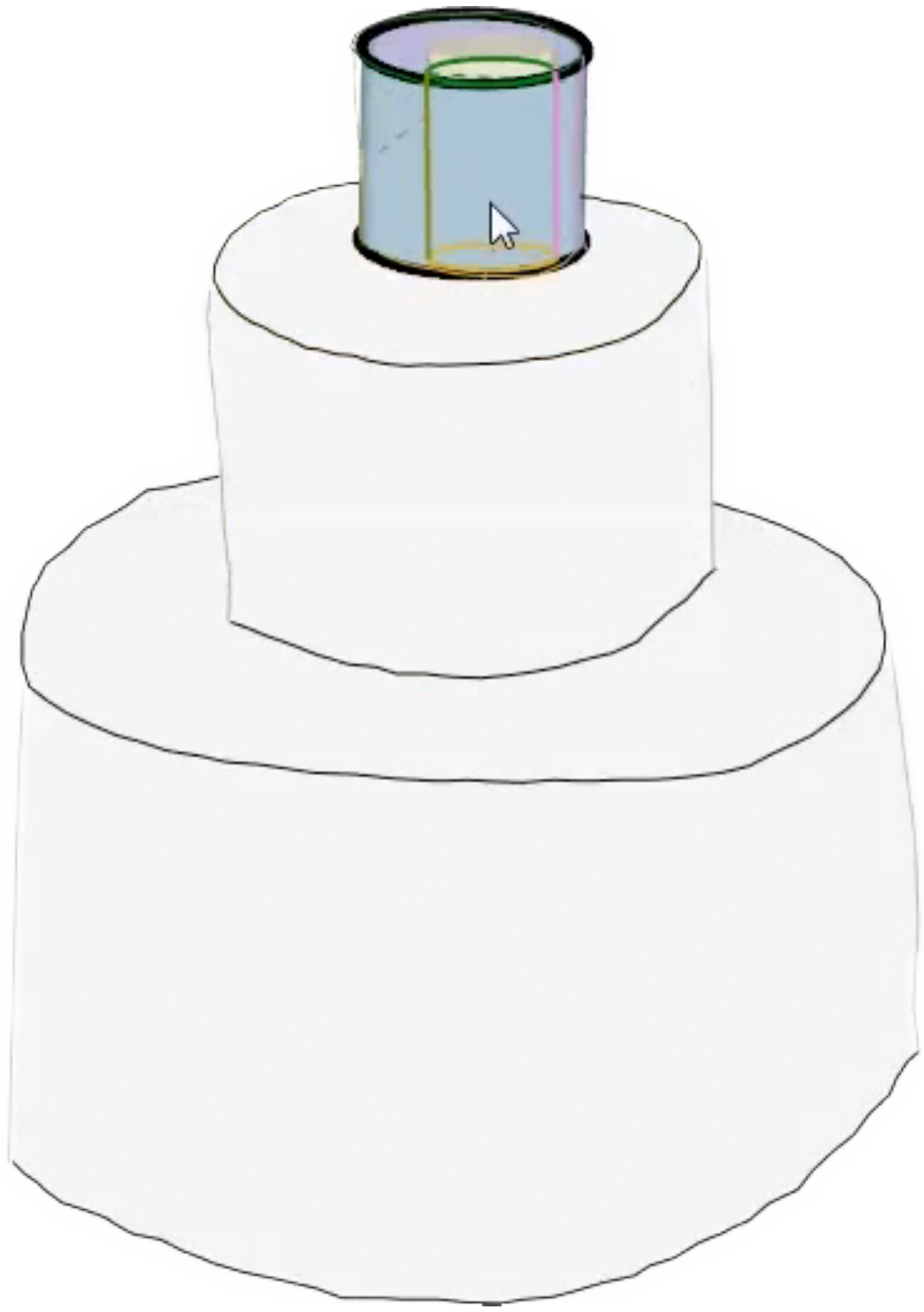
$$\begin{aligned} min: &\quad \phi_p(x_p) \\ s.t.: &\quad C_p(x_p) = 0 \end{aligned}$$

$$L_{aug}(x;\mu,\lambda) = f(x) + \underbrace{\sum_{i=1}^{m} \lambda_i \cdot c_i(x)}_{\text{Lagrangian term}} + \underbrace{\frac{\mu}{2} \cdot \sum_{i=1}^{m} |c_i(x)|^2}_{\text{Penalty term}}$$

To solve the optimization problem, we use an augmented Lagrangian method.
This solves a sequence of unconstrained problems that converge to the constrained solution.
Gradients are computed via reverse-mode automatic differentiation, which is very efficient.
These two things enable the real-time optimization, for real-time feedback to the user.
See the paper and supplemental materials for the details of our objective functions and optimization algorithm.
Also, check out our automatic differentiation library, which we have released as a standalone open source package.

Our key idea is that the users's current drag provides the starting point for this difficult, non-convex optimization problem, and
<click>
feedback is displayed to the user in real-time.

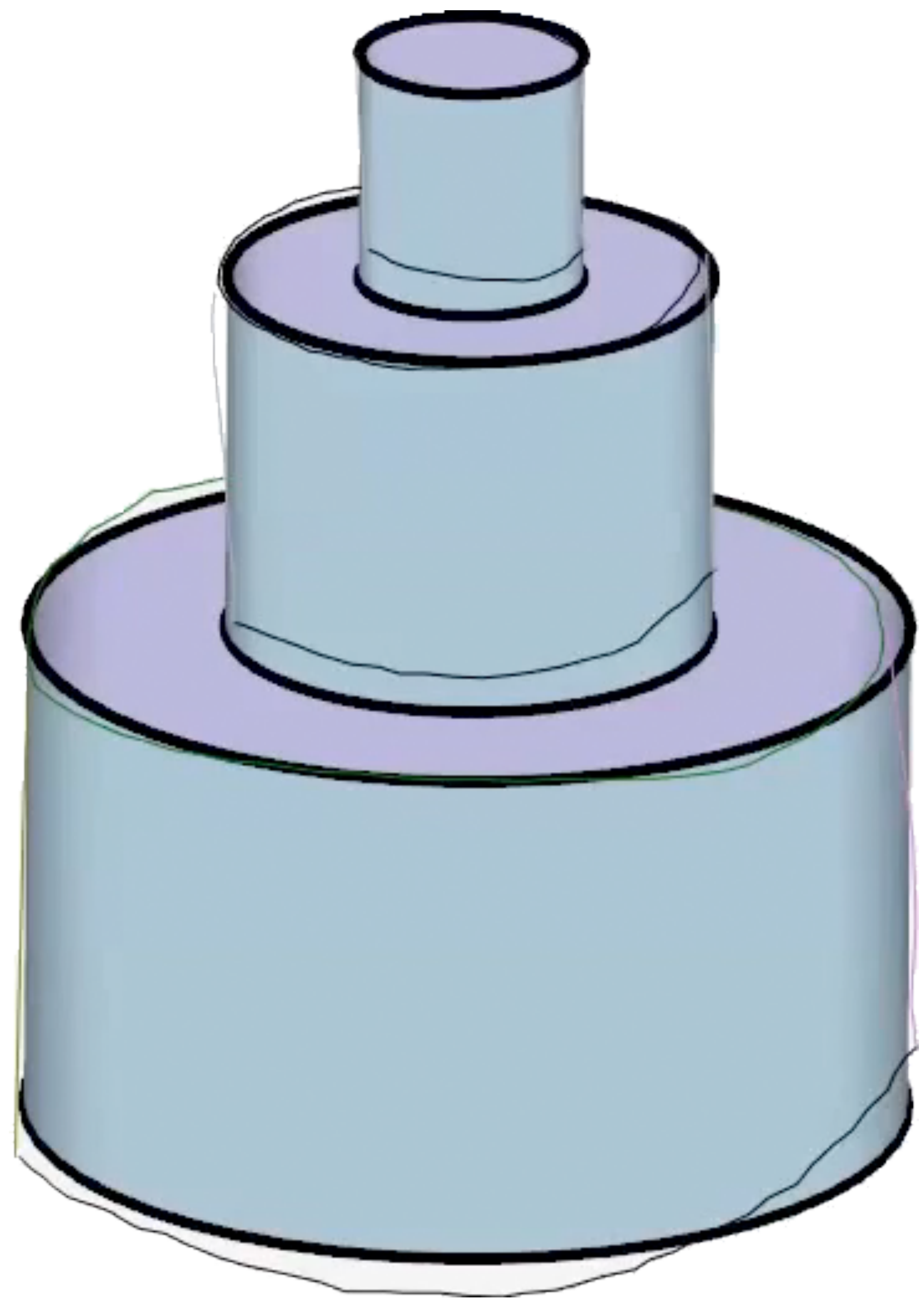
Anatomy of a Drag: Primitive Fitting



Our key idea is that the users's current drag provides the starting point for this optimization problem, and
<click>
feedback is displayed to the user in real-time.

Finally, as multiple primitives are added,
<click>
Geosemantic relations are detected and imposed as additional constraints.

Anatomy of a Drag: Geosemantic Relations

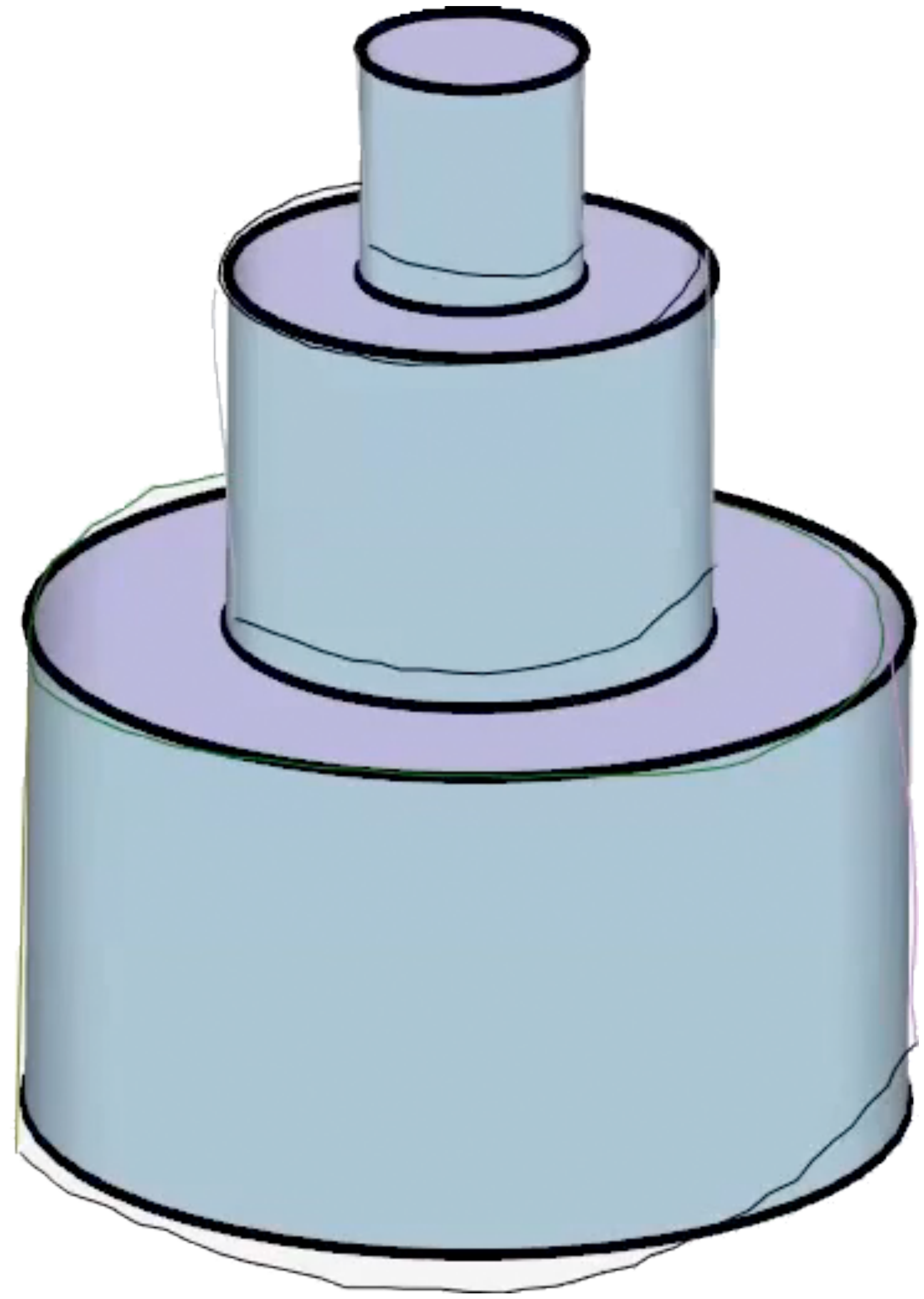


Finally, as multiple primitives are added,
<click>
Geosemantic relations are detected and imposed as additional constraints. These constraints position and orient the primitives relative to each other in space.

This occurs once the drag is finished; i.e. when the primitive is "dropped". These inter-primitive constraints trigger a
<click>
larger optimization problem involving all primitives; because the drag is finished, there is ample time for this lengthier optimization.

Anatomy of a Drag: Geosemantic Relations

$$
\begin{aligned}
\min_{x}: \quad & \textstyle\sum_{p \in P} \phi_p(x_p) & \\
s.t.: \quad & C_p(x_p) = 0 & \forall p \in P \\
& \psi_g(x_g) = 0 & \forall g \in G
\end{aligned}
$$

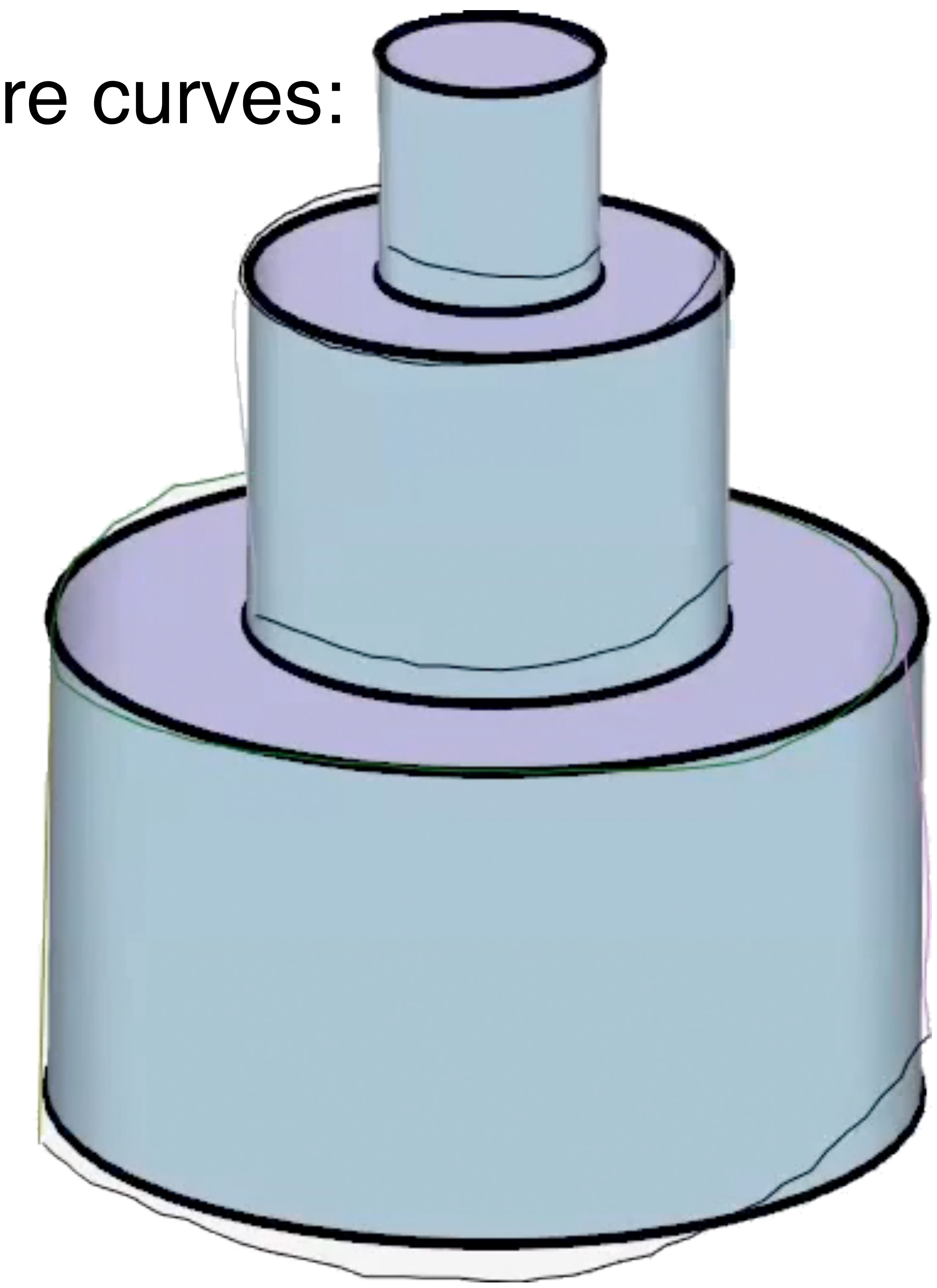


Finally, as multiple primitives are added,
<click>
Geosemantic relations are detected and imposed as additional constraints.

This occurs once the drag is finished; i.e. when the primitive is "dropped". These inter-primitive constraints trigger a
<click>
larger optimization problem involving all primitives; because the drag is finished, there is ample time for this lengthier optimization.

Geosemantic Relations

Constraints linking two or more feature curves:



Geosemantic relations are constraints that link two or more primitives' feature curves. They are automatically detected within some tolerance based on the initial fit of the newly dropped primitive. This automates tedious work for the user, who does not have to precisely position and orient the primitives in 3D. In this example, our system automatically detects that the cylinders are almost co-axial, and constraints them to be co-axial. Our system also detects that the cylinder faces are touching, and so imposes coplanar constraints.

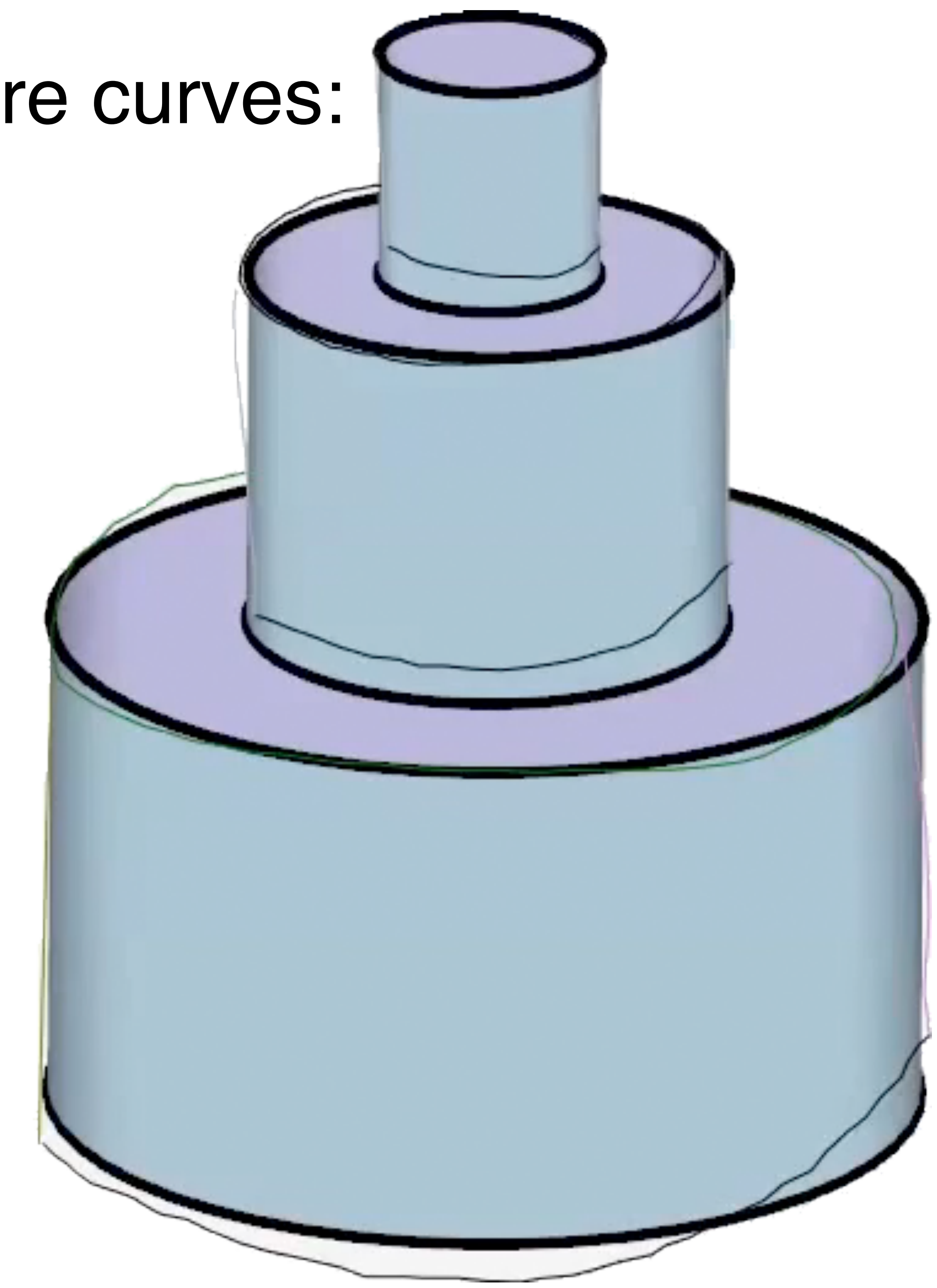
Geosemantic relations operate on primitives' feature curves, which are always planar. They can be manually overridden by the user.

The relationships that we support are <click>...:
Parallelism Orthogonality Collinear centers Concentricity Coplanarity

Geosemantic Relations

Constraints linking two or more feature curves:

- Parallelism
- Orthogonality
- Collinear centers (three or more)
- Concentric
- Coplanar



Geosemantic relations are constraints that link two or more primitives' feature curves. They are automatically detected within some tolerance based on the initial fit of the newly dropped primitive. This automates tedious work for the user, who does not have to precisely position and orient the primitives in 3D. In this example, our system automatically detects that the cylinders are almost co-axial, and constraints them to be co-axial. Our system also detects that the cylinder faces are touching, and so imposes coplanar constraints.

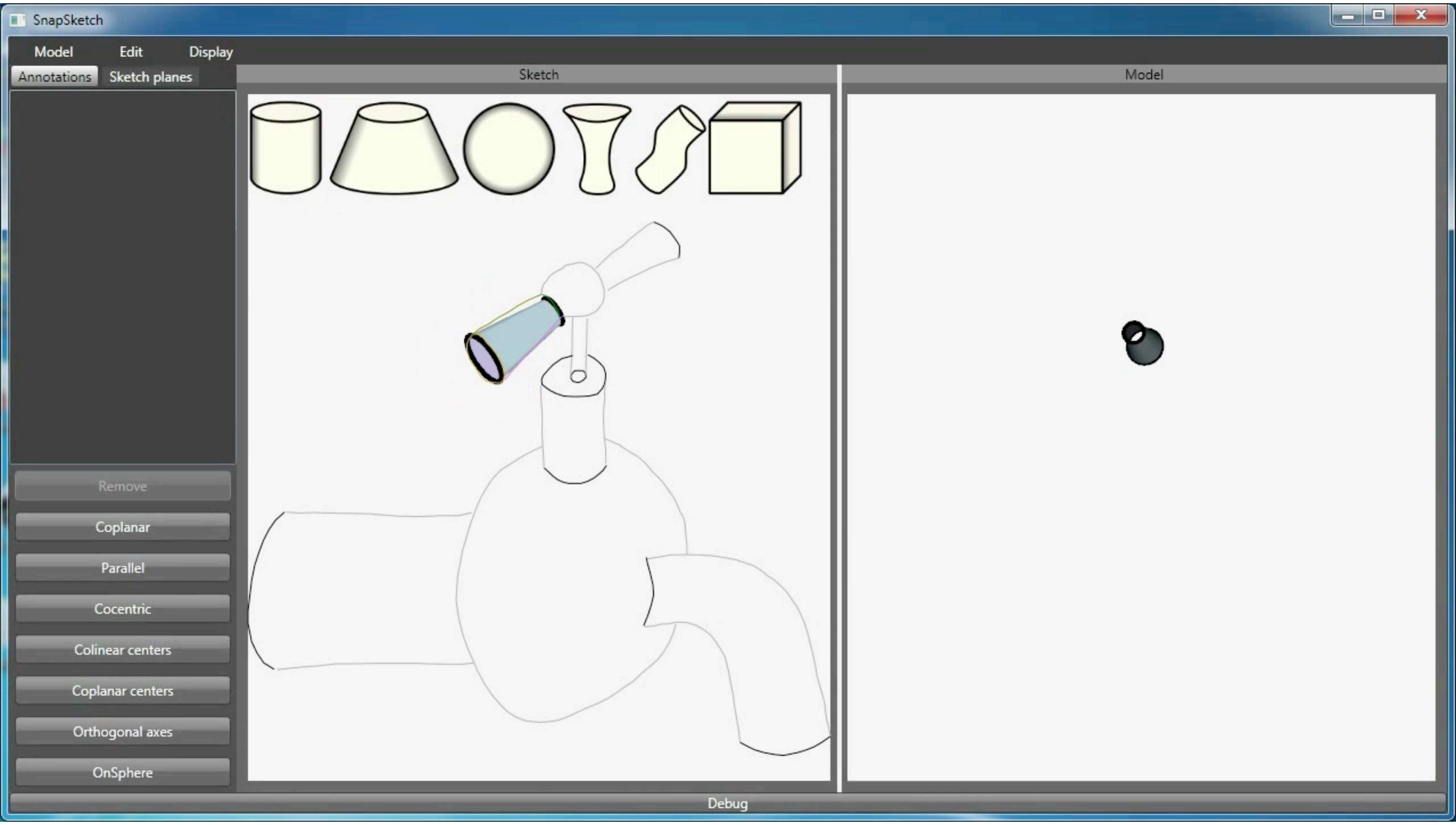
Geosemantic relations operate on primitives' feature curves, which are always planar. They can be manually overridden by the user.

The relationships that we support are <click>...:

Parallelism Orthogonality Collinear centers Concentricity Coplanarity

Results

Here are some results created with our system.

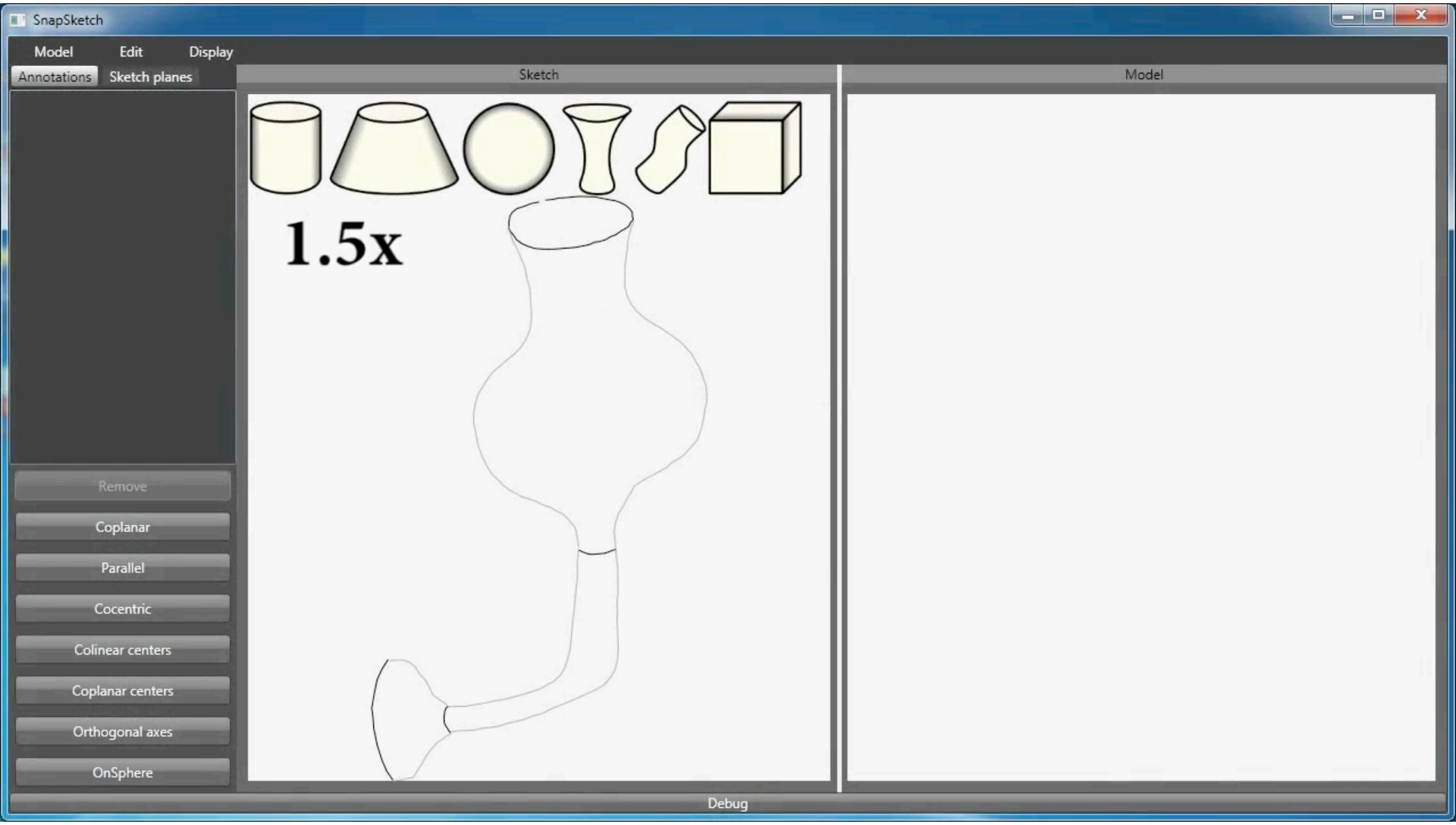


~2x

We are going to demonstrate several interactive modeling sessions.

In this first session, the user is modeling a tap.

As the user drags appropriate primitives over the sketch, the real-time snap is displayed. This involves the bipartite graph matching problem followed by the optimization, which uses the primitive's current configuration as its starting point. This allows users to be only as precise as needed.



In this session, the user models a wall lamp.

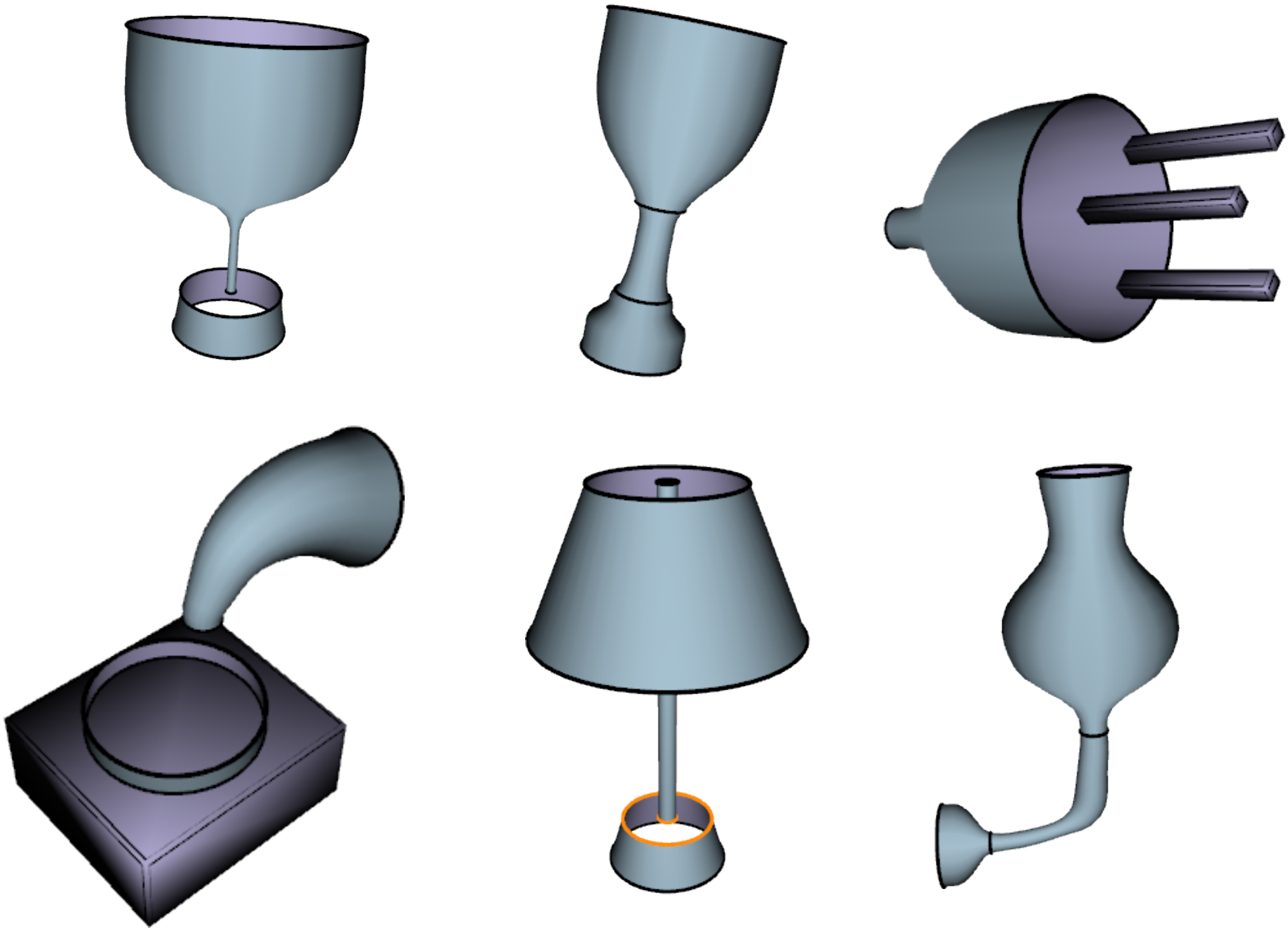
After each primitive is dropped, geosemantic relationships are inferred and a full optimization involving all primitives at once is performed.

Here is a power plug.

Despite the roughness of the sketches, the inferred geosemantic relationships result in a precise, aesthetically pleasing model, which allows users to quickly preview an idea in 3D.

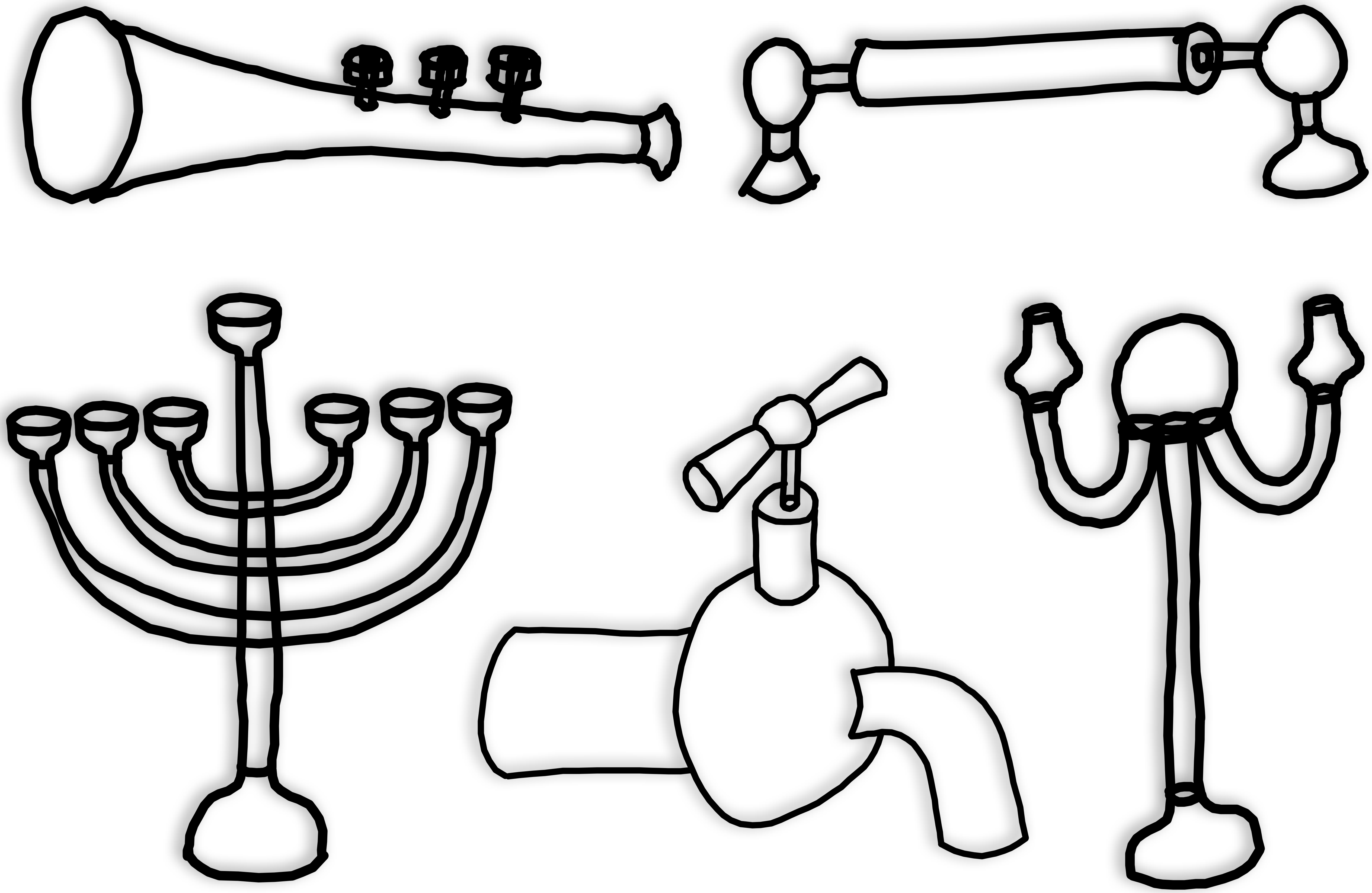
In this final session, a gramophone is modeled.
By dragging, the user simultaneously solves the segmentation, recognition, and positioning problems.

Here are several input sketches...
<click>
...and the 3D models created from them.

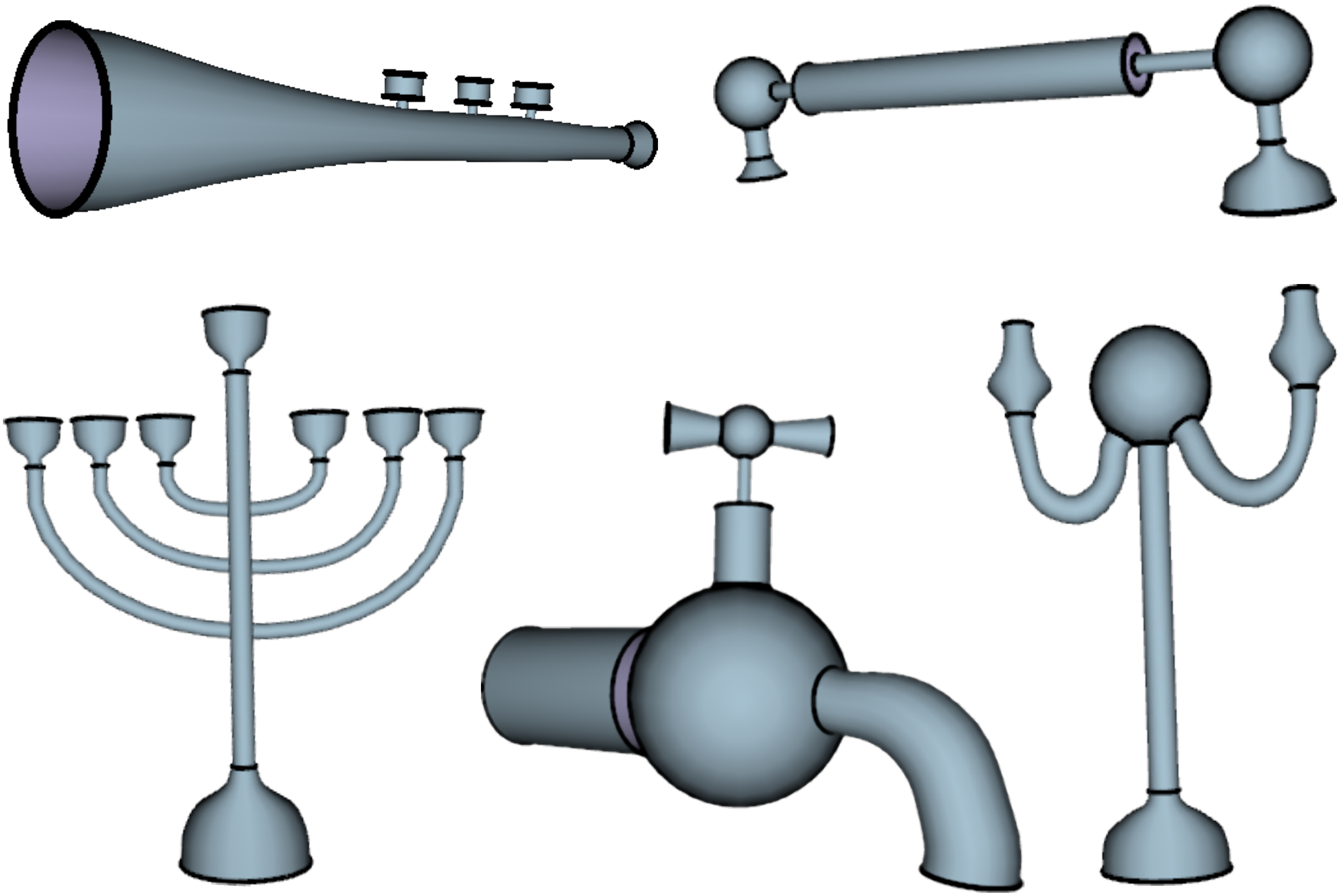


Here are several input sketches...
<click>
...and the 3D models created from them.

These simple models were created in less than 30 seconds.



Here are a few more sketches...
<click>
...and the 3D models created from them.



Here are a few more sketches...
<click>
...and the 3D models created from them.

These more complex models took between 1 and 3 minutes to create.

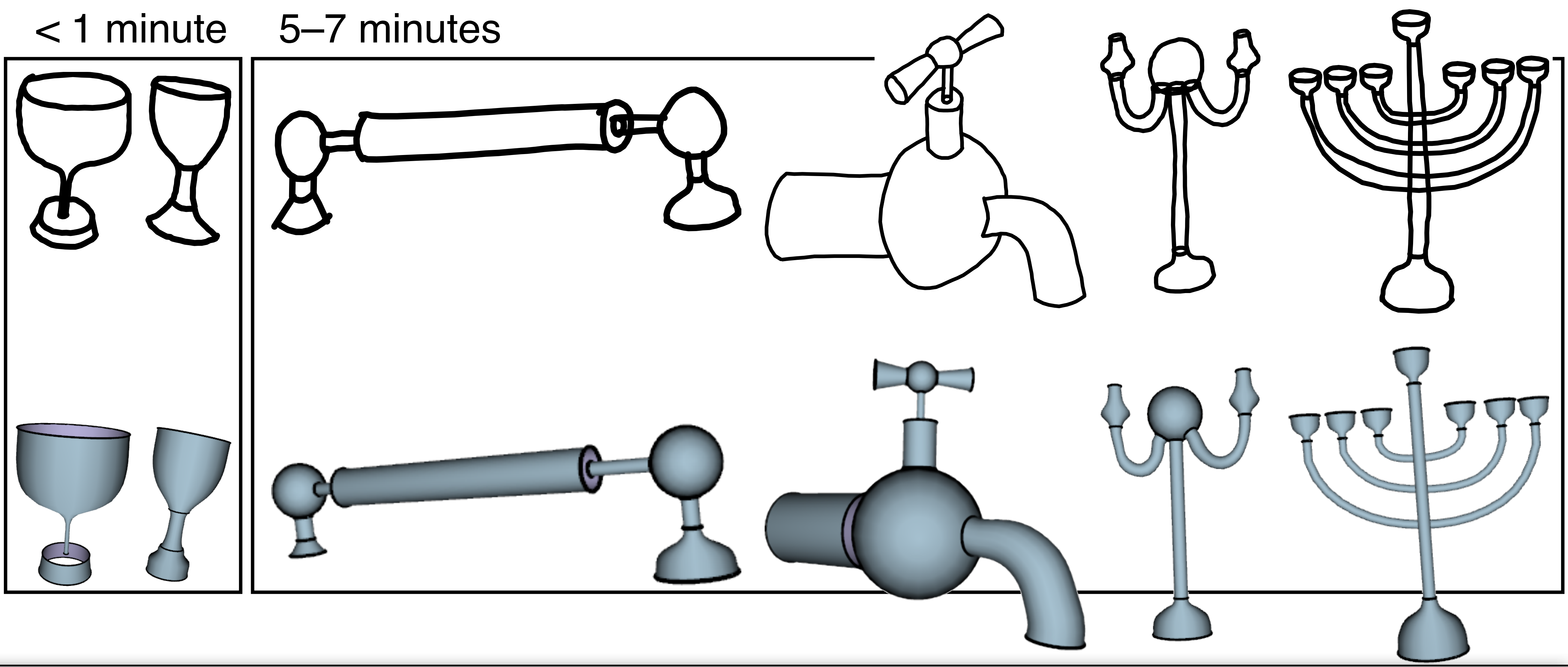
User Study

10 users (5 female/5 male)

20 minute tutorial

Tagging took less than a minute (on average)

Drag-and-drop modeling took (average and median):

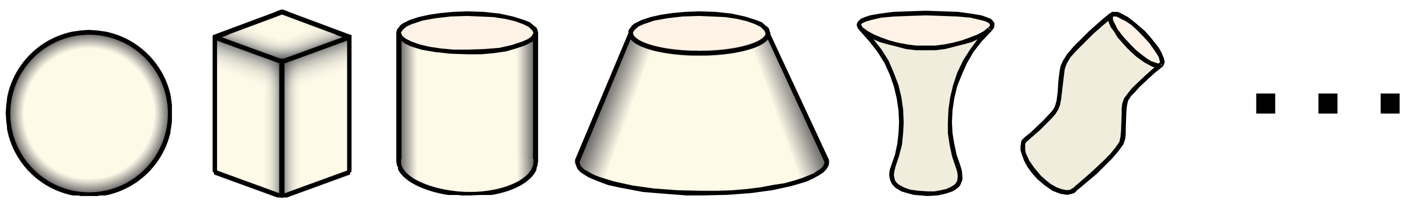


We performed a user study involving ten users.

Every user created 7 models. We found that
<click>Tagging took less than a minute (on average).
<click>Drag-and-drop modeling took
<click>less than a minute for simple models and
<click>between 5-7 minutes for more complex models.
Two users were unable to create the phone handset model; otherwise, users could create all models.
This demonstrates that even first-time users are able to create models extremely rapidly.

Limitations & Future Work

More primitives

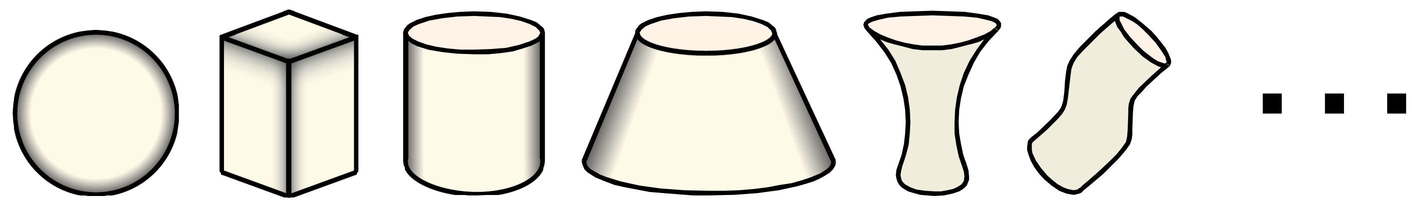


For future work, we would like to add additional primitives. Our current primitives are suitable for mechanical parts composed of spheres, boxes, and generalized cylinders.

Presently, we require a vectorized input sketch and a curve classification step. We would like to eliminate this step and operate directly on raster sketches.

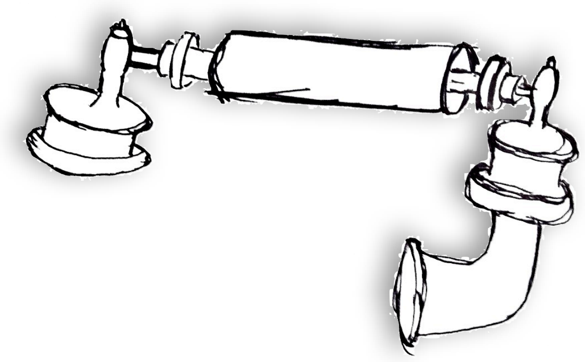
Finally, sketched occlusions, as in this trombone, are challenging and require the user to specify many geosemantic constraints on the occluded feature curves, and may require additional types of geosemantic relations.

Limitations & Future Work

More primitives 

Operate directly on raster sketches

Eliminate sketch curve classification



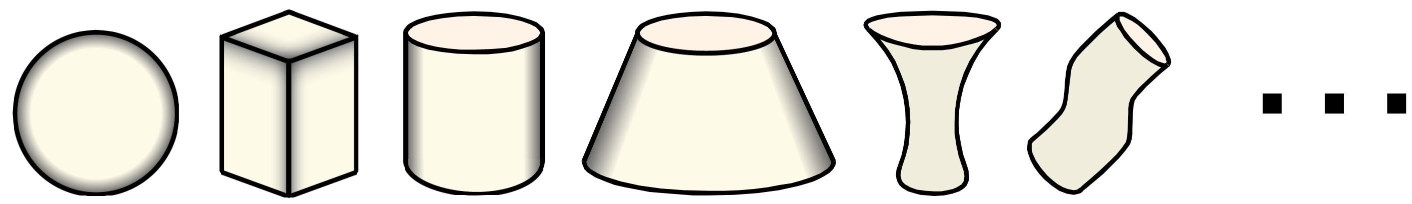
For future work, we would like to add additional primitives. Our current primitives are suitable for mechanical parts composed of spheres, boxes, and generalized cylinders.

Presently, we require a vectorized input sketch and a curve classification step.
We would like to eliminate this step and operate directly on raster sketches.

Finally, sketched occlusions, as in this trombone, are challenging and require the user to specify many geosemantic constraints on the occluded feature curves, and may require additional types of geosemantic relations.

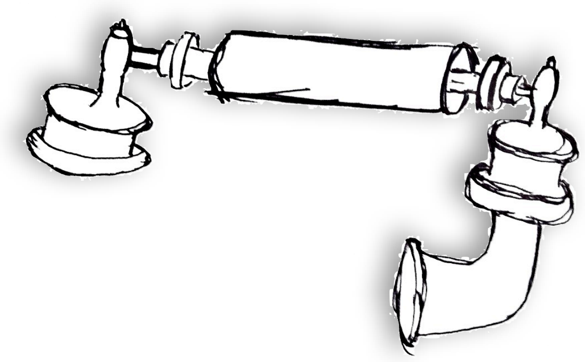
Limitations & Future Work

More primitives



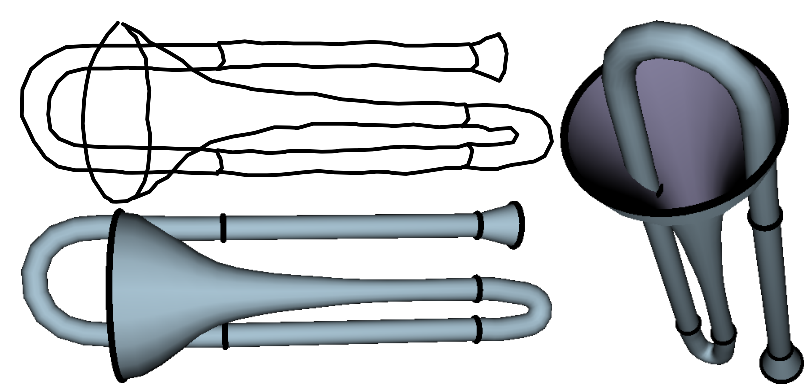
Operate directly on raster sketches

Eliminate sketch curve classification



Sketched occlusions

More geosemantic relations



For future work, we would like to add additional primitives. Our current primitives are suitable for mechanical parts composed of spheres, boxes, and generalized cylinders.

Presently, we require a vectorized input sketch and a curve classification step.
We would like to eliminate this step and operate directly on raster sketches.

Finally, sketched occlusions, as in this trombone, are challenging and require the user to specify many geosemantic constraints on the occluded feature curves, and may require additional types of geosemantic relations.

Conclusion

We have made a highly **non-convex** problem tractable by:

- Introducing an interactive solution.
- Separating that which is easy for a **human** and challenging for a **computer**.
- Providing a good starting point via **drag-and-drop**.
- Providing a flexible collection of parameterized **primitives**.
- Inferring **geosemantic relationships** for aligning primitives and placing them in depth.

Thank you

Questions?

Code for our fast reverse-mode automatic differentiation is online: http://autodiff.codeplex.com/

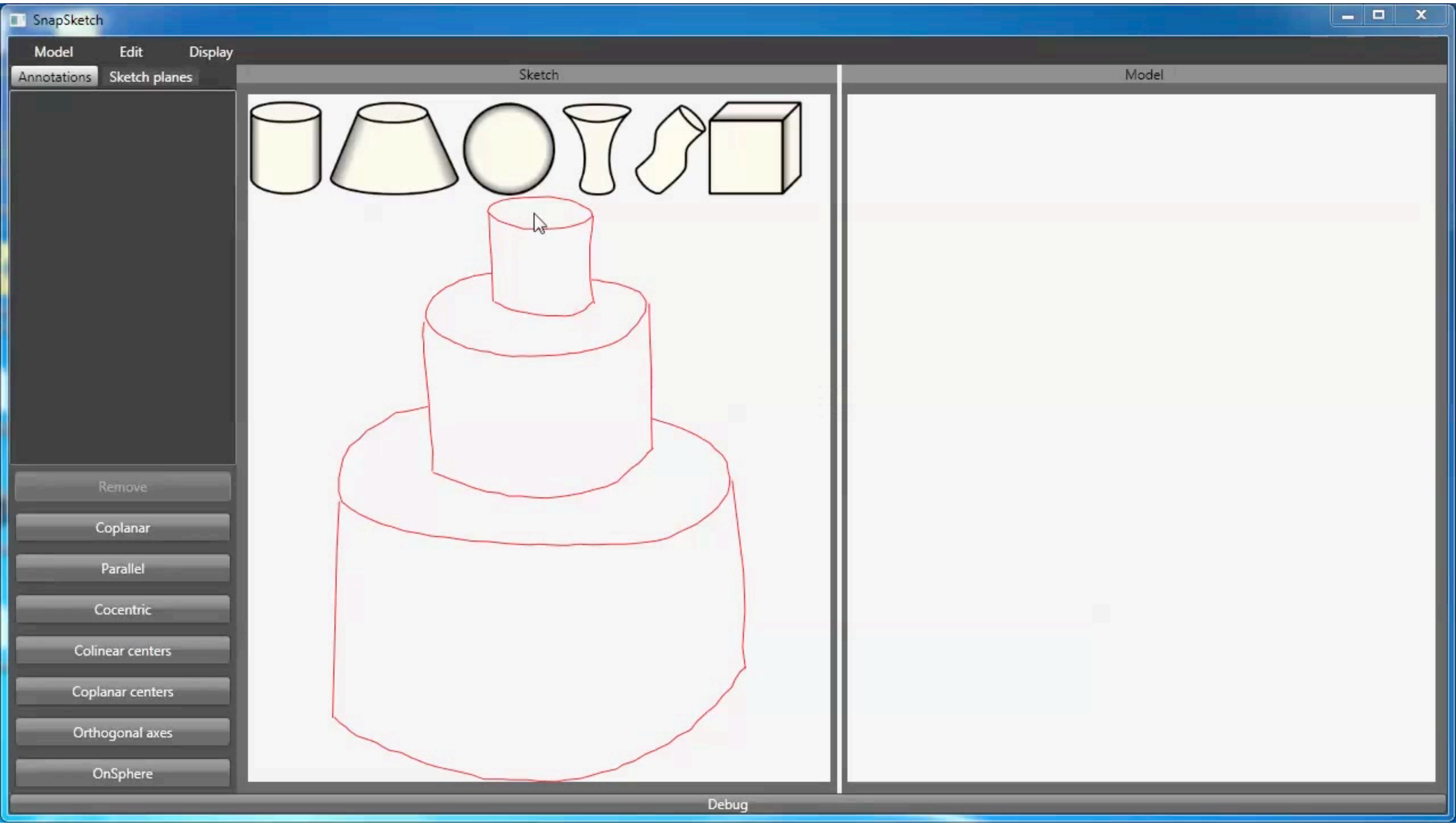
This research was supported in part by the Israel Science Foundation (grant no. 324/11), Sloan Foundation, NSF (CAREER Award CCF-06-43268 and grants IIS-09-16129, IIS-10-48948, IIS-11-17257, CMMI-11-29917, IIS-09-16845), and generous gifts from Adobe, Autodesk, Intel, mental images, NVIDIA, Side Effects Software, and the Walt Disney Company.

GEORGE MASON UNIVERSITY

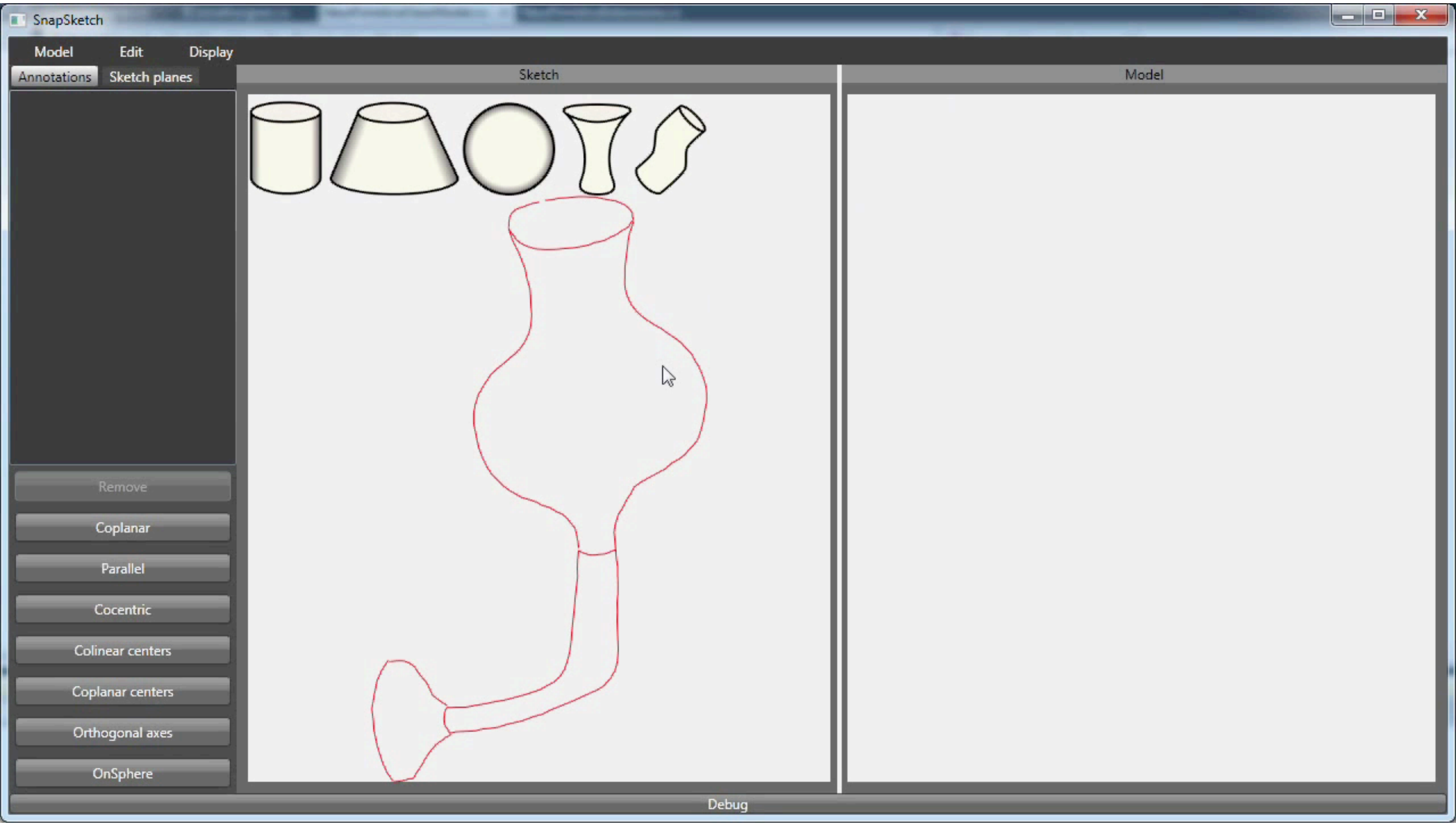
IDC HERZLIYA

Thank you, and we would be happy to take questions.

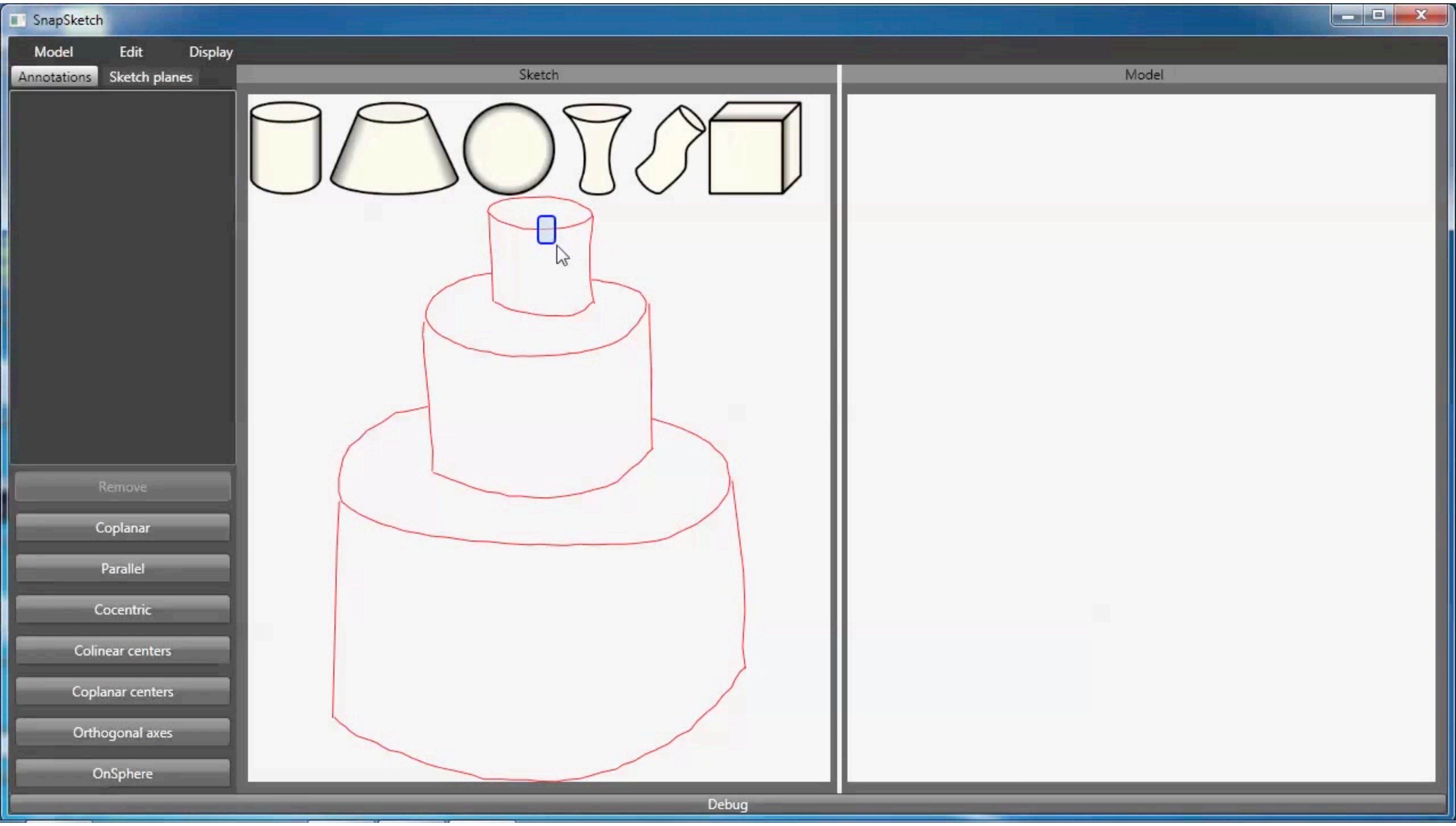
Manual Tagging



Manual Tagging



Automatic Tagging



Geosemantic Snapping for Sketch-Based Modeling — Shtof, Agathos, Gingold, Shamir, Cohen-Or — Eurographics 2013

The user classifies sketch curves as either “feature” or “silhouette”. This is a simple, semi-automated process.

In the example shown here, <click>

tagging just one curve correctly classifies all curves. The semi-automation is based on the observation that sketch curves likely belonging to a cylinder alternate between silhouette and feature, while sketch curves belonging to a box or sphere should always be classified as “feature” curves.