Integrating Open-Domain Sketch Understanding with Qualitative Two-Dimensional Rigid-Body Mechanics

Jon W. Wetzel and Kenneth D. Forbus

Qualitative Reasoning Group, Northwestern University
2145 Sheridan Road, Evanston, IL 60208-0834 USA
{jw, forbus}@northwestern.edu

Abstract

Sketching is a powerful modality for thinking through, and communicating about, mechanical designs. Qualitative mechanics reasoning has been applied to sketched input before but not without succumbing to limitations of domain-based recognition or requiring complex annotation and additional explicit knowledge from the user. This paper presents solutions to three problems in integrating qualitative mechanics reasoning with a sketch understanding platform: identifying objects and forces, discovering regions of interaction between them, and understanding the effects of these interactions. We show how the spatial knowledge available in a conceptually labeled sketch is sufficient to solve the first two problems and enables the use of qualitative mechanics to solve the third. Examples from an implemented system are used for illustration. This system is the first step in a plan to create a sketch understanding system capable of providing corrective feedback for sketched engineering designs.

Introduction

Sketching is a valuable way to work through ideas and communicate them to others. This is especially true in conceptual design, when the feasibility of basic ideas is under consideration. Qualitative reasoning seems a natural fit for such tasks, since precise details are typically unavailable during this stage of design. However, existing sketch understanding systems do not currently provide much support for these tasks. Some existing systems rely on animation for feedback. In (Alvarado & Davis 2007), ink recognition is used to set up input for a mechanics simulator, much the way that Quickset (Cohen *et al* 1997) was used to set up scenarios for a military simulator. Unfortunately, it is not clear how useful such animated output is for design tasks: If a student's design doesn't work, all of the knowledge about why something failed is hidden within the procedures of the off-the-shelf physics engine. In contrast, qualitative causal models can be used to provide explanations, both to designers and especially to engineering students learning to design. The SketchIt approach of Stahovich *et al* (1998) provides qualitative reasoning about sketches of mechanical systems that seems to be more useful in such tasks. However, SketchIt required the human user to identify surfaces of interest and describe the range of their interaction manually. This is tedious for experienced designers and not feasible for novice designers.

This paper describes how we are using a combination of qualitative mechanics and open-domain sketch understanding to enable software to reason about forces, mechanical constraint, and motion from hand-drawn sketches. We begin with our motivating context, coaching entering engineering students in how to use sketches to communicate. We then summarize CogSketch, the sketch understanding system we are using as a platform, and the qualitative mechanics theories we are building upon. We describe how we are embedding qualitative mechanics reasoning into sketch-based representations. We identify and present solutions for three problems in carrying out this integration: identifying objects and forces, discovering regions of interaction between them, and understanding the effects of these interactions. We describe some examples of our system's reasoning, and close with related and future work.

Helping Students Learn to Communicate

Communication is an important skill for engineering students. At Northwestern, 1st and 2nd year engineering students take Engineering Design and Communications, which teaches both skills in an integrated manner. Students working in teams of three or four tackle problems for real clients. Examples include patients at the Rehabilitation Institute in Chicago, who need new tools to help them achieve everyday tasks, like chopping vegetables or trimming their nails, despite physical handicaps. Students build prototypes of their designs to explore particular issues, with regular feedback from potential users. Conversations with instructors revealed that one significant problem they had was helping students learn to use their sketches to communicate ideas, both within the team and to clients. We are creating a sketch-based system to address this problem.

The idea of the *Design Buddy*, as we are calling it, is to be a "crash test dummy" for students to practice explaining their designs. They will sketch their ideas, explain the parts, what they are made of, their intended behaviors, and the intended functional roles of the parts. The system will reason through the possible behaviors itself, based on its understanding of qualitative mechanics, materials, and everyday actions. It will compare its predictions of behaviors with the student's intended behaviors, and ask the student questions about discrepancies. These may include the student not mentioning some critical aspect of the behavior in their explanation, or predicting a behavior that the system does not think is possible. This is a very challenging task, for three reasons: (1) The qualitative mechanics reasoning must be very general and robust. The design projects change constantly, and a wide range of problems arise. (2) The interface must be both sufficiently natural to not be a distraction, and must incidentally help the student learn to explain things in terms that practicing engineers would use. (3) The coaching software must have enough strategies to provide students with effective help in learning how to think about their particular design and the design process itself. This paper focuses on a particular aspect of the problem (1), namely, doing robust qualitative mechanics reasoning in a sketch-based environment.

Background

We briefly review the relevant aspects of CogSketch, our sketch understanding system, and prior work on qualitative mechanics.

CogSketch

The platform we are using is CogSketch, a publicly available sketch understanding system built on the nuSketch architecture (Forbus *et al.* 2004). CogSketch is an open-domain system. Most sketch understanding systems equate understanding ink with classifying it as a symbol drawn from a fixed vocabulary of items. By contrast, nuSketch systems treat recognition as a catalyst, not a necessity. Pieces of ink (called *glyphs*) can be labeled by the user with concepts drawn from a large underlying knowledge base[1]. CogSketch automatically computes a number of visual relationships between glyphs, such as topological relationships (using RCC8 (Cohn, 1996)) and positional relationships (e.g., above, left, etc.). CogSketch can also analyze the ink within a glyph, identifying straight-line segments within the ink, corners, and so forth.

Annotation glyphs are used to provide additional information about other glyphs. Examples of annotation glyphs include applied forces, axes of rotation, and centers of mass. Arrows are automatically recognized when drawing annotation glyphs, so that, for example, the orientation and position of application of force can be automatically computed.

[1] We use the contents of OpenCyc, containing over 58,000 concepts, plus our own group's extensions for qualitative reasoning and analogy.

Qualitative Mechanics

Our model of qualitative mechanics is based on work done by Nielsen (1989) and Kim (1993). Nielsen's work represents direction and orientation in terms of qualitative vectors (*q-vectors*). For a 2-d space, there are 8 translational q-vectors (right, up, left, down and one for each quadrant in between) and two rotational q-vectors (clockwise and counter-clockwise). All forces/torques and movements are expressed in these directions.

Objects are represented as sets of surfaces. Each surface has a parent object, a direction outward from the object (surface normal) and a direction towards the center of rotation of the object. Representations for translational and rotational freedom and constraint are used to state if/when an object can move or rotate. Forces and motion are transmitted from object to object through their surface contacts, with the net force and net motion determining the final motion of the object in a given state.

Nielsen's representations supported envisionment of a variety of mechanical systems, including clocks (Forbus *et al* 1991). Kim's work adds several more representations to Nielsen's, including notions of bounded stuff and flow fields. This enabled Kim's system to reason about non-rigid bodies. The end result was a system which could understand lift pumps, laminar flow over surfaces, and automobile engines. However, neither of these systems dealt with hand-drawn inputs. Kim's system assumed predicate calculus input representations. Nielsen's system could accept as input scanned images of parts, and automatically simplify the configuration space it computed to handle noise. For example, it was able to change resolution to see a gear train as having only one degree of freedom, despite noise in the scanned input. However, scanned photographs are far more accurate than hand-drawn sketches, making hand-drawn sketches an even harder problem. Adapting these qualitative mechanics ideas to work with hand-drawn sketches is the challenge this paper addresses.

In this paper, we restrict ourselves to a subset of qualitative mechanics, namely rigid-body mechanics involving 2D polygonal bodies. This is an important first step, because it allows for a wide variety of test cases while requiring only a subset of the qualitative mechanics.

Interpreting Rigid-Body Sketches

In this section we begin with an overview of the three main problems we encountered when integrating qualitative mechanics with a sketch understanding system. We explain our solutions to these problems through our system. This includes an overview of the sketching process, an explanation of how the sketch is translated into a qualitative representation, and how inference is used to answer questions about the sketched system.

Issues with Integrating Sketched Input

The three main problems we encounter when interpreting rigid-body sketches are: identifying objects and forces, discovering regions of interaction between them, and understanding the effects of these interactions. The first problem has often been approached before using recognition. However, in a completely open domain (even limited to two-dimensional polygons) this is not an option. Rather, we must make do with other clues that a human would understand from looking at a sketch. This includes the knowledge that arrows are not objects themselves but rather information that describes or affects the objects in the sketch.

Identifying how objects in a sketch interact has been approached by annotations—for instance, specifying objects be drawn with terminals connecting them. In a 2-d rigid-body system the "terminals" are surface contacts. At first this appears trivial, but as demonstrated in Figure 1 work must be done to disambiguate the exact nature of the interaction in a surface contact. As Nielsen (1988) showed, the decomposition of a surface into qualitatively distinct regions depends upon mechanical constraints beyond just the surface shape.

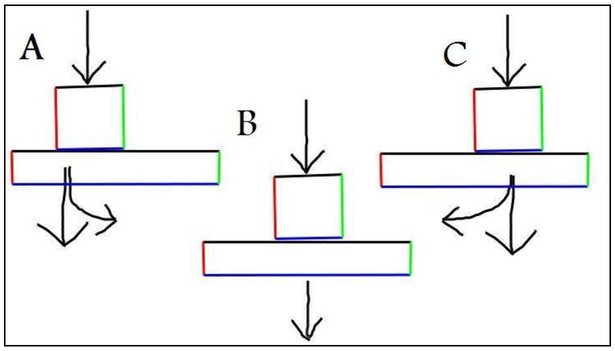

Figure 1: Slightly different positions of contact allow very different motions.

The problem of understanding the effects of these interactions is more easily solved if the representation generated by the first two steps provides the appropriate qualitative decomposition of surfaces. Next we explain how we help users generate sketches, how we construct the decompositions of surfaces, and how qualitative mechanics is applied to these representations to answer questions about the sketched mechanism.

The Sketch

The user draws the forces and objects of their mechanism as CogSketch glyphs. They label the glyphs with concepts from its knowledge base. Our system looks for four labels of glyphs in particular:

- `RigidPhysicalObject`
  Because we are working in a rigid-body domain, this category identifies all objects of interest.
- `FixedPhysicalObject`
  This category represents objects that are completely constrained from moving or rotating (e.g. the surface of the earth, walls).
- `ForceArrow`
  Force arrows are used to indicate external forces acting on the system. This includes global forces such as gravity.
- `RotOrigin`
  An annotation glyph, marking the point around which its parent object is free to rotate. An object with a `RotOrigin` is prevented from translating in all directions.

From a user interface perspective this might seem like a lot of required labeling. However, our system makes some simplifying assumptions. First, we take advantage of CogSketch's arrow recognition and assume any two or three stroke arrow glyph is a `ForceArrow`. Any annotation glyph that is not an arrow is assumed to be a `RotOrigin` for its parent glyph. Finally, all remaining glyphs are assumed by default to be `RigidPhysicalObject`. Thus the only specific labeling required of the user is the annotation glyph distinction and labeling fixed objects as `FixedPhysicalObject`.

After they are done drawing their sketch, the user can perform any of the following queries:

- Will object x move?
- Will object x rotate?
- What forces are on object x?
- Is the sketched system stable? (Will any object move?)

Performing a query begins the translation process.

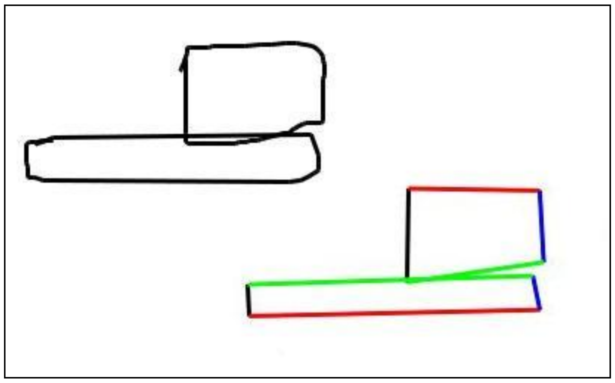

Figure 2: Blocks in a raw sketch (top left) are decomposed into idealized edges (bottom right), simplifying surface contact detection.

Translation to QM Representation

When prompted with a query, the system begins interpreting the sketch. It is here we solve the first two of the three problems addressed in this paper: identification of objects, forces, and their properties, and discovering regions of interaction.

Identifying Objects Forces, and their Properties. The first step is to know what objects and forces are depicted in the sketch, and determine their specific properties. For forces, these properties include direction and the objects they directly affect. For each object this includes identifying the surfaces of that object, and determining whether the object is fixed, fixed-axis, or free to move. If the object is fixed-axis then its center of rotation must be located.

First we check the labeling for each glyph in the sketch to see if they are a rigid object and/or a fixed rigid physical object. This gives us the fixed property. Then, each glyph representing a rigid object is decomposed into edges (Figure 2), and each edge becomes reified as one or more surfaces in our predicate representation. Every surface also has a normal vector pointing outward from the object and a q-vector towards the object's axis of rotation. The system is limited to processing polygons, so every edge will be a line segment. Thus we can infer that each edge will have only one normal but may have *multiple vectors towards the axis of rotation.* Since the edge is a line segment there will be up to five surfaces per edge. The cases of one, two, and three edges are demonstrated in Figure 3.

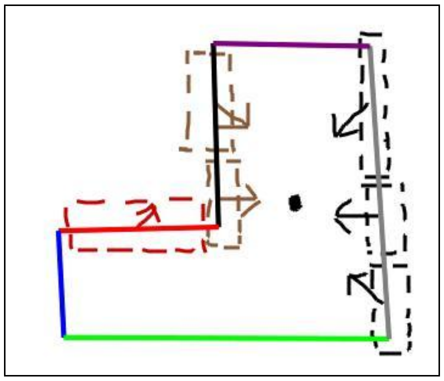

Figure 3: Straight edges of objects can be divided into up to five qualitative surfaces, each with a different q-vector in the direction of the axis of rotation.

If the axis of rotation is not given in the sketch, we assume a uniform density and choose the center of area of the shape as the axis of rotation. While multiple axes of rotation may exist over the course of a mechanism's operation, our system currently only analyzes one instant of time. Thus, we choose the axis active at the current moment indicated by the sketch and ignore the others.

Having finished with objects, we move on to forces. The force represented by each force arrow in the sketch is reified as either a force or a torque. If the force is global then a force is reified with a qvector matching the direction of the arrow. If the force arrow is on a specific surface then whether it creates a force or torque depends on the `OriginDir` of that surface. If the arrow is the inverse of its surface normal then a force is applied, otherwise a torque is applied in the appropriate direction. In Figure 3 applying a leftward force arrow to the top-right surface would create a counter-clockwise torque. Applying the same force arrow to the surface just below that would create a leftward force.

If the force arrow is an annotation glyph, a force or torque is added for each object that glyph annotates. If it is not an annotation glyph, the force is added for all objects in the sketch. This allows for global forces such as gravity to be input easily.

Discovering Regions of Interaction. This problem entails finding all surface contact relationships. This is done by first identifying all rigid objects which may be in contact. This step is straightforward since CogSketch automatically computes topological relations for each pair of glyphs in the sketch. For each pair of intersecting or connected objects, their edges are checked pairwise for contact. If two edge are nearly parallel, in close proximity to each other, and overlap by a significant amount (that is, not merely in a line one after another), they are considered in contact. If the edges contain multiple surfaces, the overlap is calculated and surface contact is only reified for the two surfaces which contain share the midpoint of the overlap.

Once all of the surfaces, surface contacts, and forces have been identified, the system is ready to make the inferences required to answer the user's query.

Answering User Queries

After the qualitative representation is complete, the system begins finding the answer to the user's query. We now are at the problem of understanding the effects of the interacting regions. The user query is passed to our backchainer, whose rules are an implementation of QM theory. These rules are written as Horn clauses in which the first statement is the consequent and the conjunction of the remaining statements is the antecedent. Some of the rules are listed here:

Constraining translation for fixed objects:
```
(<== (transConstraint ?obj ?dir)
     (isa? obj FixedPhysicalObject))
```

Determining motion constraint in a particular half-plane:
```
(<==(sufficientlyConstrained ?obj ?dir)
     (transConstraint ?obj ?dir1)
     (transConstraint ?obj ?dir2)
     (transConstraint ?obj ?dir3)
     (openHalfPlane ?dir ?dir1)
     (openHalfPlane ?dir ?dir2)
     (openHalfPlane ?dir ?dir3)
     (different ?dir1 ?dir2 ?dir3))
```
An open half-plane is defined in Nielsen's work as the set of qvectors within 90 degrees of a given direction, excluding those at exactly 90 degrees. So for direction Left, the open half-plane would contain directions Quad1 and Quad4 but not Up or Down. The above rule defines "sufficiently constrained" in a direction if motion is constrained in all directions in that direction's half-plane.

Tranferring constraint through surface contacts:

```
(<== (transConstraint ?obj1 ?dir)
     (hasSurface ?obj1 ?s1)
     (hasSurface ?obj2 ?s2)
     (surfaceContact ?s1 ?s2)
     (surfaceNormal ?s1 ?sn)
     (sufficientlyConstrained ?obj2 ?sn)
     (openHalfPlane ?sn ?dir))
```

Object 1 cannot move in direction `dir` if object 2 is constrained in all directions in that `dir`'s half-plane. Otherwise object 2 can move in one of those directions, allowing object 1 to move in `dir`.

Freedom is the absence of constraints:

```
(<== (transFreedom ?obj ?dir)
      (isa ?obj RigidOb)
      (isa ?dir 2DQVector)
      (evaluate ?x
       (CardinalityFn
        (TheClosedRetrievalSetOf ?dir
         (transConstraint ?obj ?dir))))
      (equals ?x 0))
```

Force + Freedom causes motion:

```
    (<== (transMotion ?obj ?dir)
          (force ?obj ?dir)
          (transFreedom ?obj ?dir))
```

The force predicate here is the net force on the object. In the current version this must be specified by the user when the direction of the net force is ambiguous.

Transfer of translation across surfaces:

```
(<== (transMotion ?obj2 ?d2)
      (hasSurface ?obj1 ?s1)))
      (hasSurface ?obj2 ?s2)))
      (surfaceContact ?s1 ?s2)))
      (surfaceNormal ?s2 ?sn)))
      (inverseVector ?sn ?invsn))
      (openHalfPlane ?invsn ?d1))
      (transMotion ?obj1 ?d1))
      (openHalfPlane ?invsn ?d2))
      (transFreedom ?obj2 ?d2)))
```

This rule stipulates conditions in which object 2 will move because of contact with another moving object, object 1. The `openHalfPlane` relation means that the two directions are within 90 degrees of each other. This allows an object to transfer motion through multiple directions if necessary.

Force applied to a surface via surface contact:

```
(<== (forceApplied ?s ?sn ?obj1)
      (force ?obj1 ?dir)
      (hasSurface ?obj1 ?s1)
      (surfaceContact ?s1 ?s)
      (surfaceNormal ?s1 ?sn)
      (openHalfPlane ?sn ?dir))
```

Force applied to a surface causing force on object:

```
(<== (force ?obj ?dir)
      (hasSurface ?obj ?s)
      (forceApplied ?s ?dir ?c))
```

Forces are also translated through other objects. In general, every force applied through a surface contact gets applied to the next object as a translational force if both of the following conditions hold:

1) The object is free to translate.
2) The inward normal of the contact surface points towards the object's axis of rotation.

In the version presented here, it is up to the user to resolve ambiguities in the forces. The work in progress includes rules that try to find the resultant vector of a set of forces, and resolve ambiguities by asking the user which forces are larger or by using the magnitude field of the force annotation in CogSketch.

Torque propagation is not yet implemented in the current version of the system but it will follow the same principles. These and other qualitative mechanics principles are all defined as rules. By backchaining through these rules, the system deduces what forces are acting on objects and whether they will move.

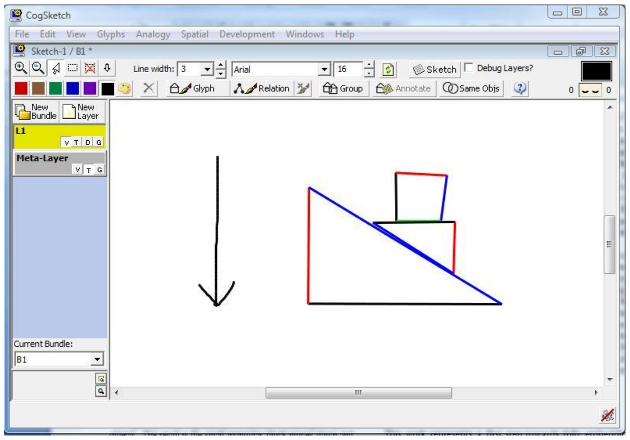

Figure 4: Two free blocks stacked on a fixed ramp. The arrow on the right represents a global downward force affecting all three objects. The result is the small triangular block moves down and right (quad 4) and the square block moves down.

The example shown in Figure 4 is based on an equivalent one in Nielsen's work. The sketch contains a ramp with two blocks stacked one upon another and one arrow pointing downward, drawn off to the left. The arrow is not an annotation glyph, and the ramp is labeled as a `FixedPhysicalObject`. When the user asks for the motion of all the objects in this sketch, the system begins to build its QM representation. First, each of the three non-arrow glyphs is decomposed into their respective edges, which become their surfaces. Since the force arrow is not annotating a specific glyph, its downward force is assumed to be affecting all glyphs in the sketch.

Next, the system searches for surface contacts by checking each pair of objects for contact or overlap. Using the topological relations calculated by CogSketch, the system finds overlap between the ramp and the triangular block and between the triangular block and the square. For both pairs it performs a line-line proximity comparison between their surfaces to find the surface contacts.

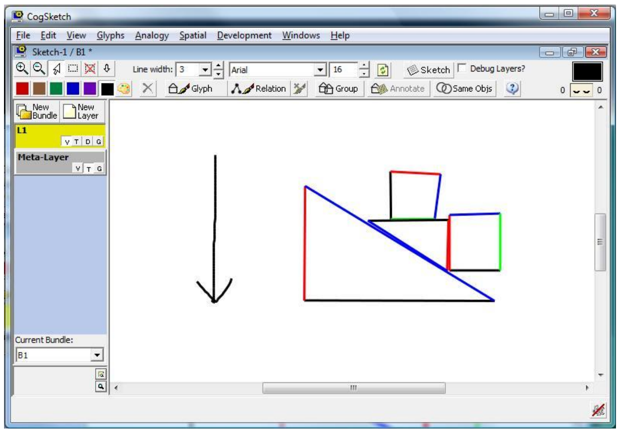

Figure 5: The two free blocks (Figure 4**) are now constrained by an additional block, labeled as fixed.**

With the surface contacts reified and the forces applied, the system begins backchaining to find any motion that each object possesses at the moment pictured. The ramp is a `FixedPhysicalObject` so it is stationary. The square block has a downward force on it, and because the triangular block is not completely constrained from moving in the downward half-plane, the square block will begin moving downward. The triangular block has a downward force and is free to move in the down-right direction; consequently, it does. Adding a stop block to the above example (see Figure 5) prevents the triangle from moving, and thus prevents the square block from moving.

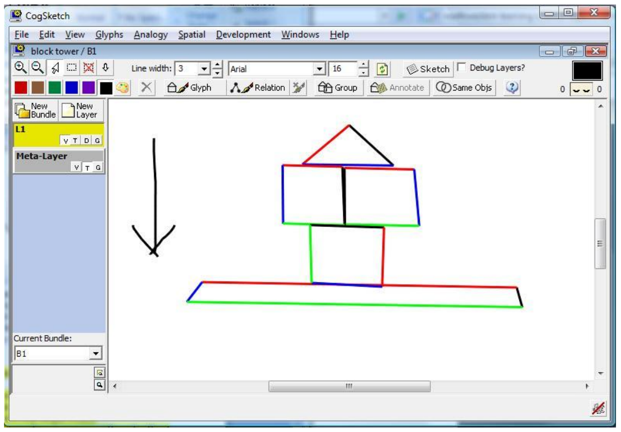

Figure 6: A stable tower of blocks on a fixed platform.

Figure 6 shows more transfer of motion constraints. The constraint is propagated upward from the fixed base to the topmost block. If the user were to draw this and query for motion, the system would return no motion. In this way, one can test a design for stability.

Related Work

Our approach of using shape edge decomposition, RCC8 relations, line-line proximity and overlap calculation is a novel solution to the problem of discovering and analyzing surface interactions between rigid bodies. Kurtoglu and Stahovich (2002) used line-line and other proximity pairs in a system to identify the type of connection between two sketched objects, with the goal of classifying those objects in categories such as rigid body or electric motor. Our system goes two steps further for rigid bodies, breaking them down into their individual surfaces and then determining exactly how the position and size of an overlap of two surfaces affects their motions.

Prior sketching systems for mechanical reasoning have relied on human input for the analysis of surface contacts. The QM theory on which our system is based had full propositional representations as its input. (Nielsen 1989; Kim, 1993) Later systems such as SketchIT (Stahovich et.al. 1998) required the designer to mark the important surfaces and build state machines describing their interactions. In the example in Figure 4 SketchIT would require the user to highlight the contact surfaces.

Progress has also been made in the area of automatically recognizing the objects in sketches (Alvarado & Davis, 2004; Kurtoglu & Stahovich, 2002). By eliminating the need for extra human input we have moved closer to a sketch-understanding system that can reason deeply about hand-drawn sketches.

Discussion and Future Work

This work represents a first step towards fully embedding qualitative mechanics in systems that reason with hand-drawn sketches. This required tackling the problems of identifying objects, forces, and their properties; discovering the interactions between said objects and forces; and finally, computing the exact effects of these interactions. The advances which enabled us to do this include using a combination of shape decomposition, RCC8 relations, and line-line proximity and overlap calculation, allowing us to identify the different surfaces of two-dimensional objects, their areas of contact, and compute the consequences of those interactions.

Our goal is to have a complete, robust qualitative mechanics reasoner that can operate over a wide range of hand-drawn sketches. We see two key next steps. First, handling curved surfaces is important for many kinds of designs. This presents new challenges to segmentation. Second, the system currently only reasons about instantaneous force/motion transfers. Reasoning about motion over time, including automatically deducing plausible changes in contacts (cf. Nielsen 1988), is also important. Longer term, we plan to extend the system to handle 3D shapes, flexible bodies, laminar flow situations, and fluids as well as rigid bodies.

Acknowledgements

This work was supported by NSF SLC Grant SBE-0541957, the Spatial Intelligence and Learning Center (SILC).

References

Alvarado, C., Davis R. 2004. Multi-domain sketch understanding. Massachusetts Institute of Technology, Cambridge, MA.

Alvarado, C., Davis R. 2007. Resolving ambiguities to create a natural computer-based sketching environment. In *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 2007 courses*. San Diego, California.

Cohen, P., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L. and Clow, J. 1997. QuickSet: Multimodal interaction for distributed application. In *Proceedings of the Fifth Annual International Multimodal Conference,* 31-40. Seattle, WA.

Cohn, A. 1996. Calculi for Qualitative Spatial Reasoning. In Calmet J., Campbell J. A., Pfalzgraph J., Verlag S. (Eds.), *Artificial Intelligence and Symbolic Mathematical Computation*, LNCS 1138, 124-143.

Forbus, K., Nielsen, P., and Faltings, B. 1991. Qualitative spatial reasoning: The CLOCK project. In *Artificial Intelligence*, 51(1-3).

Forbus, K., Lockwood, K., Klenk, M., Tomai, E., and Usher, J. (2004). Open-domain sketch understanding: The nuSketch approach. Proceedings of the AAAI Fall Symposium on Making Pen-based Interaction Intelligent and Natural, October, Washington, D.C.

Kim, H. (1993). Qualitative reasoning about fluids and mechanics. Ph.D. dissertation and ILS Technical Report, Northwestern University. Evanston, IL.

Nielsen, P.E. (1988). A qualitative approach to rigid body mechanics. (Tech. Rep. No. UIUCDCS-R-88-1469; UILU-ENG-88-1775). Urbana, Illinois: University of Illinois at Urbana-Champaign, Department of Computer Science.

Stahovich T.F., Davis R., Shrobe H. 1998. Generating multiple new designs from a sketch. In *Artificial Intelligence* 104 (1998) 211–264.

Kurtoglu T., Stahovich T.F., 2002. Interpreting Schematic Sketches Using Physical Reasoning, In *AAAI Spring Symposium on Sketch Understanding*, AAAI Technical Report SS-02-08, 78-85.