There are two times in his life when a man should not speculate: when he can afford to, and when he cannot.
— Mark Twain

Chapter 10

Evaluation and Future Work

In this thesis, I have presented a differential approach to graphical interaction: a systematic approach to realizing graphical manipulation based on numerical constraint techniques. In this concluding chapter, I will review the contributions of the thesis, evaluate the work in light of the goals, and suggest some directions for future work.

10.1 Contributions

The central contribution of this thesis is to develop and demonstrate an approach for realizing graphical manipulation based on numerical constraint techniques. The thesis extends previous work in interactive physical simulation into a general approach for graphical manipulation with contributions in the following general areas:

- abstractions for graphical interaction;
- numerical constraint techniques for graphical applications;
- implementation techniques for numerical computations;
- software architecture for graphical applications;
- interaction techniques;
- graphical applications.

This section reviews the specific contributions of the thesis.

I feel that the biggest contribution of the thesis is not any of the individual pieces, but rather, how all the pieces can be fit together. To create the differential approach requires a new set of abstractions that define graphical manipulation as constrained optimization, mathematical methods to solve the constraint problems, implementation methods to address the practical issues in employing mathematics in interactive systems, and a software architecture to enable applying the approach and building applications. Only with all of these pieces can novel interaction techniques and applications be created. Here we discuss the specific contributions in more detail.

10.1.1 New Abstractions for Graphical Manipulation

At the core of the differential approach is a view of graphical manipulation as a constraint solving problem. Interaction techniques are created from connectors, controllers, and the passage of time.

- **Equation solving as a general abstraction of graphical manipulation:** the differential approach treats graphical manipulation as an equation solving problem. Previous work has used numerical equation solving and optimization for interactive control in specific cases and non-interactive control of geometric objects with general constraints. However, no one has previously explored numerical optimization methods as a general mechanism for implementing graphical manipulation.

- **Connectors and controllers as basic abstractions for building graphical manipulation techniques:** the concise set of abstractions provided by the differential approach are quite different than any previous ones. They can be combined and composed. One key feature of the interactive controls in the differential approach is that they are combinable: if we can control A, and we can control B, then we can control A and B simultaneously.

- **Application of numerical constraint solving to interaction technique design:** interaction techniques are described by sets of controls and constraints that are enabled and disabled at appropriate times. This leads to concise, directly executable descriptions of classes of interaction techniques that have been difficult to specify, such as 3D widgets. Freeman-Benson's Kaleidoscope '91 system [FBB92] defined interaction techniques and widgets by dynamically switching sets of constraints and previous work such as Olsen and Allan [OA90] and the Garnet system [MGD$^+$90] have used constraints to describe the behavior of 2D widgets. However, by employing numerical techniques, the differential approach permits a richer set of constraints including inequalities, to be used to specify widgets in a way that automatically defines its inverse mapping from input to variable.

- **A method for describing interactions independent of the underlying representations of the graphical objects:** interaction techniques are defined in terms of the objects' attributes. The equation solving mechanisms map changes to the underlying parameters.

10.1.2 Numerical constraint methods for graphical applications

The use of numerical constraint techniques for graphical applications began with the relaxation techniques used in Sketchpad[Sut63]. More recently, there has been interest in the related methods of non-linear constrained optimization, non-linear equation solving, and physical simulation. The problem solved by the differential optimization is very similar to the problems solved at the core of these methods. The solution techniques discussed in this thesis are variants of standard mathematical techniques for solving relatively standard problems.

The methods used in the differential approach are the methods used for interactive simulation. The techniques described in [WGW90] have been extended by applying some standard mathematical techniques, such as damping (Section 3.2.1). This thesis brings these methods to the domain of user interface design and construction. Previous work attempting to apply numerical controls to the design of interactive systems has been plagued by starting from inferior mathematical foundations.

- **Demonstrations of the practicality and utility of the differential approach:** the running systems show that general controls for graphical manipulation can be implemented, and perform acceptably on existing computers.

- **Application of damping to constraints in interactive graphics:** the differential approach uses damping methods (Section 3.2.1) to provide stability in ill-conditioned cases, and to handle over-determined cases. These singular cases have plagued many previous systems. Damping methods are well known in the optimization community, and have been applied to similar domains, such as animation and robotics. This thesis introduces the use of selective damping to achieve constraint-hierarchies.

10.1.3 Implementation Techniques for Numerical Computations in Interactive Systems

Another contribution of this thesis is the methodology for solving the numerical computations required for constraint approaches in a manner that addresses the issues of interactive systems. The work of this thesis refines the earlier work on interactive physical simulation.

- **Snap-Together Mathematics:** the work described in Chapter 5 is a descendant of earlier tools for dynamic function composition and derivative evaluation. The basic ideas behind Snap-Together Mathematics, such as explicit representation of expression graphs, compile time definition of new function blocks using a symbolic math system, and sparse-vector-passing, forward mode, automatic differentiation remain from its earliest ancestor described in [WK88]. Several details, such as how variables are handled, have been refined with evolution. The current implementation is the first to be incorporated into a higher level toolkit, and the first to exploit the generality in a range of applications. Snap-Together Mathematics and its ancestors are very different from most other automatic differentiation tools which focus on large, dense problems statically defined at compile time[BGK93].

- **An encapsulation of numerical constraints:** Snap-Together Math's simple protocol for connector outputs, combined with its objects for constraints and constraint engines, provide facilities for defining (and solving) constraints on equations without seeing the underlying mathematics. The `ConstEngine` class has been used with other solvers, such as Newton-Raphson.

- **Generalization of performance methods for numerical constraints in interactive systems:** the performance methods of Chapter 4 have they been used in previous systems. However, the work of this thesis shows how all can be integrated in a general fashion. In particular, the scatter–gather mechanism of Snap-Together Mathematics, coupled with variable selection mechanisms and variable merging mechanisms provides a generalized substrate for speeding numerical computations through switching between explicit and implicit constraints and partitioning.

10.1.4 Software Architecture for Graphical Applications

This thesis described Bramble, an object-oriented toolkit based on the differential approach. The main contributions of Bramble are:

- **A graphics toolkit encapsulating numerical constraint methods:** Bramble encapsulates the differential approach in a manner that provides services inside the abstractions of an object-oriented graphics toolkit. In fact, Bramble permits building graphical applications with constraints without referring to anything other than the typical abstractions of an object-oriented graphics toolkit. This is important because designers of interactive systems seem to be resistant to learning about mathematical techniques.

- **Demonstrations of the impact of the differential approach on application architecture:** by providing generalized methods for mapping between controls and parameters and by permitting connectors to be attached without knowing what is behind the connector, the differential approach fosters modularity, parameter independence and common code. This is seen in Bramble since objects, constraints, and interaction techniques can all be created independently and hooked together as needed.

- **A graphics toolkit that facilitates providing general constraints and controls to users:** Bramble aids applications in providing constraints and multiple controls to the user. Previous toolkits have used constraints as an internal abstraction to aid the programmer, but Bramble's focus is providing them as a user level service.

- **Use of a time continuous input model in a graphics toolkit:** the differential approach provides a continuous model of time, with events integrated as instantaneous impulses. For a graphics toolkit, the model allows simultaneous asynchronous actions to be handled without explicitly programming them into each event handler or providing for multi-threaded flow of control. A fundamental difference between this strategy and standard approaches is that handlers for continuous actions such as dragging are instantaneous event handlers, they do not remain active over the duration of the operation, for example from mouse down to mouse up.

- **An object oriented graphics toolkit specifically designed for prototyping of 2D and 3D applications and techniques.** Much of the rationale for Bramble is based on the intention of its use for prototyping a variety of interactive graphical applications. This led to exploring a different point in the design space of interface toolkits. Because the art of toolkit design, especially for 3D, is still evolving, explorations of new points in the design space are still useful. Interestingly, other 3D-only graphics toolkits with emphasis on rapid prototyping, including UGA [CSH$^+$92], MR [SLGS92], Alice [PT94], and VB2 [GBT93], have independently made some similar choices.

10.1.5 New Interaction Techniques

In order to demonstrate the differential approach, this thesis presents a variety of interaction techniques, many of which are novel enough to be considered contributions. Without user testing, it is in many cases difficult to claim that these are clear improvements over previous methods. In fact, some of the new techniques may be unusable in practice. However, I think that a few are interesting. Also, I think the differential approach will lead to many new good interaction techniques because it permits providing the advantages of direct manipulation, such as continuous feedback and kinesthetic coupling, to a broader class of interactions and objects.

- **Through-the-lens camera controls:** these techniques permit manipulating virtual cameras and graphical objects by controlling and constraining points in the image seen through the cameras lens. The methods can be used to address important problems in computer graphics, such as the image registration problem of Section 8.2.4.

- **Generalized snapping:** (Section 8.4.1) the snapping controller permits any attribute of a graphical object to be snapped to precise values, even if they are away from the cursor.

- **Interactive manipulation of color to control object and light geometry:** the use of the lighting model equation as a control permits manipulating light color, light position, surface colors, and surface geometry. Previous systems contain special case code for such techniques for controlling light colors and surface properties.

- **Generalization of interactive control of reflections:** the methods of Section 8.1.6 permit constraining and directly controlling the positions of reflections on planar and curved mirrors. The only previous technique was restricted to manipulating object translations by dragging reflections in planar surface[HZR$^+$92]. The reflection controls also provide a generalization of the specular highlight positioning methods of Poulin and Fournier[PF92], permitting the positions of specular highlights to be constraints, and to control the geometry of the objects in addition to that of the light source.

- **Generalization of the use of shadows as controls:** Section 8.1.6 introduced techniques to constrain and control object geometry and light positions by controls placed on shadows on both plane surfaces and other objects. This permits a generalization of the Brown interactive shadow widgets[HZR$^+$92] and Poulin and Fournier's positioning of lights by moving shadows[PF92].

- **Scene composition by mixing and matching controls:** (Section 9.7) while some of the scene composition controls have appeared as special purpose hard-coded controls, no previous system has allowed interactively specifying and combining constraints and controls for camera positioning, lighting specification, and object manipulation.

- **Augmented snap-dragging:** Briar (Section 9.1) introduces an extension of previous constraint inferencing techniques that permits automatic generation of a variety of geometric constraints from drawing operations. Conversely, the drawing methods of snap-dragging [BS86, Bie89], are augmented to also specify constraints.

- **Techniques for displaying and editing constraints in a constraint-based drawing program:** Briar (Section 9.1) introduces a number of novel methods for displaying and editing constraints in a drawing program. It introduces a visual syntax for displaying constraints "in-place" in a manner that parallels their definitions, and two methods for deleting constraints.

10.1.6 Graphical Applications

The primary contribution of the applications described in this thesis is to demonstrate the viability of the differential approach, and the flexibility of the implementation. Almost all of the applications are re-implementations of other applications using the techniques of the differential approach.

- **An interactive Tinkertoys application:** the Tinkertoys application permits interactive assembly and experimentation with 3D objects mimicking the real toys, including their kinematic behavior. Previous systems, such as those by Surles [Sur92c] and Schroeder [SZ90], either do not provide for interactive editing of the objects or they do not permit the simulation of the kinematic behavior.

- **User defined shapes with constrained behaviors:** the `define-shape` facility in Bramble permits end users to dynamically define new types of shapes in a 2D drawing program, and have these objects be manipulated without specifying the inverse transformations. Some systems, like the commercial drawing package Visio [Sha93], permit users to define shapes with equations, but do not allow these shapes to be manipulated except by directly controlling the parameters, and do not allow the use of constraints in defining shapes.

10.2 Evaluation

This thesis takes a step towards the goal of improving the quality and range of interactive graphical applications, by providing a substrate on which interaction techniques and applications can be built.

10.2.1 Basic Questions

The central premise of this thesis is that the differential approach can provide an implementation of direct manipulation that uses numerical constraint methods to map controls to object parameters in a systematic fashion, and that several issues in direct manipulation interfaces can be addressed by such an approach.

The differential approach does provide a way to implement direct manipulation, as shown by the recreations of other direct manipulation interfaces using the approach. The approach uses a single numerical solver to map a wide variety of controls to a wide variety of objects. This thesis presents a sufficient set of mathematical and implementation techniques to realize the approach, as illustrated by the prototype implementations. These prototypes show that the methods have acceptable performance on current computers. Traditional direct manipulation operations take only a few milliseconds of computation per frame, and reasonable sized constraint problems can be handled at interactive rates (performance of the prototypes is discussed in Appendix B).

The examples show that the differential approach is interesting. The approach can create desired interfaces, often in ways that are more flexible and general than traditional implementations. The approach also permits the creation of new interfaces that would be difficult or impossible to create with conventional means. For example, despite the utility of methods like the image alignment technique of Section 8.2.4, users are typically provided with controls less suited for the task because of the difficulty in deriving the mapping from the controls to the underlying parameters. While the special-case example of the table controls for image alignment might have been derived by hand, the more general approach of mixing and matching though-the-lens controls could not have been. In fact, without the view that anything that can be computed can be a control provided by the differential approach, techniques like through-the-lens camera control might never have been dreamed of.

10.2.2 Evaluating Abstractions

The abstractions for creating interfaces provided by the differential approach have many desirable properties:

- the set of abstractions is small;
- they provide concise definitions for many interaction techniques;
- they describe manipulation without reference to the underlying parameterization;
- they can be combined and composed;
- they allow modularity by permitting objects, constraints, and controls to be defined independently and hooked together as needed;
- they map directly to the data structures in the implementation;
- they foster a view of manipulation that helps lead to novel interaction techniques.

The abstractions have their problems and limitations, which will be surveyed in Section 10.2.7.

10.2.3 Evaluating Differential Techniques

Mathematical and numerical methods are crucial to the differential approach. Without methods that solve the constrained optimization problems in a manner that meets the demands of interaction, an approach to graphical manipulation based on constrained optimization is impossible. The prototypes show that methods exist that permit the differential approach to be realized. Section 1.1.4 introduced a number of goals for the methods. Here, we review them:

1. **Flexibility in the types of controls:** the methods permit the use of arbitrary differentiable functions over continuous valued variables as controls. The range of controls discussed in Section 8.1 illustrates this flexibility. In practice, the range of controls is limited by numerical considerations.

2. **Freedom to combine controls arbitrarily and dynamically:** the solver handles arbitrary numbers of simultaneous controls, without concern for what they are. This allows controls to be combined arbitrarily. In contrast to many other methods, there are no restrictions on acyclic dependencies or existence of a procedural solving plan. Adding a new control requires only passing more equations to the solver. The flexibility in combining controls is an essential part of the approach.

3. **Keep the good properties of direct manipulation:** the differential approach allows the creation of interfaces with the good features of direct manipulation such as rapid feedback, continuous motion, and kinesthetic correspondence.

4. **Choose the "best" solution in under-determined cases, and find a "reasonable" answer if there is no exact solution:** in underdetermined cases, the optimization objective defines the "best" solution. The damping methods of Section 3.2.1 handle overdetermined situations, providing robustness in ill-conditioned cases, allowing for redundant controls, and blending conflicting controls.

5. **Provide freedom in picking representations independent from user concerns:** because controls are defined in terms of object attributes, rather than their internal parameters, programmers have the freedom to select representations based on other concerns.

6. **Allow a standard procedure for defining new controls that minimizes the amount of difficult mathematical work in defining a new type of control:** to serve as a control, a function must compute its value and the derivatives of its value with respect to its variables. The derivatives can be computed using automatic methods given the attribute's function, which must be known anyway.

7. **Allow a solving mechanism that is general purpose and encapsulatable:** the solving mechanism for the differential approach works on a set of functions, variables, and controllers that specify desired derivative values. Therefore it applies to problems of controlling differentiable functions of continuous variables. Snap-Together Mathematics encapsulates the solving mechanism, providing the solver as a black box object whose details are hidden from the programmer using the toolkit. Bramble demonstrates this: it does not even provide the application programmer with the ability to look at solver internal data structures.

8. **Work in a variety of domains:** the approach is not specific to any particular domain. A wide range of graphical applications can make use of it.

9. **Be fast and scale well:** as detailed in Appendix B, the prototype implementations show that the methods can provide sufficient performance on the present generation of computers. The $O(n^2)$ computational complexity is a potential

problem. At the present time, models with 50-100 simultaneous constraints can be handled.

10. **Not require sophisticated numerical techniques:** the methods only require numerical algorithms to solve easy forms of standard numerical problems. Standard algorithms from textbooks were sufficient for the prototype implementations. Fancier numerical algorithms, such as an efficient sparse singular-value decomposition solver or a better ODE solver, may improve performance.

10.2.4 Evaluating the Interaction Techniques

This thesis provides tools for creating graphical interaction techniques. The tools do not necessarily lead to better interfaces, in fact, the tools give interface designers new ways to create bad interfaces, for example by applying controls "behind the users back." However, the differential approach would not be interesting if it only permits the creation of bad interfaces.

The differential approach can be used to create good interaction techniques. Although no formal evaluations were done, the techniques demonstrated with the approach include several time-tested standard interfaces, as well as some interfaces that apply direct manipulation to problems that it previously could not be applied to.

Using the differential approach to recreate existing interaction techniques is not necessarily overkill: the methods provide a new way to define these techniques in a representation-independent fashion and permit the techniques to be coupled with other constraints. The approach can also make implementation easier as it does not require the programmer to derive the mappings from controls to parameters.

A more exciting use of the differential approach than recreating existing interfaces is creating of new interaction techniques and constraint-based interfaces. I believe that the differential approach can often lead to interesting new techniques for graphical manipulation tasks. The approach has the benefits that:

1. it preserves the continuous motion animation and rapid feedback properties generally desired in graphical interfaces;
2. it permits the creation of controls directly relevant to the users task;
3. it speeds experimentation with new control types as the mathematics does not need to be derived for each new one;
4. it speeds experimentation by allowing controls to be combined.

The image registration interface of Section 8.2.4 exemplifies all these points. It provides an interface to an important problem, for which conventional interactive controls are not convenient. The through-the-lens controls are directly relevant to the task, as they permit directly specifying where in the image a point in the 3D scene should appear. The creation of the through-the-lens control required knowing only functions that were needed for drawing[1]. Initial experimentation with the alignment technique merely required applying 4 through-the-lens controls. Since through-the-lens controls were already available in the toolkit, the only new code required to create the interaction technique was code to place the video image as the screen background.

The image alignment technique would have been very difficult to create with traditional approaches for implementing direct manipulation. Deriving the mapping from controls to camera parameters is extremely difficult, as can be seen in the photogrammetry literature mentioned in Section 2.4.2. The more general approach of mixing and matching controls, used in Showoff (Section 9.7), would be impossible because such derivations must be done for every new combination of controls.

Unfortunately, the range of interaction techniques that I have developed with the tools of this thesis is not as extensive as I had hoped. Much more of the effort of the thesis research went into tool building. Through-the-lens controls were the biggest success. Most other techniques, such as generalized snapping, the scene composition controls such as shadows and reflections, and 3D widgets were not explored sufficiently well to make persuasive evaluations of. The success of color controls, something I had high hopes for initially, was thwarted by technical problems. Numerical issues cause the lighting equations not to serve well as controls. Further exploration of these controls may find solutions to these difficulties, especially now that the solver has damping correctly implemented.

Generalized snapping, described in Section 8.4.1, is another interaction techniques that I was unable to explore sufficiently. Some issues are described in Section 10.3.

10.2.5 Evaluating the Architecture and the Prototypes

The prototype implementations described in this thesis consist of a few major parts. Code sizes listed are for C++ source files, with comments, not counting headers.

- **Math and Data Structures Library —** (5K lines of C++ in 22 files) My C++ math and data structures library includes various basic mathematical elements such as matrices, vectors, and ODE solvers as well as more general data structures such as lists, queues, and hash tables.

  The math library has been used extensively over the past 5 years in a variety of applications. Particularly good design features were the object-oriented encapsulation of sparse matrices and ODE solvers. The code is very portable, and runs on a range of machines from notebook PCs to high performance workstations.

[1] Arguably, the viewing transformations are in the graphics toolkit, and therefore not known to the applications programmer. However, under this argument, through-the-lens controls are also in the toolkit, so their functions are not known to the programmer.

It is significant to note that despite the more than 50,000 lines of C++ code in the complete system, for reasonable sized problems, the majority of time is spent in a few inner loops of the routines in the library. A fast and robust implementation of basic mathematical data types such as vectors and matrices is crucial to the differential approach. Fortunately, such routines are not hard to create and can be nicely encapsulated in a manner that allows reuse.

- **Snap-Together Math Library —** (3K lines of C++ in 16 files, several hundred automatically generated by BlockMaker, 300 lines of Mathematica to implement BlockMaker) The Snap-Together Mathematics library is discussed in Chapter 5. It supports function composition and evaluation, the differential solver, and a number of predefined function blocks.

  Snap-Together Mathematics has proven to work really well. The tool has evolved since the earliest versions, but the basic design is the same. Sparse sparse vector passing, global time-stamps, and the minimal protocol have all remained. The library has been used for a number of applications besides Bramble, including Spacetime motion control, physical simulation, and statistical data fitting.

  The biggest successes of Snap-Together Mathematics are the places where it is the simplest, such as the minimal protocol and the global time-stamp cache validation. Better support for features like partitioning and representation switching should be easy to add. One design flaw is that Snap-Together Mathematics objects, in particular function blocks, are a bit "heavy." That is, they can be expensive to create and destroy.

- **Whisper Interpreter —** (9K lines of C++ in 18 files, 3K of which were generated semi-automatically by EMACS macros to implement primitives for Snap-Together Mathematics and GL, 800 lines for an interface to an image processing library) The Whisper interpreter is a C++ library for embedding in interactive applications. The library contains interfaces for Snap-Together Mathematics, the GL graphics library, and a small image processing library. There are approximately 400 built-in primitives and 20 predefined types, not including what Bramble adds.

  Whisper has been a very useful tool for the research described in this thesis. It has been particularly pleasant to program in and to use as a basis for Bramble. Because it was designed for embedding, it provided the right set of functionality to the systems it was put into. It is extremely easy to extend. The simple dynamic object systems is particularly well suited for the kinds of exploratory programming done in an experimental toolkit like Bramble.

- **Bramble —** (40K lines of C++: including 15K lines of C++ for basic parts including some automatically generated code, 5K lines C++ for the 2D library including automatically generated code for a variety of objects, 7K lines C++ for the 3D library, 7K lines of automatically generated function blocks, 700 lines of Whisper library code) Bramble is the C++ interactive graphics toolkit described in Chapter 7.

Bramble has been an interesting testbed for the development of the differential approach. It has facilitated experimentation with the variety of interaction techniques and applications discussed in the thesis. I think the design of Bramble is quite solid. However, the implementation itself was hastily put together, and shows signs of its unusual evolution, for example before Whisper, other mechanisms for many of the object services had to be provided and remnants of these still haunt Bramble.

The abstractions of Bramble were sufficient to create a variety of applications without bypassing the approach. This is best illustrated by Toys which contains many features, but is written entirely in Whisper. Bramble is missing many of the features of standard toolkits, such as text handling. This makes it hard to build complete applications.

- **Applications —** (230 lines of Whisper for `Boxer`, 800 lines of Whisper for `Mechtoy`, 500 lines of Whisper for `Poly`, 500 lines of Whisper for `ShowOff`, several hundred lines Whisper for `Showoff` demos, 900 lines of Whisper for `Tinkertoys`) The complaint with the applications is that there are too few, that they are too small, and not "complete" enough. They do demonstrate the differential approach. However, there is much more to a graphical application than manipulating graphical objects. I have been disappointed by my inability to find a "killer" 3D application.

I believe that the architectural organization of the system works out very well. Snap-Together Mathematics provides a good degree of separation between the solver and the client application. It also allows objects to be defined independently, yet still connected together, by providing a common protocol.

Besides Snap-Together Mathematics, the most significant architectural decision in Bramble was the use of Whisper. I believe that this is a good strategy. The dynamic object system is extremely useful in a graphics toolkit. The facility for rapid prototyping is extremely useful in an experimental system. The embedded interpreter approach also facilitated additions to the toolkit: each new feature or object type did not require extensive installation into all of the applications. Instead, each new feature simply defined new primitive function calls that could be called as needed from applications, or even interactively from the interpreter prompt. The same functionality might have been attainable if a development environment such as LISP or Scheme were used, however, such environments did not offer the floating point performance of C++, the ease of transporting executables, the availability of graphics and numerics packages, nor were good compilers available at the time that the project was begun.

As in the process of constructing any system of this size, a lot of design decisions were made along the way. Some of the best design decisions that I made were (in no particular order):

- the simple protocol for Snap-Together Mathematics;
- using an embedded interpreter in the toolkit;
- using the Whisper object system for graphical objects;
- half sparse matrices;
- using a conjugate-gradient solver;
- the scatter gather handling of state variables.

Some of the worst design decisions I made were:

- using the simple GL event model of handling events globally, rather than handling events on a per-window or per-object basis. Also, GL does not time-stamp events, nor record the mouse state when an event occurs. This causes problems when event handling is delayed as the mouse may move between when the event happens and when the event is processed;
- having a single world, rather than explicit scene graphs like Inventor [SC92]. This makes it impossible to edit multiple documents, and difficult to show different data in different views.
- creating my own widget library. Many toolkits containing basic widgets such as buttons and sliders are available. Using these would have given more attractive looking widgets, and probably some richer functionality such as file-selector dialogs.

10.2.6 Experience with the Implementations

Because of the large overhead of making tools available to others, an explicit decision was made in the research plan for this thesis not to make the tools available to others.

After many requests, I have made the Snap-Together Mathematics library publicly available by anonymous FTP access. Because I did not offer support or documentation, I suspect that few people have made extensive use of it. A small number have reported successful experiments built with Snap-Together Mathematics. One CMU undergraduate student was able to build a portable 2D version of Point Tinkertoys that ran under the X window system, without learning anything about the constraint mathematics. I also know of at least three re-implementations of Snap-Together Mathematics by researchers who had access only to the papers. These researchers were able to build 3D constraint demonstrations.

Bramble was not offered to others because of the amount of support it would require. Because of this, I cannot make any claims about how "usable" Bramble is. Personally, I find Bramble a joy to use. Part of this may be attributable to the fact that I designed Bramble based on my personal tastes and program development style. The interpretive environment and high level language of Whisper extremely appealing compared to C++. However, I believe that a good part of Bramble's attraction is how well the abstractions fit the needs of graphical application development.

Other than myself, the main users of the applications in Chapter 9 have been people who have "taken over the drivers seat" during demos. This has allowed informal assessments of how users react to interaction techniques created with the differential approach. Such users are atypical: since they are only playing with the applications for a few minutes, their usage patterns are different than those of users' attempting to solve real problems. For example, many people permitted to use the systems immediately to try to torture the solver by making impossible connections to see what happens.

Response to the applications has been very positive. Few complain about the lag between the pointer and the object being dragged. In fact, many comment on how they like the feel of it. Some things, such as dragging the spiral of Section 8.1.1 generally require a little practice before people can achieve the effects that they intend. However, almost everybody has been able to figure it out without instruction. There is little else to compare these systems to in terms of usage.

I am consistently impressed by how quickly people learn to use the mousepole after getting past the initial issue of using multiple buttons on the mouse simultaneously. In general, people seem to like the Bramble standard 3D interface. Part of this may be attributed to relative novelty of interactive 3D graphics, and to the attention to aesthetic details such as the use of shadows and the selection of colors.

10.2.7 Limits and Drawbacks

The drawbacks of the differential approach, as seen in this thesis, fall into three categories:

1. fundamental limitations or drawbacks of the approach;

2. limitations and drawbacks of the techniques that should be resolved by future work;

3. artifacts of the prototype implementations.

The fundamental limitations of the approach are:

- **The differential approach is time continuous.** Things do not happen instantaneously. While this is generally a feature, some times we want objects to jump to their destinations, for example, when a new object is created.

- **The differential approach only is applicable to continuously valued parameters and attributes.** The numerical methods do not apply to discrete data. Some alternate mechanisms, such as propagation constraints, must be used.

- **The methods do not scale as well as those for other approaches.** Without restricting the problems, the complexity bound of $O(n^2)$ seems to be unavoidable.

Some major limitations that I think can be addressed in future work are:

- **scalability:** While $O(n^2)$ is the limiting factor, methods to reduce n and the constants can allow solving larger problems. Implicit constraint methods can potentially enhance scalability considerably, as demonstrated in the Tinkertoys application. Also, it is unclear how large the number of controls a system must handle. For predefined objects and interaction techniques, the number of controls will be a (usually small) fixed number (e.g. it will not grow with problem size). However, with a constraint-based interface, a user may create arbitrary numbers of controls. But, this number may not be unbounded as eventually, the limits of how much simultaneous behavior a user can comprehend may be reached.

- **precision:** Methods for more precisely achieving the user's goals will facilitate many applications. Generalized snapping is a first attempt.

- **generality:** As described in Section 8.1, mathematical considerations cause some functions to work better as controls than others. Some formal characterization for which controls which and what do not would further simplify the construction of interfaces.

- **real-time:** Adding a coupling to real (e.g. wall-clock) time would increase the expressibility of the abstractions. Interface designers would gain control over the time constants involved in the approach.

- **integration:** The differential approach applies only to the continuous motion dragging parts of applications. Therefore, integration with other techniques, such as propagation constraints, to handle other parts of applications is essential.

- **hierarchy:** The methods for differential constraints provide only two levels of constraint hierarchy. The work of Borning et al.[BFBW92] shows the usefulness of more levels.

Many of the problems in the prototypes are artifacts of the implementations, rather than problems with the approach. They fall into three general categories:

- **Bad design decisions:** Several bad design decisions were listed in Section 10.2.5.

- **Artifacts of evolution:** The prototypes evolved from earlier versions, so in many places there are leftovers from early versions. Bramble's property sheets and Snap-Together Mathematics's confusing mechanisms for soft controls and generalized snapping are two examples.

- **Incompleteness:** The prototypes were built in time to finish a thesis, so many features that are not essential to the points of the thesis have been omitted, such as text-handling in Bramble.

10.3 Directions for Future Work

This section discusses some topics to extend the differential approach, make effective use of it, or repair its deficiencies.

For this thesis, a number of prototype implementations were constructed. An entire class of future work involves the creation of more "industrial strength" tools and applications that can be widely distributed. For Snap-Together Mathematics, this may mainly involve completion of some partially implemented features, documentation and support. However, for Bramble, it is probably not worthwhile to go through the effort of making a distributable tool since too many design decisions were made with a "get it done for the thesis" attitude. This section focusses on future work on the differential approach more generally, rather than on the specific artifacts of the toolkit.

10.3.1 General Issues

The differential approach's model of time does not correspond with real time. This limits what can be expressed with the approach. For example, it makes it impossible to specify desired rates in a meaningful way. Some mechanism for correlating simulation time and real time would be a useful addition to the approach. However, it would require tackling a number of difficult technical issues in synchronization, especially with the variability of the solving methods. This issue will also be increasingly important in future multi-media applications. Faster processors might help with synchronization issues since programs will have the possibility of finishing their work quickly and then waiting for synchronization. However, such "busy waiting," is not the most effective use of processor cycles, and fails to handle cases where the fast processor's performance is needed to handle larger problems.

Future high-performance computers will make the lack of real-time coupling difficult because things may happen too fast. It will be important to provide the interface designer with some control over the time constants of the interactions.

The approach lacks a good characterization for what makes a function work well as a control. The rules of thumb from Section 8.1 need to be formalized. If the characterization could be sufficiently codified it would allow an even more automated tool for creating new controls to be created.

Precision in manipulation is another issue for the differential approach. Even for `Snap` and `Click` controllers (Section 6.4), the exactness of solving is limited because controllers only get to specify velocities at sampled increments. For greater precision, controllers might need to be given some information about step size. This is also important for better tracking of input devices as it would permit better prediction. Enhancing mouse tracking, for example by predictive filtering, is another area for study.

The lag between pointing device and dragged object needs to be better understood. In applications where the lag is detrimental, it might be reduced by techniques such as predictive tracking. Developing predictive tracking strategies for input devices will be generally useful, not only for the differential approach, but also for things such as remote collaboration.

10.3.2 Mathematical Techniques

The weakest link in the mathematical techniques is the way that over-constrained systems are handled. Many of the deficiencies of the damping techniques are listed in Section 3.2.1. An alternate method of dealing with the ill-conditioned or singular matrices might be found. To date, I have not found any that retain the performance characteristics of the methods discussed in this thesis on well-conditioned problems.

A primary drawback of the damping methods as described in this thesis is that they also affect constraints that are not ill-conditioned. This drawback can be avoided by using methods, such as the one presented by Nakamura[Nak91], that adaptively set the amount of damping. These methods are expensive, as they effectively must solve the system to determine its condition. Another strategy would perform some potentially expensive computation to determine if there are bad constraints, and applying the damping methods only in these cases. The hope would be that the expensive computations would not be needed often or could be computed incrementally.

Alternate methods for handling over-constrained matrices, such as singular value decomposition, have better numerical properties than damping, but would require work to make them as efficient. Similarly, more reliable methods for handling inequalities, such as those of Baraff [Bar94], could be applied if efficiency concerns were addressed.

Another deficiency of the methods presented in Chapter 3 is that they offer few opportunities for levels of importance of controls (hierarchies in the terminology of Borning et al[BFBW92]). Soft controls provide two levels of constraints, but require better methods to give them the degree of control of hard controls. Extending differential techniques to support more levels of hierarchy would be useful.

The methods of the differential approach are designed to work with the lowest common denominator of general functions, avoiding special cases and anything beyond the minimal information about the functions. Each of these restrictions could be relaxed to create methods that better handled important special cases. For example, special methods for articulated figures exist. Also, more information about functions, such as higher derivatives and intervals, could also be used.

The techniques provided for handling inequality constraints are really a hack, and would be improved by using numerical methods better suited for this task. Finding methods with suitable performance characteristics could be challenging.

10.3.3 Numerical Techniques

As computers become faster, the size of the problems that can be handled at interactive rates will increase. This means even more of the computation time will be spent in the complexity limiting step of solving the linear systems. Selecting appropriate algorithms will be increasingly important. Other conjugate-gradient solvers, such as those from [PS82] might apply. In particular, some of these methods may better handle ill-conditioned or singular matrices, allowing less use of damping techniques. Dynamic selection among multiple solvers, as done in Converge[Sis90], may also provide better performance, particularly when the problem can be partitioned.

The best weapon against the computational complexity of solving the linear systems will be to reduce their size while giving the user the illusion that the entire problem is being solved. There are many possibilities for new implicit constraint mechanisms. For example, representations of objects could be dynamically switched to maximize the number of constraints that are represented cheaply. Algorithms similar to multiple-output multi-way local propagation solvers, like [San94], could implicitly solve as many constraints as possible. Techniques that divide the problem based on numerical results are possible. For example, Sistare[Sis90] partitions constraints empirically. The solver attempts to solve the constraints on a small subset of the objects. If these objects do not have sufficient degrees of freedom, more objects are added to the subset.

Better methods for solving the ODEs would enhance the stability of the constraints and permit a wider class of controls. Making use of knowledge about the expected behavior of the objects should provide information that can be used to adapt the step size of the solver. Finding ways of making adaptive step sizes unobtrusive to the user might be required to make them acceptable.

The simple caching mechanisms of Snap-Together Mathematics are reasonable for the experiments conducted so far with the differential approach. Cache analysis shows varying performance of the global cache validation scheme. A better caching scheme, perhaps based on incremental attribute evaluation[Hud91], might be more efficient. However, as problem sizes grow, the cost of the Snap-Together Mathematics computations will become smaller relative to that of solving the linear system. Therefore, extensive optimizations are unwarranted. Better support for switching representations would be another improvement in the Snap-Together Mathematics implementation.

10.3.4 Interface Toolkits

The differential approach addresses only a part of the problem addressed by modern interface toolkits. If the differential approach is to be used in a complete toolkit, it must be made to integrate cleanly with existing approaches. Employing propagation solvers for discrete data inside the differential approach seems to primarily be an engineering problem. However, incorporating the differential approach, with its different model of time, into a conventional event-driven toolkit poses difficult questions such as how to integrate discrete state changes into the continuous flow of time.

The ways that the Whisper interpreter supports Bramble bring interesting questions as to the architecture of graphics toolkits. Fast turnaround is only one of the reasons for the widespread acceptance embedded interpreters in graphics toolkits. However, much of the utility of the Whisper/Bramble connection is that many of the services that Bramble must provide, such as object management and on-the-fly definition of behaviors are similar to those provided by the run-time support for a modern programming language. An interesting strategy might be to design run-time support specifically for interactive graphical applications.

10.3.5 Interaction Techniques and Applications

While the differential approach has many flaws that may be addressed with future research, the techniques and tools are evolved enough that they can be used to create novel interaction techniques and applications. The invention of new graphical techniques will be the real payoff of the approach. Developing new interaction techniques may require finding new attributes of objects to constrain and control, new combinations of these controls, and new ways to present the controls and constraints to users.

Generalized snapping is a feature of the differential approach that has not been sufficiently explored. There are many issues in making it into a successful interaction technique, such as how to provide adequate feedback to the user and how to determine the many parameters. However, I think it is still a promising technique.

The availability of a variety of controls and the ability to combine these controls as constraints opens up a new domain of questions about how to select, specify, display, and edit the controls and constraints. Some tasks, like scene composition, seem particularly appropriate for interfaces that give palettes of controls to users. I think the strategy of manipulation from structure is a promising way to help make these tasks easier by building more constraints into the system before the user has to specify anything.

Good controls might also serve to facilitate manipulation when the "user" is some computational mechanism automatically controlling the graphical objects. For example, an automatic picture composition system like those described by Feiner[Fei93] might be easier to develop in terms of controls like "point the light at this object" rather than directly altering the underlying parameters of the objects.

The ability to handle controls asynchronously may help explore multi-input device interaction techniques. In general, the approach may serve to help incorporate new input mechanisms as it decouples the input device from the objects being controlled.

There are many questions which must be addressed before constraint-based interfaces for drawing and modelling can be successful. Constraints must be made easy to display, debug, understand, and edit. While restricted problems, such as permitting users to define the behaviors of individual objects, simplify these issues, they do not remove them.

The techniques for specifying, displaying, and editing constraints introduced in Briar also deserve to be examined in more detail. Future work must address issues in extending the techniques to larger classes of constraints, scaling them to larger models, and applying them in 3D.

10.3.6 Usability Evaluation

This thesis provides little in the way of formal evaluation of the differential approach. Because the new interaction techniques were meant only as examples of the differential approach, not as wide-ranging solutions to interface problems, it was not appropriate to study the usability of these techniques. However, because some show promise as interaction techniques, more formal evaluation may be useful.

Much of the lag between pointing device and target object in the differential approach can be reduced with better tracking. However, many users report enjoying this "spring" behavior. A study to evaluate the usability of the spring dragging might be interesting. Similarly, the usability of constraint-based graphics has never been formally studied. Such a study might determine people's abilities to use multiple simultaneous controls, or help understand how complicated a constrained model might be created by a user before comprehensibility is sacrificed.

10.4 Final Remarks

This thesis has provided an approach to implementing direct graphical manipulation that uses the numerical and graphical performance of modern processors. The approach views manipulation as an equation solving or constrained optimization problem, allowing interactions to be defined in terms of objects' attributes, rather than their representations. To implement the approach, mathematical techniques for solving the optimization problems had to be selected and implemented in a way that addresses the issues of interactive systems. This implementation permitted encapsulating the mathematics in a manner that allowed the construction of a graphics toolkit that hid the mathematics of the approach from the applications programmer. With the approach and the prototype implementations, a number of interaction techniques and applications were created.

Direct graphical manipulation has been widely successful, and future systems offer new possibilities and issues for these interfaces. As decreasing cost makes high performance computers more widely available, graphics tools will have a wider potential audience. Many of these users will require more fluent and transparent interfaces for types of tools now only used by experts. New classes of applications, such as virtual reality, offer whole new classes of issues. New users and new applications will challenge interface designers. The ad-hoc methods traditionally used to devise manipulation techniques will hinder the development of new interfaces. This thesis has offers an alternative that can provide a substrate for future direct manipulation interfaces.