A Study of Delays and De-Synchronisation in a Multiple-View Direct Manipulation Task

by

Fabrice Jaubert

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 1995

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Computer Graphics are used in increasingly complex situations, often involving multiple dynamic views. This poses the problem of maintaining proper synchronisation among the various views. The effect of de-synchronisation and delays in command-line and single-view graphical environments has been well studied; this thesis aims to do the same with multiple view environments. To that end, an experimental platform is developed, which requires subjects to complete a placement task in a multiple-view representation of a three-dimensional world. Delays are imposed on the different views, and the effect on the subject's performance (in terms of completion time and number of manipulation operations required) is noted. The results show that delays on the view under manipulation have a definite detrimental effect on all subjects; furthermore, certain subjects are affected by delays on other views as well. To conclude, the experiment reveals that the problem of de-synchronisation in a multiple-view environment is a complex one, requiring further research; several new experiments are suggested to help answer the questions posed in this study.

Acknowledgements

I wish to thank my two supervisors, William Cowan and Marceli Wein, for their guidance and advice, as well as for their patience and continuing support throughout the many terms required to complete this degree. I would also like to thank my faculty readers, Richard Bartels and Gordon Cormack, and my student reader, Sandra Loop, for their feedback and suggestions, making this thesis more polished and readable.

The members—student, staff and faculty—of the Computer Graphics Lab deserve credit for their ongoing support and understanding with respect to my dual role within the lab. In particular, Greg Veres and Rob Kroeger assumed many administrative duties, freeing some of my time for research work. Glenn Pauley was instrumental in helping me format this document with LaTeX. Last, but not least, my thanks go to Anne Jenson for all her help in day-to-day matters, large and small.

This research was made possible by an NSERC scholarship and an ITRC fellowship. Some of the computer equipment used to conduct the experiment was purchased with the help of ITRC funding.

Finally, I would like to thank the members of the University of Waterloo Outers Club and the Conestoga Sailing Club, for the various opportunities to enjoy the outdoors and for the welcome relief from staring at computer monitors all day.

Trademarks

Solaris and **SPARCstation** are registered trademarks of Sun Microsystems Inc. **DEC OSF/1 AXP** and **Alpha AXP** are registered trademarks of Digital Equipment Corporation. **X Window System** is a trademark of the Massachusetts Institute of Technology. **DataDesk** is a registered trademark of Data Description, Inc. **OpenGL** is a trademark of Silicon Graphics, Inc.

All other products mentioned in this thesis are trademarks of their respective companies. The use of general descriptive names, trade names, trademarks, etc., in this publication, even if not identified, is not to be taken as a sign that such names may be used freely by anyone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer graphics, and especially animated computer graphics, is no longer the exclusive playing field of leading-edge technology and research and it has made its way onto the display monitors of normal, every-day computer users. As the trend towards democratising computer graphics progresses, we observe a growth in complexity in common applications. In particular, there is ever more demand for graphics involving multiple views, often with the added requirement that all views must be animated.

Multiple views are useful, for instance, in displaying three-dimensional scenes for simulations, architectural presentations, or even games. For these types of applications, the current high-technology approach is to use a Virtual Reality (VR) setup (such as that described by Cruz-Neira, Sandin, and DeFanti (1993)) to obtain a realistic three-dimensional representation; but this involves expensive and mostly experimental equipment, thus the more general solutions must rely on multiple-view presentations that can be accommodated by normal office hardware. However, even true VR often requires multiple views,

if for instance multiple displays (or projection screens) are used. Multiple views are also useful in visualising complex data and in scientific and engineering applications. In these cases, the data may be inherently three-dimensional, as in the paper by Osborn and Agogino (1992), where multiple two-dimensional cross-sections are used to describe a three-dimensional object (this is typical of CAD and architectural applications); alternatively, the data may be two-dimensional but complex enough that multiple aspects of it need to be shown in different views (Mackinlay, Robertson, and Card 1991). Finally, there is a growing interest in tele-conferencing, electronic white-boards and media spaces in general. While these applications are still in their infancy, as they develop, there will clearly be the need for multiple views, showing, for instance, each participant's perspective on a common problem.

The natural implication of using multiple views in a dynamic environment is that multiple updates must occur—either when the scene itself changes, or when the point of view is modified. It is true that in certain situations, some of which are described by Osborn and Agogino (1992), static views may provide sufficient information, but in general all views are required to be dynamic, and thus all views must be updated. So for instance, in a CAD package that presents an object with three cross-sectional views, all three views are affected, and must therefore be updated, when the object is rotated (i.e., when the scene changes). For a slightly less serious example, suppose a flight simulator shows the enemy target (say a building that must be bombed) both as the pilot would see it out the wind-screen and as a dot on the radar display; now when the airplane is banked into a turn (this is the point of view changing, not the scene), both the pilot's view and the radar view must be updated to reflect the new bearing.

Inherent in this discussion of the need for updating all views is the implication that all updates must happen synchronously. Indeed, as the object in the CAD program rotates, all views are expected to show the change at the same time, and similarly for the flight simulator, when the airplane turns. But what happens if coordination between the various views is disturbed, if delays are introduced in some of the views and updates are de-synchronised? The pertinence of this question lies in the fact that some form of delay is unavoidable, due to either hardware response time or software overhead.

The effect of delays in command line-based systems has been very well studied over the years—Teal and Rudnicky (1992) give a brief summary of this research—and the general consensus is that delays in the response time degrade the user's performance. Furthermore, several mathematical models have been established to predict the effect of delays, while taking into account user strategy and other environmental factors. Moving one step closer to the focus of the research at hand, researchers such as MacKenzie and Ware (1993) have examined the effect of delays (or lag) in single-view direct-manipulation tasks. Once again, the conclusion is that de-synchronisation between the user's actions and the feedback on screen increases the difficulty of the assigned task, as evidenced by longer completion times and higher error rates. Similar results were found in a so-called "fish-tank" Virtual Reality environment, which, again, is single-viewed (Ware and Balakrishnan 1994). In these last two studies, as in many others in the field, the results were used to derive a mathematical expression of Fitts' Law (Fitts 1954; Fitts and Peterson 1964), which can then be used to predict the difficulty of a task under given conditions.

In a report on an experimental multi-view Virtual Reality environment—the CAVE (Cruz-Neira, Sandin, and DeFanti 1993)—the authors describe a very specific problem, which was in fact related more to the stereo-scopic nature, rather than the number, of views: when the frames on different screens (where each screen is a separate view) were de-synchronised, the stereo effect failed, and thus the illusion of three-dimensionality collapsed. They solved the problem with customised hardware, and have not investigated other potential effects of delaying views. In even more esoteric research, Friedman, Starner, and Pentland (1992) examine synchronisation (and, subsequently, lack thereof) between audio and visual streams—which can be loosely equated to multiple "views."[1] Unfortunately the focus of that study, once again, is on maintaining proper synchronisation, and not on measuring the effect of de-synchronisation.

Thus the question posed above remains: what measurable effect is there on the user of a multi-view system when the various views become de-synchronised? Knowing the answer to this would be a valuable guide in designing and using interactive, multi-view systems: it could suggest which areas require the most performance to eliminate particularly detrimental delays, and it could offer hints and strategies to reduce the effect of unavoidable delays.

[1] Using an audio stream as a "view" conveying information is also studied by Kroeger (1993).

Chapter 2

Rationale

2.1 Objectives

The objective of this research is to model de-synchronisation between two or more views as it might occur in a complex real-life application, and to study its effect on the user's ability to perform tasks. For instance, traditional windowing systems such as X11 and Microsoft Windows serialise access to the display, hence an application with multiple views will draw each one in turn. Any delay in the window server will then cause de-synchronisation among the various views. An alternative might be to interleave the drawing of several windows, but this could be very costly, and introduce another set of delays. The general question, therefore, is how this de-synchronisation hinders a user, and how much of it, if any, is tolerable.

Some aspects of this topic bear similarities with the question of delays in a single-view application, such as a typical two-dimensional direct manipulation task. In that

well-studied case, a delay in redrawing the scene—in other words, a de-synchronisation between the input and the output—causes a delay in the time to complete the task, as demonstrated, for example, by MacKenzie and Ware (1993). In that study, as in other research in delays (Friedman, Starner, and Pentland 1992; Cruz-Neira, Sandin, and DeFanti 1993), lag is assumed to be caused partly by uncontrollable physical device characteristics, such as mouse response time or screen retrace time. On the other hand, as mentioned above, this study will concentrate on software-caused delays, which are potentially much larger.

In order to test whether the results for single-view cases extend to multi-view situations, it is first necessary to find an application which uses more than one view. For this research, it is not sufficient to simply draw multiple instances of the same view, since it seems likely that the extra views will be ignored, and the results will not provide any new insights into the problem (although it might prove interesting to test whether this assumption is, in fact, true). What is needed is a representation and a task in which integration of multiple views is essential.

On the other hand, if the objective is to test the effect of introducing de-synchronisation, care must be taken that other factors do not influence the results. In particular, the complexity of a task that requires multiple views could introduce large learning-curve effects, which could mask any measurable consequence of introducing delays. Thus, whatever task is chosen, it must be kept as simple as possible—it must be reduced to a minimal configuration that still requires multiple views. The final requirement in the selection of a task is that there must be a direct way of measuring performance, to

provide a means of evaluating the effect of delays and de-synchronisation. Furthermore, this measure ought to be numerical in nature so that a mathematical relationship between the stimulus—the amount of de-synchronisation—and the response—performance at the assigned task—can be established.

2.2 Task Selection

One representation model which satisfies the requirement for multiple views is a natural extension to the two-dimensional environment used in the research mentioned previously: a three-dimensional, direct-manipulation application. Since no form of single-view representation can fully and completely describe a three-dimensional scene, multiple views—at least two—are indeed a necessity. There are many common forms of presentation for three-dimensional information, such as the side-by-side plan, elevation and front views used in architectural drawings (see also Osborn and Agogino (1992)). However, of all these, the model which appears to be the most direct extension of the two-dimensional case is the one presented by Herndon et al. (1992), called "Interactive Shadows."

This model presents a three-dimensional scene in four views: a perspective view and three "shadows" on the side and bottom walls of the "world," where each shadow is cast by its own light source at infinity. In other words, the shadows are parallel projections onto a plane perpendicular to the direction of projection. It is interesting to note that, despite an apparent lack of interest in research literature, shadows have been used in this way to convey three-dimensional information for over a decade in commercial applications; to name but one example, in a video game of the early eighties—*Zaxxon*—the player had

to navigate a spaceship through a three-dimensional obstacle course and was aided by a shadow (a single one in this case) projected on the "ground." To return to the Interactive Shadow model, one will note that in fact, it provides no more information than would be contained in a plan, elevation and front view decomposition—the bottom shadow is the plan view, the side wall shadow is the elevation, etc. However, the shadow metaphor and the fact that all views share common reference points (they are all drawn in the same viewing cube) make this representation more natural, and less synthetic. For these same reasons, one would expect that proper synchronisation of the different views is critical; if shadows start lagging, the metaphor can be broken. This aspect of the model makes it very well suited to research in de-synchronisation.

Another important advantage of Interactive Shadows is that it has an inherent interaction model. This is of great help as, in general, interaction in a three-dimensional environment, using a two-dimensional interactor (typically a mouse) is a difficult problem. Much research has been done on this topic, with solutions ranging from overlaying the scene with three-dimensional widgets (Bier 1987; Conner et al. 1992) to mapping mouse motion to different planes depending on position or direction (Chen, Mountford, and Sellen 1988; Evans, Tanner, and Wein 1981), but none of the solutions have the elegant simplicity offered by Interactive Shadows, where direct manipulation occurs on the shadows themselves. Since the shadow views are two-dimensional (they are projections), there is a direct mapping between interactor (mouse) input and resulting effect on the scene; there is no need for more or less complex schemes to map two dimensions to three. The types of manipulation which appear ideally suited to this interaction model

are translation (in the shadow plane) and rotation (around the normal to the shadow plane), both accomplished through "click-and-drag" operations.

Having chosen a representation model, one needs to find a suitable application and task to use as a test bed for this study. Since this research stems from the observation that de-synchronisation does, in fact, occur in real-life applications, it would be preferable to use such an application—or at least a simulation thereof—as the platform for the experiment. On the other hand, it is important to achieve simplicity, both in terms of the task to be accomplished and in terms of visual cues and visual information. Thus, while an engineering design (CAD) package, for example, would provide the opportunity to use Interactive Shadows, there is the danger that the task might be too complex; even a simple wire-frame object in such an environment might require a large number of criss-crossing line segments and cause visual confusion. This confusion certainly increases the difficulty of any task presented to the subject. A chemical modelling package, on the other hand, shares many of the attributes of a CAD program—notably, the need to visualise and manipulate three-dimensional scenes—while using objects composed only of spheres and simple line segments. Visually speaking, a chemical model will yield a simpler and less cluttered scene (assuming the molecules are kept small!) than an engineering diagram.

A simulation of a chemical modelling program was therefore chosen as the platform for the experiment of this study. Obviously this platform implements none of the algorithms needed in chemistry; it is simply designed to allow manipulation of molecules. In this context, the simplest task requiring three-dimensional information—the essence of all

three-dimensional tasks, really—is alignment: a molecule must be placed and oriented in a particular way. Depending on the initial set-up, completion of this task requires translation or rotation, or both, where manipulation may occur in one, two or three dimensions (for example, the molecule may only need rotation about X, but translation along both X and Z). Note that most research on system lag, delays and de-synchronisation to date has focused on pointing (or acquisition) tasks, rather than the dragging task described here. But an acquisition task, in which only the pointer moves, is inappropriate for this experiment, where the goal is to obtain motion (animation) in multiple views. In any case, MacKenzie, Sellen, and Buxton (1991) have shown that although pointing and dragging are different tasks, requiring different skills, they elicit the same type of user response in terms of variations in difficulty levels. Thus the results from this study should be generally comparable to other research in delays.

Due to the possibilities afforded by this type of placement task, it was felt that the molecule itself could be kept simple—a large and a small atom, connected by a single bond—even if this made it only a two-dimensional object. In other words, at most two views (two shadows, or the perspective view and one shadow) are needed to obtain a precise picture of the molecule's orientation; however, as long as placement is part of the task, three views are required to evaluate position with respect to the target.

The next question posed in the task definition process relates to the combination of placement and orientation. It has just been shown that placement is necessary, but is it also sufficient? Or is orientation also necessary for a minimally complex but fully three-dimensional task? The concern is that orientation might introduce too much complexity:

it is often difficult to visualise a priori the effect of a rotation, especially in more than one dimension (Finke and Shepard (1986) discuss the various cognitive difficulties associated with three-dimensional rotations). Thus, while the manipulation itself would be fairly simple (by rotating a shadow), and the result immediately apparent, it is possible that the user would have trouble deciding which rotations were required to obtain the proper orientation. This complexity would certainly increase the learning-curve effect. No such conceptual difficulties exist with translation, although the concern is that a poorly designed placement-only task can be reduced to a strictly two-dimensional problem, requiring, in the worst-case scenario, information from only a single view. Such simplification is undesirable when the point is to study de-synchronisation effects between multiple views. However, with some care in initial positioning, it is possible to ensure that a placement task will require information and manipulation of more than one view. For this condition to hold, it is sufficient to guarantee that the initial and target positions are misaligned in all three dimensions.

Defining the target position poses certain problems, involving a three-way tradeoff, between clearly marking the final position so that the task is unambiguous, reducing clutter on the screen to avoid confusion and finally ensuring that the task is not overly simplified by explicit guide marks, which might again reduce the need for multiple views. In this experiment, the decision was made to use a second molecule, similar in shape and size to the first one, to define the target area. With this scheme, no new forms of markings are introduced—the two molecules are displayed in the same manner—which reduces clutter and possible confusion. The only potential pitfall is that one molecule could be mistaken for the other; to avoid this, the molecule which is to be manipulated—the *user*

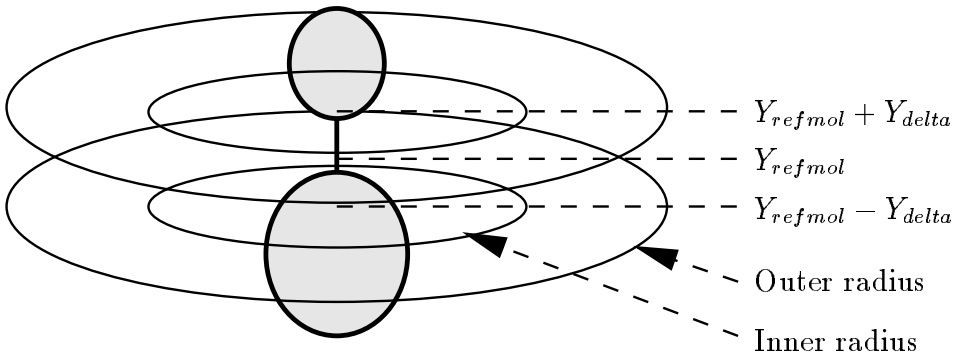

Figure 2.1: Illustration of the Target Area, defined by an inner and outer radius and a Y_{delta}

molecule—is drawn in bright colours and has black shadows, whereas the immovable one defining the target area—the *reference* molecule—has dull colours and grey shadows. This colour scheme was selected because it is analogous to current windowing system designs, where menu items or buttons are "grayed-out" when they are unavailable for selection.

While the selected molecule provides a reference frame for the target area, it does not actually define anything until design decisions are made. How should the target position relate to the reference molecule? The simplest approach would be to define the destination as the position occupied by this molecule, so that when the task is successfully completed, both user and reference molecules overlap completely. This solution was rejected for two reasons. First, it seems that the task might be over-simplified, because the subject would simply have to overlap shadows, one after the other, thus reducing the task to

a single-view process. The second, and most important, reason was the fear that the overlapping shadows and perspective views could cause visual confusion and complicate selection of shadows for manipulation. Therefore, instead of requiring the molecules to overlap, the task used in the experiment calls for placement of the user molecule inside a "nearly-toroidal" region around the reference molecule, illustrated in Figure 2.1. The region is in reality defined by sections of an inner and an outer cylinder centred at the reference molecule and with radii in the XZ plane (the "horizontal" plane), clipped by two planes $Y = Y_{refmol} \pm Y_{delta}$, where Y_{refmol} denotes the centre of the reference molecule. This scheme requires information from at least two views for proper placement, since neither the $X = 0$ or $Z = 0$ shadows convey enough information to define the circular part of the region, and the $Y = 0$ shadow conveys no information about vertical placement. In order to avoid screen clutter and confusing markings, the target area is not explicitly marked, and must therefore be inferred from the position of the reference molecule. Proper placement—and therefore end-of-trial—is signalled by re-drawing the bonds between the atoms to form two new molecules, leading to an intuitive explanation for the shape of the target region, as the subjects can be told they must bring the two molecules within "chemical bonding" distance.

The final issue to be addressed in defining the target area is its size, as controlled by the cylinder radii and the Y_{delta} parameter. Once again, this choice involves a tradeoff between over-simplifying the task to the point of triviality or making it so complex that learning curve effects dominate the results. Indeed, if the target region is made very large (large Y_{delta} and large difference in the two radii), the task can be completed with almost no visual feedback at all, as the user simply needs to drag the shadows in the general

direction of the target. On the other hand, a very small region requires a great deal of precision for which some of the subjects may not have enough hand-eye coordination. The difficulty in this case is further compounded by the fact that the region, small as it is, is not even drawn. A region sized somewhere between these two extremes hopefully yields a task requiring enough precision so that information from the various views is necessary, while remaining feasible for an average subject. The size of the region must remain constant throughout the trials, otherwise subjects need to constantly adjust to new conditions, thus introducing learning effects.

The trials in this experiment end when the subject releases a view from a manipulation operation and the user molecule is within the target area. An alternative method—named *Auto-bind* because the two molecules would automatically form the "end-of-trial" bonds—was considered, where the trial would end as soon as the user molecule entered the target region, without waiting for the subject to end the current click-and-drag operation. This reduces the need for precisely pre-visualising the target area, with the advantage that the learning period is reduced, but it might also greatly reduce the need for visual feedback: the user simply needs to wave the various shadows back and forth in the general vicinity of the target until auto-bind tales place. Performance—in terms of time required to complete a trial—is then completely unrelated to the delays of the various views. It is possible that the auto-bind method combined with a very small target region would work well and produce valid results, as the user would be forced to rely on visual cues to get near the small area, but this was not tested in the experiment.

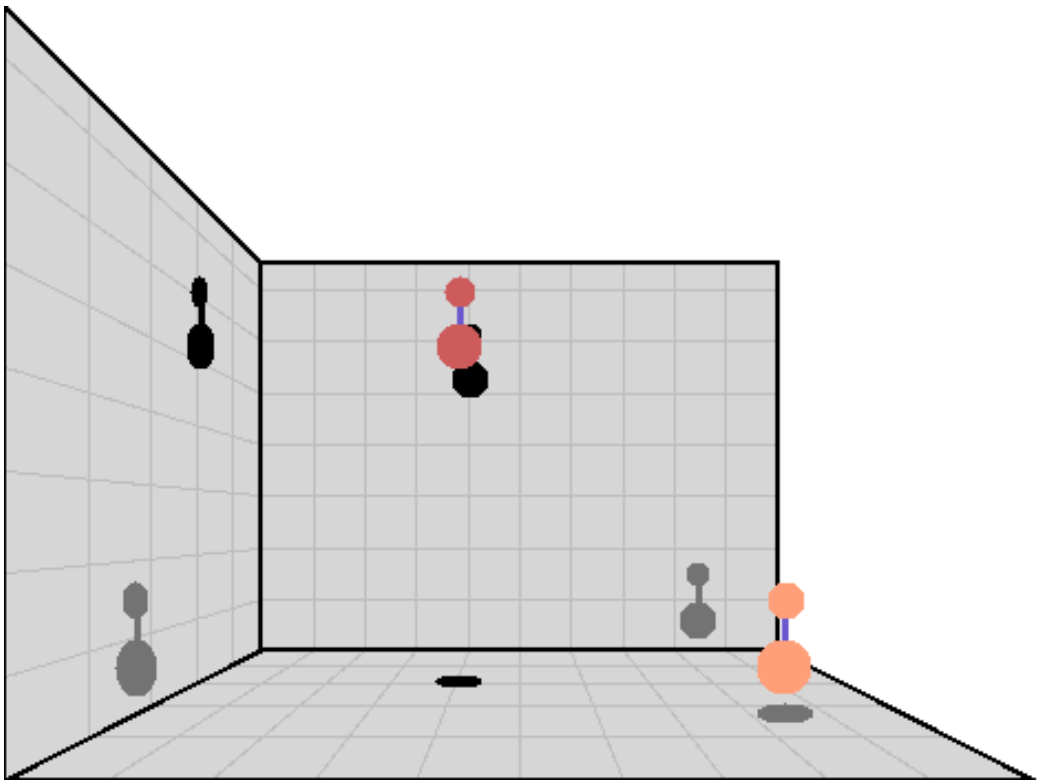

Figure 2.2: A snapshot of the system at the start of a trial

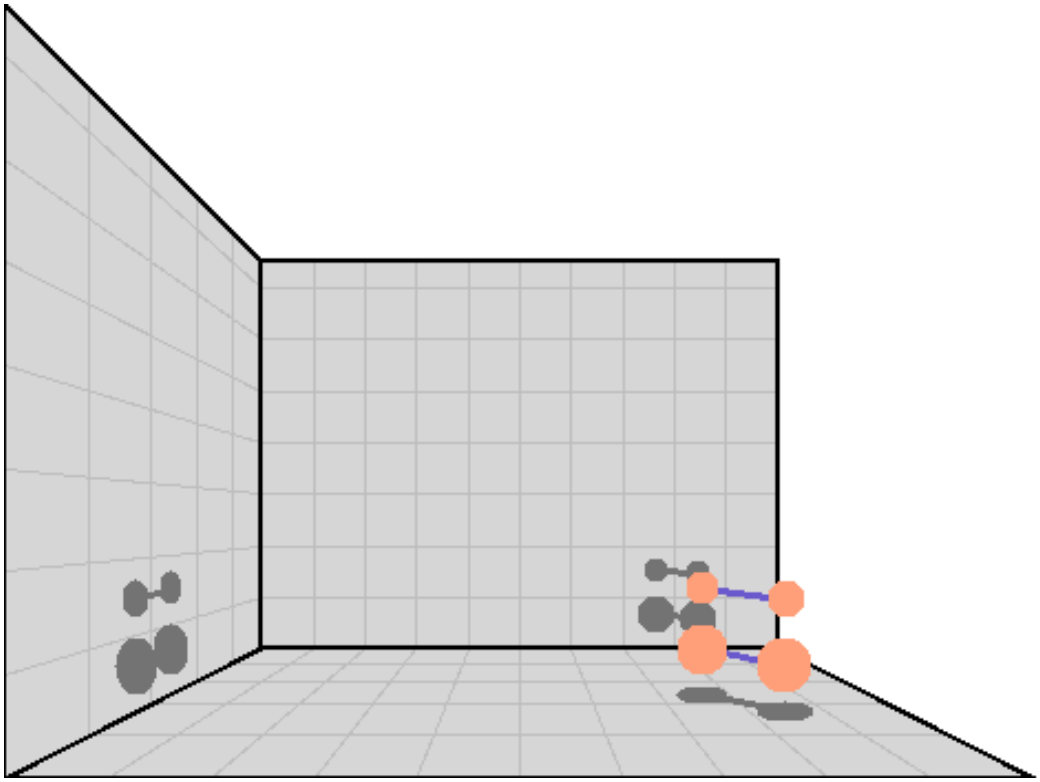

Figure 2.3: A snapshot of the system at the end of a trial

To summarise the experiment platform, the subject is presented with a three-dimensional "world" defined by three walls—a floor, a back wall and a side wall to the left—on which grid-lines are drawn for reference. Within this world, a reference molecule and a user molecule (distinguishable by their different colour schemes) are randomly placed at the start of each trial; both molecules project shadows onto all three walls. The subject manipulates the user molecule—by using click-and-drag operations on any of the three shadows—until it is within the (invisible) target area around the reference molecule, at which time the subject signals the end of trial by releasing the current manipulation operation. End of trial is confirmed by the system by re-arranging the bonds between the atoms. Figure 2.2 shows the system at the start of a trial, and Figure 2.3 shows the end of that same trial.

2.3 Measures of Performance

The principal measure of performance in this study is the time required to complete a trial, based on the assumption that the greater the difficulty of the task—due to the various delays introduced—the longer the subject takes to successfully place the molecule in its target position. Timing starts when the first interaction begins (i.e., when the user starts the first click-and-drag operation), rather than when the trial begins (i.e., when the scene is first displayed). As the scene for a new trial is drawn, the subject requires some time to locate both molecules, form a mental image of the target region, and perhaps also plan the sequence of moves necessary to complete the trial. Since in the experiment described here, these preliminary tasks are of consequence only because they facilitate the

next stage, where delays are introduced, they should be allowed to complete un-hindered, and un-timed. Only when the first click-and-drag operation is initiated do delays come into play, and it is at this point that performance needs to be measured. Obviously, timing ends when the trial ends: when the user releases the molecule within the target region.

It is possible that in addition to requiring more time, difficult trials require a larger number of interaction operations. Indeed, while it is conceivable that with a good "plan of attack" (i.e., a well thought-out sequence of moves) for placing the molecule, and enough patience to simply wait out the delays, a subject could perform the task in the same number of operations regardless of configuration, it is an hypothesis of this research that the delays cause errors and loss of precision, requiring, in turn, more operations to obtain correct placement. To verify this hypothesis, the number of click-and-drag operations (counted as either the number of mouse-down or mouse-up events) is noted as a secondary measure of performance, alongside the time-to-complete.

Finally, it is possible that certain difficulties inherent in the task—notably, the lack of explicit marks defining the target area—might cause a "fine-tuning" effect, similar to the one observed by Schwarz (1985), in a study which involved adjusting colours to match a target. He identified two phases of operation: a coarse adjustment phase, where the current setting is wrong and the subject knows in what way it is wrong, and how to improve it, and a fine-tuning phase, where the setting is still wrong, but the subject is unable to quantify the error and is therefore unable to formulate a logical strategy for improvement. In other words, most of the useful work is done during the

coarse adjustment phase, while the fine-tuning phase consists of small, apparently random adjustments to obtain a perfect match. Relating those findings to this experiment, it is conceivable that once the subject has placed the user molecule within a few pixels of the target region (in a coarse adjustment phase), a large number of extra operations and a significant amount of extra time are spent micro-adjusting the position to end the trial, because it is not clear exactly why the molecule is not yet within the target region. As in the experiment on colour, this fine-tuning stage has very little relevance to the study at hand: while delays may increase fine-tuning time even further, the main cause of the difficulty in this particular case is the task itself (the target area is not marked explicitly enough), not the configuration. Thus, ideally, the experiment should provide a means to determine if there is indeed a fine-tuning effect, and if so, should allow identification of the components of performance (in terms of time-to-complete and number of operations) attributable to this stage. To this end, a full set of intermediate results (although not full enough, as it turns out—see Section 3.7) is kept, noting both distance between the user molecule and the target region and time for each user event. With this information, it should be possible to isolate sequences of events occurring when the molecule is nearly in the target area, and filtering out the extra time and number of operations from the final results. Teal and Rudnicky (1992) present a more detailed discussion on the benefits of keeping intermediate results as well as aggregate measures of performance, such as total time-to-complete.

Chapter 3

Implementation

The goal of the experiment is to measure the effect of view delays and de-synchronisations. Achieving this goal requires that all delays in the system be controllable. Specifically, the experimental platform may not introduce any inherent delays of its own. The guiding objective during the implementation phase was to minimise unintended delays. Another important objective, of course, was a clean code structure and maintainability: so that the system could easily be modified for future experiments, if needed.

The entire experimental platform consists of an input script describing the delay parameters for each trial, used as input to an experimental engine, which controls the task and measures the subject's performance. The measures are stored in results files, that can then be analysed with a variety of tools. This chapter describes the structure of the engine code, outlining the main challenges as well as the various performance tweaks necessary to obtain a delay-free experimental platform, and explains the format of the input and output.

3.1 C++ Structure

The experimental system is written in C++ and is based on a number of classes, in the usual "object-oriented" approach. This software model is used because it provides maintainability and encourages good code structure. From the top down, the experiment itself is a class which embodies the concepts of trials and trial configurations. This experiment class handles the main event loop, but relies on other classes to parse the input script, to output results and to draw the scene.

The input script parser and the results printer (called the Stats Agent) are simple classes; they presented no design or implementation challenges. The encapsulation of these functions in distinct units allows a great deal of flexibility; for example, the specification for the input script was changed mid-way through the implementation. This change affected the entire input parser class (about 100 lines of code in total), but required no modification to the experiment class.

Molecules are implemented as a tree of atoms. The nodes of the tree are encapsulated in a simple class (called MolAtom to avoid conflicts with the "Atom" type defined in X11) which keeps track of the atom's size, position relative to its parent and position relative to the world coordinate system. In addition, the class has the usual functionality (pointers to siblings and children, etc) needed for tree management. The tree class itself—the Molecule class—implements tree-management functions, such as adding nodes, as well as specialised traversal routines, such as placing, picking and rendering, tailored to the experiment.

When a molecule moves, the experiment class passes to the placement routine the new position of the root atom, in world coordinates. This routine then traverses the tree and uses each atom's relative position information to compute its new world position. Then, when a draw request arrives, the coordinates for each component of the scene are known, and the traversal routine need only call the appropriate drawing function, thereby improving efficiency.

The drawing routines themselves are implemented in a Graphics I/O class (GrIO), which is itself derived from two base classes: a projection class (Project) and a viewport mapping class (ViewPort). The derived class deals only in world coordinates (as does the Molecule class), and the base classes handle the conversion to and from screen coordinates. The GrIO class is, in fact, the experiment's only interface to the graphics subsystem. As such, it must provide graphics routines for both the Molecule class and the Experiment class itself. In the first case, the routines embody "primitives" useful in drawing molecules: spheres, lines and shadows, and are described in more detail below. In the second case, GrIO provides abstractions for X window system related functionality, such as creating and managing windows and colormaps, and double-buffering. This part of GrIO is straightforward, and requires no further explanation.

Initially, the C++ approach permitted using some existing C++ base classes from the Splines Project (Bartels 1994). In particular, the molecule class was derived from the SNTree class, designed to support general-purpose trees. Atoms were nodes in the tree, and as such were derived from the general-purpose tree node class SNTreeNode. This approach made it possible to rapidly get beyond the first stage of coding and into the

more significant aspects particular to the experiment. Unfortunately, they also added several layers of dependency to the experimental platform: the code could compile and run only in environments where the Splines project ran, which in turn depended on its underlying tools, the RogueWave classes (Keffer 1991).

The second, inherent, disadvantage of using general-purpose base classes is that a great deal of extra functionality is included in the system, even though the application discussed here does not use it. In fact, the program requires only a very simple tree structure, and ends up using about one tenth of the capabilities of the SNTree and SNTreeNode classes, which themselves only use a fraction of the possibilities of the RogueWave tools. Yet the program bears the weight of the extra complexity—in terms of size of the executable code image as well as processing time.

Thus it was decided to re-write a simplified, application-specific tree structure and eliminate the dependency on external classes and tools. This tree structure is now an integral part of both the Molecule and Atom classes, as described above. Comparing the old executable—with the Splines and RogueWave classes—to the new one, shows a 50% decrease in size, roughly down from 2Mb to 1Mb. Such a reduction in size is of course beneficial to performance in many ways—less code to execute as well as less chance of filling memory and causing swapping, to name but two.

3.2 Graphics Optimisations

Concern for efficiency and optimal performance, and above all, control over all aspects of the code, led to the decision not to use any commercial libraries or other external aids in

meeting the application's needs for graphics. For example, many of the details discussed here are unnecessary if OpenGL (Segal and Akeley 1992) is used as the basis for the experiment. On the other hand, the implementation is then at the mercy of OpenGL; while the complexity of projecting spheres and shadows, for instance, is no longer a concern, there is no opportunity to implement the efficient approximations described below in order to increase performance. As is the case with the Splines classes, the price for functionality and generality beyond the application's needs is probably inadequate performance.

In addition to performance considerations, another goal was to achieve direct control over the entire graphics "pipeline," as this type of experiment may require non-standard behaviour which might not be supported by OpenGL. Thus, while it may have been possible to use OpenGL on an SGI Onyx (where the graphics hardware pretty well guarantees satisfactory performance), it was preferable to write experiment-specific routines "from scratch," using the lowest-level tools available. In this case, the lowest level is the X11 programming environment: Xlib (Jones 1989; Nye 1990), and the Multi-Buffering X extension (Friedberg, Seiler, and Vroom 1990), an extension providing a double-buffer mode for smoother animations. The above extension appears to be implemented entirely in software on both of the platforms used in the experiment (DEC Alphas and Sun Sparc 20s), making no use of potential hardware support.

As an aside, this software implementation—presumably using bit-blitting—has interesting performance implications on the Alphas. It appears that pixels are copied in blocks two scan-lines high; when the target of the copy starts on the wrong line (i.e., starts in

the middle of a block), the entire operation is slowed down by an order of magnitude. It was therefore necessary to hard-code the position of the window to properly line up with the block size. No such constraint appears to exist on the Suns, which was used in the final stages of the development, and in production.

The low level libraries—Xlib and Multi-Buffering—provide basic graphics primitives only: lines, arcs, etc. There is, of course, no support for the type of three-dimensional objects present in the experiment and tools had to be built to meet the specific needs of the application. The first level of tools consists of a viewport mapper and a projection model. This is the usual, well-known structure for displaying three-dimensional scenes, and the only details that are discussed are those relevant to the experiment. For a general overview of these topics, see for example Chapter 6 of Foley, van Dam, Feiner, and Hughes (1990).

The ViewPort provides the mapping between the two-dimensional projection space and the screen space; here it is the matter of simple translation and scaling operations based on a View Extent (i.e., size of the "world" in which drawing will occur) specified by higher-level components (GrIO). The mapping is bi-directional: for drawing purposes, projection-space coordinates are transformed into screen coordinates, which can be passed directly to Xlib routines; for input (picking) operations, screen coordinates (such as the location of the pointer) are mapped back to projection space.

The Projection model transforms the three-dimensional world coordinate system into the two-dimensional projection space which can then be passed to the ViewPort. This transformation can generally be done with either a perspective or a parallel projection.

The latter type usually involves simpler mathematics, but is not well suited to the experiment. It yields images which appear less "natural," more synthetic than a perspective view. Furthermore, since shadows are, in fact, orthographic parallel projections themselves (i.e., each shadow appears to be cast from a separate light source at infinity), confusion could result from also using a parallel projection for "the world."

Consequently, a perspective view transformation is used in this experiment, using a projection plane which is normal to a single principal axis (a one-point perspective). While this is perhaps slightly less realistic-looking than a 2-point perspective, it allows simplifications in the computations (in particular, shadows on the projection plane are trivial to compute) and better performance. For this reason, projection is onto an arbitrary $Z = Z_p$ plane, normal to the Z-axis.

For performance reasons, the projection class provides only basic services to meet the specific requirements of the Molecule and GrIO classes. This class can transform a point specified in world coordinates into projection space, and can map a point in projection space back onto a specified plane ($X = 0$, $Y = 0$ or $Z = 0$) in the world coordinate system.

Also, no attempt is made to perform any of the clipping usually associated with this stage in general graphics pipelines, because the higher-level components never attempt to draw outside of the specified drawing extent. This assumption is possible because the placement of the Molecules (the components which do the drawing in this case) is governed by the interaction method, which excludes operations outside of the defined extent. At worst, the centre-point of the Molecule can be on the outer edge of the extent,

and thus parts of the Atoms may indeed be drawn "outside" of the viewport. In practice, this happens rarely and is not noticeable in the experiment.

Another simplification from the general model of a pipeline is how primitives more complex than single points are projected. Theoretically, molecules should be drawn in three dimensions and let the tools, from the projection level down, handle the conversion to two-dimensional primitives. However, this requires complex algorithms in the lower-level tools, algorithms with a large number of calls to simple 2-D primitives such as points and line segments. This approach would be an inefficient way to draw the geometric shapes of this experiment. Instead, GrIO (which draws on behalf of the molecules) relies on the projection and viewport models to obtain a single screen coordinate (corresponding, for example, to the centre of an atom), and renders directly to the screen using its own algorithms, which make more efficient use of the available Xlib primitives. This type of simplification is not new—it was suggested nearly twenty years ago by Knowlton and Cherry (1977).

In effect, the GrIO class contains code to implement projections, and is specific to the projection model. This is unfortunate from a maintainability point of view, since the choice of a projection model can not change without a complete rewriting not only of the Project class but also of the GrIO class. On the other hand, such simplification allows for significant optimisations that would not be possible in a truly general system and it is these optimisations that allow the experiment to run in real-time.

For example, the correct perspective transformation of a sphere is, in general, an ellipse. When the centre of the sphere lies on the normal to the projection plane passing

through the Centre of Projection (call this normal the "line of sight"), then both major and minor axes of the projection are identical, and the sphere projects as a circular disk. The further the sphere lies from the line of sight, the greater the difference in the axes, and the greater the eccentricity of the projected ellipse.

Now, under the conditions of the experiment, the spheres actually never lie very far from the line of sight, and the correct projection would at most be very slightly elliptical. It is therefore reasonable to simply draw a disk without affecting realism, thus avoiding computing both of the ellipse's axes. Mathematically, this is a rotation of the plane of projection so that it is perpendicular to the line passing through the Centre of Projection and the centre of the sphere (condition under which a sphere projects as a disc); finding the disc's radius is now a simple matter of computing similar triangles.

Initially, there was to be an additional "catch-light" effect (i.e., pseudo-shading), to enhance the impression of three-dimensionality. However, it turns out that the scene appears convincing enough, and that no extra effects are necessary.

More difficult than drawing the molecule spheres, two of the three shadows of the spheres—themselves theoretically parallel projections onto the "shadow" planes—project as skewed ellipses, that is ellipses whose axes are not even aligned with the coordinate system's major axes (the shadow on the plane parallel to the plane of projection is, of course, always a disc). These shapes are accurately described by generalised conic equations (Smith 1953) and produce complex mathematics. A more tractable approach is given in Chapter 19 of Foley, van Dam, Feiner, and Hughes (1990), in the form of a scan-

conversion algorithm. However, even this approach is too complex for the experiment, which needs to output four such shadows per redraw.

The work presented by Wanger (1992), which concludes that the actual shape of the shadow has no effect on the perception of spatial positioning and sizing, suggests that the rendering of these shapes can be approximated as well. Thus, the bounding box of the sphere is projected onto the shadow wall, yielding the quadrilateral which inscribes the skewed ellipse. The average height and width of this quadrilateral is obtained (height and width changes for different positions of the sphere: the size of the shadow is partly determined by its position), and these values are used as the axes for a regular ellipse. In effect, the skewed ellipse is "straightened" so that it can be inscribed in a rectangle. The result is, again, very convincing and does not detract from the illusion of three-dimensionality.

As a further refinement, some shadow shapes could be stored in pixmaps; the drawing routine could then simply bit-blit them onto the screen rather than needing to re-compute the axes and use the *XFillArc()* Xlib call. For shadows on the $Z = 0$ plane (which is parallel to the plane of projection), the process is straightforward: the shadow of a sphere of given size is always the same disc. Therefore, a simple list structure which stores pixmaps with the radius of the sphere as a search key can be used for subsequent retrieval.

But this algorithm does not extend well to the other two shadows. Recall from the discussion above that the size of the shadow varies according to position—due to the perspective transformation, the shadows which are further from the point of view look

smaller. Thus, in order to draw the shadow one needs to know the radius of the sphere—as is the case for the $Z = 0$ plane—but also its position. It follows that the pixmap list-management routines needs to be extended to retrieve elements based on both these criteria, instead of just the radius.

That extension in itself is simple. The only difficulty stems from the fact that a pixmap is required for every (x,z,radius) or, depending on the shadow plane, (y,z,radius) triplet. This is not intractable: the shadows can only vary in size in discrete steps—at the very most, one such step per pixel difference in position. Since there are a finite (and computable) number of possible positions in the viewing cube, there are a finite number of shadow sizes. But this number is certainly large (it is bounded above by roughly 120000; the true number is probably about half that), and at this point the simple list management becomes extremely inefficient. In addition, the memory required by all these pixmaps is guaranteed to cause swapping activity on the machine configurations used in the trials. Thus the alternative was chosen, even though it involves computing the height and width each time, and issuing the *XFillArc()* instruction.

There remained the possibility of using the pixmap list for shadows on the $Z = 0$ plane, since this causes no storage problems. However, some simple timing tests demonstrated that the gain from copying pixmaps (over *XFillArc()*) was not appreciable; in the interest of simplicity, all shadows are handled in the same way—by computing and re-drawing them each time.

A pixmap proves to be very useful, however, in displaying the background grid. This grid requires a number of different graphics primitives, unlike the single *XFillArc()* used

for shadows: three filled polygons form the shadow planes onto which 30 grid marks are drawn (each requiring an *XDrawLine()* primitive). Since the result never changes—the grid would change only if the projection or the view extent changed, which does not happen in the experiment—the grid is stored in a class variable (no need for a list) the first time it is drawn, and then *XCopyArea()* is used in all subsequent re-draws.

The final optimisation involved depth cues. In a general 3D graphics pipeline, such as OpenGL, a Z-buffer algorithm is used, whereby a depth (Z-axis) value is associated with each pixel in the frame buffer; an element in the frame buffer will only be over-written if the new pixel has a "closer" (smaller) depth value. This algorithm handles occlusion properly, such that closer objects are never obscured by further ones, regardless of the order of drawing or the complexity of the scene.

The scene in this experiment, however, is not complex: there are two molecules, each composed of two spheres and one connecting line segment. Thus there is no need for the generality of the Z-buffer algorithm; it is not necessary to carry depth information with each pixel, and this simplifies both the projection computation—depth information need not be generated—and the data structures. Instead, drawing order is chosen to ensure that depth cues are presented properly.

For example, the molecules draw the bond between the atoms first, followed by the atoms themselves. This ensures that the line segment—which is actually drawn between the centres of the spheres—stops where the spheres begin. It is also important to draw all shadow views first, before the object (*perspective*) view; otherwise the object would occasionally be obscured by a shadow, giving the impression that it had passed behind

the wall. Or rather, since the wall itself is always drawn first (immediately after the screen clearing command), the impression would be that the object had passed between the wall and the shadow—an impossibility which would be sure to distract the subjects!

Initially, these were the only constraints on drawing order; out of simplicity, the two molecules were always drawn in the same sequence, in effect always placing one of them (the reference molecule) in front. This was not actually very noticeable in the early preliminary tests—the two molecules infrequently overlapped, due to the placement strategy used by the first pilot subject (they were more often side by side). However, one of the later subjects complained that this incorrect depth-cueing caused some confusion. The code now compares the centre-points of the two molecules (in the World Coordinate system), and draws the further one first.

This simplistic algorithm relies on the assumption—true in the experiment—that both objects are identical, simple shapes, and that all parts of the closer molecule are closer than all parts of the further one. In a more general environment, the further molecule could have atoms which reach out in front of its neighbour and to correctly handle occlusion in this case would require comparing the positions of the molecules component by component. Also, the molecules used here are never viewed at an angle from which their own atoms overlap; if this were not the case (if, for example, molecules could be rotated), it would be necessary to compute which atom to draw first, for each molecule. It is clear therefore that the occlusion-handling scheme is far from generalisable, but it does manage all the cases which arise in the experiment layout, and it does this as efficiently as possible, ensuring adequate performance.

3.3 Input Processing

Most general-purpose three-dimensional graphics environments support input as well as output. Usually this support consists of "picking" primitives. The experiment naturally requires the ability to accept mouse input and as was the case for graphics output, these requirements can be met by a few special-case functions. There is no need for general-purpose algorithms, such as those in OpenGL, where picking is achieved by simulating a redraw of the scene and checking what objects would be rendered at a given screen location (the pick point). The normal graphics pipeline—all transformations, clipping, etc—is used, except pixels are not actually drawn into the frame buffer. But the system does check if the picked pixel(s) would be affected, and if so, a label identifying the current object (usually a path in the scene description hierarchy) is written to a "pick buffer." The application can then scan through the pick buffer and choose the object it wants to select, according to context. This algorithm eliminates the need to convert a two-dimensional screen coordinate (the mouse location) to a three-dimensional world coordinate in order to determine position within the scene—because in general this mapping is not possible, without extra information about the Z coordinate.

The application described here is considerably simpler than the general case. Picking in this case can only occur on one of the "walls," and furthermore these walls are non-overlapping. Thus there is no need to incur the overhead of a complete scene redraw for every mouse-button event, as would be the case with a general picking algorithm. Instead, the picking operation starts by mapping the screen coordinates of the mouse to projection space (i.e., the inverse View Port mapping), and then "back-projecting" this

intermediate result to each of the three major planes. For the XY plane, for example, reverse view port mapping followed by back-projection answers the question: "which (x,y,0) world coordinates project to these (u,v) screen coordinates?" The answer to this question is a unique point on the XY plane; it is then possible to test if this point lies within the defined viewing extent. If it does not, then the mouse was not in the area drawn as the $Z = 0$ wall, and the process can be repeated with the other planes (XZ and YZ).

Every screen location can be back-projected to a unique point on each of the three major planes. However, of the three resulting points in 3-D space, at most one will be completely within the bounds of a shadow wall—this is guaranteed to be true because none of the walls overlap. It is possible, of course, that none of the three back-projected points fall entirely within the viewing extent, in which case the mouse is not on any of the walls, and the picking operation fails. The mouse might still actually be within the three-dimensional "viewing cube" when this happens (see Figure 3.1), but since picking is only allowed on the shadow walls, there is no need to worry about this complexity.

When the picking operation has identified the proper shadow plane, it passes the world coordinates of the mouse to the molecule, to check if the position is within the pick region. In a general algorithm, this would involve looking at each component—each atom and each bond—to see if it overlaps the given location. To simplify that process, the pick region is defined to be a bounding box (described by two points in space) which completely inscribes the molecule plus an extra margin (to simplify picking small objects). With this information, completing the pick operation is simply a matter of checking if the mouse

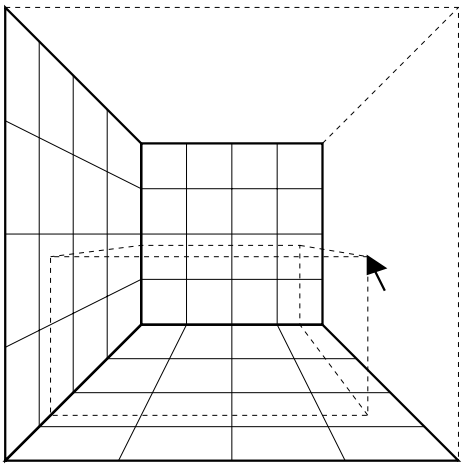

Figure 3.1: The mouse pointer is in the viewing cube (as evidenced by the dashed lines), but it does not lie on any of the three shadow walls. Thus, it is not in a valid pick region.

location is inside the bounding box. Since these are shadows—parallel projections onto the plane—the comparison only involves two out of the three coordinates (the other is set to zero). This implementation of picking would also work for rotation manipulations, as long as the centre of rotation is fixed (the centre of gravity of the object, for instance). If this is not the case, if the subject determines the centre of rotation as well as the angle, a more precise form of picking is needed, that can select a point rather than a region.

A few more details regarding the bounding box used as the pick region need to be considered. In order for the comparisons described above to be direct and simple, the cube must be defined in the World rather than the Object coordinate system. Thus its position must be re-computed whenever the molecule moves. When the pick operation is invoked, it first checks if the molecule has moved since the last call (by way of an internal

flag, set during move operations). If it has not, then the previously-computed bounding box is still valid, and is re-used for efficiency. Otherwise, the molecule is traversed and the minimum and maximum extents in all three axes are recorded. A constant—the margin—is added to the maxima and subtracted from the minima, and the two resulting triplets define the new picking cube.

The implication of this simplified picking cube is that the user can select a molecule by clicking on some "white space," which does not contain any shadows (but which is enclosed in the picking region). While this method might sound awkward, there is actually no noticeable distraction. Because the molecules are composed of two vertically-aligned atoms, the bounding box fits relatively "tightly" (non-aligned atoms would result in a much looser fitting cube). Most of the white space surrounds the bond, which users will rarely attempt to select.

To complete this discussion of input processing, it is appropriate to describe what follows a successful picking operation, that is the assignment of labels to the different views. The shadows are examined in the following order: $X = 0$, $Y = 0$ and $Z = 0$; the picked view becomes the interactive shadow and the other two are assigned the shadow 1 and shadow 2 labels, in that order. The latter two views are therefore indistinguishable in terms of results. For example, if the $X = 0$ shadow is picked, the $Y = 0$ shadow is assigned the label 1; but if the $Z = 0$ shadow is picked, the $Y = 0$ shadow becomes view 2.

3.4 Introducing Delays

With the support libraries in place, defining all the necessary input and output routines, the next step is the application itself. The purpose of the application, of course, is to manage multiple views, and to add delays and to introduce de-synchronisation. For this to happen, it is necessary to accept user input—mouse input, for example, channelled through the modelling transformations as described above—and apply it to different views, at different times. Associating the views with delay values—the experiment configuration phase—is described in the section below; this section concentrates on controlling the delays.

In the early stages of development, it seemed sufficient to define a single delay value per trial, a delay which would then apply to some combination of views. Thus, in a given trial, any lagging view was delayed by the same amount of time. Initially, this assumption seemed to simplify the design of the program. Two copies of the user molecule (i.e., of the molecule which could be manipulated) were defined: one that would always be "on time," glued to the current pointer location, representing the *leading* views, and one that would be delayed by the amount specified by the current configuration: the *lagging* views. To draw a given view—say the $X = 0$ shadow—it would first be necessary to determine how it was affected by the current configuration, and then call the appropriate molecule's rendering algorithm. Thus for example, if the configuration specified that the $X = 0$ shadow was delayed, the lagging molecule's *draw()* routine would be invoked.

As user input (in the form of mouse movement) arrived, it was immediately applied to the leading molecule, so that this molecule's position was always up-to-date, and was

also stored in a queue, for later use in the lagging views. Again, with a single delay value defined, there was only need for a single queue, apparently simplifying matters. Each entry in the queue would contain a position and a time at which the entry would be de-queued and processed.

This simple scheme worked well with delays of a few hundred milliseconds; delays short enough that the lagging views had time to catch up before the user could initiate a new manipulation operation. With longer delays, in the thousands of milliseconds, however, the limitations of the one-queue design were rapidly discovered.

To illustrate the main problem, suppose the current configuration specifies that the *interactive shadow* be delayed by $T = 2000$ milliseconds. The user selects the $X = 0$ shadow, and moves it. At this point, the $X = 0$ shadow is drawn using the lagging molecule, and is two seconds behind the other three views, which are drawn using the leading molecule. The user then releases the shadow, and less than two seconds later (so the lagging molecule is still catching up) selects the $Z = 0$ view. Now it is necessary to draw this new shadow using the lagging molecule, which is still in use by the previous ($X = 0$) view. If one attempts to merge the two uses, either the $Z = 0$ shadow jumps backwards to the not-yet-caught-up position, or the lagging molecule is updated and the $X = 0$ shadow jumps ahead to the caught-up position. Both of these options introduce inconsistencies in the delay paradigm, and while the second case might appear less disruptive to the user, it has a serious drawback in terms of the experiment. Indeed, subjects never need to wait for a long delay to end: they simply need to click momentarily on one

of the other shadows for the previous view to catch up immediately (and be ready for re-use). In effect, this artifact invalidates the results for long delays.

In an attempt to work around this problem, all views were forced to catch up after each manipulation ended and before the next one could begin. The user would now have to wait for all views to be synchronised before a new shadow could be selected for interaction. In this way, conflicting delay configurations never occurred. On the other hand, this solution introduced a new set of problems, namely how to handle the forced re-synchronisation.

Two options were considered: either the delayed shadows could catch up at the normal rate, keeping the artificial delays consistent, or they could catch up as fast as possible by having all position updates dequeued with no regard for proper dequeue time. In the first case, the time-to-complete measure is skewed by forced pauses after each individual interaction operation. For example, if the configuration specifies a one second delay and it takes the user four operations to complete the task, then four seconds of the final time-to-complete will have been spent simply waiting for views to catch up. The second solution mostly avoids this problem; a pause is only introduced for as long as it takes to empty the queue (in the order of hundreds of milliseconds). However, this solution also breaks the delay paradigm and makes the system appear inconsistent. Also, it re-introduces the initial problem. In the case of very long delays, if the user wishes to get feedback on the current state, all she needs to do is release the mouse button. Within an instant, all views are synchronised and ready for interaction—so again, the experiment can not effectively measure the effect of long delays in the feedback loop.

The crucial observation with the initial design was the realisation that it was in fact desirable to be able to delay different views by different times within a single trial (see Chapter 4). Thus the design was modified to accommodate arbitrary delays on a per-view basis; as the following description will show, this not only met the extended requirements, but also solved the problems described above.

The new design uses four separate internal event queues—one per shadow view and one for the perspective view, even though the latter allows no interaction. The structure of the queue elements remains unchanged, with position and time-to-dequeue fields; the difference is that now, this position information applies to exactly one view. With this scheme, a single lagging molecule is no longer sufficient, so there are four "working" copies of the user molecule. Each copy is associated with one queue, which is to say it holds the current position of the molecule in a given view. Note that the original molecule is still needed, to store accurate up-to-date position information even if all views are delayed—so in fact the program uses five copies of the same object.

It is interesting to note that this design would be easy to adapt to a multi-processor environment. Each view could be handled on its own processor; a central task would dequeue position updates and communicate them to the various view-handling tasks, which would update their private copy of the molecule. This type of design, built on a multi-processor real-time platform such as described by Gentleman, MacKay, Stewart, and Wein (1989), would allow extremely rigorous control over the timing of the various views, especially if a non-serialised graphics output mechanism is used (as described, for example, by Bertrand, Cowan, and Wein (1993)). For the purposes of the experiment,

however, sufficient control over delays can be obtained by using a simple-minded, single-tasked, serial approach to drawing the different views.

As in the previous design, incoming position update requests (mouse movement information) are immediately applied; in this case, however, they are applied to the original "user molecule," which never gets drawn. As a second step, each view is examined in turn and handled according to the current configuration. For example, if the configuration calls for a delay of t milliseconds for the $X = 0$ shadow, the new position is placed at the tail end of the queue for that view, with a dequeue time-stamp set to the current time plus t.

Counter-intuitively, the situation is slightly more complex if the configuration does *not* call for a delay. The instinctive action of immediately updating the corresponding molecule could be wrong. Suppose for example that a position update arrives at time-stamp T_0 for a view—say the $X = 0$ shadow again—which was delayed by some time t in the previous interaction, and that this interaction completed less than t milliseconds before T_0 (in practice, this will only occur if t is long, in the order of thousands of milliseconds). In this case, the view is still catching up, and there are still some position updates in the $X = 0$ queue, with dequeue time-stamps greater than T_0. If the current position of the $X = 0$ view is simply updated with the most recent information, the shadow will appear to jump to the new location and, even worse, will then jump backwards when the next "catching-up" event is dequeued. To avoid this problem, when faced with an un-delayed view, the associated queue is first checked for entries. If it is empty, then the position of the view's molecule can indeed be updated immediately; otherwise, the

position update is placed at the tail end of the queue for later processing. In all this discussion, entries are always added to the tail of the queue. One might be tempted, instead, to sort the queue according to time-stamp. But this would be wrong and would lead to inconsistent results similar to those described above: short and long delays would mix in the queue, and the view would jump back and forth between potentially very different positions.

While the enqueuing process operates explicitly on the four different queues, so that each view can function independently of the others, the dequeuing operation, on the other hand, is simplified by viewing the internal queues as a single "virtual" queue. For example, the first step in retrieving a position update is to check if there are any such updates stored in the queues; while internally, all four queues must be checked individually, the external interface is a single call that returns empty only if all queues are empty. Nonetheless, there is also a version of the call that checks a single, specified queue and which is used to determine whether to enqueue an update or apply it immediately, as described above.

To return to dequeuing, if at least one entry exists in the "unified" queue, the next step is to obtain the time remaining before the next position update is valid. Again, this is done with a single call—*tmToEvent()*—which selects the oldest of the head elements in the four queues. Recall that entries are not sorted by time-stamp, so the oldest head element may not be the oldest queue element; nonetheless it is the correct "next event," for the previously-mentioned consistency reasons. The routine returns the number of milliseconds before the chosen entry needs to be dequeued; this number is 0 if the time has passed (i.e., the entry's time-stamp is smaller than the current time). It will also return

the view type of the event. This last piece of information—the view type—is necessary when an entry is actually dequeued, so that the correct molecule can be updated and redrawn.

In summary, the second design provides a much cleaner and much more flexible experiment environment. Not only can the configuration specify different delays for each view, but also the system is, in general, more consistent than with a single delay queue. There is no longer any need to resort to artificial means to force views to catch up during a trial, nor does the user need to wait for the views to catch up before proceeding to a new interaction.

In the first design, end of trials were as difficult to handle as end of interactions: should the end be flagged immediately, keeping time-stamps consistent but causing problems with up-coming position updates, or should the end wait until all views have caught up? There are no such problems in the new, multi-queue design: when an interaction ends and the molecule is in the target location, the end of trial is immediately recognised, so that the timings are completely accurate. Immediate visual feedback is provided—by changing the "bonds" between the two molecules—to signal that the trial is indeed finished, even if some (or all) of the views are still catching up. The queues are then left to empty at the normal rate, unless the user signals the start of a new trial. In that case, it is necessary to interrupt the normal course of events and purge all queues. As the new trial starts, the molecules are placed in randomly chosen initial locations, and it would be wrong to continue using the queued-up position from the previous trial. Furthermore,

since the molecules must jump to a new location anyway, interrupting the catching-up motion remains un-noticed.

3.5 Merging Internal and External Event Queues

The discussion of event handling has so far focused mainly on storing and retrieving internal events. Incoming external events have also been mentioned as the initial source of information for these internally-queued events. The nature of this external source should now be examined in more detail, along with some of the difficulties involved in managing it.

Events, such as mouse movement or button actions, are delivered to the experiment code by the X11 event queue common to all X-applications. The normal method of handling this queue is to build the program around an event loop, which waits for input to arrive, processes the input and returns to wait for further events. At first glance, this organisation appears to fit very well into the design at hand. However, the particular needs of the experiment require a more complex treatment of the event loop structure because two sources of events must be merged: the external X11 queue and the internal set of queues, which require real-time attention. So if one were to use the usual *XNextEvent()* call—a blocking call, which will only return when an X event is available—anywhere in the event loop, there is the risk of being blocked on the external queue while internal events come due, thus causing some dequeuing deadlines to be missed. If this happens, the system will appear to "freeze," and views in the process of catching up will halt until an external event occurs.

One work-around to this problem would be to interrupt a blocked *XNextEvent()* call when an internal event becomes ready for processing. A scheme was considered where the internal queue mechanism sets an alarm for the next dequeue time; the signal handling code (which would be separate from the main event loop, and so shouldn't actually process the event) would then use *XSendEvent()* to place the item in the external queue, where it would be available to the blocked *XNextEvent()*. Apart from the complications of setting and handling signals, this solution has the disadvantage of introducing an unknown—and unpredictable—amount of delay (due, for instance, to signal-handling overhead) in the pseudo-real-time event loop.

The ideal solution would consist of a version of *XNextEvent()* that accepts a time-out value. The main loop could then determine the time remaining until the next internal event, and block on the external queue until that time. If an X event arrived in the meantime, it would be processed immediately; otherwise the blocking call would eventually time out, and the internal event would be processed. Unfortunately, this version of the blocking call is not available, nor is it easy to implement it as it would require some re-coding of Xlib.

The solution finally adopted for this experiment attempts to simulate the functionality described above using standard library calls and a polling loop (see Figure 3.2 for a pseudo-code description of the algorithm). From a theoretical standpoint, the use of polling is an unfortunate design choice as it continually keeps the CPU busy—even during idle periods—and prevents proper scheduling of other tasks. However, since the code is contained in a single task, scheduling problems (due to busy-waiting) do not cause

contention among various parts of the experiment; only system processes and other users' tasks are initially affected. Nevertheless, allowing proper scheduling of other jobs is a concern—a queue of back-logged jobs will eventually interfere with the pseudo-real-time nature of the event loop—so to ease the process, the task is repeatedly put to sleep within the polling loop as it waits for internal events to be dequeued (see line 8 of the pseudo-code). As a result, some blocking is introduced in the algorithm, and this reduces the amount of CPU busying during idle periods.

Note that the longer the sleep time, the less this algorithm ties up the CPU and in this respect, the longer the sleep period the better. Also, since it is known exactly how long until the next internal event is ready, there is no risk of "over-sleeping" and missing the deadline. However, what is *not* known is time until the next external (X) event arrives, thus it is important to regularly wake up and check (poll) that queue. The longer the wait between polls, the longer an event such as mouse motion can remain un-processed. This has the potential for introducing delays—uncontrolled and undesirable delays which could affect measurements. On the other hand, the shorter the sleep period, the more responsive the system will be, with quicker response to user input. This is true all the way down to zero sleep time, where the algorithm will process incoming events as soon as they appear, thus introducing no extra delays at all. But now the algorithm is once again based entirely on polling, keeping the CPU constantly busy. Thus finding the ideal sleep time is a matter of striking a balance between lowest possible delays in processing incoming events and lowest possible utilisation of the processor in idle periods.

```
1.  loop forever
2.    if internal queue not empty
3.      Tnext = time to next internal event
4.      if Tnext has arrived (or passed)
5.        dequeue and process internal event
6.      else
7.        do
8.          sleep (small constant time)
9.          poll X event queue
10.       until Tnext has arrived or X event has arrived
11.       if X event arrived
12.         dequeue and process X event
13.       else
14.         dequeue and process internal event
15.       endif
16.     endif
17.   else (internal queue empty)
18.     block on X event queue
19.     dequeue and process X event
20.   endif
21. end loop
```

Figure 3.2: Pseudo-code description of the main event-processing loop

For this experiment, a constant value of ten milliseconds was chosen, which is roughly one order of magnitude smaller than a "short delay" in the experiment configurations. While choosing a constant value certainly simplifies matters, and appears to be sufficient in this case (no scheduling problems seem to arise during the running of the code), it is by no means the only option. One could imagine a scheme where the delay increases (geometrically or exponentially) at each pass through the loop, to be reset to a small value when an external event does arrive. The reasoning behind this is that the user will tend to generate bursts of input followed by relatively long pauses; in particular in this experiment, it could be expected that pauses often last until all views catch up—justifying, perhaps, less frequent checks of the X event queue. However, the delay should not be allowed to increase beyond a pre-determined value, so that the user does not experience an extremely long response time after having paused for a while.

Of course, if there are no events in the internal queue, it is safe to block on the X queue without resorting to the polling loop (see line 18 of the pseudo-code in Figure 3.2). Indeed, new internal events can only be generated as a direct result of user input; if all views are currently synchronised (as indicated by an empty internal queue), none will need catching up until the user moves. Until that move, which will be signalled by an external event, the system is completely idle.

Much of the discussion so far has focused on timings and time-stamps, but without describing the source of the timing information. There are two such sources available to the program: the X-server's time-stamp in event messages and the time values returned by system calls. At first, the X time-stamp was appealing: the information is in every

event message and can be retrieved with absolutely no overhead. But the drawback was soon apparent: if the current time can be obtained only when an event arrives, some form of "dead reckoning" must be used between external messages to evaluate the time remaining until the next internal queue event. In other words, it would be necessary to keep track of the number of milliseconds slept and estimate (or ignore) the number of milliseconds spent processing the algorithm since the last known time-stamp. And of course, there would be no choice but to ignore the time spent processing other tasks (recall that this is a multi-tasking environment), since there is no means of receiving that type of scheduling information.

Consequently, system calls were chosen as the means to obtain a time-stamp. This choice presumably introduces extra overhead (cost of context-switching), but it means the time can always be accurately determined, both when events are enqueued and when the queue is checked, within the polling loop. Although the system calls used return a value in microseconds (on DEC OSF/1 AXP) or nanoseconds (on Sun Solaris), they are not necessarily micro- or nanosecond accurate. Thus, in both cases, millisecond accuracy only is assumed, which is sufficient for the experiment, since there may be up to a ten millisecond delay before the X queue is checked, so that a millisecond difference represents a (non-cumulative) 10% error.

This completes the discussion of implementation of the core functionality of the experiment. As mentioned at the beginning of this chapter, development of the remainder of the code—the input script parser and the statistics generator—involved no difficulties or challenging design choices. However, while the code itself is uninteresting, the infor-

mation on which it operates requires some attention, as it has a direct effect on both the flexibility of the entire system and on the final results. What follows is a description of the input—in the form of configurations scripts—and output—statistics on the subject's performance—as well as some tools for managing and processing both. Finally, an outline is presented of the measures taken to ensure maximum performance during the actual runs of the experiment.

3.6 Experiment Customisation: the Input Script

In allowing the experiment to be driven by an input script, which is parsed at run-time, flexibility in configuring the system is ensured, especially during the pilot trial period. Performance considerations may have suggested a pre-compiled driver (in the form, perhaps, of a dynamically-linked library) or perhaps even a hard-coded configuration compiled in with the experiment code itself. This would have saved the overhead (which is in fact low) of parsing the input script, but would have made it difficult to make repeated minor adjustments to the various input parameters. As it stands now it is extremely easy to change a delay value or an option setting in the script, to re-run the experiment and to get immediate feedback.

The main focus of the input script, then, is to define the delays that affect each trial. Initially, when the concept was that a single delay value would affect all lagging views in a trial, this definition took the form shown in Figure 3.3: the delay itself, expressed in milliseconds, followed by a list of views to which it applies. A view which

```
[...]  Delay 1000 Config i2 [...]
```

Figure 3.3: Excerpt from first-generation input script

was not mentioned in the definition would by default be a "leading" view: it would be in synchronisation with the pointer.

Recall from previous discussions that the labelling of the four views is as follows: *i* is the *interactive* shadow (the view with which the subject is interacting), *p* is the *perspective* view (which can never be manipulated), and views *1* and *2* are the other two shadows (those not currently being manipulated). Thus, in the example above, whatever shadow the user selects for manipulation and one of the other two shadows are delayed by 1000 milliseconds, while the perspective view and the last remaining shadow keep up with the mouse pointer.

In time, it was realized that this setup could be improved. For a variety of reasons (some of which were described in a previous section and others which will be explained in the next chapter), it was decided to allow each view to be delayed by a different amount. For this, a change in the format of the input script was necessary, to allow specification of multiple delays. As can be seen in Figure 3.4, the only difference is that the *Delay* keyword may now be repeated any number of times; views, introduced by the *Config* keyword, are configured with the most recently defined delay value. Once again, views that aren't mentioned have a delay of zero milliseconds.

```
[...]  Delay 100 Config i Delay 400 Config p1 [...]
```

Figure 3.4: Excerpt from the revised input script

[**Autobind** 0 | 1] [**Delay** *t* **Config** *cfg*]* [**NumPractice** *nmprct*] **NumTrials** *nmtrl*

Figure 3.5: Full description of the input script language

The examples shown so far are only excerpts of a full trial configuration. The complete description is shown in Figure 3.5, where square brackets denote optional fields.

As this shows, the only mandatory field is the one defining the number of measured trials to run with the current configuration (while this value must be present, it can be zero). Prior to this, an optional field defines the number of practice trials which, during a run, are signalled by a flag. The purpose is to give subjects time to adapt to new configurations, if the experimenter thinks this is necessary, without having final results tainted by a learning-curve effect.

The delay configuration field is described above; note that this field is optional: by default, all views are configured as un-delayed.

The first field allows modification of the end-of-trial behaviour. By default, *Autobind* mode is turned off; this is the behaviour described in the section on input processing, where the user molecule is checked for proper positioning only when the subject ends an interaction (releases the mouse button). If Autobind is turned on (with a non-zero

parameter), however, the user molecule's position will be monitored throughout an interaction (i.e., during the dragging process). As soon as a correct position is reached, the end of trial is signalled. This theoretically simplifies the placement task should pilot trials indicate that practice trials do not sufficiently reduce the learning curve; this wasn't the case, however, and Autobind mode was not used in any of the trials. (See Section 2.2 for a more detailed description of Autobinding and related issues).

The final input script involves 81 trials (see Chapter 4), with one configuration line per trial. It may strike the reader that such an input script would be tedious to write, and that it would be difficult to modify delay values so that they remain consistent across all trials (i.e., so that all short delays, for example, are identical; this is important in the analysis phase). For this reason, a "meta-script" was written, which uses symbolically defined constants—such as `LONG_DELAY` and `SHORT_DELAY`—instead of actual numbers in the configuration commands. The script is then run through the C pre-processor to generate a valid input script. The advantage of doing the pre-processing ahead of time, rather than at the time of the trials (by piping the output of the pre-processor into the input of the experimental engine, for instance) is that it reduces the CPU load during the experiment, contributing to overall performance.

3.7 The Results Files

At the output end of the system the concern is to summarise the information gathered during the experiment in a format that is as descriptive, efficient and manageable as possible, which requires compromise. Indeed, it is desirable to be able to reconstruct the

subject's actions in as much detail as possible, providing answers to complex questions about behaviour and reaction to delays. Yet the analysis phase should not be swamped by meaningless numbers, lest the processing effort be too high to answer even a simple question. And finally, the information must be stored in a fashion that facilitates access (by humans, standard tools and custom processing tools), not only to the body of data as a whole, but also to specific parts of it so that precise questions might be addressed.

When a new trial starts, the output module of the experiment addresses two of these concerns, by printing a comment string fully describing the current configuration, noting the date and time, and describing the format of the results themselves. This comment string gives valuable information for interpreting the numbers, and also provides identification useful in data set management. Each line of the comment is prefixed with the "#" symbol, adhering to the shell programming standard, and facilitating further processing with tools such as *awk*, *sed*, etc.

The data itself is arranged in four fields. The first three, which are always present, are those described in the comment string mentioned above: *event*, *time* and *distance*. Strictly speaking, there are only two types of events: *mouse down* and *mouse up* (except in AutoBind mode, see below). However, to aid analysis of the data, this classification is refined to include a *start* and an *end* event for each trial, referring to the initial mouse-down and final mouse-up respectively. This refinement allows analysis tools to parse the results in a context-free manner, without any need, for example, to look ahead to determine if the current mouse-up is the last one in the trial. Note also that in AutoBind mode, the end of the trial is not associated with a mouse-up event (it occurs as soon as

the user molecule is in position), so that in this case a separate *end* event is a necessity, not simply a convenience.

Associated with each event is the information used most in the analysis, the time-stamp, used to measure the subject's performance. Recall from previous discussion that there are two possible sources of timing information. For internal queue management, the system call approach was used, favouring timeliness (not having to wait for an X event and its time-stamp to arrive) over reduced overhead. In the case of generating statistics, however, time-stamp information is needed only when a significant X event occurs—usually a mouse button event, but also, in AutoBind mode, a mouse move event. In all cases, the event messages come bundled with an X time-stamp, which can be used directly, thus saving some system-call overhead and, more importantly, generating more accurate data. Indeed, the time-stamp included in the event message will be as close as possible to the "true" event time, whereas time obtained from a system call would necessarily include some variable amount of delay (time for the X message to be placed on the queue, time for the application to receive and process it and time needed for the system call itself to return).

The times included in the results files, expressed in milliseconds, are therefore taken directly from the X server. Of course, the "absolute" times (where time zero is presumably when the X server was initialised) are meaningless in themselves, and are only useful when taken relative to the time of the first event in a trial; it seemed simpler to leave the required subtraction for the data analysis phase rather than build it into the output module of the experiment.

Distance, the last mandatory field in the experiment output, provides information for a secondary metric of subject performance: accuracy. By including the distance with each event, it should be possible to track the subject's progression, and visualise the speed / accuracy tradeoff. By isolating interaction operations (i.e., mouse-down, mouse-up pairs) that result in very small changes in distance, it might also be possible to identify "fine-tuning" stages, which might then be analysed separately. The distance information, expressed in World Coordinate units, is computed as the norm of the vector that connects the centres of the two molecules. Unfortunately, this is insufficient to determine distance from the user molecule to the target area, defined as a toroidal region around the reference molecule. To determine if the target has been reached, it is necessary to compare the distance, as computed above, to the inner and outer radii of the torus *and* the vertical position of the user molecule to the height of the target region. By relying only on distance and ignoring vertical position—as must be the case when this incomplete data is analysed—one actually defines the target region to be a sphere.

Thus the information included in the results can not reliably provide information on subject accuracy in placing the molecule. For instance, a move (a mouse-down, mouse-up pair) that changes the distance only by a small amount, just outside the outer torus radius might, at first, appear to be a small, fine-tuning adjustment, suggesting that the user is on the verge of placing the molecule correctly. But in fact the reality may be quite different: perhaps the molecule is just outside the outer *sphere*, but high above the torus, and the user is making significant vertical adjustments while remaining (inadvertently, perhaps) just outside the correct radius. The assumption that this move is a fine-tuning stage is incorrect.

This deficiency in the results file was discovered after all the subjects were tested, and at that point it was too late to change the format. Should any future work require the same experimental platform, however, the output statistics should be augmented to include, at the very least, the vertical distance between the two molecules. Alternatively, for even greater detail, the full coordinates of both molecules could be used; this would allow finer tracking of the subject's movements, although that would come at the cost of more complex computations during the data analysis stage.

Finally, the data files contain a fourth, optional, field consisting of the keyword *Practice*, indicating that the current trial was marked in the input script as a practice run, and should not count in the final scores. It is therefore up to the analysis tools to check for this field, and react accordingly. It would have been possible, of course, simply to omit printing data generated by practice trials. However, it might be useful in some cases to compare practice trials to real trials. For example, this would make it possible to determine if there are strong learning curve effects, and perhaps even decide if some of the real trials are tainted by such effects.

Two of the goals mentioned at the start of this section have now been addressed: providing descriptive and efficient data. The final goal—manageability of the body of data—is the responsibility of the storage scheme, which dictates that one file be created for each configuration encountered in the input script (so if multiple trials are identically configured, all their results are placed in the same file, facilitating retrieval of closely related data). For performance considerations, data is only written to the file at end of trial, ensuring that I/O delays (disk wait) have no adverse effect on either the subject

or timing information. The result files' names are automatically derived from the trial configuration, and are of the form: `prefix-p-i-1-2`, where *p*, *i*, *1* and *2* are replaced by the delay values (in milliseconds) of the respective views. The *prefix* is specified by a command-line option when the experiment is started; by convention, in this experiment, the subject's initials were used.

The advantage of this scheme over the simpler one of placing all results in a single large file is that it is trivial to isolate the data for any particular subject and configuration. Furthermore, through judicious use of wildcard file-name matching in shell commands, it is easy to analyse sets of configurations. For example, `fj-2000-0-*` would retrieve all trials with very long *p*-delays but no *i*-delay for subject *fj*. Note that the most common analysis—on the entire body of data for a given subject—can be performed in the same way, using wildcards, which is no more difficult than if all results were in a single file.

3.8 Processing the Results

Processing and analysis of data is a vital component of the overall study and this section describes the tools used to interpret the data. These tools are completely separate from the experiment itself and were developed even though any number of other analysis methods (such as the *SAS* package, for example) could have been used on the experiment data.

The first step in processing the results is to summarise the detailed information. Indeed, while data on the subject's every move can be useful in a refined analysis, as a first pass it is preferable only to compare performance (in terms of time-to-complete and

number-of-trials) between different configurations. For this purpose, a simple *awk* script was written to pick out the start and end of a trial—using the corresponding keywords in the results file, and ignoring practice trials—and write out a one-line summary with the configuration, total time and number of operations.

A more advanced version of this script is capable of ignoring moves when the molecule arrives within a certain distance of the target, in effect simulating an early end to the trial. The goal of this approach is to eliminate the fine-tuning stage, on the assumption that it is not relevant to the study (see Chapter 2). Unfortunately, as described in the previous section, the script must rely on incomplete information for this type of processing. Without proper vertical placement information, it is not possible to determine how close the molecule is to the actual toroidal target area, and the sphere inscribing this torus must be used instead. Thus some trials may be terminated early incorrectly, when in fact the molecule is still far from its target area.

Finally, for the statistics themselves, the "exploratory analysis" approach (Tukey 1977) was thought to be the best option, as it may identify unexpected patterns and behaviours, which might otherwise remain hidden. To this end, the summary files produced by the awk scripts were copied to an Apple Macintosh, and used as input to the DataDesk statistical analysis package (Velleman 1989); the actual analysis steps are described in detail in Chapter 4.

3.9 Fine-Tuning the Run-Time Environment

Much was said throughout this chapter about designing the code with performance in mind. However, in the end, the biggest boost in performance came from carefully preparing the run-time environment. The first step here, of course, is selection of the platform itself. When work on the experiment code was started, the fastest machines available to the author were single-processor DEC Alphas (DEC 3000 model 600 AXP). The prototypes gave acceptable results on minimally-loaded systems, provided certain precautions were taken to allow for the idiosyncratic behaviour of the hardware. Proper vertical alignment of the window on even scan lines, for instance, was a *sine qua none* condition of performance (see Section 3.2).

With the arrival of dual-processor Sun workstations (Sun SPARCstation 20) in the laboratory came a more reliable platform for the experiment. Performance, measured subjectively during prototyping sessions, appeared more consistent, with less uncontrollable delays in the event loop. When the mouse was moved, the (un-delayed) views on the screen were perfectly glued to the pointer, whereas on the Alphas, there were occasional lapses. A probable explanation to this is that with their two processors, the Suns can devote one CPU to the program (where the event loop busy-waits) while the other is free to run the X server and manage mouse input. On the Alphas, both these tasks—as well as the other inevitable Unix tasks—must share a single processor, causing slight delays (even though the Alpha chip is faster than the SPARC). In fact, should the need arise to display the experiment on an Alpha's display, better performance could be obtained by running the code remotely on a Sun rather than running it locally. This surprising

result—true only if the load on the ethernet is low—confirms that the bottleneck really occurs at the CPU.

Even on the Suns, however, performance degraded rapidly as soon as other processes vied with the experiment code for CPU time. Something as seemingly innocuous as an *xterm* could introduce significant uncontrollable delays. To ensure nothing of the sort could invalidate the results, the machine was brought to a state as close to single-user as possible before starting the experiment on a subject. This involved killing "superfluous" jobs (such as *xclock* and *dsdm*, the drag-and-drop manager), asking all other users (if any) to log out, and modifying the password file so that no further logins could occur until the experiment was over. These measures do not guarantee that no other process will run during the experiment (in fact, Unix guarantees that daemons and other system processes *will* run), but they were successful in preserving near-real time performance throughout the eighty-one trials for all nine subjects.

Chapter 4

Experimental Design

4.1 Pilot Study

The initial thrust of this research was to examine the effect of de-synchronisation among the various views of a task, using a simple multi-view three-dimensional environment as a test bed. In this context, the general issue is to examine the effect on subject performance (time and number of operations required to complete a set task) of introducing delays in the various views of the scene. To address this, the experimental platform initially allowed a simple form of control: a given view could be either synchronised or de-synchronised and a single delay value was sufficient to characterise each trial (see Chapter 3 for the implementation details of this limitation). While this simplified view of de-synchronisation can not provide answers to the entire question, it was hoped that it would provide insight into meaningful subsets of the problem, and narrow down the research to a size suitable for the scope of this work.

Pilot experiments were designed around this implementation to test combinations of synchronised and delayed views in order to answer some preliminary questions. For instance, how useful is a given view to the completion of the assigned task? This question, which may appear to be outside the scope of this research, is an instrumental first step in addressing the larger issue: how is that usefulness affected by a delay? Thus the pilot input scripts were designed to measure the subject's performance under various delay configurations for a given view. For example, in a sequence of three trials, the subject might face an un-delayed perspective view, followed by a short and long delay on that same view. Since nothing except that delay changed in the three configurations (and assuming there were no learning effects), the results could be compared to one another to determine the role and usefulness of the view.

An important assumption in this phase of the study was that a long enough delay (600 milliseconds, in the pilot studies) guarantees that the view is unusable. If so, all changes in performance between the no delay and large delay trials can be attributed to the loss of the information normally provided by the view, indicating the usefulness of the view. With this information as a baseline, the short delay trial evaluates the consequences of de-synchronising a particular view.

Unfortunately, this experimental method considers each view in isolation. It is likely, however, that some views are useful only in conjunction with others. For example, the perspective may not be useful alone, but combined with one of the shadows it may provide sufficient information to complete the task. Identifying such pairs (or perhaps triplets) of views and introducing appropriate de-synchronisation is clearly important. For this

purpose, a refinement of the pilot study looked for meaningful groupings (including, for instance, the two non-interactive shadows, or the interactive shadow and the perspective view) by repeating the no-delay, short-delay, long-delay sequence of trials.

The pilot trials were conducted principally on the author with two additional subjects used when the input scripts reached maturity. The trials showed the inadequacy of the 600 millisecond delay (the long delay) in rendering a view useless: a much longer delay is necessary. Unfortunately, because of technical constraints in the implementation (see Chapter 3), longer delays were impractical. Analysis of the results also showed that the chosen groupings of views into "useful subsets" were not all meaningful, and that some groupings were missing. Indeed, the data seemed to indicate that the main effect of delays was on the interactive shadow, and it hinted that if there were effects on any other view, then there were possible interactions between all different views.

These early results led to a re-design of the experimental platform and a narrowing of the research objectives. The focus of this study was shifted to the interactions between the four views, to determine exactly which combinations of delays would affect a user. The redesigned experimental system is able to test any combination of delays and views and to handle extremely long delay values. The technical aspects of this re-design are outlined in Chapter 3.

4.2 The Main Experiment

As explained above, the main experiment aims to identify which views or combinations thereof are sensitive to delays, and to determine the *quantitative* effect of delays: how

much a delay affects a user, on a per-view and per-combination of views basis. To satisfy the first of these objectives, all meaningful combinations are studied, using all permutations of delays and views in a factorial design. Furthermore, in the interest of the second objective, the set of delays to be applied to each view should include as many different values as possible; the larger the number of (stimulus, response) data points, the better the resolution of the resulting curve.

From a practical standpoint, however, the set of trials must be small enough to complete the experiment without exhausting the subject. Indeed, the pilot study showed the repetitiveness of the task rapidly leads to boredom, which may skew the results.[1] Preliminary results showed that each trial takes about fifteen to twenty seconds, and that a session can be about thirty minutes. It was also observed during pilot trials that approximately ten trials were necessary for the subject to become fully accustomed to the task. (This applies to subjects who are already generally familiar with the task, not to complete novices, who need an initial training session.)

With these constraints, only three delay values (none, short and long) per view are possible, yielding 81 (3^4) measured trials, plus eleven trials for practice. Each individual configuration (for instance, no delay in any of the four views) is tested only once, however, during analysis, data can be grouped to increase the number of samples. So for instance, the effect on performance of the interactive shadow delay (i-delay) is analysed by dividing the 81 samples into three subsets, one for each value of the i-delay; there are then three

[1] Parasuramon (1986) and Hockey (1986) both present evidence that repetitiveness, boredom and fatigue contribute to degrade performance in an assigned task. Parasuramon also reports that reaction time at a vigilance task can increase by hundreds of milliseconds over the course of an hour-long test. Problems like these would clearly interfere with the effects of de-synchronisation.

(i-delay, performance) pairs, defined by twenty-seven samples each. Three distinct data points per stimulus do not yield a very detailed curve, but do give a satisfactory idea of the effect of delay on performance. At approximately thirty minutes for 92 trials, it appears to be the best compromise between extra data and shorter experiments.

The actual values used for the delays were obtained by trial-and-error during the pilot study phase. Delays of less than a hundred milliseconds—which has been found, experimentally, to be the typical cycle time of the perceptual processor in the human brain (Card, Moran, and Newell 1986)—are barely noticeable, and were thus deemed inappropriate for the experiment. A value of twice this threshold (i.e., 200 milliseconds) appears to be the shortest clearly noticeable delay, and was chosen as the experiment's "short delay." The preliminary trials also revealed that the perceived difficulty of the task, in subjective terms, starts increasing dramatically around delay values of 800 to 1000 milliseconds, depending on the subject. A "long delay" of 2000 milliseconds (two seconds) was therefore chosen, to ensure that all subjects would be faced with a difficult task.

When all the delay values were chosen, the only remaining work was to generate the configurations for the trials, and to decide on their sequencing. The configurations were written in the meta-scripting language described in Section 3.6; to ensure that no permutation of delays was missed, the trials were initially enumerated in strict order (first all the zero i-delays, then all the short i-delays, etc.). However, if the experiments are conducted in this order, subjects might recognise the pattern and anticipate up-coming configurations, or some more subtle forms of learning effect might occur; in either case,

this may skew the results. Thus the order of the trials was randomised. In this way, it is impossible for first-time subjects to predict the configuration of up-coming trials.

4.3 The Secondary Experiment

As mentioned above, in order to keep the experiment from running too long, only three delay values were tested for each stimulus. The first few tests with the finalised input script confirmed that this is enough data to obtain a general picture of the effect of delays on the various views (see Chapter 5); however, as expected, the curves obtained from this data are sketchy, as they are only based on three points. It would be interesting to focus on the most relevant curve and fill it in with more data points. Preliminary results suggested that performance varied the most as a function of the i-delay, so a second script was created to study in detail the effect of delays on the interactive shadow—and on that view only, making it possible to look at a larger number of separate delays without running an impossibly long experiment.

The range of delays for this secondary experiment was chosen to match that of the main study: from no delay to two seconds. Rather than a single value in between, however, this experiment has configurations every 200 milliseconds, including the two end-points, yielding eleven data points to define the response curve. Since no parameters vary other than the i-delay, it is not possible to partition the total body of data into sets of multiple samples per data point, as is the case for the main experiment. Therefore, to obtain sufficient data for statistical significance, it is necessary to run each configuration

multiple times: twenty-five samples per i-delay increment, with eleven practice trials to start.

Work on this secondary experiment occurred in parallel with the running of the main set of tests. When the script was ready, a prototypical subject was selected, based on his results in the main experiment, and the detailed test was administered. Due to the extra time commitment required—in this subject's case, the 176 trials took roughly 45 minutes to complete—this experiment was only run once.

4.4 Subject Selection

The placement task, while not complicated in itself, calls upon several concepts that might appear confusing to the uninitiated. For instance, the very basis for the experiment—multiple, de-synchronised views—is a phenomenon which computer-proficient users will find unsurprising, having in most cases observed it in complex real-life applications. But the same might not apply to a novice who has never even experienced delays in single-view applications; for this user, the de-synchronised movements of the various views might be startling and distracting. This research aims to study the lasting effect of de-synchronisation, not the initial effect due to confusion. Furthermore, the experiment relies heavily on the subject's efficient mousing technique; while to a power-user this seems a very natural interaction method, beginners might occasionally have coordination problems, which would certainly affect results.

For these reasons, the subjects are all chosen from a pool of computer experts: graduate students and faculty in the Department of Computer Science. In fact, all but one

subject belongs to the Computer Graphics Laboratory; this last condition ensures prior exposure to complex multi-dimensional tasks and to severe delays in interaction feedback loops. As expected, none of the subjects had any problems with the underlying concepts of the experiment.

4.5 Conducting the Experiment

As mentioned above, the subjects were given a chance to familiarise themselves with the task before the experiment began. It was initially thought that this could be achieved by a dozen or so practice trials immediately prior to the measured trials, within a single experiment. But running the pilot experiment on a subject other than the author (who had, by then, one year of practice with one form or another of the task!) quickly proved that a dozen trials were insufficient to absorb the entire learning curve. In fact, it was clear that practice trials alone were inadequate to ensure proficiency at the task, and that the subjects needed some "coaching" as well. Thus a second, short, input script was created as a preliminary, "first-contact" experiment, during which the author would demonstrate the task and supervise the subject for the first few attempts.

The two pieces of advice most often needed were a reminder that all three shadow views were available for manipulation—the "floor" (XY plane) shadow tended to be overlooked—and a clarification of the topology of the target area, around the reference molecule. In the first few trials, subjects spent much time looking for this seemingly elusive target area, moving the user molecule back and forth a few pixels each time. By

the end of the preliminary experiment, however, all subjects seemed to have mastered the task and were ready to proceed to the measured trials.

At this point, the subjects were reminded that response time was critical once a trial had started, but that frequent breaks *after* the end of a trial were encouraged, to rest the eyes and the wrist. Also, if the subjects were uncomfortable with the default mouse response (many felt that it was too sensitive and difficult to control with precision), the acceleration parameters of the X server were changed, using the *xset* command. Finally, the main experiment, including a lead-in of eleven practice trials, as described previously, was started, and the subjects were left alone, both to minimise external influences and to avoid having someone watch over their shoulder and cause stress. When the experiment ended, the author returned and asked for a brief subjective evaluation of the difficulty of the various configurations, as well as any other relevant comments. These comments are presented at the end of Chapter 5.

4.6 Analysis

The first, obvious, step in the analysis is to look at the entire body of data, and attempt to isolate general patterns. For this, as for most of the remaining steps in the analysis, the tool of choice is an Analysis Of Variance (ANOVA); in this case, the variances of time-to-complete (*time*) and of the number of tries (*numtries*) are analysed with respect to all four delay values and the subject identifier (five factors in all).

Since the analysis without interactions shows that only the interactive shadow and the perspective view play a significant role in either *time* or *numtries* (see Chapter 5),

the analysis is refined to exclude the other two views (shadows *1* and *2*). Furthermore, because the results indicate large inter-subject variability, the data is analysed subject by subject. This leads to two ANOVAs per subject—one for each of *time* and *numtries*—where the independent variables are the i-delay and the p-delay.

These results contain some outlying data that inflates the computed variance. Data like this occurs in most experiments, usually caused by momentary inattention or distraction. In this experiment, these outliers can also be explained as trials in which the subject does excessive fine-tuning (see Section 2.3). These trials can be removed based on the *numtries* field. As described in Chapter 5, subjective and numerical results both suggest difficult configurations (those involving long delays) slow the subject down, but without causing many extra click-and-drag operations—rather, each operation takes much longer. Thus a high *numtries* value, reflecting a large number of click-and-drag operations, probably indicates fine-tuning problems.

Based on these findings, a box plot is drawn of *numtries* against the i-delay—subdividing the data according to this delay value, the most significant one, allows for increases in the number of operations normally caused by the difficulty of the configuration—and outlying points are identified (DataDesk marks outliers with a star burst symbol). These points are then excluded from the body of data, and the two ANOVAs described above are repeated: *numtries* and *time*, respectively, against i-delay and p-delay. The two sets of results—before and after the removal of outliers—can be compared; if the assumption about the noise introduced by fine-tuning is correct, the second set of results should be stronger.

Finally, one of the objectives of the experiment is to obtain a quantitative relationship between the stimulus (the delays) and the response (the time to complete and the number of operations required). In other words, is the difficulty of the task (as measured by *time* or *numtries*) directly proportional to the duration of a delay, and if so, what is the scaling factor in the linear relationship? Or is the relationship non-linear, and how so? To answer these questions, the ANOVAs described above can be used to obtain predicted values for *numtries* and *time*, as a function of either of the two delays (or both), depending on which appears to be significant for the subject in question. These predicted values are then plotted against the delay, thus graphing the relationship. From the graph, it is possible to estimate the slope of the line connecting the two end-points, which gives an estimate of the magnitude of the effect of delays.

For a more detailed look at this stimulus-response relationship, the data from the secondary experiment must be examined. Here ANOVA is not an appropriate tool, so a linear regression (of both measures of performance against the i-delay) is performed. Once again outliers are removed. Finally, analysis of the main experiment hints at possible non-linear effects (i.e., an asymptotic relationship between time-to-complete and i-delay); to test for this, non-linear terms can be introduced in the regression.

Chapter 5

Results

5.1 Main experiment

This section presents the results of the analyses performed on the data from the main experiment, starting with a global view, then focusing on individual subjects. To recapitulate, the main experiment was conducted on nine subjects, and involved three different delay values affecting four separate views, yielding eighty-one trials (in a factorial design).

5.1.1 Global Results

Tables 5.1 and 5.2 present the global view: the response variables (time-to-complete and number-of-tries, respectively) analysed against all controlled variables. This includes all stimuli (the four delays) as well as the subject identifier. Of immediate note in Table 5.1 are the very high F-ratio values for the i (interactive shadow delay) and *sbt* (subject) variables: 38.2 and 25.7, respectively. Also, in both cases, *Prob*, which denotes the

Analysis of Variance For **time**

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	2350744843	1175372422	38.145	0.0000
p	2	139615639	69807820	2.2655	0.1045
1	2	112995382	56497691	1.8336	0.1606
2	2	30483254	15241627	0.49465	0.6100
sbt	8	6334738668	791842333	25.698	0.0000
Error	712	21938969176	30813159		
Total	728	30907546963			

Table 5.1: ANOVA of time-to-complete against all controlled variables and all subjects

probability that this apparent causal relationship has arisen out of random variance, is zero (to four significant digits). Thus two preliminary conclusions can already be drawn: subjects are sensitive to delays in the view that they are manipulating (the interactive shadow in the case of this experiment), and different subjects have markedly different performance responses to the task. Table 5.2 reveals that the interactive shadow delay and the subject also both affect performance as measured by the number-of-tries (i.e., the number of manipulation operations required). Although the F-ratio numbers are lower than in the previous ANOVA—$F = 18.3$ and $F = 15.6$—they are nevertheless conclusive evidence of an effect. The *Prob* values of zero further strengthen this evidence.

In both analyses above, the *1* and *2* variables (representing the two shadows which the subject is not manipulating at the time) yield very low F-ratios, ranging from 0.4 to 1.8, which indicate that there is no significant variation.

Analysis of Variance For **numtries**

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	147.896	73.9479	18.274	0.0000
p	2	5.02332	2.51166	0.62070	0.5379
1	2	13.6159	6.80796	1.6824	0.1867
2	2	3.33608	1.66804	0.41222	0.6623
sbt	8	504.571	63.0713	15.587	0
Error	712	2881.12	4.04651		
Total	728	3555.56			

Table 5.2: ANOVA of number-of-tries against all controlled variables and all subjects

The last variable in these tables, the p-delay, does not yield conclusive results. In terms of number of attempts to complete the task, delays in the perspective view apparently do not affect the subject's performance, as indicated by a very low F-ratio ($F = 0.6$). Analysis of the effect of p on the time-to-complete, however, is more uncertain: the fairly low F-ratio ($F = 2.3$) significant only to $p = 0.105$ is close to significance (and is markedly different from the results of shadows 1 and 2), and should therefore be analysed more carefully. Further study below reveals that the global results are weakened by strong differences between the subjects.

Figure 5.1 shows the distribution of the number of manipulation operations per trial, subject by subject. This plot illustrates the inter-subject variability shown numerically by the ANOVA of Table 5.2. The plot also shows that all subjects except *rjk* and *sll* have outlying data points, perhaps indicative of the fine-tuning problems described in the previous chapters. The boxplot of Figure 5.2 identifies these outliers algorithmically, by

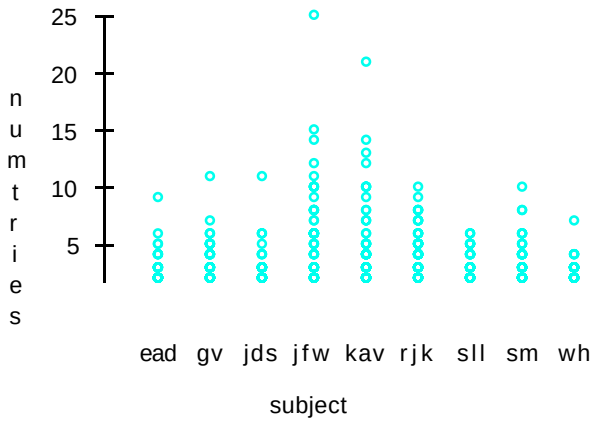

Figure 5.1: Plot of number-of-tries against subjects

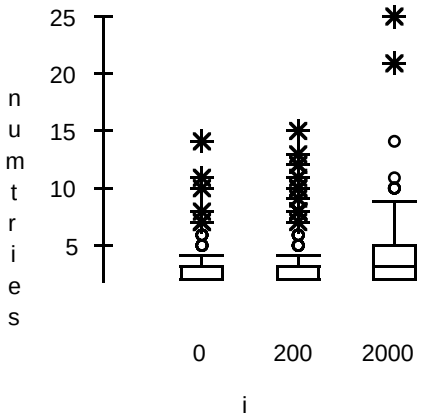

Figure 5.2: Boxplot of number-of-tries against the i-delay

Analysis of Variance For **time**
cases selected according To glob no outliers
729 total cases of which 23 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	2296040448	1148020224	48.025	0
p	2	75297642	37648821	1.5749	0.2078
Error	701	16757309843	23904864		
Total	705	19127856350			

Table 5.3: ANOVA of time-to-complete against i-delay and p-delay, all subject, with outliers removed

showing number-of-tries against the i-delay; here, DataDesk represents extreme outliers with the star burst symbol.[1] Once identified, these points can be excluded from the body of data for the next set of analyses.

Tables 5.3 and 5.4, which analyse the two response variables with respect to i-delay and p-delay only, and with extreme outliers removed, strengthen the previous conclusion about the effect of delaying the interactive shadow: extremely high F-ratios ($F = 48.0$ and $F = 39$, respectively) with zero probability that the effect is due to random sampling. On the other hand, this revised analysis further weakens the case that delays in the perspective view play a role in subject performance.

[1] In a boxplot, low and high *hinge* values are computed, corresponding approximately to the 25^{th} and 75^{th} percentiles. The *H-spread* is the difference in values between the two hinges. An outlier is any point greater or smaller than the high or low hinge, respectively, by more than 1.5 times the H-spread. DataDesk further refines this by defining *extreme* outliers as any point beyond a hinge by more than 3 times the H-spread. See Howell (1992) and Appendix 8A of the DataDesk Manual (Velleman 1989) for more details.

Analysis of Variance For **numtries**
cases selected according To glob no outliers
729 total cases of which 23 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	171.485	85.7423	39.120	0
p	2	1.05098	0.525492	0.23975	0.7869
Error	701	1536.45	2.19179		
Total	705	1709.00			

Table 5.4: ANOVA of number-of-tries against i-delay and p-delay, all subjects, with outliers removed

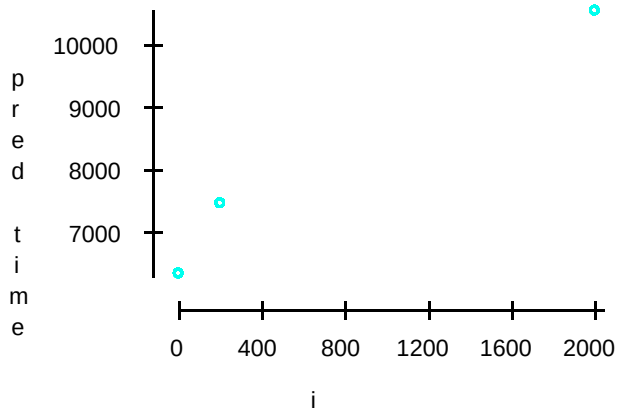

Figure 5.3: Plot of predicted time-to-complete against i-delay, all subjects with outliers removed

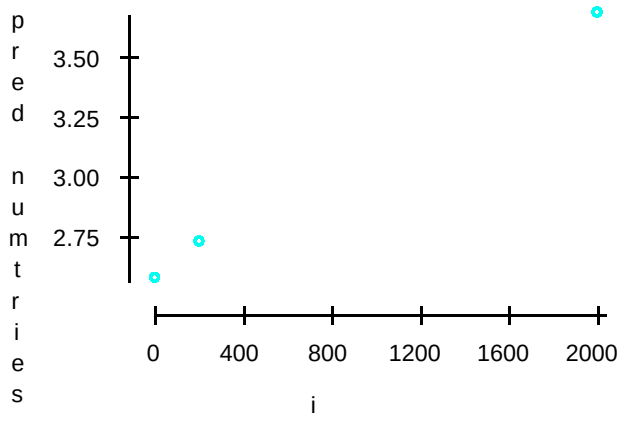

Figure 5.4: Plot of predicted number-of-tries against i-delay, all subjects with outliers removed

Figure 5.3 plots the expected time-to-complete, based on the ANOVA of Table 5.3, against the i-delay. If a straight line is drawn connecting the two end-points, its slope can be used to obtain the sign and approximate magnitude of the effect of i-delays on performance. This slope is positive, confirming that an increase in i-delay causes an increase in time-to-complete; a rough estimate (computed directly from the graph) yields a magnitude of 2: an increase of T in the i-delay causes a corresponding $2T$ increase in the time to complete. However this is imprecise and in particular, the line does not pass through (or near) the third point, suggesting either that this is an inaccurate estimate of the linear relationship, or even that the relationship is not linear at all (this latter possibility is tested in the analysis of the secondary experiment).

Analysis of Variance For **time**
cases selected according To ead
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	48860548	24430274	6.3703	0.0028
p	2	982026	491013	0.12803	0.8800
Error	76	291464118	3835054		
Total	80	341306693			

Table 5.5: Subject *ead*: ANOVA of time-to-complete against i-delay and p-delay

The same estimate can be done on Figure 5.4, which plots the expected number-of-tries against the i-delay. The slope here is in the order of $6 * 10^{-4}$ (in this case, the units are "number of operations required per millisecond of delay").

5.1.2 Individual Results

Of the nine subjects, three showed results remarkably similar to the overall analysis: *ead*, *sm* and *wh*. The initial analyses for these subjects—time-to-complete and number-of-tries against i-delay and p-delay, with all trials included—are shown in Tables 5.5 through 5.10.

In all three cases, the i-delay is a significant factor in performance, in terms of both time-to-complete and number-of-tries, as indicated by high F-ratios and low probabilities that the effect is random. The effect is felt the strongest on the time-to-complete measure, with ratios ranging from $F = 6.4$ to $F = 10.2$, significant to $p = 0.005$ in all cases; using number-of-tries as a measure yields weaker, but nevertheless conclusive, results, in the range $F = 2.6$ to $F = 6.4$, significant to $p = 0.1$ ($p = 0.005$ for subject *sm*). The F-ratios

Analysis of Variance For **numtries**
cases selected according To ead
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	6.39506	3.19753	2.8241	0.0656
p	2	1.28395	0.641975	0.56700	0.5696
Error	76	86.0494	1.13223		
Total	80	93.7284			

Table 5.6: Subject *ead*: ANOVA of number-of-tries against i-delay and p-delay

Analysis of Variance For **time**
cases selected according To sm
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	252550483	126275242	8.7066	0.0004
p	2	10253278	5126639	0.35348	0.7034
Error	76	1102252459	14503322		
Total	80	1365056220			

Table 5.7: Subject *sm*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To sm
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	28.9136	14.4568	6.3943	0.0027
p	2	1.06173	0.530864	0.23480	0.7913
Error	76	171.827	2.26088		
Total	80	201.802			

Table 5.8: Subject *sm*: ANOVA of number-of-tries against i-delay and p-delay

Analysis of Variance For **time**
cases selected according To wh
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	96569144	48284572	10.179	0.0001
p	2	2560681	1280340	0.26991	0.7642
Error	76	360508720	4743536		
Total	80	459638544			

Table 5.9: Subject *wh*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To wh
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	3.06173	1.53086	2.5890	0.0817
p	2	0.172840	0.086420	0.14615	0.8643
Error	76	44.9383	0.591293		
Total	80	48.1728			

Table 5.10: Subject *wh*: ANOVA of number-of-tries against i-delay and p-delay

are understandably much lower than in the global analysis since there are only a fraction (1/9) of the number of data points. As in the analysis of the full body of data, there is no significant dependence on p-delay on either measure of performance.

Figures 5.5 through 5.7 show boxplots of these subjects' number of tries against i-delay. These graphs reveal outlying data points—5 to 10 per subject—suggesting possible fine-tuning problems. The extreme outliers thus identified (the points marked as star bursts on the plots) are excluded from the body of data, and the ANOVA tests are repeated, as shown in Tables 5.11 through 5.16.

The i-delay is confirmed as a significant factor, with F-ratios for the time-to-complete measure ranging from $F = 8.0$ to $F = 14.8$ (a marked increase over the previous results which included outliers) and a probability of error never above $p = 0.001$. The effect of removing outliers is even more visible in the analyses for number-of-tries, as the F-ratios are now between $F = 5.8$ and $F = 11.4$, $p < 0.005$. On the other hand, removal of outliers

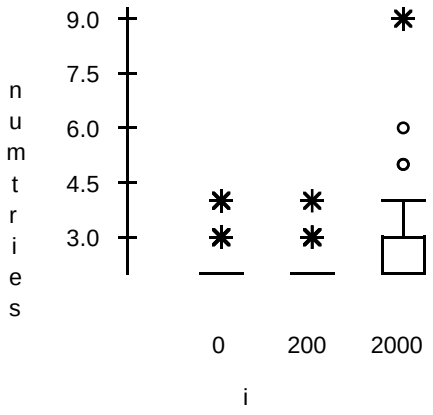

Figure 5.5: Subject *ead*: Boxplot of number-of-tries against i-delay

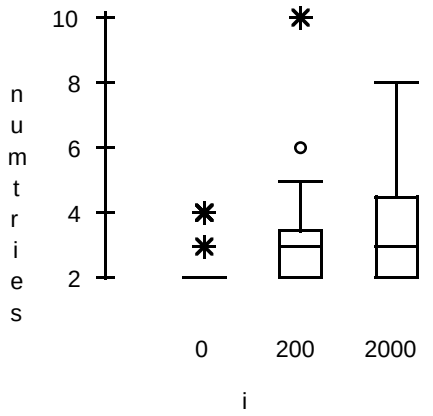

Figure 5.6: Subject *sm*: Boxplot of number-of-tries against i-delay

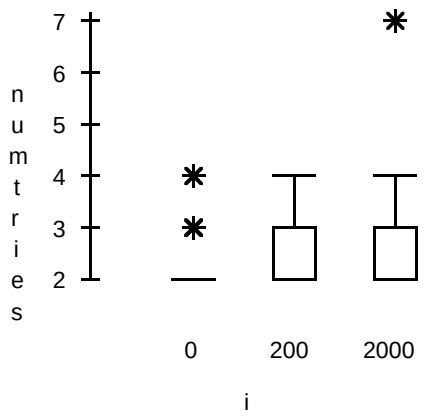

Figure 5.7: Subject *wh*: Boxplot of number-of-tries against *i*-delay

Analysis of Variance For **time**
cases selected according To ead no outliers
729 total cases of which 658 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	35998709	17999355	8.0255	0.0008
p	2	799022	399511	0.17813	0.8372
Error	66	148022177	2242760		
Total	70	185571270			

Table 5.11: Subject *ead*: ANOVA of time-to-complete against *i*-delay and *p*-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To ead no outliers
729 total cases of which 658 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	5.59181	2.79591	5.7838	0.0048
p	2	0.441591	0.220796	0.45675	0.6353
Error	66	31.9046	0.483402		
Total	70	37.8310			

Table 5.12: Subject *ead*: ANOVA of number-of-tries against i-delay and p-delay, with outliers removed

Analysis of Variance For **time**
cases selected according To sm no outliers
729 total cases of which 653 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	279479846	139739923	10.475	0.0001
p	2	9505223	4752611	0.35626	0.7015
Error	71	947148489	13340120		
Total	75	1234199247			

Table 5.13: Subject *sm*: ANOVA of time-to-complete against i-delay and p-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To sm no outliers
729 total cases of which 653 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	36.0027	18.0014	11.362	0.0001
p	2	0.984675	0.492337	0.31074	0.7339
Error	71	112.491	1.58438		
Total	75	149.526			

Table 5.14: Subject *sm*: ANOVA of number-of-tries against *i*-delay and *p*-delay, with outliers removed

Analysis of Variance For **time**
cases selected according To wh no outliers
729 total cases of which 655 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	66145786	33072893	14.813	0.0000
p	2	1774726	887363	0.39745	0.6736
Error	69	154050713	2232619		
Total	73	222592282			

Table 5.15: Subject *wh*: ANOVA of time-to-complete against *i*-delay and *p*-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To wh no outliers
729 total cases of which 655 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	3.88309	1.94155	8.0596	0.0007
p	2	0.020543	0.010271	0.04264	0.9583
Error	69	16.6219	0.240897		
Total	73	20.5541			

Table 5.16: Subject *wh*: ANOVA of number-of-tries against i-delay and p-delay, with outliers removed

does not reveal any hidden effect due to the p-delay, with results weaker, if anything, than before.

The plots in Figures 5.8 through 5.13 show the same pattern of points as the general cases (Figures 5.3 and 5.4), suggesting that there may or may not be a linear relationship. In this case, however, if a linear relationship is assumed, the slope for the predicted time-to-complete is much lower than in the general case: 0.8 for *ead*, 1.5 for *sm* and 1.2 for *wh*. The slopes for predicted number-of-tries are in the 10^{-4} to 10^{-3} range.

Subject *rjk*'s results, shown in Tables 5.17 and 5.18, are similar to those presented above. With $F = 6.6$ $(p < 0.005)$, the i-delay certainly affects time-to-complete; similarly, it affects the number-of-tries measure, with $F = 7.3$ $(p < 0.005)$. And as in the previous cases, the p-delay does not play a role in performance. Figure 5.14 shows that this subject has no extreme outlying data points. Thus, the outlier-removed analysis is not done. Figures 5.15 and 5.16 show predicted values for time-to-complete and number-of-

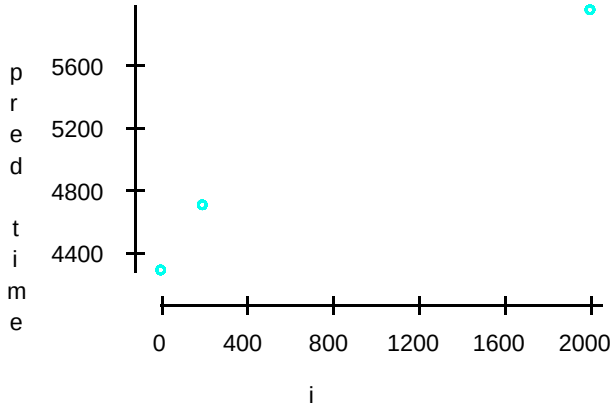

Figure 5.8: Subject *ead*: Plot of predicted time-to-complete against i-delay, with outliers removed

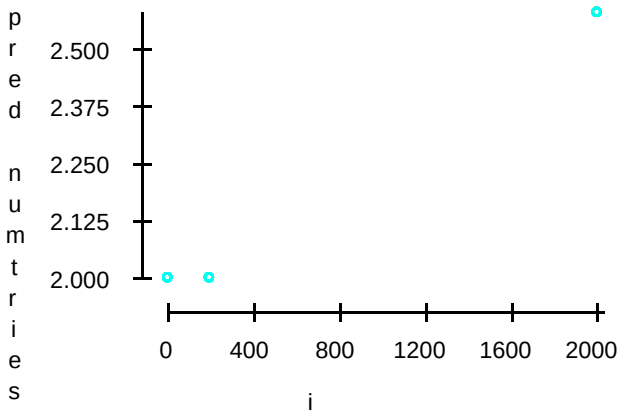

Figure 5.9: Subject *ead*: Plot of predicted number-of-tries against i-delay, with outliers removed

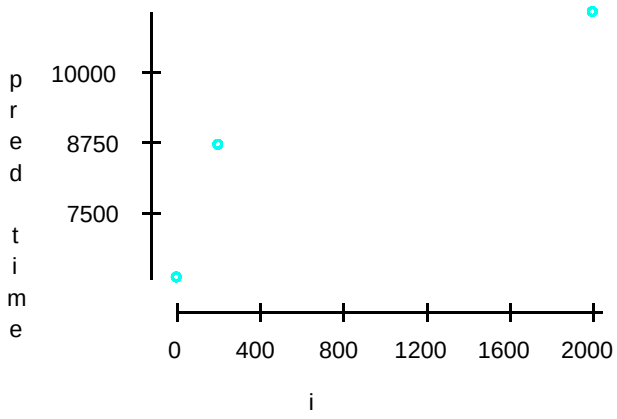

Figure 5.10: Subject *sm*: Plot of predicted time-to-complete against i-delay, with outliers removed

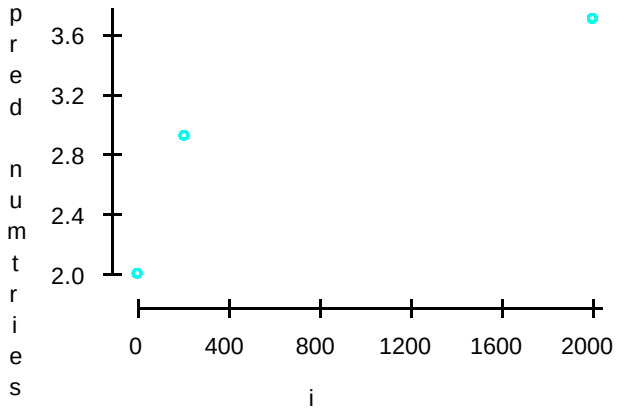

Figure 5.11: Subject *sm*: Plot of predicted number-of-tries against i-delay, with outliers removed

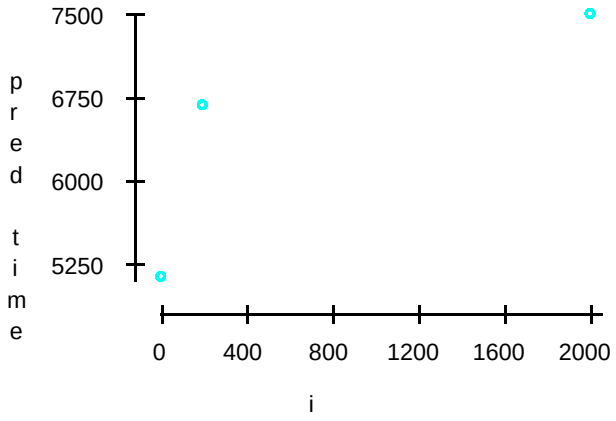

Figure 5.12: Subject *wh*: Plot of predicted time-to-complete against i-delay, with outliers removed

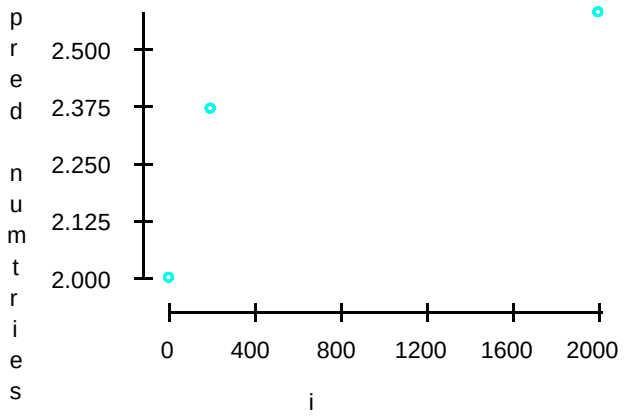

Figure 5.13: Subject *wh*: Plot of predicted number-of-tries against i-delay, with outliers removed

Analysis of Variance For **time**
cases selected according To r j k
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	485747244	242873622	6.5815	0.0023
p	2	18498023	9249011	0.25063	0.7789
Error	76	2804593755	36902549		
Total	80	3308839022			

Table 5.17: Subject *rjk*: ANOVA of time-to-complete against *i*-delay and *p*-delay

Analysis of Variance For **numtries**
cases selected according To r j k
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	45.7284	22.8642	7.2838	0.0013
p	2	1.65432	0.827160	0.26351	0.7691
Error	76	238.568	3.13905		
Total	80	285.951			

Table 5.18: Subject *rjk*: ANOVA of number-of-tries against *i*-delay and *p*-delay

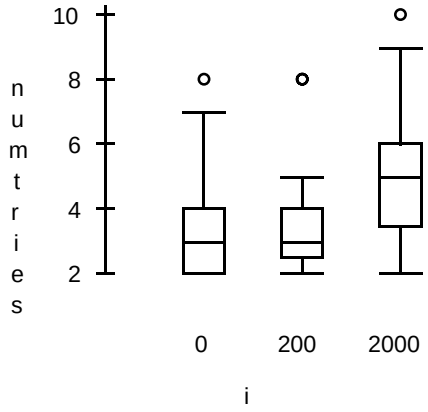

Figure 5.14: Subject *rjk*: Boxplot of number-of-tries against i-delay

tries. The time-to-complete slope is approximately 3, which is higher than the results obtained thus far.

Subject *kav*'s results in terms of time-to-complete, shown in Table 5.19, are also similar: $F = 3.9$, $p < 0.05$. Furthermore, in terms of number-of-tries (Table 5.20), the results are inconclusive, with $F = 1.5$ and a very low statistical significance ($p > 0.2$). However, the boxplot in Figure 5.17 reveals outliers, so the analysis may be refined by the removal of these points.

Tables 5.21 and 5.22 show that the outlier-removed data does indeed yield stronger results, in particular for the number-of-tries. With $F = 3.8$, $p < 0.05$, it is possible to say that the i-delay has an effect on this measure of performance. Note that this subject follows the general trend in that the p-delay plays no role at all, as demonstrated by the statistically insignificant results ($p > 0.5$), even when outliers are ignored.

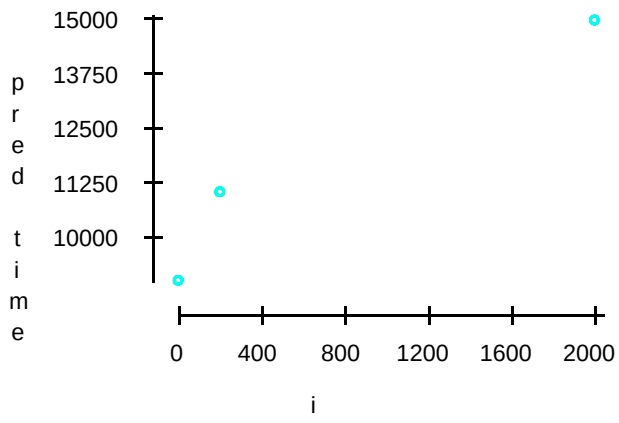

Figure 5.15: Subject *rjk*: Plot of predicted time-to-complete against i-delay

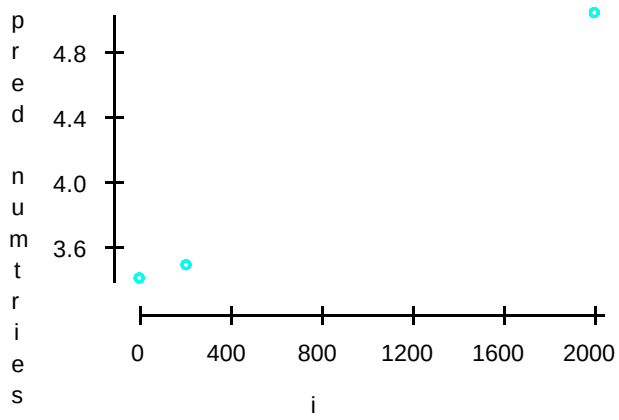

Figure 5.16: Subject *rjk*: Plot of predicted number-of-tries against i-delay

Analysis of Variance For **time**
cases selected according To kav
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	326816365	163408183	3.8606	0.0253
p	2	32539063	16269532	0.38438	0.6822
Error	76	3216853928	42327025		
Total	80	3576209357			

Table 5.19: Subject *kav*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To kav
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	33.8765	16.9383	1.5404	0.2209
p	2	1.95062	0.975309	0.08870	0.9152
Error	76	835.679	10.9958		
Total	80	871.506			

Table 5.20: Subject *kav*: ANOVA of number-of-tries against i-delay and p-delay

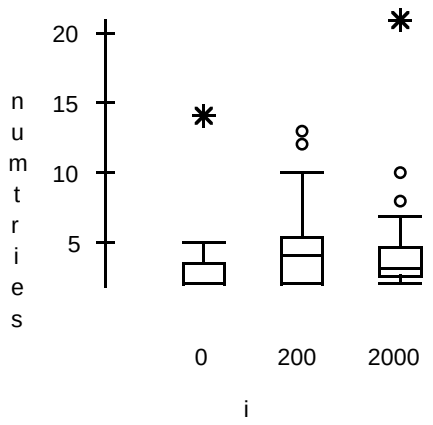

Figure 5.17: Subject *kav*: Boxplot of number-of-tries against i-delay

Analysis of Variance For **time**
cases selected according To kav no outliers
729 total cases of which 650 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	197421562	98710781	4.7221	0.0117
p	2	6749838	3374919	0.16145	0.8512
Error	74	1546905203	20904124		
Total	78	1753811663			

Table 5.21: Subject *kav*: ANOVA of time-to-complete against i-delay and p-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To kav no outliers
729 total cases of which 650 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	44.2930	22.1465	3.7773	0.0274
p	2	2.31436	1.15718	0.19737	0.8213
Error	74	433.867	5.86306		
Total	78	480.759			

Table 5.22: Subject *kav*: ANOVA of number-of-tries against i-delay and p-delay, with outliers removed

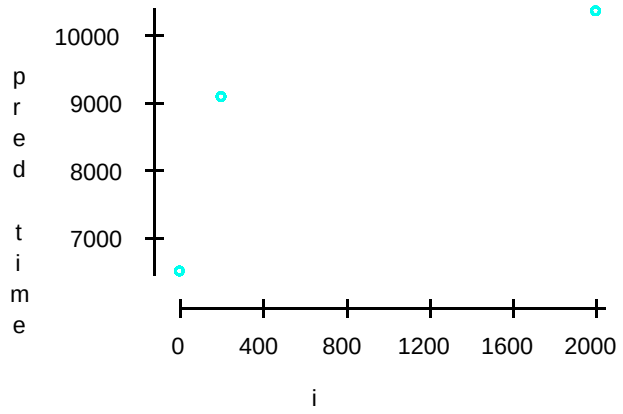

Figure 5.18: Subject *kav*: Plot of predicted time-to-complete against i-delay, with outliers removed

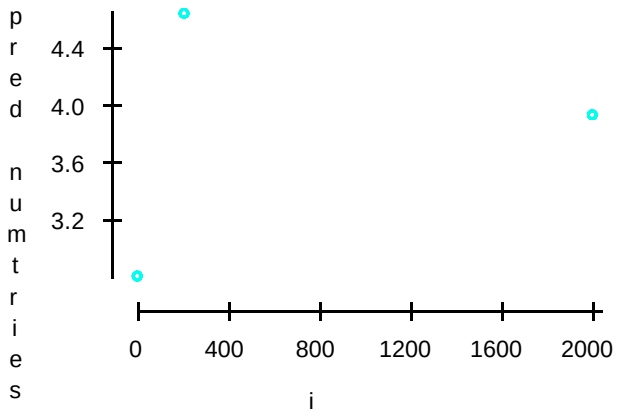

Figure 5.19: Subject *kav*: Plot of predicted number-of-tries against i-delay, with outliers removed

The plot of the predicted time-to-complete (Figure 5.18) shows the usual pattern, with an approximate slope of 1.9 (close to the general case). The plot of number-of-tries (Figure 5.19), on the other hand, is atypical, with a clearly non-linear arrangement of points. While this deviation from the normal relationship may be indicative of an effect particular to the subject—for instance, one might theorise that for short i-delays, this user prefers to start new manipulation operations rather than wait for the manipulated view to catch up, but that for very long delays, it is simpler to wait, thus requiring fewer operations—but it is more likely a sampling effect. Indeed, note that the number-of-tries measure may only change in discrete, integer steps, and that the last two points on this graph (representing averages) are less than one such step apart.

Analysis of Variance For **time**
cases selected according To jds
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	23584950	11792475	2.5060	0.0883
p	2	6827542	3413771	0.72545	0.4874
Error	76	357633749	4705707		
Total	80	388046242			

Table 5.23: Subject *jds*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To jds
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	4.83951	2.41975	1.4968	0.2304
p	2	1.65432	0.827160	0.51166	0.6016
Error	76	122.864	1.61663		
Total	80	129.358			

Table 5.24: Subject *jds*: ANOVA of number-of-tries against i-delay and p-delay

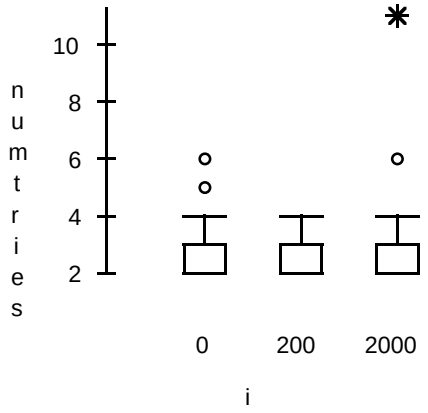

Figure 5.20: Subject *jds*: Boxplot of number-of-tries against i-delay

Analysis of subject *jds* (Tables 5.23 and 5.24) yields a weaker result than subject *kav*: the F-ratio ($F = 2.5$) is low, significant only to $p < 0.1$, and may leave some doubt as to the effect of the i-delay on the time-to-complete. Furthermore, the i-delay does not affect the number-of-tries. And in this case, removing the extreme outliers (shown graphically in Figure 5.20) actually disimproves the analysis, even though only one point is removed (see Tables 5.25 and 5.26). In fact, it degrades it so results for both measures of performance are inconclusive: the F-ratios are $F = 1.6$ or less, with $p > 0.2$ probability that this is simply random. This subject particularly demonstrates the importance of removing outliers because the entire marginally significant effect in Table 5.23 proves to be caused by a single point. Nevertheless, the effect of the i-delay on the time-to-complete can be plotted (see Figure 5.21), which reveals the same pattern found in other subjects (with a slope of approximately 0.45). By themselves, the results

Analysis of Variance For **time**
cases selected according To jds no outliers
729 total cases of which 649 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	10605791	5302895	1.5648	0.2159
p	2	3461583	1730792	0.51072	0.6021
Error	75	254169341	3388925		
Total	79	268213105			

Table 5.25: Subject *jds*: ANOVA of time-to-complete against i-delay and p-delay, with outliers removed

from this subject are inconclusive. But because they follow the same trends as the other results, they strengthen the group result.

The first-pass analysis of subject *gv*'s performance, show in Tables 5.27 and 5.28, reveals results weaker still than subject *jds*. In fact, in this case, the only potential effect is of the p-delay on the number-of-tries measure ($F = 2.6$, $p < 0.1$), a result that is inconsistent with the subjects examined to this point. All other variations of performance among the different trials are more likely due to random sampling effects rather than experiment configuration. However, the boxplot of number-of-tries against the i-delay (Figure 5.22) reveals two extreme outliers, which might be skewing results. Indeed, after removal of these two points, the ANOVA analyses (Tables 5.29 and 5.30) yield significant results, in accord with general trends: the i-delay affects both the time-to-complete ($F = 3.4$, $p < 0.05$) and the number-of-tries ($F = 5.3$, $p < 0.01$), and the p-delay affects neither (so the apparent effect of p-delay on number-of-tries is gone). The

Analysis of Variance For **numtries**
cases selected according To jds no outliers
729 total cases of which 649 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	1.16243	0.581215	0.78257	0.4609
p	2	1.11827	0.559136	0.75285	0.4746
Error	75	55.7022	0.742697		
Total	79	58			

Table 5.26: Subject *jds*: ANOVA of number-of-tries against i-delay and p-delay, with outliers removed

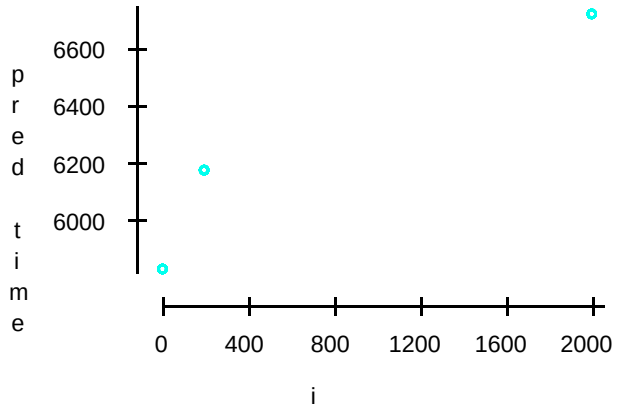

Figure 5.21: Subject *jds*: Plot of predicted time-to-complete against i-delay, with outliers removed

Analysis of Variance For **time**
cases selected according To gv
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	30648783	15324391	1.1264	0.3296
p	2	45679309	22839655	1.6788	0.1934
Error	76	1033982106	13605028		
Total	80	1110310198			

Table 5.27: Subject gv: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To gv
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	8.96296	4.48148	2.0472	0.1362
p	2	11.5556	5.77778	2.6394	0.0779
Error	76	166.370	2.18908		
Total	80	186.889			

Table 5.28: Subject gv: ANOVA of number-of-tries against i-delay and p-delay

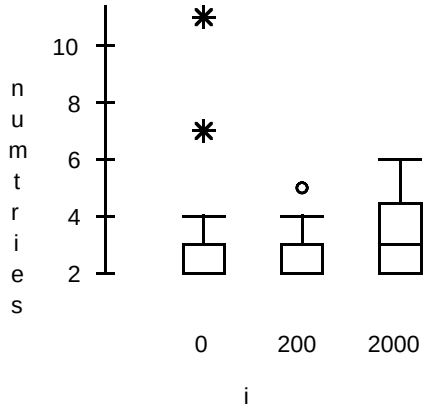

Figure 5.22: Subject *gv*: Boxplot of number-of-tries against i-delay

plots of the i-delay effects (Figures 5.23 and 5.24) are typical, with an approximate slope of 1.2 for the time-to-complete.

The analyses for subject *sll* (Tables 5.31 and 5.32) are similar in many ways to those of subject *gv*: the very weak results reveal no statistically significant variances, except perhaps for an effect of the p-delay on the time-to-complete ($F = 2.6$, $p < 0.1$). The two subjects differ, however, in the boxplots used to identify outliers. In the case of *sll* (Figure 5.25), there are no extreme outliers, only "near" outliers. There can be no refinement phase, and only the first-pass analysis may be used.

Although the effects are not statistically significant, the relationships between the i-delay and the two measures of performance are graphed (in Figures 5.26 and 5.27), as they may support the general result. Interestingly, despite the notable differences in the

Analysis of Variance For **time**
cases selected according To gv no outliers
729 total cases of which 650 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	65782681	32891340	3.4335	0.0375
p	2	8255914	4127957	0.43092	0.6515
Error	74	708875886	9579404		
Total	78	784724387			

Table 5.29: Subject *gv*: ANOVA of time-to-complete against *i*-delay and *p*-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To gv no outliers
729 total cases of which 650 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	12.6689	6.33446	5.2737	0.0072
p	2	2.31929	1.15965	0.96546	0.3856
Error	74	88.8837	1.20113		
Total	78	104.152			

Table 5.30: Subject *gv*: ANOVA of number-of-tries against *i*-delay and *p*-delay, with outliers removed

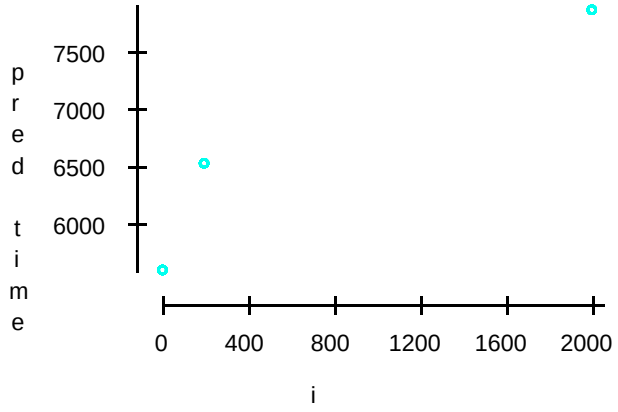

Figure 5.23: Subject *gv*: Plot of predicted time-to-complete against i-delay, with outliers removed

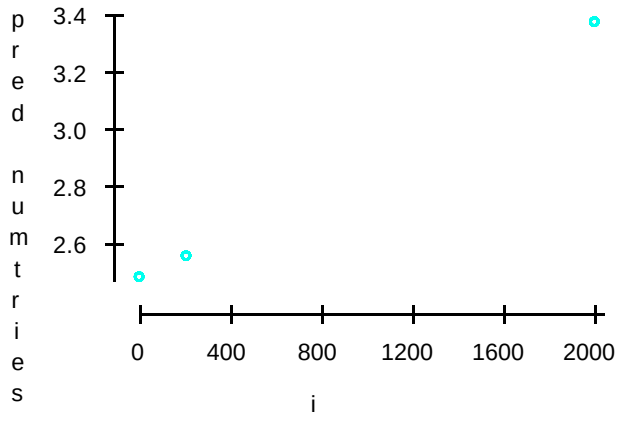

Figure 5.24: Subject *gv*: Plot of predicted number-of-tries against i-delay, with outliers removed

Analysis of Variance For **time**
cases selected according To sll
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	26977019	13488509	1.5537	0.2181
p	2	44677663	22338832	2.5731	0.0829
Error	76	659810047	8681711		
Total	80	731464729			

Table 5.31: Subject *sll*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To sll
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	2.39506	1.19753	1.1082	0.3354
p	2	1.50617	0.753086	0.69693	0.5013
Error	76	82.1235	1.08057		
Total	80	86.0247			

Table 5.32: Subject *sll*: ANOVA of number-of-tries against i-delay and p-delay

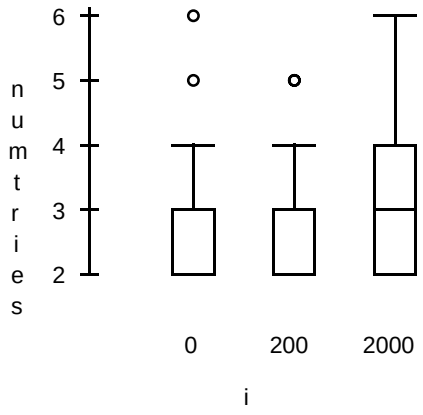

Figure 5.25: Subject *sll*: Boxplot of number-of-tries against i-delay

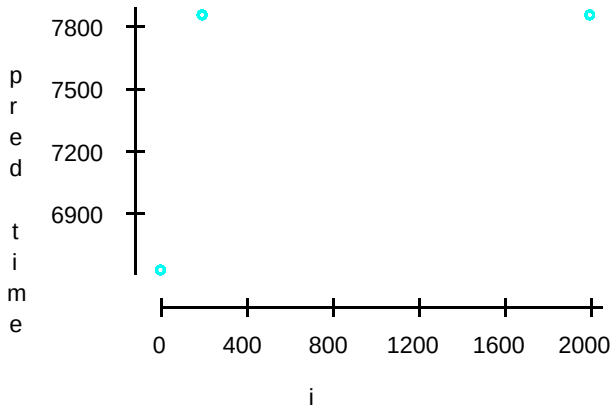

Figure 5.26: Subject *sll*: Plot of predicted time-to-complete against i-delay

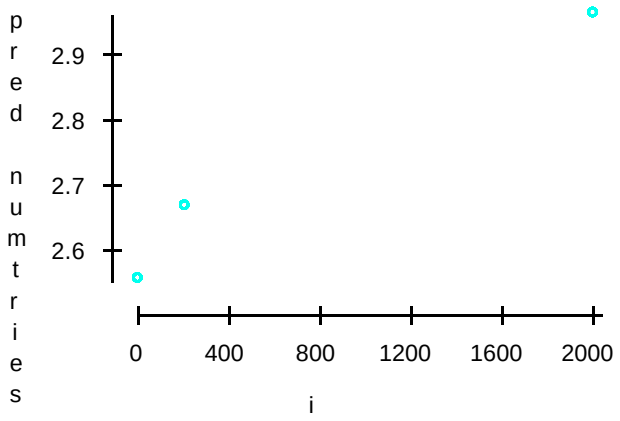

Figure 5.27: Subject *sll*: Plot of predicted number-of-tries against i-delay

ANOVAs compared to previous subjects, the graphs reflect the general trend identified to this point, with a positive slope (0.6 for time-to-complete).

As the results of Tables 5.33 and 5.34 show, the final subject, *jfw*, confirms some of the conclusions drawn so far and contradicts others. Indeed, the effects due to the i-delay found in the other subjects are strongly present here, both on the time-to-complete ($F = 16.9$, with zero probability of error) and on the number-of-tries ($F = 6.2, p < 0.005$). However, while the p-delay has no effect for other subjects, for subject *jfw* this delay has an undeniable effect on the time-to-complete ($F = 8.7$, $p < 0.0005$) and a no less remarkable effect on the number-of-tries ($F = 4.6$, $p < 0.05$). Furthermore, because the boxplot of Figure 5.28 reveals an extreme outlier, the analysis should be refined.

Analysis of Variance For **time**
cases selected according To jfw
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	3528417533	1764208767	16.872	0.0000
p	2	1816472851	908236425	8.6857	0.0004
Error	76	7947046907	104566407		
Total	80	13291937291			

Table 5.33: Subject *jfw*: ANOVA of time-to-complete against i-delay and p-delay

Analysis of Variance For **numtries**
cases selected according To jfw
729 total cases of which 648 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	145.407	72.7037	6.1786	0.0033
p	2	107.852	53.9259	4.5828	0.0132
Error	76	894.296	11.7671		
Total	80	1147.56			

Table 5.34: Subject *jfw*: ANOVA of number-of-tries against i-delay and p-delay

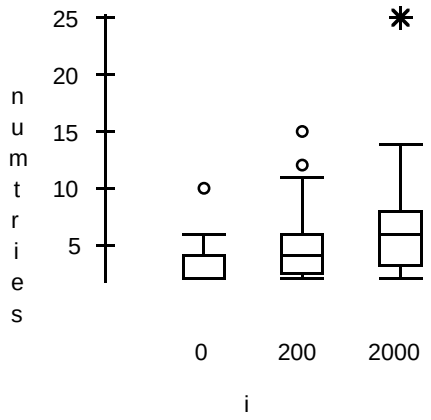

Figure 5.28: Subject *jfw*: Boxplot of number-of-tries against i-delay

Analysis of Variance For **time**
cases selected according To JFW no outliers
729 total cases of which 649 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	2504329792	1252164896	20.084	0.0000
p	2	1176663040	588331520	9.4363	0.0002
Error	75	4676067622	62347568		
Total	79	8290946023			

Table 5.35: Subject *jfw*: ANOVA of time-to-complete against i-delay and p-delay, with outliers removed

Analysis of Variance For **numtries**
cases selected according To JFW no outliers
729 total cases of which 649 are missing

Source	df	Sum of Squares	Mean Square	F-ratio	Prob
i	2	97.1306	48.5653	6.2413	0.0031
p	2	61.0038	30.5019	3.9199	0.0240
Error	75	583.593	7.78124		
Total	79	739.550			

Table 5.36: Subject *jfw*: ANOVA of number-of-tries against i-delay and p-delay, with outliers removed

When the ANOVAs are repeated on the outlier-removed data (Tables 5.35 and 5.36), the results are even stronger. Four separate graphs are shown, one for each of the i-delay and the p-delay on both the time-to-complete and number-of-tries.

Figures 5.29 and 5.30 show the usual relationship between the i-delay and the predicted time-to-complete and number-of-tries. Note however in Figure 5.29 that the slope for the predicted time-to-complete is approximately seven, which is far greater than any other subject. The slope for the predicted number-of-tries is also higher than for other subjects. Figures 5.31 and 5.32, which plot the predicted time-to-complete and number-of-tries, respectively, against the p-delay, are a surprise: the slopes are negative (approximately -4 for the time-to-complete and -10^{-3} for the number-of-tries). For this subject, an increase in the p-delay appears to improve both measures of performance.

Could this result also be present in other subjects? The only other subject to show a near significant effect of p-delay on the time-to-complete is *sll* (no other subject showed a

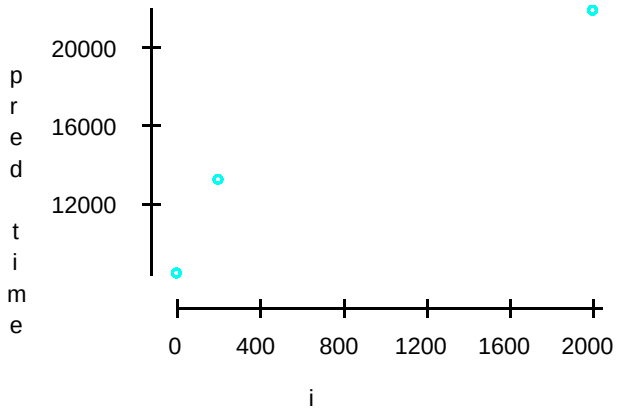

Figure 5.29: Subject *jfw*: Plot of predicted time-to-complete against i-delay, with outliers removed

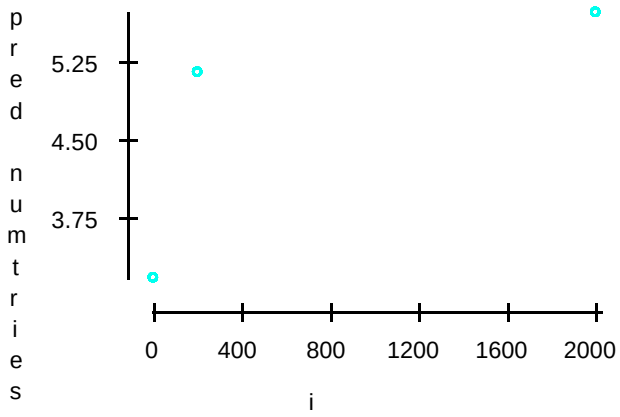

Figure 5.30: Subject *jfw*: Plot of predicted number-of-tries against i-delay, with outliers removed

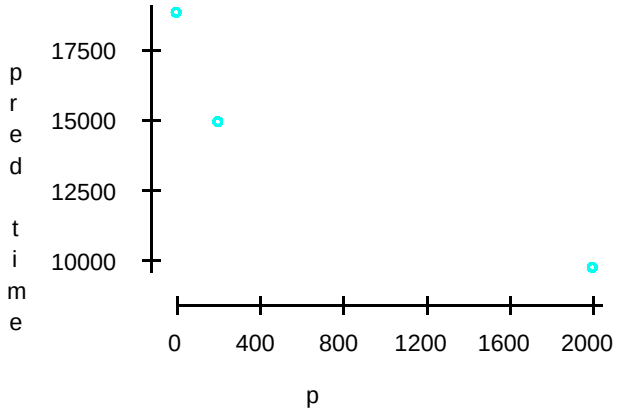

Figure 5.31: Subject *jfw*: Plot of predicted time-to-complete against p-delay, with outliers removed

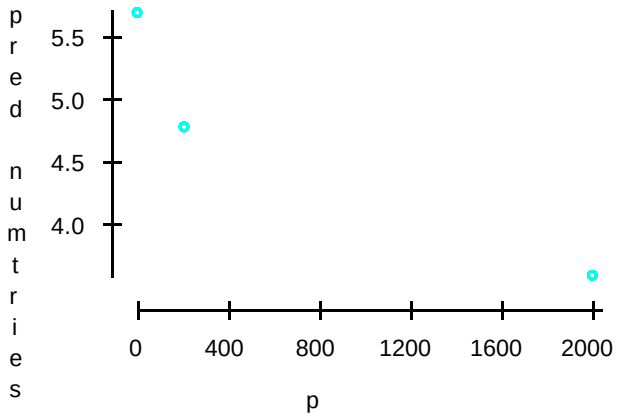

Figure 5.32: Subject *jfw*: Plot of predicted number-of-tries against p-delay, with outliers removed

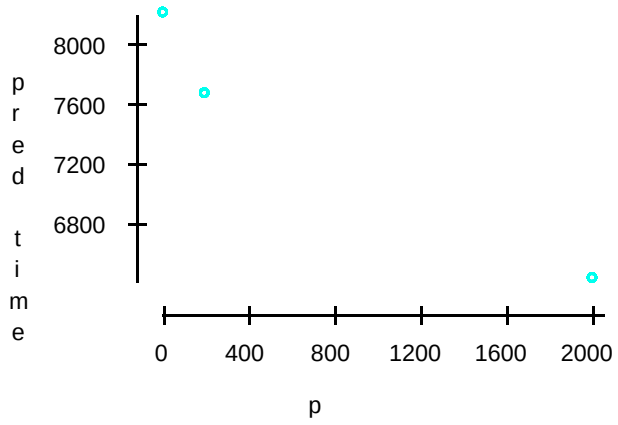

Figure 5.33: Subject *sll*: Plot of predicted time-to-complete against p-delay

significant effect of p-delay on the number-of-tries). Figure 5.33 plots the corresponding relationship; from this, it is clear that slope is also negative, although smaller in magnitude (approximately -0.9). Thus this subject's performance also improves as the p-delay increases.

5.2 Secondary experiment

The secondary experiment, which is designed to investigate the effect of delays on the interactive shadow in greater detail, was conducted on a single subject. It uses eleven different delay values (for the i-shadow only, no other view is delayed in this experiment), and twenty-five trials per value.

Dependent variable is: **time**
R = 5.3% R (adjusted) = 4.9%
s = 6440 with 275 - 2 = 273 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	628545162	1	6e+8	15.2
Residual	11323736680	273	41478889	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	5765.14	726.6	7.93
i	2.39041	0.6141	3.89

Table 5.37: Regression analysis of time-to-complete against i-delay

Dependent variable is: **numtries**
R = 2.3% R (adjusted) = 1.9%
s = 3.437 with 275 - 2 = 273 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	75.2818	1	75.28	6.37
Residual	3225.29	273	11.8142	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	3.40182	0.3878	8.77
i	0.000827	0.0003	2.52

Table 5.38: Regression analysis of number-of-tries against i-delay

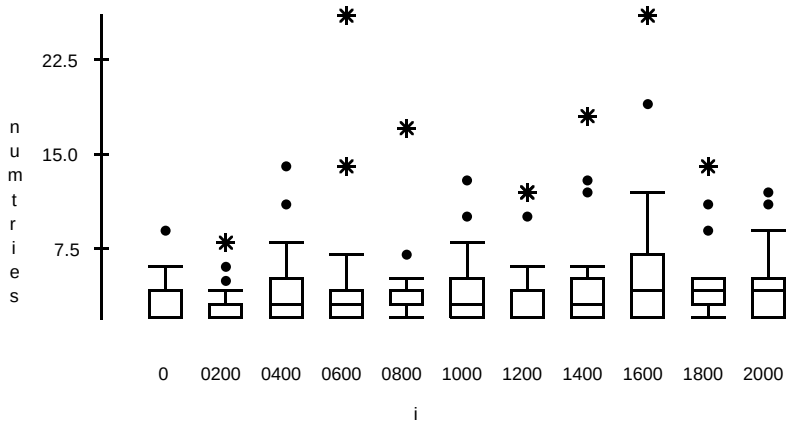

Figure 5.34: Boxplot of number-of-tries against i-delay

Tables 5.37 and 5.38 show the linear regression analysis of time-to-complete and number-of-tries, respectively, against the i-delay (which is, in this part of the experiment, the only controlled variable). The fairly high F-ratios ($F = 15.2$ for time-to-complete, $F = 6.37$ for number-of-tries) indicate that the data can indeed be fitted with a linear model. The slope of the relationship is shown as the "Coefficient" of the i-delay variable; for the time-to-complete, the slope is 2.4, which is in the same range as the numbers derived from the graphs in the previous analyses. Similarly, the slope for the number-of-tries—$8 * 10^{-4}$—is also in the same order of magnitude as was previously computed. The "Coefficient" of the Constant indicates the y-axis intercept, or the baseline performance if no i-delays are introduced. These numbers suggest that with no delays, the task requires, on average, 5.8 seconds and 3.4 manipulation operations.

Dependent variable is: **time**
cases selected according To no outliers
275 total cases of which 9 are missing
R = 8.0% R (adjusted) = 7.6%
s = 4529 with 266 - 2 = 264 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	467709353	1	5e+8	22.8
Residual	5414625763	264	20509946	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	5323.34	516.6	10.3
i	2.08334	0.4363	4.78

Table 5.39: Regression analysis of time-to-complete against i-delay, with outliers removed

Dependent variable is: **numtries**
cases selected according To no outliers
275 total cases of which 9 are missing
R = 3.6% R (adjusted) = 3.2%
s = 2.439 with 266 - 2 = 264 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	58.6926	1	58.69	9.87
Residual	1570.65	264	5.94942	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	3.08265	0.2782	11.1
i	0.000738	0.0002	3.14

Table 5.40: Regression analysis of number-of-tries against i-delay, with outliers removed

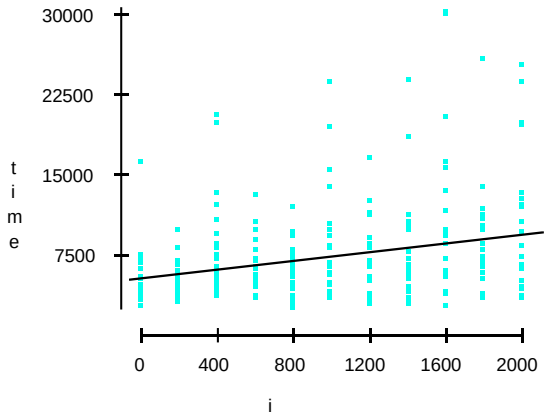

Figure 5.35: Plot of regression of time-to-complete against i-delay, with outliers removed

This analysis can be extended in the same manner as before, by identifying and removing potential outliers. The boxplot of Figure 5.34 shows that there are indeed a number of outlying points; these are removed from the body of data, and the regression analyses are repeated, as shown in Tables 5.39 and 5.40. It is immediately apparent that the removal of outliers has strengthened the results, as evidenced by the higher F-ratios ($F = 22.8$ and $F = 9.87$). The slope of time-to-complete against the i-delay is down slightly, to 2.1; in other words, a given increase in the i-delay will cause roughly twice that much increase in the time taken to complete the task, which agrees with the results of the previous section. Looking at the number-of-tries analysis, the slope is also down slightly, to $7 * 10^{-4}$, which is still well within the range of the other results. The y-axis intercepts now suggest that with no delays, the task requires on average 5.3 seconds and 3.1 manipulation operations. The two relationships are plotted in Figures 5.35 and 5.36.

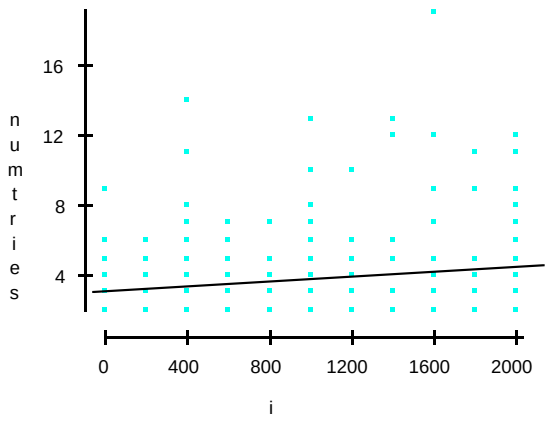

Figure 5.36: Plot of regression of predicted number-of-tries against i-delay, with outliers removed

All of the time-to-complete *vs.* i-delay relationships plotted in the previous section show a common pattern (where the middle data point lies above the line segment connecting the end points), suggesting possible non-linear effects. Furthermore, a plot of the residuals against the predicted time-to-complete values, based on the regression of Table 5.39 (see Figure 5.37), seems to reveal a pattern, once again suggesting there are non-linear effects that have not been accounted for by the regression. Finally, simple logic dictates that the time required to complete the task can not increase without bound as the delay increases; there must be limiting factors, that are not described by a linear relationship. However, finding an appropriate mathematical description of this non-linear relationship is difficult. A few preliminary attempts to model the effect (using squared terms and exponential terms) failed to fit the data (in all cases, the results were statisti-

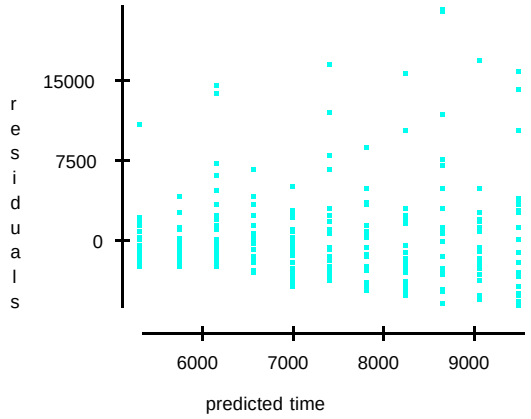

Figure 5.37: Plot of residuals against predicted time-to-complete, based on the regression of Table 5.39

cally insignificant). This question remains open, and further study is required (initially on the current set of data, although more subjects may be needed later to obtain statistical significance) to find the exact nature of the non-linear effects.

Finally, it is interesting to apply the linear regression analysis technique described above to the full body of data from the previous section, to verify that they agree with the results of subject *kav* from this section. Since it has already been determined that the full body of data includes extreme outliers (see Figure 5.2), this linear regression is performed directly on the outlier-removed data. From the regression of time-to-complete against the i-delay (Figure 5.41), the slope of the relationship is 2.0, which is exactly what was estimated from the graph of Figure 5.3. From Figure 5.42, the slope of the number-of-tries against the i-delay relationship is $5.4 * 10^{-4}$, which again is very close to

Dependent variable is: **time**
cases selected according To glob no outliers
729 total cases of which 23 are missing
R = 11.7% R (adjusted) = 11.6%
s = 4899 with 706 - 2 = 704 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	2233479682	1	2e+9	93.1
Residual	16894376668	704	23997694	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	6660.37	239.0	27.9
i	1.96432	0.2036	9.65

Table 5.41: Regression analysis of time-to-complete against i-delay, on full body of data (from main experiment) with outliers removed

the value estimated from Figure 5.4 (that estimate was rounded up to $6 * 10^{-4}$). Based on the constant coefficients in these regressions, the baseline performance for the task (with no i-delays) is, on average, 6.7 seconds and 2.6 manipulation operations. The data is graphed in Figures 5.38 and 5.39, along with the regression lines.

5.3 Subjective Results

After all trials were completed, each subject was asked to comment on perceived difficulty of the various configurations, as well as on strategy employed. The comments were verbal and unguided. They are paraphrased and summarised here.

Dependent variable is: **numtries**
cases selected according To glob no outliers
729 total cases of which 23 are missing
R = 10.0% R (adjusted) = 9.9%
s = 1.478 with 706 - 2 = 704 degrees of freedom

Source	Sum of Squares	df	Mean Square	F-ratio
Regression	171.260	1	171	78.4
Residual	1537.74	704	2.18429	

Variable	Coefficient	s.e. of Coeff	t-ratio
Constant	2.59493	0.0721	36.0
i	0.000544	0.0001	8.85

Table 5.42: Regression analysis of number-of-tries against i-delay, on full body of data (from main experiment) with outliers removed

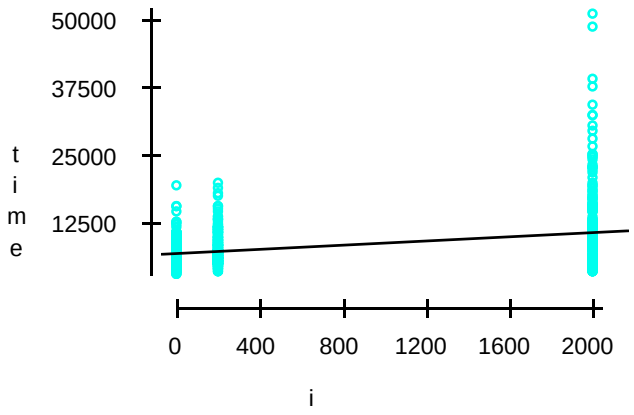

Figure 5.38: Plot of regression of time-to-complete against i-delay, on full body of data (from main experiment) with outliers removed

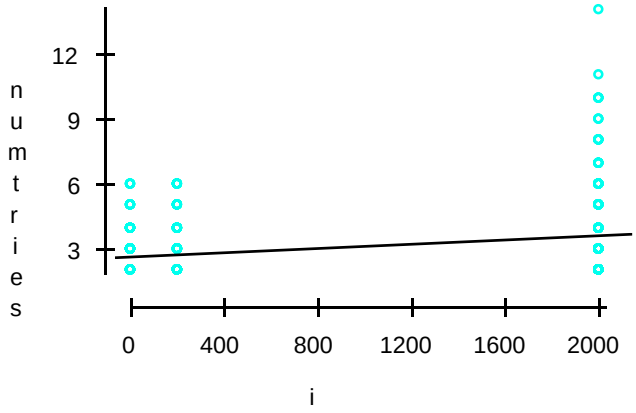

Figure 5.39: Plot of regression of number-of-tries against i-delay, on full body of data (from main experiment) with outliers removed

In terms of strategy, five of the nine subjects (*ead, gv, jds, kav* and *wh*) claimed to use the back wall (XY plane) first, followed by adjustments along the left wall (YZ plane), and never to have considered the floor shadow (XZ plane). Subject *ead* listed two reasons for ignoring this bottom view: the shadow itself is smaller (because of the vertical orientation of the molecule, this shadow presents the smallest cross-section), and it offers no opportunity to adjust the Z-component of the object (and vertical placement of the molecule is a critical factor in this task). Nevertheless, subject *wh* stated that he made use of the floor view for visual hints to aid in the placement process.

Three subjects normally used either the left wall (*sll, sm*) or the back wall (*jfw*) for vertical alignment only, following which they used the bottom shadow exclusively for XZ

placement, thus eliminating the risk of vertically de-aligning the molecules in the latter phases of the task. The last subject, *rjk*, claimed to use all three views in turn.

Subjects *ead, rjk* and *wh* agreed that any delay in the interactive shadow greatly increased the difficulty of the task (*rjk* claimed the task became "intolerable"); they also agreed that delays in the other views were usually ignorable. Subject *gv* stated that the only "annoying" configuration was when all views were delayed, and that any other configuration made for a simple task. Subject *jfw* felt that, in general, long delays were difficult but short delays were ignorable, although she admitted to being affected by any type of delay in two cases: when the interactive shadow was delayed, and when the perspective view was delayed. In the first case, the subject would manipulate the back-wall view but focus attention on the left-wall shadow (assuming it wasn't also delayed), although she found this confusing. Interestingly, in the second case—with the perspective view lagging—she observed that the task appeared to be easier, a comment echoed by both *rjk* and *sm*. All three subjects felt that when this view was delayed it stayed out of the way of the back-wall shadow, which it would otherwise occlude.

Subject *sll* was the only one to claim it easier to perform the task "blind" rather than to deal with delayed feedback: she would manipulate the first shadow or two without waiting for the views to catch up, achieving coarse placement; only for the final, precise, adjustment phase would she use the visual feedback of the various views—and found this phase to be significantly more difficult.

Finally, both subjects *rjk* and *sm* mentioned that delays in shadows 1 or 2 could actually affect them, if the view they were planning on using next was currently delayed.

In this case, they would either have to wait for the view to catch up before commencing the next manipulation operation, or change their strategy to employ another shadow.

Chapter 6

Discussion

6.1 Effect of Delays on Shadows 1 And 2

It was noted very early in the analysis of the data that shadows 1 and 2—i.e., the two shadows that are not being manipulated by the user—play no detectable role at all in performance as measured by this experiment. This result can be viewed from two different angles. On the one hand, it is important that both shadows exhibit the same behaviour. Indeed, as explained in Section 3.3 (on page 37), the labels 1 and 2 are assigned randomly to the two non-manipulated shadows; thus it would be difficult to explain should there be any difference in effect due to delays in the two views.

On the other hand, the fact that delays in the non-manipulated shadows have no effect at all on performance is an interesting result in itself. It implies that either these views are not useful in completing the task—and thus their delays are irrelevant—or that somehow their usefulness is unaffected by delays and de-synchronisation, which is the

issue this study hopes to address. Unfortunately, it is possible that the first conclusion is at least partly true, and that either or both of these views are often ignored. Nonetheless, from the verbal comments, it appears that many subjects did refer to one or the other of these shadows to aid in assessing correct placement. Perhaps the secret of the apparent lack of effect is the very fact that either shadow can be used as an aid to placement, so that the most delayed one can be ignored in favour of the other—but since both views are merged into one common set of results, this effect is invisible.

To verify these tentative conclusions, an experiment should be designed with the following objectives: that the two views be distinguishable in the body of data and that they both be critically useful in the completion of the task. The first of the two criteria ensures that a subject's preference for the least-delayed view, for instance, is clearly visible in the results; the second criteria, of course, eliminates all doubts that the views are used at all.

6.2 Effect of Delays on Perspective View

The subjective comments of *rjk, sm* and *jfw*, claiming that delays in the perspective view actually simplified the task, initially seemed peculiar. It was likely that the objective measurements would not bear out such a claim, and that at best, increased p-delays gave the impression of a simpler task. Surprisingly, however, analysis of the results does in fact confirm the subjects' impression: longer delays in the p-view affords better performance, for some subjects, at least. The reason given by all three subjects to explain this phenomenon is that the perspective view often occludes the back-wall (XZ plane)

shadow, which according to the majority of subjects is one of the most useful views. Thus when the p-view is delayed, the shadow can be seen, and manipulated, without obstruction. If this is true, it follows that the longer the p-delay, the longer the shadow remains un-obstructed, and the easier the task.

The experiment as it stands has no provision for testing this hypothesis, although a simple modification may be sufficient for this purpose: by changing the perspective of the scene (by moving the viewpoint up and to the side, for instance), it is possible to minimise (but not eliminate completely) shadow occlusion by the p-view. In this modified setup, if the hypothesis is true, the effect of p-delays should disappear, or at least be greatly diminished. For this current study however, the fact that the same intuitive explanation was given by several subjects suffices to give some legitimacy to the hypothesis.

This result is interesting because of its implications for task-completion strategy. Indeed, it suggests that when faced with a new trial, a user visualises a sequence of manipulations to achieve the goal, and adheres to this plan even if it involves a view which later proves to be problematic. Thus, rather than change the pre-conceived plan while the trial is underway and use an easier view, the user persists with the occluded view, at the expense of performance. As the obstruction of the view lessens (by lengthening the delay of the p-view), the difficulty of executing the plan decreases.

Alternatively, the explanation might be that when faced with a difficulty in the plan, the user does indeed change strategies, but that this requires time. In the worst case, with no p-delay, the shadow is constantly occluded and a new plan must be elaborated every time. Then, as the perspective view delay increases, so does the chance of finding

the shadow un-obstructed when it is required; consequently, there is a decreasing need to re-think strategy, and performance increases.

To distinguish between these two possible explanations, two modifications could be made to the experiment. First, the detailed logs of user actions should include information on which shadow is being manipulated. A comparison of trials with and without p-delay would then show if the subject makes less use of the back-wall shadow when the perspective view occludes it (favouring the second hypothesis). A second test would be to include trials in which some of the views are completely blanked out, so that they provide no information and, more importantly, can not be manipulated. In this case, the user has no choice but to change strategies and the resulting effect can then be compared to the one found in this study.

These theories on occluded views are indirectly relevant to the study of delays: changes in strategy in response to a view made unusable by obstruction are likely similar to changes in response due to delayed views. If the first theory above holds, and users insist on using occluded views, then it is likely that they also continue to use views with extremely long delays. Once again, the modifications to the experiment suggested above can confirm this, by comparing very long-delay trials to blanked-out trials. If indeed users strive to keep their strategy as much as possible, the two effects should differ (since in the second case, it is not possible to keep the same strategy).

6.3 Effect of Delays on Interactive View

Delaying the interactive shadow, unlike all other views discussed above, has a clear effect on performance, visible throughout the collected body of data. And unlike the initially surprising effect of delays on the perspective view, delaying the interactive shadow acts, as expected, to reduce performance: the strong results presented in the previous chapter leave no doubt that longer i-delays cause increases in task time and number of operations. This is similar in many respects to the results of studies of delays in two-dimensional, single-view environments, described in Chapters 1 and 2.

It would appear, then, that although the two environments and task requirements are different, they are nevertheless approached in a similar manner by a user, so that delays have a similar effect. So for instance, it is conceivable that a subject could shift the focus of attention to another view when the manipulated one begins lagging (something which is not possible when the environment is composed of a single view), but the results shown here suggest that this does not occur. It is difficult to say at this point where the similarities between the two environments end; for instance, perhaps the multi-view three-dimensional task is mentally decomposed into a series of purely two-dimensional, single-view operations.[1] If this is true, then the results from this experiment should be statistically indistinguishable from those of a two-dimensional equivalent of the same placement task. On the other hand, it is also possible that the task maintains its multi-

[1] For example, many subjects stated that they first achieved correct vertical placement, which is in fact one-dimensional. Horizontal placement is then two-dimensional, and can be achieved by using only the bottom shadow (XZ plane), although the comments indicate that this was infrequently done. In this respect, it might have been preferable to set a spherical target area, in which correct placement in any one dimension depends on current placement in the other two dimensions.

view characteristics at all times, but that subjects focus mainly on the view they are manipulating, thus weighing that delay more than the others in the final results. In this case, one would expect to find a lessened effect compared to a pure single-view version of the task.

As the above discussion points out, the key to deciding whether or not the task is in fact approached as a sequence of two-dimensional operations is to obtain results from a true two-dimensional task which is as similar as possible to the task used here. Otherwise, if a different setup is used (such as the various experiments described in Chapter 1), many factors other than the number of available views could account for variations in results. The best approach to designing the equivalent two-dimensional task might in fact be to extend the experiment suggested in the previous section, where certain views are "blanked out" at various times during the trials. This time, however, all views except the one currently under manipulation should be blanked out, ensuring that the task can be nothing more than a sequence of single-views operations. Throughout the trials, the i-shadow—the only visible view—should of course be subjected to a range of delays, so that the performance curves can be compared to the ones presented in the previous chapter.

Regardless of whether or not subjects approach the task in a single-view mind set, there is once again the question of strategy, and of how often it can be changed. The much larger effect of i-delay, compared to other delays, is certainly due in large part to the phenomena discussed above, but it is possible that some of it is due to an inability to adapt plans to new conditions (where the new condition is a delayed shadow). The same

modified experiment as described above can be used to gauge this effect, this time by blanking out the i-shadow during some trials; in the meantime, however, the current set of results shows hints that when pushed beyond certain limits, subjects do in fact change strategies; this is discussed in the following section.

6.4 Possible Non-Linear Relationship Of i-delay

Indeed, a result which was apparent early, and recurred throughout the analysis, is the possibly non-linear relationship between the controlled variable (principally the i-delay) and the response (either time-to-complete or number-of-tries). Although the graphs all show an arrangement of points which is not far from linear, and although analysis of the secondary experiment does not support the existence of a non-linear effect, there is a consistent similarity in the shape of all the plots—a shape which is suggestive of a possible asymptotic relationship, as shown in Figure 6.1.

This type of asymptotic relationship could have many explanations; in particular, it is possible that in order to complete the placement task, subjects call upon two different processes. The shape of the curve in Figure 6.1 would then suggest that one of these processes dominates for low i-delays, but that for delays larger than some key value, the second process takes over. In other words, it is possible that in extreme cases, subjects finally resort to a new task-completion strategy. In Figure 6.1, the curve is sketched asymptotic to a line of zero slope, but this is not necessarily the case, and a more general view of the relationship is shown in Figure 6.2. In this case, slope m_1 approximates the magnitude of the response time to controlled variable relationship for

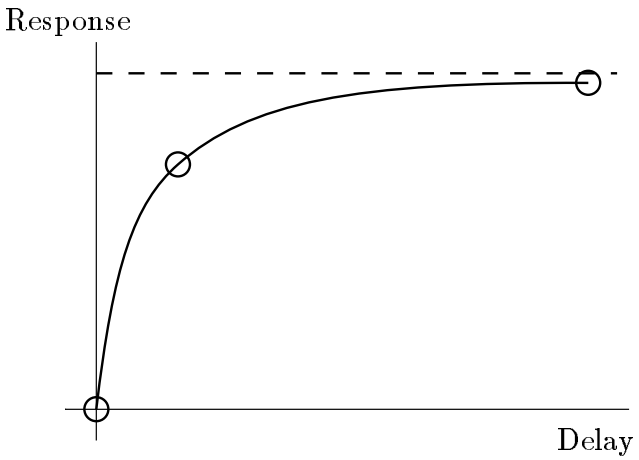

Figure 6.1: Sketch of response (e.g., time-to-complete) against delay, showing a possible asymptotic relationship

"less difficult" cases, where subjects rely on a *feedback* process for completing the task: placement is monitored throughout the interaction, and feedback information is used to adjust the operation. Slope m_2 represents the same relationship for "more difficult" configurations, corresponding to a *ballistic* process, where the subject pre-visualises the required operation and performs it without any regard for feedback (in a process analogous to saccadic eye movements). The partitioning of delays into "less difficult" and "more difficult" occurs at point p, which is where the ballistic process becomes the critical factor in the subject's performance.

It would be interesting to examine the various features of this model, if indeed it is believed to accurately describe a subject's mode of operation. In particular, what is the slope m_2? A value of zero indicates that p is in fact the point of "maximum difficulty,"

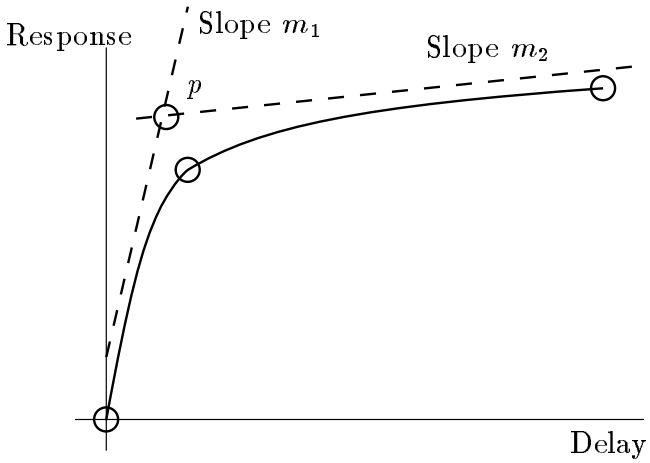

Figure 6.2: Sketch of response (e.g., time-to-complete) against delay, showing a possible asymptotic relationship to two separate line segments

and implies that for very long delays, subjects no longer rely on any information from the view. A slope of one is no less significant, and suggests that the difficulty of the task no longer increases beyond point p, but that subjects are simply waiting for the view to catch up (and thus completion time is directly proportional to delay). Both these slopes are consistent with a ballistic process, as described above. In many respects, determining slope m_1 (where the feedback process dominates) is even more interesting. From a practical standpoint, real-world applications are more likely to encounter shorter delays, and knowing the effect of these delays on users allows designers and programmers to gauge difficulty levels, and find possible work-arounds. Since the experiment described here focuses mainly on the shorter delay end of the spectrum (notably by starting at zero delay), it should give a fairly accurate idea of m_1 (approximately two), even though it was

not designed with the two-process model in mind. Finally, since point p appears to play such an important role, it would be important to uncover the factors which determine its location.

6.5 Strategy

From the discussions above, it appears that the common thread tying together the effects of various delays is strategy, defined as the sequence of operations employed by the subject to complete the given task. The theory so far is that a "plan of attack" is conceived at the start of a trial—presumably before the first manipulation operation, and thus before the completion-time clock starts ticking—and that this plan is then followed as closely as possible, with little regard to external factors such as delays. However, although this hypothesis could explain many of the effects revealed by the results of this study, nothing in the experiment addresses it directly, and it remains for future work to prove or disprove it.

The single most interesting point for further study therefore appears to be subject strategy. Precise knowledge of how a user plans to achieve the task will help explain the effects found in this study, and will provide insight on ways to reduce performance degradations caused by delays. This last result is important for the design of real-world applications, as it provides an efficient framework for environments where delays are unavoidable.

Several modifications to the current experiment have already been suggested to study different aspects of strategy. The most important change is to include more detail in

the trial output. In addition to timestamp and distance information, the selected view should be recorded. With this information in hand, the simplest analysis is to search for patterns and determine whether or not subjects truly have a common strategy across trials, as suggested by their verbal comments. To help identify patterns, it might be necessary to run multiple trials with identical configurations, as changes in configuration are likely to cause changes in strategy. And in fact, the next step is to determine if such modifications to strategy do actually occur (by comparing whatever patterns have emerged for various configurations), thus addressing many of the points raised in previous sections of this discussion. Finally, as was suggested above, it would be interesting to force subjects to modify their plans, to obtain a point of comparison (a pattern of operation which is guaranteed to be different) and to see what the fall-back strategy might be. One previously-suggested method of inducing a change of plans is to temporarily remove a view from the scene, making it unavailable for both feedback and manipulation. Depending on the types of strategy detected, another approach might be to "gray-out" certain views, thus marking them as unavailable for manipulation but retaining them for visual feedback. It should be noted that in general, research on strategy is costly in terms of time required of the subjects: this type of study typically requires a very large number of trials, so that patterns can be correctly identified. Furthermore, it is often necessary to consider blocking effects (some of the subjects are presented with similarly-configured trials blocked together, while others are presented with a randomised set of configurations), thus requiring a larger number of subjects.

There are potentially other effects of delays associated with strategy, which have not been mentioned yet. For instance, some of the verbal comments mention difficulties

caused when the next view to be manipulated (as per the pre-formed plan) is currently delayed—so that the subject must wait before attempting the next operation. In the current setup, views are labelled according to the current operation in progress (except for the perspective view), so that in general each manipulation is considered in isolation of all others. Thus no carrying forward of delay effects into the next manipulation is visible in the results, making it impossible to verify the claims mentioned in the comments. One way to overcome the current limitation of the experimental platform is to assign delays to specific views (such as bottom or back-wall shadow), rather than linking them to the currently selected view. With this model, it is possible to correlate observed behaviour—such as a change in strategy—at any given time with prior delays, although it is now more difficult to identify causal effects on views as they relate to the user (effect of delaying the interactive shadow, for instance). In fact, this approach changes the thrust of the study significantly, as subjects can now simply avoid manipulating—or even looking at—delayed views. Thus, before such an experiment is attempted, thought should be given to the exact nature of the task, as modifications may be required to ensure that full use of all views is required.

6.6 Fine-Tuning Problems

The final issue which needs to be addressed is fine-tuning. Schwarz (1985) interpreted fine-tuning as a difficulty inherent in the task rather than effects of interaction mode. And indeed, to some extent this is the case with the raw data, as demonstrated by the results of the filtering script: it reveals, on average, a few data points per subject for which the

molecule is very close to the target area, and little appreciable change in position is made. The conclusion must be that the problem in these trials is finding the boundary of the target area, rather than dealing with delays.

However the results of analysing the filtered data are very similar to those of the raw data with outliers removed during analysis. This suggests that trials with fine-tuning problems generally appear as outlying points in the data, and that the process of removing outliers (a standard procedure in human experiments, for reasons described previously) manages to remove these points. Thus, while fine-tuning problems do exist, they do not interfere with the results. And although a specific filter tailored to the information contained in the raw data (information which is unfortunately absent in the results of this experiment, but which could easily be added in the future) is the preferred method of masking fine-tuning problems, a statistically-based approach such as was used here appears adequate. In summary, then, fine-tuning problems are not a major issue, and the platform described and developed here is a viable one for future studies.

6.7 Conclusions

As expected, the problem of de-synchronisation and delays in a multiple view environment is a large and complex one, and this study only scratches the surface. Nevertheless, it identifies avenues for continued research, ranging from studying strategy to determining the relative importance of the various views. Furthermore the results obtained here suggest at least one general statement: the view which is the focus of the user's interaction plays the most important role in determining task difficulty, more so than all other views.

Thus, if an application designer were to look for a rule of thumb for creating an optimal environment, the suggestion should be to make every effort to reduce delays to a minimum in the interactive view, possibly even at the expense of increases in delays to other views. Further study will help refine and quantify this rule of thumb.

In particular, designers might wish to have a mathematical expression relating desynchronisation and delays in given views to task difficulty; this could help them choose which areas to optimise for greatest improvements in the overall environment. Such formulas exist for single view tasks, usually as a variation of Fitts' Law; research could be done to extend and reformulate the Law for multiple view situations. The experiment described here could then be modified to provide results suitable to the mathematical expression and to compute the index of difficulty.

Bibliography

Bartels, R. H. 1994. Object-oriented spline software. In Laurent, Le Méhauté, and Schumaker (Eds.), *Curves and Surfaces in Geometric Design (Proceedings of the Second Chamonix Conference on Curves and Surfaces)*, pp. 27–34. A. K. Peters, Wellesley, MA.

Bertrand, P. F., W. Cowan, and M. Wein. 1993. A window architecture providing predictable temporal performance. In *Proceedings of Graphics Interface '93 (Toronto, Ontario, May 19–21, 1993)*, Toronto, Ontario, pp. 146–154. Canadian Information Processing Society.

Bier, E. A. 1987. Skitters and jacks: Interactive 3-D positioning tools. In *Proceedings of 1986 Workshop on Interactive 3D Graphics (Chapel Hill, NC, October 23–24, 1986)*, New York, pp. 183–196. ACM.

Card, S. K., T. P. Moran, and A. Newell. 1986. The model human processor. In K. R. Boff, L. Kaufman, and J. P. Thomas (Eds.), *Handbook of Perception and Human Performance*, Volume 2 (Cognitive Processes and Performance), Chapter 45. John Wiley And Sons, Inc.

Chen, M., S. J. Mountford, and A. Sellen. 1988. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of SIGGRAPH '88 (Atlanta, GA, August 1–5, 1988)*, Volume 22 of *Computer Graphics*, New York, pp. 121–129. ACM SIGGRAPH.

Conner, D. B. et al. 1992. Three-dimensional widgets. In *Proceedings of 1992 Symposium on Interactive 3D Graphics (Cambridge, MA, March 29–April 1, 1992)*, New York, pp. 183–188. ACM.

Cruz-Neira, C., D. J. Sandin, and T. A. DeFanti. 1993. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proceedings of SIGGRAPH 93 (Anaheim, CA, August 1–6, 1993)*, *Computer Graphics* Proceedings, Annual Conference Series, New York, pp. 135–142. ACM SIGGRAPH.

Evans, K. B., P. P. Tanner, and M. Wein. 1981. Tablet-based valuators that provide one, two or three degrees of freedom. In *Proceedings of SIGGRAPH '81 (Dallas, TX, August 3–7, 1981)*, Volume 15 of *Computer Graphics*, New York, pp. 91–97. ACM SIGGRAPH.

Finke, R. A. and R. N. Shepard. 1986. Visual functions of mental imagery. In K. R. Boff, L. Kaufman, and J. P. Thomas (Eds.), *Handbook of Perception and Human Performance*, Volume 2 (Cognitive Processes and Performance), Chapter 37. John Wiley And Sons, Inc.

Fitts, P. M. 1954. The information capacity of the human motor system in controlling amplitude of movement. *Journal of Experimental Psychology 47*, 381–391.

Fitts, P. M. and J. R. Peterson. 1964. Information capacity of discrete motor responses. *Journal of Experimental Psychology 67*, 103–112.

Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. 1990. *Computer Graphics Principles and Practice* (Second ed.). Addison-Wesley.

Friedberg, J., L. Seiler, and J. Vroom. 1990. Extending X for double-buffering, multi-buffering and stereo, version 3.3. A proposed standard, for public review.

Friedman, M., T. Starner, and A. Pentland. 1992. Device synchronization using an optimal linear filter. In *Proceedings of 1992 Symposium on Interactive 3D Graphics (Cambridge, MA, March 29–April 1, 1992)*, New York, pp. 57–62. ACM.

Gentleman, W., S. MacKay, D. Stewart, and M. Wein. 1989. Using the harmony operating system: Release 3.0. Technical Report ERA-377, NRCC No. 30081, National Research Council of Canada, Division of Electrical Engineering, Ottawa, Ontario.

Herndon, K. P., R. C. Zeleznik, D. C. Robbins, D. B. Conner, S. S. Snibbe, and A. van Dam. 1992. Interactive shadows. In *Proceedings of ACM Symposium on User Interface Software and Technology (UIST) (Monterey, CA, November 15–18, 1992)*, New York, pp. 1–6. ACM.

Hockey, G. R. J. 1986. Changes in operator efficiency as a function of environmental stress, fatigue and circadian rhythms. In K. R. Boff, L. Kaufman, and J. P. Thomas (Eds.), *Handbook of Perception and Human Performance*, Volume 2 (Cognitive Processes and Performance), Chapter 44. John Wiley And Sons, Inc.

Howell, D. C. 1992. *Statistical Methods for Psychology* (Third ed.). Wadsworth Publishing Company.

Jones, O. 1989. *Introduction to the X Window System*. Prentice Hall.

Keffer, T. 1991. *Tools.h++ Introduction and Reference Manual, Version 4*. Corvallis, OR: Rogue Wave Software, Inc.

Knowlton, K. and L. Cherry. 1977. Atoms—a three-D opaque molecule system for color pictures of space filling or ball and stick models. *Computers and Chemistry* (1), 161–166.

Kroeger, R. J. 1993. Sonification: Adding streams of sound to a user interface. Master's thesis, University of Waterloo.

MacKenzie, I. S., A. Sellen, and W. Buxton. 1991. A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of CHI, 1991 (New Orleans, Louisiana, April 28–May 2, 1991)*, New York, pp. 161–166. ACM.

MacKenzie, I. S. and C. Ware. 1993. Lag as a determinant of human performance in interactive systems. In *Proceedings of INTERCHI, 1993 (Amsterdam, The Netherlands, April 24–29, 1993)*, New York, pp. 488–493. ACM.

Mackinlay, J. D., G. G. Robertson, and S. K. Card. 1991. The perspective wall: Detail and context smoothly intergrated. In *Proceedings of CHI, 1991 (New Orleans, LA, April 28–May 2, 1991)*, New York, pp. 173–179. ACM.

Nye, A. 1990. *Xlib Programming Manual* and *Xlib Reference Manual* (Second ed.), Volume One and Two. O'Reilly & Associates, Inc.

Osborn, J. R. and A. M. Agogino. 1992. An interface for interactive spatial reasoning and visualization. In *Proceedings of CHI, 1992 (Monterey, CA, May 3–7, 1992)*, New York, pp. 75–82. ACM.

Parasuramon, R. 1986. Vigilance, monitoring, and search. In K. R. Boff, L. Kaufman, and J. P. Thomas (Eds.), *Handbook of Perception and Human Performance*, Volume 2 (Cognitive Processes and Performance), Chapter 43. John Wiley And Sons, Inc.

Rees, P. K. 1963. *Analytic Geometry, 2nd edition.* Prentice-Hall.

Robertson, G. G., S. K. Card, and J. D. Mackinlay. 1993. Information visualization using 3D interactive animation. *Communications of the ACM 36*(4), 56–71.

Schwarz, M. W. 1985. An empirical evaluation of interactive colour selection techniques. Master's thesis, University of Waterloo.

Segal, M. and K. Akeley. 1992. The OpenGL graphics system: A specification. Technical report, Silicon Graphics Computer Systems, Mountain View, CA.

Smith, C. 1953. *Conic Sections, Co-ordinate Geometry.* Macmillan And Co.

Teal, S. L. and A. I. Rudnicky. 1992. A performance model of system delay and user strategy selection. In *Proceedings of CHI, 1992 (Monterey, CA, May 3–7, 1992)*, New York, pp. 295–305. ACM.

Tukey, J. W. 1977. *Exploratory Data Analysis.* Addison-Wesley.

Velleman, P. F. 1989. *Data Desk.* Northbrook, IL: Odesta Corporation.

Walpole, R. E. and R. H. Myers. 1978. *Probability and Statistics for Engineers and Scientists* (Second ed.). Macmillan Publishing Co.

Wanger, L. 1992. The effect of shadow quality on the perception of spacial relationships in computer generated imagery. In *Proceedings of 1992 Symposium on Interactive 3D Graphics (Cambridge, MA, March 29–April 1, 1992)*, New York, pp. 39–42. ACM.

Ware, C., K. Arthur, and K. S. Booth. 1993. Fish tank virtual reality. In *Proceedings of INTERCHI, 1993 (Amsterdam, The Netherlands, April 24–29, 1993)*, New York, pp. 37–42. ACM.

Ware, C. and R. Balakrishnan. 1994. Target acquisition in fish tank VR: The effects of lag and frame rate. In *Proceedings of Graphics Interface '94 (Banff, Alberta, 18–20 May 1994)*, Toronto, Ontario, pp. 1–7. Canadian Information Processing Society.