Design Decisions for Implementing Backside Video in the SomeProduct

Prepared by
A Student
Student ID: 20000000
User ID: uwid@uwaterloo.ca
3A Software Engineering
September 9, 2005
Dear Sir:

This report, titled “Design Decisions for Implementing Backside Video in the SomeProduct”, is my third work term report. It deals with a problem I faced during my 3A work term experience, under the employment of SomeCompany Limited.

SomeCompany is a privately owned, rapidly growing company dedicated to designing and manufacturing the best in television equipment used in live broadcasting. I was employed in the Research & Development Facility, which is responsible for designing, developing, and maintaining the products that SomeCompany supplies to the world market. During my work term, I was assigned to work in the graphical stream of the newest model of production switchers, assisting the current development team in adding new features and effects.

This report describes different possible solutions for implementing the Backside Video feature in the SomeProduct switcher, and investigates how well each solution satisfies the specific constraints and criteria expected from this feature. The report’s audience would be the management in charge of supervising the 3D graphical card features, as well as any software engineer who wishes to understand the reasoning behind the design of Backside Video.

I would like to thank all my co-workers at SomeCompany for their help and support during my stay with them. I would especially like to thank Steve Robinson for mentoring me, for his advice and suggestions regarding my work, and guidance throughout the term. I hereby confirm that I have received no help, other than what is mentioned above, in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

A. Student

Student ID: 20000000
Executive Summary

In July 2005, work on the Backside Video feature for the SomeProduct production switcher started. Backside Video was a feature that allows the user to control two video sources on-screen as if they were different sides of a flat rectangle, called a flykey. There was a serious glitch discovered, and the feature was eventually implemented with difficulty and some imperfections. The purpose of this report was to elaborate on the glitch, define design criteria and constraints for solutions, and describe the best solution implemented compared to other alternatives.

The backside glitch was the result of the video source switching 3 video frames (fields) after the 3D card knew it should change. The main goal was to make the two changes synchronized on screen. Any solution had to appear professional enough to ship, and fast enough to not crash the switcher. Appearance was the most important criterion, followed by the speed and maintainability of the solution.

The problem was split into two scenarios since no single solution we analyzed would fix everything. The Automatic Control scenario was when the flykey would move in a predetermined way from recalling a state in memory. The best solution found would precalculate a future state 3 fields ahead, using the same code that would normally move the flykey along its path. The Manual Control scenario was when the flykey would move according to direct human control via a control panel. The best solution found would look at the current flykey state and the previous flykey state, and from that linearly predict what a future state would be 2 fields ahead.

Future work on Backside Video should use the weighted criteria listed in this report. Demo artists need to be told about the slight problem with Manual Control scenarios in order to work around it during presentations or explain to customers that it looks better under Automatic Control. Also, it is recommended during the next iteration of hardware design to make a fix for the 3 field delay, as that is what caused this issue in the first place.
# Table of Contents

Executive Summary ........................................................................................................ iii
List of Figures .................................................................................................................. v
List of Tables .................................................................................................................. vi
1 Introduction .................................................................................................................. 1
   1.1 Purpose of the Report ......................................................................................... 1
   1.2 Background Information ............................................................................... 2
2 Problem Specifications ............................................................................................... 4
   2.1 Detailed Problem Description ...................................................................... 4
   2.2 Design Constraints and Criteria ................................................................... 5
3 Accepted Solutions ..................................................................................................... 6
   3.1 Automatic Control and Precalculation ......................................................... 6
   3.2 Manual Control and Linear Prediction ....................................................... 8
4 Problem Specifications ............................................................................................... 10
   4.1 Flykey Delay ............................................................................................... 10
   4.2 Panel Calculation ....................................................................................... 10
   4.3 Polynomial Prediction .............................................................................. 13
5 Conclusions ................................................................................................................. 15
6 Recommendations .................................................................................................... 16
References ...................................................................................................................... 17
Acknowledgements ........................................................................................................ 18
Appendix A – Detailed Timing Figures .......................................................................... 19
   A.1 Automatic Control ....................................................................................... 19
   A.2 Manual Control ......................................................................................... 19
List of Figures

Figure 1.1: A high-level view of the various components of a production switcher.....2

Figure 1.2: An example of flykey usage........................................................................3

Figure 2.1: Example of the display rectangle defining an alpha pattern..................7

Figure 3.1: How predicting too far might cause linear prediction solution to be incorrect.................................................................................................................................9

Figure 4.1: The message timing between the original and panel calculation........11

Figure 4.2: Showing the accuracy of polynomial prediction.............................13
1 Introduction

1.1 Purpose of the Report

SomeCompany is a company that “designs, markets, manufactures, and supports a wide range of innovative products for use in broadcast, distribution, live event and production applications” [1]. Their products can be “found in over 100 countries and are used 24 hours a day, 365 days a year to produce and distribute video and audio signals” [1]. They are well known for their commitment to quality products. In July 2005, I was assigned by the software development team at SomeCompany to implement a particular feature for the SomeProduct, SomeCompany’s newest model of production switcher. This particular feature, known as “Backside Video”, allows the user to control two video sources on-screen as if they were different sides of a flat rectangle. Rotating the rectangle would show one of the two video sources, depending on whether the front or the back was visible. Backside Video is a useful and important feature for our customers, and can be used to create stylish effects and transitions in their broadcasts.

However, I soon discovered that the straightforward solution used in the previous model would not be acceptable. If implemented in that way, Backside Video would suffer from a visual glitch noticeable enough that it would better to leave the feature out rather that sell it in that state. I was asked by the development team to design a solution that would reduce the appearance of the Backside Video glitch as much as possible.
After much analysis, I submitted my solution to the product manager and it was deemed acceptable to be shipped in the next version release.

This report covers the design decisions made for the implementation of Backside Video, and is intended for SomeCompany employees who wish to understand why Backside Video was designed the way it was at the time of this writing. This report will describe the Backside Video glitch in more detail, and list the design specifications and criteria that were used to judge possible solutions. This report will then describe the implemented solutions, and show how well they satisfied the criteria as well as any disadvantages that they have. Finally, this report will describe alternative solutions and explain why they were not chosen compared to the accepted solution.

1.2 Background Information

The SomeProduct is a live production switcher that “utilizes the latest technology to operate in all popular Standard and High Definition [television] formats” [1]. Live production switchers are devices that are able to control and manipulate multiple television sources in real-time, meaning that there is negligible delay between the incoming video signals, the switcher modifying the signal, and the outgoing television signal being broadcast. They are used to manage the directing of any live broadcast, like news bulletins, sporting events, or weather reports. Just about every television studio would have a live production switcher as the heart of their facility.

Switchers consist of multiple components, which work together to form the complete product, as shown in Figure 1.1. The control panel is the physical interface by which the user can manipulate the system. From the panel, you can select the video sources to broadcast and how you wish to broadcast them. Several specialized hardware cards within the switcher handle the processing of the video. All of this is connected to the frame motherboard that acts as the heart of the switcher, making sure all the different components cooperate. Finally, the output video is broadcast to the intended audience.
One of the specialized cards handles 3D-graphic manipulation (and will be called the “3D card” for the remainder of this report). The 3D card is an add-on to the basic Synergy-MD, and its primary function is to manipulate a live video source as if it were a rectangle in 3D space. This rectangle (to be called a “flykey” for the remainder of this report) is placed in front of the normal background video and can be warped and rotated however the user wants. Any part that the flykey doesn’t cover the screen will reveal the background video behind it. The system can currently support 8 simultaneous flykeys per 3D card.

There are two different ways that flykeys are manipulated. The first and most professional way is to use automatic control, which “enables you to slew a switcher setup from its current setting to a new recalled setting. The switcher interpolates between the two settings at a given rate to create a smooth two-keyframe effect” [2]. The second and less common method is to have manual control, where the switcher user controls the
flykey directly with the control panel. During manual control, the user would be able to modify the flykey anyway s/he pleased, but probably would not be able to recreate the same effect twice due to human limitations.
2 Problem Specifications

2.1 Detailed Problem Description

As one of the listed features of the Synergy-MD’s 3D card, Backside Video allows the user to show a different video source on the other side of a flykey. Rotating the flykey would reveal the secondary video source, as if the flykey was a double-sided physical object. It is a useful feature and widely used by customers to create video effects, and to make transitioning between broadcast segments more attractive.

In order to create the backside effect, the 3D card computes whether the front or the back of the flykey is visible. Whenever the visible side changes, the 3D card notifies the frame to switch the flykey’s video source. The computation is done by converting the flykey’s information (like position, rotation, aspect, scaling) into a transformation matrix, and then comparing the z-axis of the matrix to the vector from the viewpoint.

It was originally thought that the system would be able to swap video sources in sync with the orientation of the flykey on screen. However, while implementing this feature in the new Synergy-MD, it was discovered that a simple, encompassing solution would not be adequate. Because of a patch for a critical feature more essential than Backside Video, all messages to switch video sources needed for the flykey became delayed by 3 fields. Thus the video switch would happen 3 fields after the back of the key became visible, which ruins the Backside Video effect as described in Table 2.1.

<table>
<thead>
<tr>
<th>Timeline</th>
<th>Before frontside/backside switch</th>
<th>Flykey has just switched</th>
<th>2 fields after switch</th>
<th>3 fields after switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>Rotating flykey to the right</td>
<td>3D card sends message to frame</td>
<td>Synchronization delay – B should be showing but isn’t</td>
<td>Frame finally switches video sources</td>
</tr>
<tr>
<td>Video Output</td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>
2.2 **Design Constraints and Criteria**

Because the quality of the product was paramount, there were some design constraints that any presented solution had to comply to. First, solutions must not cause any part of the switcher’s runtime to exceed the shortest supported duration between fields, or bring the switcher dangerously close to doing so. If some component cannot process all its calculations for the current field before the next field starts, that component will essentially lose all functionality. Since the highest frequency rate the SomeProduct supports is 60 Hz, the shortest duration allowed for runtime is 16.666 milliseconds.

Second, solutions must be in presentable condition enough to sell to clients. This means that visual flaws, if any, should be small enough to be essentially unnoticeable to a regular user, and completely hidden for any viewers watching the user’s broadcast.

If a solution managed to pass the constraints, then three criteria are used to judge the remaining solutions. The most important criterion is the appearance of the solution. The more accurately a solution follows the behavior expected, the more the customer will be pleased with the feature. The original sync delay of 3 fields would not pass the design constraints, but 2 fields would be somewhat acceptable, 1 field would be very good, and 0 fields would be perfect.

The next important criterion used to judge solutions is the speed of the solution. Because of the limited amount of runtime available in the switcher, every new feature or addition reduces the total available runtime. Keeping the speed of solutions fast allows more flexibility for modifications in the future.

The last criterion would be how maintainable the solution is. The SomeProduct is a massively large and overwhelming project, and “programmers who use every trick in the book to save... computing time at the expense of clarity [would] not in tune with the cost structure of today’s world” [3]. Even though maintainability is a criterion hidden from clients, it is still essential, since complex and obtuse code makes developing new features and fixing bugs more costly in the future.
3 Accepted Solutions

3.1 Automatic Control and Precalculation

Automatic control of the flykey is the expected primary usage for our customers, since shows usually rely on effects that are easily reproducible. It is unheard of for a live broadcast to manually modify a flykey for working purposes. As such, making sure that Backside Video worked well under automatic control was of the highest priority. The expected audience of this situation would be the broadcast viewers, so as long as the glitch was hidden from them, it would be fine. The expected usage during broadcast would be to activate the dissolve simply, one at a time, without any unexpected interruptions. As such, the flykey conditions are very predictable during typical use of automatic control.

The solution that was implemented was to precalculate the flykey’s orientation 3 fields into the future, using the same code that would usually perform automatically controlled dissolves. When the 3D card receives notice that it is dissolving a flykey with Backside Video on, it does an extra call to the dissolving function to check the state of the flykey 3 fields into the future. It sends that future result to the frame instead of the current orientation to remove the backside glitch. If the final state is less than 3 fields away, it calculates the final state instead, so the solution doesn’t overshoot.

Table 3.1. How the precalculation solution works.

<table>
<thead>
<tr>
<th>Timeline</th>
<th>Before frontside/backside switch</th>
<th>3 fields before switch</th>
<th>Flykey has just switched</th>
<th>1 field after switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>Rotating flykey to the right</td>
<td>3D card sends message to frame</td>
<td>Frame switches video sources</td>
<td>Effect recreated perfectly</td>
</tr>
<tr>
<td>Video Output</td>
<td><img src="image1.png" alt="Image A" /></td>
<td><img src="image2.png" alt="Image A" /></td>
<td><img src="image3.png" alt="Image A" /></td>
<td><img src="image4.png" alt="Image A" /></td>
</tr>
</tbody>
</table>

The appearance of this solution is almost perfect for typical use. Because the path of a dissolve is calculated the same way every time, we can calculate the future state of a...
flykey with perfect accuracy. The dissolve can be as complex as possible, and the future state will not be fooled.

However, there are still weaknesses that stem from the 3 field delay. Any situation where the dissolve switches sides less than 3 fields from the start will obviously not change in time. This kind of usage would be unlikely from a client though, and the glitch would be seen in any practice runs before it was used in live broadcast. Also, another glitch is found when a dissolve is manually interrupted just after it sends the message to the frame to switch video sources, but before it actually switches sides on the screen, as illustrated in Figure X.X. Again, such usage would not be done during an actual broadcast, and is not significantly detrimental to the solution as a whole.

Table 3.2. How precalculation solution can glitch.

<table>
<thead>
<tr>
<th>Timeline</th>
<th>3 fields before switch</th>
<th>2 fields after first message sent</th>
<th>3 fields after first message sent</th>
<th>5 fields after first message sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>3D card sends message to frame</td>
<td>Dissolve is manually halted, 3D card sends second message to frame</td>
<td>Frame receives first message, switches video</td>
<td>Frame receives second message, switches video back</td>
</tr>
<tr>
<td>Video Output</td>
<td><img src="Image1" alt="Image" /></td>
<td><img src="Image2" alt="Image" /></td>
<td><img src="Image3" alt="Image" /></td>
<td><img src="Image4" alt="Image" /></td>
</tr>
</tbody>
</table>

The speed of this solution is fairly slow when compared to most of the other solutions discussed later in this report. Calculating the entire future state of a flykey during an effects dissolve and then converting that to a transform matrix requires an additional 0.957 ms for the current maximum of 8 flykeys. This is 5.74 percent of the absolute constraint of 16.666 ms, and is quite a significant percentage to spend on just one feature. One idea that was dropped later on was to save the future calculations locally, and access those saved values when that future state became the current state. Doing this would reduce the additional calculations for the most part. However, the 3 starting fields would still have to do the double calculation in order to stay in sync, and thus if the runtime was going to go over the limit with this solution, it would go over in those 3 starting fields anyway.
The maintainability of this solution is quite good. Much of the solution uses existing, tested code, and any additional code can be self-contained in a few files within the 3D card project. The estimated time to code and verify this solution would only be about 3 workdays. The design used here can also be used for the upcoming sequence feature, which is essentially several dissolves linked together in a row.

### 3.2 Manual Control and Linear Prediction

The second way that a flykey can be manipulated is to manually adjust the flykey’s settings using the control panel. This type of control is mainly used by customers to position flykeys to store in memory, and by our company’s demo artists who might use this feature manually in their presentation to impress a potential client. As such, the audience will not normally see flykeys under manual control. Because manual control is used to experiment and play around with the flykeys, the path of the fly is human controlled and can become erratic.

The solution that was implemented for this situation was to use a linear extrapolation algorithm to predict the future state of the flykey from its previous movements. This solution uses the current state and the previous state of the flykey to calculate a predicted future state. If the predicted flykey happens to show a different side than the current state, then the 3D card notifies the frame to switch sources.

The appearance of this solution is satisfactory when the solution predicts only 2 fields into the future. This number of fields was specifically chosen due to a loss in visual quality when predicting more or less fields. If 1-field prediction were used, then the Backside Video glitch would be much more noticeable than with 2-field prediction. However, if 3-field prediction were used, then the predictions became much more prone to the type of errors seen in Figure 3.1. Predicting too far leads to situations where erratic movements can easily prevent accurate predictions from being. 2-field prediction struck a pleasant bridge between the two types of major appearance flaws.
The speed of this solution is moderately fast. Calculating the predicted state of all 8 flykeys requires 0.516 ms of runtime. This is 3.09 percent of the absolute limit of 16.666 ms, which is quite small when compared to the alternative solutions. Also, the maintainability of this solution is fairly good. This solution required a significant amount of new code, but most of the code is already simple in nature, and can be self-contained within the 3D card project. The estimated time to code and verify this solution would be about 5 workdays. Overall, linear prediction was a good solution for manual control. It was also considered as a solution for automatic control, but it was rejected primarily because the appearance of the precalculation solution was much more accurate.
4 Alternative Solutions

4.1 Flykey Delay

When the Backside Glitch was first discovered, one of the first ideas suggested was to simply delay the current flykey’s appearance on screen by 3 fields, in order to give the frame enough time to switch video sources. This kind of solution would have been very easy to implement and very fast in terms of processing speed. It also would have completely removed the Backside Glitch in both the automatic and manual control scenarios.

However, the fatal flaw that such a solution would affect various 3D card features as a whole. Adding the delay to the current flykey state would cause controlling flykeys to become laggy and inconvenient. Features that have found solutions around the 3-field delay, or even rely on the 3-field delay, would have to be changed. Because this solution did not pass the design constraints in the eyes of the product managers, this solution was rejected.

4.2 Panel Calculation

One of the alternative solutions that would only work for manual control focused on the fact that during manual control, the panel had access to flykey-modifying information before the 3D card did. As mentioned before, when the flykey is manually controlled by a user, the panel parses what the user is doing and sends that information to the 3D card. The 3D card would then process that information, modify the flykey on screen, and message the panel back to display specific details about the flykey’s current state to the user. Occasionally, the 3D card would message the frame whenever the backside video needed to change.

However, during discussions about Backside Video, a coworker suggested the idea that the panel could notify the frame instead of the 3D card. If the panel already had the flykey’s current orientation and the information on what the user was changing, it would be possible to calculate the frontside/backside state directly on the panel. Doing this
eliminates one of the time delays in notifying the frame, and would decrease the glitch delay. This pathway is shown as the dashed line in Figure 4.1. A downside of this design is that panel calculation only works for manual control. Using this as a solution for automatic control doesn’t work since this solution relies on the panel having access to information before the 3D card, which is only true in the case of manual control.

![Figure 4.1. The message timing between the original and panel calculation.](image)

In theory, the appearance of this design would be quite acceptable. The time saved from sending the switch message from the panel makes the glitch only 1 field off sync, which is a vast improvement from the usual 3-field sync error. Because the panel is not using any type of prediction-like algorithm and is simply calculating the flykey’s orientation as is, there are no glitches due to erratic movements or awkward starting positions like with the prediction solution.

However in actual practice, there were some complications. Occasionally the panel would fail to calculate the correct side of the flykey immediately after the user command that would have changed it on the 3D card, and the full 3 field delay would occur. Throughout the duration of testing I was unable to figure out the cause of this failure, even though it was believed that all the relevant code had been ported successfully to the panel. The fact that there was such difficulty in finding this bug and why it didn’t match indicated that it would be hard to maintain panel calculation in general.
When compared to the prediction method, panel calculation is definitely less maintainable. First, code must be duplicated between both the 3D card and the panel, and must work exactly the same way in order for the backside video to sync up correctly. If the way that the 3D card handled flykeys changed, the panel code would also have to be changed. Also, there was the conceptual problem of putting 3D calculation code into a module that was not designed for such a task. Many significant code changes needed to be added in order to send messages from the panel to the video source hardware. The estimated time to code and verify this solution would be about 15 workdays, 3 times that of linear prediction.

The speed of this solution could not be analyzed by normal means. On the 3D card, this solution actually improved performance by a small amount since all the existing backside code was moved to the panel. We didn’t have the equipment to record runtime on the panel, so I could only assume that this method would put a similar, if not more substantial, strain on the panel runtime like how the other solutions strained the 3D card. There is also the fact that we are not aware of the existing time left over in the panel code. No signs of runtime-related problems were observed in testing, so panel calculation seemed safe to use on a runtime standpoint. Still, the fact that we don’t know too many details is an unfortunate matter.

Overall, this solution was not used because the tradeoff of accuracy for maintainability was too high. For manual control, the theoretical accuracy of panel calculation was slightly better than the linear prediction solution. However, one must consider the amount of work needed to debug the fault found in the design. Also the inherent maintenance of panel calculation is much worse in comparison than the linear prediction solution by a significant factor. Even if linear prediction had some accuracy weaknesses that panel control improved on, it is not enough of an improvement to justify using panel calculation.
4.3 Polynomial Prediction

The last solution to be discussed here is a more complex form of prediction algorithm. Unlike linear prediction, which only used the current state and the previous field’s state as plot points, polynomial prediction would use several past states as reference to calculate an expected future state. An example of this is shown in figure X.X below. The future state itself would be 2 fields in advance, using the same reasoning as with linear prediction.

![Diagram showing polynomial prediction](image)

The appearance of this solution is better in every way to linear prediction. With polynomial prediction, it is roughly 33 percent less likely for the user to cause a situation where the prediction is incorrect due to misleading erratic movements. However, it is still possible to trick the solution into creating a video glitch, and thus it is less accurate than the precalculation solution.

The speed of polynomial prediction is quite slow. Even with the minimum of 3 plot points from which to predict from, the solution’s runtime is 1.413 ms, almost a full 3 times slower than linear prediction, and 50 percent slower than precalculation.

The maintainability is also worse off when compared to the solutions accepted. The code needed to handle the polynomial prediction is much harder to verify and debug, especially when compared to the relatively simple code needed for precalculation or...
linear prediction. The estimated time to code and verify this solution would be about 10 workdays, about double the work than that of linear prediction.

Overall, polynomial prediction was not a candidate for automatic control since it scored much worse than precalculation in appearance and maintainability, while only being about 50 percent faster. Regarding manual control, it is really only going to be seen by a show director during testing or a demo artist that knows exactly how the switcher will respond, so polynomial prediction’s slight gain in appearance is not very significant, and not worth the large losses in speed and maintainability in comparison to linear prediction.
5 Conclusions

Based on the specifications of the 3D card and the needs of the Backside Video feature, solutions were judged primarily on their appearance, speed, and maintainability. Because the feature had to look presentable to the intended audience to maintain our high standards of quality, appearance was the most important criterion to consider. The speed of the solution was only slightly less important than appearance, as is important due to the limited amount of runtime when working with real-time video feeds. The maintainability of the solution was the least important since it is not visible to the audience, but is still important to think about in the long term.

The solution implemented for automatic control would use the saved states that are part of an automatic control effect to precalculate the future state of the flykey. This solution took advantage of how precise automatic control would be, and was much more accurate than any other solution tested. It was a little slow, but its maintainability was quite good.

The solution implemented for manual control used the past state and the current state of the flykey to predict a future state using linear extrapolation. The appearance of this solution was good, although it was still possible for the user to make the backside glitch appear. The speed of the solution was fast, and the maintainability was okay.

There were other solutions that were passed over in favour of the current solutions. One alternative was to delay showing the current flykey state by 3 fields so that the video switch would be synchronized again. However, this solution was rejected since the delay would adversely affect many other features in the 3D card. The next alternative would only work for manual control, and involved having the panel notify the frame instead of the 3D card. It did not beat the linear prediction solution due to its appearance and maintainability. The last solution mentioned used several previous flykey states to predict a future state with polynomial extrapolation. It performed worse than precalculation in all three areas, and when compared to linear prediction, the slight increase in appearance was not worth the losses in speed and maintainability.
6 Recommendations

Future work on the Backside Video feature should use the weighted criteria discussed in the report. The differing requirements between automatic and manual flykey control are also important facts to note. The precalculation solution for automatic control uses a lot of runtime in its current implementation, so it is recommended to try and optimize this solution as much as possible.

Regarding manual control, the demo artists must be told about the visual glitch that exists in the linear prediction solution, so that they may work their presentations around it, or reassure clients that such glitches would not happen on air, since they would then be using automatic control.

Also, when the next version of hardware specifications are scheduled to be designed and printed, this report recommends adjusting the intermediary hardware between the 3D card and the frame so that the frame can be told to switch video sources without the 3-field delay. This change would allow the Backside Video to have a simple solution with perfect appearance and very fast speed.
References


Acknowledgements

I would like to acknowledge Steve Robinson for providing me with the SomeProduct Backside Video project, and for his help and recommendations on certain design problems. I acknowledge Jean-Francois Gagnon and Silvain Beiruit who helped me start the Backside Video project off, and for letting me bounce ideas off of them about possible solutions I could implement. I acknowledge SomeCompany Limited for letting me use their computers, video equipment, and software for my work.

I would also like to acknowledge the staff of the Software Engineering department, the staff of CECS at the University of Waterloo, and the staff of CECS at the University of Windsor for their own individual Work Report Guidelines, and finally Ed Papazian for his help over the work term.
Appendix A – Detailed Timing Figures

All figures are taken from the 3D card’s internal clock functionality, and are averages of 3 observations each. All figures are with the maximum of 8 flykeys being modified with the Backside Video feature on.

A.1 Automatic Control

Maximum runtime limit = 16.666 ms
Normal runtime with no solution = 10.144 ms

Table A-1. Runtime observations for automatic control.

<table>
<thead>
<tr>
<th></th>
<th>Precalculation</th>
<th>Linear Prediction</th>
<th>Polynomial Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall runtime (ms)</td>
<td>11.101</td>
<td>10.654</td>
<td>11.461</td>
</tr>
<tr>
<td>Solution-specific runtime (ms)</td>
<td>0.957</td>
<td>0.510</td>
<td>1.417</td>
</tr>
<tr>
<td>Percentage of maximum runtime used by the solution</td>
<td>5.74 %</td>
<td>3.06 %</td>
<td>8.50 %</td>
</tr>
</tbody>
</table>

A.2 Manual Control

Maximum runtime limit = 16.666 ms
Normal runtime with no solution = 9.309 ms

Table A-2. Runtime observations for manual control.

<table>
<thead>
<tr>
<th></th>
<th>Linear Prediction</th>
<th>Panel Calculation</th>
<th>Polynomial Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall runtime (ms)</td>
<td>9.825</td>
<td>8.998</td>
<td>10.722</td>
</tr>
<tr>
<td>Solution-specific runtime (ms)</td>
<td>0.516</td>
<td>-0.311</td>
<td>1.413</td>
</tr>
<tr>
<td>Percentage of maximum runtime used by the solution</td>
<td>3.09 %</td>
<td>-1.86 %</td>
<td>8.47 %</td>
</tr>
</tbody>
</table>