HELPING ARTISTS CREATE DIGITAL MUSIC VIDEOS USING SPACETIME CONSTRAINTS

A Thesis
In TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

by

John Middleton Rhoads

March 25, 2002

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

John Middleton Rhoads

Approved ____________________________________ Date_____________
Technical Advisor—Dave Brogan

Approved____ Date_______________
Technical Advisor—Bryan Pfaffenberger

Preface

As a double major in computer science and music my interest in this project is natural. I am quite interested in using my skills as a computer scientist to aid musicians in any way possible, and this project is an attempt to create a tool that will do just that. My secondary goal in taking on this project was to enable future students to more easily understand how to use spacetime constraints. The literature that currently exists on the subject is either organized poorly, or does not give enough detail to allow a skilled programmer to implement a spacetime constraints system without an extended amount of outside research. It is my goal to make this method accessible to more people in the hope that this powerful tool will be used more frequently.

I'd like to take this opportunity to thank some of the people that made this project possible. My TCC advisors, Bryan Pfaffenberger and Helen Benet-Goodman, were of great help with the writing and organization of this paper. Jason Zeibel did a great job of explaining some of the tougher physics concepts to me. I'd especially like to thank Dave Brogan, my technical advisor, for his help on the project. This project is the result of a year's worth of weekly meetings, and I really couldn't have gotten as far as I did on this project without his help.

Table of Contents

Glossary of Terms

Column Orthogonal Matrix – A matrix whose columns are linearly independent and whose lengths are one[8].

Column Vector – A matrix consisting of just a single column.

Constrained Optimization – The field of mathematics concerned with minimizing a function while still satisfying various restrictions.

Degrees of Freedom – Any of the minimum number of coordinates required to specify completely the motion of a mechanical system[11].

Diagonal Matrix – A square matrix in which all elements in the matrix that are not located on the diagonal are zero.

Feasible Solution – The point in a constrained optimization problem at which all of the constraints are satisfied; a feasible solution may or may not be optimal.

Finite Difference Formula – A means of approximating the derivative of a function by sampling it at different points in time.

Null Space – Set of inputs for which a function maps to zero.

Parse – To analyze or separate (input, for example) into more easily processed components[12].

Physical Simulation – Creating animation by using a physical model of the objects in a virtual scene.

Pseudo-Inverse ($\mathbf{A}^+$) – The psuedo-inverse of a matrix A is the matrix for which the following equality holds: $\mathbf{A}\,\mathbf{A}^+\,\mathbf{A} = \mathbf{A}$

Sequential Quadratic Programming (SQP) – A method for solving constrained optimization problems in which the objective function can be any second order equation.

Singular Matrix – A non-invertible matrix.

Singular Value Decomposition – A method used to find the pseudo-inverse of a matrix. It decomposes a matrix into the orthogonal matrices **U** and **V** and the diagonal matrix **W**. The matrix **A** is equal to $\mathbf{U}\ \mathbf{W}\ \mathbf{V}^{\mathbf{T}}$

Solution Space – The set of possible solutions to a system of equations.

Spacetime Constraints – A method used to apply constrained optimization to physically simulated computer animations.

Virtual World – A description of a computer graphics environment.

Table of Mathematical Variables

Variable	**Description**
$R(\mathbf{S})$	Objective Function(function to minimize)
$\mathbf{C}(\mathbf{S})$	Constraint Equations
$\mathbf{S}$	Vector Containing all state variables(y_i and f_i for example problem)
$\mathbf{H}$	Hessian Matrix(second derivative of objective function with respect to state variables)
$\mathbf{J}$	Jacobian Matrix(first derivative of constraint function with respect to state variables…the rows map to the different constraint equations and the columns map to the different state variables
$\mathbf{A}^+$	Pseudo-Inverse of the Matrix **A**
$\mathbf{U}$	First orthogonal matrix in Singular Value Decomposition
$\mathbf{W}$	Diagonal matrix in Singular Value Decomposition
$\mathbf{V}$	Second orthogonal matrix in Singular Value Decomposition
$\mathrm{diag}(1/w_j)$	The diagonal matrix formed by inverting the diagonal elements in **W**

$\frac{\partial R}{\partial \mathbf{S}}$	Column vector containing the first derivative of the objective function with respect to each state variable
$\hat{\mathbf{S}}$	Column vector containing the values to add to each respective element of the **S** vector in order to minimize the objective function after the first step of the SQP algorithm
$(\hat{\mathbf{S}} + \tilde{\mathbf{S}})$	Column vector containing the values to add to each element of the **S**’ vector in order to satisfy the constraints after the second step of the SQP algorithm.
S’	Column vector containing the intermediate **S** values obtained from the first step of the minimization algorithm(adding $\mathbf{S} + \hat{\mathbf{S}}$)
P_i	The ith physics constraint in the example problem
h	The time between samples(timesteps) of the physics constraints

Abstract

A common problem in computer music is that audiences are dissatisfied with sounds that are not performed. Audiences are used to seeing music performed, and since computer music has no performer, it is missing this key element of a satisfying concert. Computer musicians needs the tools to be able to add a visual component to their music.

Spacetime constraints is a method that combines physical simulation of objects and constrained optimization to create realistic digital animations. Physical simulation involves using principles from physics to determine where objects should be positioned in a scene, and constrained optimization is a field of mathematics that allows one to find a particular solution to a set of infinite equations based on some set of optimization criteria.

Spacetime constraints can be used to solve the aforementioned performance problem in computer music. Given a digital representation of a musical score, one can generate a digital animation that will appear to "play" the music specified by the score in a virtual (or computer animated) world. Each note in the score can be represented as a physics constraint in a spacetime constraints system. Using this information, a system can completely remove the technical problem of timing the animation to fit the music from the visual artists. This prevents a great hindrance to the creative process that previously existed.

While the project presented here is not yet fully functional, the content of this document should allow a researcher to fully implement a spacetime constraints system from scratch. The system successfully implements Spacetime Constraints on

a simple, one-dimensional example, but it still needs to be scaled to work in multiple dimensions for the project to work completely.

Chapter 1: Generating Digital Art

Perhaps the most significant obstacle to the effective creation of compelling digital art is that most artists lack the full technical knowledge necessary to successfully use the available technology. Using Spacetime Constraints, a method developed by Andrew Witkin and Michael Kass, I have begun to develop a system that automatically generates digital music videos from musical score files[2]. This allows artists to use their talents without worrying about technical details.

1.1 The Performer Problem in Computer Music

One of the major dilemmas that composers of computer music face is that audiences are frequently unwilling to accept music without a performer. When all of the sound is coming from a set of speakers with no visual cues to go with the music, it can lessen the effectiveness of a concert. For this reason, computer musicians are frequently interested in combining their talents with visual artists to produce a multimedia work of art.

Unfortunately, in the world of digital media, a number of obstacles prevent artists from fully utilizing the available technology to create their works. Many musicians may lack the visual artistic skill to be able to produce interesting, multimedia works on their own, so collaboration is usually necessary to produce satisfactory results. This doesn't solve the problem though. Lack of technical knowledge is the greatest impediment to the artist's ability to get the most out of the digital medium. Current work in digital media requires a combination of significant amounts of technical knowledge and artistic ability. This hinders the creative talents of those with a significant amount of artistic ability but virtually no technical knowledge.

Computer scientists have made a number of efforts to help alleviate this dependence on technical knowledge. Tools such as Maya, RenderMan, 3DStudioMax, and RTcmix are designed to allow artists to ignore the technical details so that they can focus on the areas of their own expertise. My system builds off of these previous tools to aid musicians and visual artists alike in attacking the performer problem.

1.2 Creating a Virtual Performer

The product presented here is a step toward providing a general tool that multimedia artists can use to make digital music videos. Given an RTcmix score file and a set of digital instruments and performers created with 3DStudioMax, my system, once fully functional, will be able to generate an animation that will show this set of virtual performers playing the predefined digital instruments. The goal is to allow artists to work on visual and aural concerns without worrying about the timing issues involved in creating such an animation. Specifically, the system will allow a user to specify a series of movable cannons and place them in the scene with the instruments. These cannons will fire projectiles that will strike the instruments in the scene, appearing to play the music. Based on the data in the score file, the system will determine where each cannon needs to be at any moment in time and when each cannon needs to fire in order to "play" the instruments scattered around the scene. This idea of using cannons to perform music is inspired by the 2001 SIGGRAPH movie *Pipe Dream*.[1] The movie is based on a similar idea, but it is a proprietary piece and does not create a general tool to assist artists in music visualization, as my project will do.

1.2.1 Physical Simulation Overview

Physical simulation is the idea that an animation can be generated by modeling the motion of all of the objects in the scene using physical principles. The laws of kinematics are used to plot where every object in a scene will be at any given moment in time. What this amounts to is a set of differential equations that model the motion of objects in the scene through time. These differential equations will have an infinite number of possible solutions, which demonstrates that objects can move through a system in an infinite number of physically valid ways. Typically, such systems will choose a solution for these equations by specifying a set of initial conditions, for example, initial velocity and position of an object. This will narrow the solution space (the set of possible answers to the equations) of the equations from an infinite number to a single solution. In this way, the particular way that the objects will move through the scene will be determined, and the values given in the solutions to the equations will specify the positions of the objects at any given time. The advantage of this method is that it provides a means for producing realistic motion without requiring the animator to create said motion by hand.

The problem with physical simulation is that it is difficult to control. It is hard for an animator to know exactly how he should set the initial conditions for the system in order to achieve a desired result. In the context of this system, if the animator wanted to make a cannon ball hit an instrument at a certain time it would be very difficult for the animator to know what initial conditions to use to create that motion. In 1988, Andrew Witkin and Michael Kass came up with spacetime constraints, an idea that would more intelligently choose which of the infinite solutions to the physical equations would be the best solution[2].

1.2.2 Spacetime Constraints Overview

Spacetime constraints is a powerful method that allows an animator a great deal of control over a physical simulation. Since an infinite number of physically valid solutions exist, the trick is to pick which set of solutions to use in the animation. Suppose a programmer could add more restrictions on the motion than just the laws of physics, for example, requiring an object to be in a certain position at a certain time; one could continue to add limits like this to the system until there is only one possible solution remaining. The above logic serves as the basis for spacetime constraints. In fact, if there are not enough restrictions to place on the system, the animator can specify optimization criteria, such as minimizing the total force of the system, and the method will provide the answer. The power of this method lies in the fact that one can now choose the "best" answer based on some set of defined criteria and not just any arbitrary answer.

The system I built depends heavily on spacetime constraints. A musical score is essentially a list of constraints of objects in a system. "This note has to be played at this time." As such, the score itself provides the much needed restrictions to limit the physical possibilities of the system. Since an RTcmix score file is text based, it is easy to integrate with a simulation system. Add to this the notion of minimizing the force used by the cannons that will be striking the instruments, and it becomes clear that this method will solve all of the timing issues involved in solving the performer problem in computer music. An artist can take an RTcmix score file involving an arbitrary number of instruments and create a video from this without having to worry about any of the timing issues involved.

1.3 Contents of Report

The rest of this paper will describe the details of the system I have been working on. Chapter two reviews previous work involving spacetime constraints and describes how my project fits into that paradigm. Chapter three explains how spacetime constraints works and how one could go about implementing a spacetime constraints system. Chapter four gives a detailed description of my specific application of spacetime constraints to the performer problem described above. The final chapter describes the results and future possibilities of the system.

Chapter 2: Previous Work In Spacetime Constraints

Using physical simulation to drive computer animation is not a new field. It is central to the idea of creating true virtual reality, and in principle removes the task of determining what "looks realistic" from the animator. The idea of creating mathematical models of physical phenomena has been around since antiquity, and it was natural that computer scientists would try to apply physical principles to computer graphics.

Spacetime constraints arose to facilitate the idea of physical simulation. The problem is that in a given complex system, there are generally an infinite number of physically possible things that could occur. Physical models of different systems are usually obtained by solving a series of differential equations. Consequently, there are an infinite number of physically possible motions that a system can have. Before Spacetime constraints, a particular solution would be obtained by specifying the initial conditions of the system. These would constrain the system to one possible set of equations. However, spacetime constraints allow a user to specify a set of desirable goals for the animation (have an object be at a certain position at a certain time, minimize energy consumption, etc.) and then have the system determine how to achieve those goals in a physically realistic manner. There are a number of different numerical methods that can achieve the desired constraints and optimization criteria.

In 1988, Andrew Witkin and Michael Kass introduced the idea of spacetime constraints in their paper of the same name[2]. Their hypothesis was that animations that adhere to strict physical principles will, by their very nature, look realistic. The key aspects of standard animation practices could be reproduced by physical principles. They applied this idea to Pixar's 1986 animation *Luxo, Jr.* which was possibly the most impressive computer animation at that time[3]. The film displayed a cartoon lamp "jumping" across the screen. Witkin and Kass found that the traditional animation techniques used to make the *Luxo, Jr.* movie could be reproduced by a simple optimization of physical principles. They used optimization criteria combined with physical principles to create a quite impressive animation of the jumping lamp. By combining physics with mathematical constraints such as "jump to this point" and "minimize energy consumption," they reproduced the most successful tricks of traditional animators. Since their method involved using a physical model that was evaluated through both space and time, they called their method "spacetime constraints." Witkin and Welch later extended this idea to non-rigid bodies[4] . They make a basic non-rigid body model and then build "attachment constraints" to combine these different base elements together to form complex, non-rigid bodies.

Spacetime constraints is an unusual research field, because it is not an idea with a number of potential improvements (the numerical methods used to solve the optimization problems inherent in the method have been around since before the beginning of computer graphics). Therefore, the research in spacetime constraints lies in applying these methods intelligently to new computer graphics problems. A discussion of some of the more significant applications is found below.

Rose et al used this method to create a physical model of the human body in a variety of contexts[5]. This allowed for reasonable computation time for a simulation of a human body with 44 degrees of freedom (the number of coordinates needed to fully specify the motion of an object). One advantage of the spacetime constraints paradigm is that it not only chooses a solution from an infinite number of solution sets, it chooses one that is optimal or near optimal for some set of criteria. This paper was an important proof of concept in that respect.

Michael Gleicher used this method to adapt motion capture data to similarly proportioned bodies of different sizes[6]. He used motion capture data from a person walking and carrying a box, and used this data as constraints on the motion for other humanoids for whom motion capture data is not available. He also applied this method to the morphing problem and successfully changed the size of a character walking as he moved.

Popovic et al. designed a dynamic interface for more intuitive control of physical simulation data[7]. They designed a system that allows a user to change certain aspects of the system, such as position or orientation, at any time during the simulation. The system would then recalculate the motion in real time, to allow the user an intuitive interactive control. This allows the user to specify the constraints for the motion rather than the programmer based constraints seen previously.

The project presented in this report is yet another instance of applying spacetime constraints to a new and interesting problem. Once the system is fully functional, an artist will be able to create digital music videos from previous compositions without having a significant amount of technical knowledge. Spacetime constraints are fundamental to my system, because without spacetime constraints, the level of automation required to remove the dependence on technical knowledge from the artist would not be possible. The next chapter describes in detail how spacetime constraints uses constrained optimization methods to solve physical problems.

Chapter 3: Spacetime Constraints

In order to understand my system, it is important to understand exactly how spacetime constraints work. This chapter details the theory behind the method and describes how the underlying theory is applied to physical principles.

3.1 Constrained Optimization Overview

Spacetime constraints is a specific application of the more general mathematical field of Constrained Optimization. Constrained Optimization problems involve two main parts: an objective function "R" and a series of constraint equations C_i. The idea is to make the objective function as small as possible, while making sure that all of the constraint equations are satisfied. There is generally the implied constraint that all variables have to be positive, to prevent trivial answers of negative infinity for any of the variables. For example, consider the following equations:

$$R(X_1,X_2) = X_1 + 2X_2 \qquad (1)$$

$$C_1 = X_1 + X_2 = 1 \qquad (2)$$

$$C_2 = X_1 \geq 0 \qquad (3)$$

$$C_3 = X_2 \geq 0 \qquad (4)$$

In this case X_2 would be equal to 0 and X_1 would be equal to 1, because this is the minimum value that R can have. Since constrained optimization problems generally involve many more than two variables, it is not common that such problems can be solved by inspection as we did with this simple problem. There are a number of different numerical methods that allow us to find solutions to more complex problems than the one shown above.

Typically, constrained optimization problems have many more variables than constraint functions. In this case an infinite number of solutions will exist, so the algorithm uses the objective function to determine the best of the infinite number of solutions. In the case where the number of linearly independent, constraint functions is equal to the number of variables, the constraint functions will specify a unique solution to the problem, and no minimization will occur. Effective algorithms to solve these types of problems must be able to handle both of the cases outlined above.

The methods used to solve constrained optimization problems involve algorithms that involve using first and second derivatives of the objective and constraint functions to find the optimal solution.

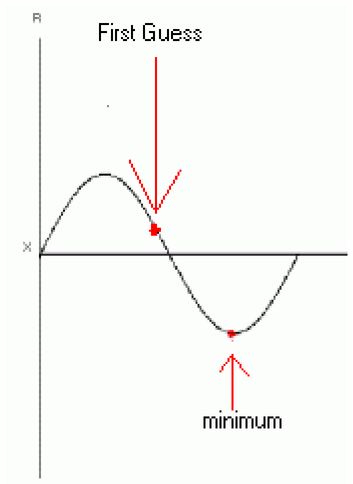

Figure 1: Constrained Optimization Visualization (drawn by author)

Figure 1 demonstrates a high-level view of what is going on in the typical algorithm. The system makes an initial guess for where the optimal solution will be. It will then use derivative information to walk along the solution curve to find a locally optimal solution. It is important to note that all these algorithms will do is find a locally optimal solution. If the left side of the curve in figure 1 were to continue in the same direction towards negative infinity, the algorithm would not detect this, given its initial guess. Each successive step in constrained optimization algorithms will always be closer to a minimum than the preceding step. As such, the algorithm will not be able to move up and to the left in figure 1 to arrive at the global minimum of negative infinity unless the initial guess is on this path.

The way that many constrained optimization algorithms use this information is by constructing matrices of derivative information for the various curves involved. The algorithm makes an initial guess at the optimal solution without regard to the constraints. It will then place this minimal solution on the curve by projecting the first guess onto the solution curve. This gives a first guess, called a feasible solution, which is a valid, possibly non optimal, solution that satisfies the constraints. The algorithms will then iterate through the two steps mentioned above using the matrix of derivative information; this effectively amounts to walking down along the solution curve until the algorithm reaches a local minimum. How this is done is described in more detail in the next section.

3.2 Sequential Quadratic Programming

The particular method of constrained optimization that I use for my system is Sequential Quadratic Programming (SQP). This section describes how SQP works in general and specifically how it can be applied to physics. Alternative descriptions of the method can be found in [2] and [9]. The description given here is based loosely on the one give in [2].

3.2.1 Overview of Method

Quadratic Programming is a subset of constrained optimization problems in which the objective function is an n variable quadratic function, and the constraint functions are n variable linear equations. In matrix notation the general form of the problem is:

$$R = (1/2)\,\mathbf{x}^{\mathbf{T}}\,\mathbf{A}\,\mathbf{x} + \mathbf{B}\,\mathbf{x} \tag{5}$$

$$\mathbf{C} = \mathbf{D}\,\mathbf{x} = \mathbf{e} \tag{6}$$

To facilitate discussion of this problem, we will use matrix notation and use the vector **S** to refer to all of the state variables used in the problem. In order to solve this problem, we need to determine two matrices of derivative information: The Hessian and the Jacobian. The Hessian is a second derivative matrix that allows the minimization step of the algorithm. The elements in the Hessian are given as:

$$H_{ij} = \frac{\partial^2 R}{\partial S_i \partial S_j} \tag{7}$$

for $0 \le i \le n$ and $0 \le j \le n$, where n is the number of state variables. And the elements of the Jacobian are:

$$J_{ij} = \frac{\partial C_i}{\partial S_j} \tag{8}$$

for $0 \le I \le m$ and $0 \le j \le n$, where n is the number of state variable and m is the number of constraint equations. These two matrices form an integral part of the SQP algorithm.

To solve the equations that constitute the SQP step (described below) we will need to know the inverses of the Hessian and Jacobian matrices. However, since there are usually more variables than constraint equations, these matrices are both likely to be singular (non-invertible) matrices. Luckily, we can compute the “pseudo-inverse” of these two matrices, and this will work just as well for our purposes.

The pseudo-inverse of a matrix ($\mathbf{A}^+$) is the matrix that satisfies the following equations:

$$\mathbf{A} = \mathbf{A}\mathbf{A}^+\mathbf{A} \text{ and} \tag{9}$$

$$\mathbf{A}^+ = \mathbf{A}^+\mathbf{A}\mathbf{A}^+ \tag{10}$$

To obtain the pseudo-inverse of the Hessian and Jacobian, I use the Singular Value Decomposition algorithm found in *Numerical Recipes in C* [8]. This algorithm separates the matrix (A) into three matrices, **U**, **W**, and **V** such that the following equation is satisfied:

$$\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^\mathbf{T} \tag{11}$$

U and V are both column orthogonal matrices (meaning that each column of the matrix is linearly independent and has a length of 1), and W is a diagonal matrix (the only non-zero values are along the diagonal). If the Matrix is square, the pseudo-inverse is:

$$\mathbf{A}^+ = \mathbf{V} * [\mathbf{diag}\ (\mathbf{1/w_j})] * \mathbf{U}^\mathbf{T} \tag{12}$$

If the matrix is not square, we can make it square by adding a row of zeros to the bottom until it becomes an N X N matrix. The only problem is that if an element of the diagonal matrix W (w_j) is 0, then $1/w_j$ is undefined. The algorithm actually works if you set 1/0 = 0 in this case [8]. Now that we know how to find the pseudo-inverses of the Hessian and Jacobian Matrices, we need to define a few more matrices before moving to the SQP step algorithm. For convenience, let **S** be the N X 1 column vector representing all of the independent variables in the problem. This vector can be initialized to any possible values. We will also need an N X 1 vector, $\frac{\partial R}{\partial \mathbf{S}}$, that will contain first derivative information for the objective function. The algorithm will also require two temporary N X 1 column vectors $\overset{\wedge}{\mathbf{S}}$ and $\overset{\sim}{\mathbf{S}}$ that store how much to alter **S** to obtain values for the next **S** in the algorithm.

The first part of the SQP algorithm involves minimizing R without considering the constraints. Solving the equation:

$$-\frac{\partial R}{\partial \mathbf{S}} = \mathbf{H} \cdot \overset{\wedge}{\mathbf{S}} \tag{13}$$

for $\overset{\wedge}{\mathbf{S}}$ will give an intermediate step that minimizes R without considering the constraints. To obtain this intermediate value, simply add the values in $\overset{\wedge}{\mathbf{S}}$ to the values in **S** to form the new vector **S'**. This vector will be used to obtain the values to the constraint equations **C(S')** used in the next equation.

It is important to note that there is no mystery involved in this first step. It is simply a Taylor series expansion. The equation:

$$f(x + x_0) = f'(x_0)(x - x_0) \tag{14}$$

represents the first order Taylor series expansion. If we let f(x) = $\frac{\partial R}{\partial \mathbf{S}}$ then

f '(x) = $\frac{\partial^2 R}{\partial \mathbf{S}^2}$. By the above Taylor equation this gives that:

$$f(x + x_0) = 0 = \frac{\partial R}{\partial \mathbf{S}} + \frac{\partial^2 R}{\partial \mathbf{S}^2}\overset{\wedge}{\mathbf{S}} \tag{15}$$

$\hat{\mathbf{S}}$ simply represents the change (x – x_0). Since $\frac{\partial^2 R}{\partial \mathbf{S}^2}$ is simply the Hessian matrix **H** that we've already defined, simply subtracting $\frac{\partial R}{\partial \mathbf{S}}$ from both sides of equation 15 gives equation 13.

From here, the algorithm solves the second equation:

$$-\mathbf{C}(\mathbf{S}') = \mathbf{J} \cdot (\hat{\mathbf{S}} + \tilde{\mathbf{S}}) \qquad (16)$$

for $\hat{\mathbf{S}} + \tilde{\mathbf{S}}$. This step is projecting the result $\mathbf{S} + \hat{\mathbf{S}}$ from the optimization step (equation 13) onto the null space (set of values for which a function is equal to zero) of the constraint Jacobian [2]. In other words, it is altering the minimized value from the first step by just enough to satisfy the constraints. The algorithm then sets **S** = **S'** + $\hat{\mathbf{S}} + \tilde{\mathbf{S}}$ and is now one step closer to an optimal solution. The algorithm continues performing these two steps, using the previous **S** values to find the next, until decreasing R any further would violate the constraints. The **C** vector used in the second equation is just the values from each of the constraints with respect to the values found in **S'** from the first step. When **C**(**S**) = **0**, the constraints are satisfied. The algorithm terminates when decreasing R any further requires **C(S)** ≠ 0. In other words, at this point, any decrease in R requires the algorithm to violate one of the constraints.

3.2.2 Application to Physics

The key insight that Witkin and Kass presented in their spacetime constraints paper [2] is that the laws of physics can be treated as constraints in a constrained optimization problem. In a physical system, one can make the total force in a system the objective function, and the physical laws the constraints, in an SQP problem. The advantage of this idea is that the algorithm will choose the physical system that minimizes the total force used in the system while still following the laws of physics. In addition, an arbitrary number of additional constraints can be added to give more control over the system. A user can specify that certain objects be located in certain positions at certain times, and if there is a physically valid way to achieve these goals, the algorithm will find it. To demonstrate more clearly how SQP works and how it can be applied to physical simulation, I will walk through a simple example. This example is similar to the example presented in the Witkin and Kass paper, except it only considers one dimension for simplicity [2].

Consider a ball falling off of a building. Suppose this ball were sentient and had a jetpack strapped to it that would allow it to slow its fall. Now suppose the ball is dropped from a height of 10 meters at time 0 and wants to end up at a height of 5 meters at time 2. This is an obviously ridiculous situation, but it illustrates how spacetime constraints works quite well.

Assume that the jetpack exerts a force f(t) upward and that it is capable of producing as much force as it wants instantaneously at any time. We want to minimize the force that it uses so we will declare our objective function to be:

$$R = f^2(t) \tag{17}$$

We square the force so that a force in either direction will still incur a positive cost in the objective function. Using Newton's second law of motion we know that the ball's mass (m) times it's acceleration ($\ddot{y}$) is equal to the total force exerted on the system.

In other words:

$$m\ddot{y} = f - mg \qquad (18)$$

Since this calculation is going to be performed by a computer, we need to sample the functions y(t) and f(t) that represent the force and position of the ball through time. In other words, f(t) can be represented as a series of n discrete samples f_i for $0 < i < n$. A similar sampling will be done for y(t). Since we have previously declared that **S** will contain all of the state variables needed for the problem, and these variables S_j will be the combination of the f_i and the y_i, we need to express the physics constraints in terms of these two variables. To obtain the time derivatives of y(t) in terms of the y_i we need to use the finite difference formula [2]. Let h represent the amount of time between samples (taken to be one in this example for simplicity). These give that:

$$\dot{y} = \frac{Y_{i+1} - Y_i}{h} \qquad (19)$$

and

$$\ddot{y} = \frac{Y_{i+2} - 2Y_{i+1} + Y_i}{h^2} \qquad (20)$$

If we now take these values and plug them into equation 18 we will get n – 2 physics constraints specified by:

$$P_i = m\frac{Y_{i+2} - 2Y_{i+1} + Y_i}{h^2} = f_i - mg \qquad (21)$$

or:

$$P_i = f_i - mg + m\frac{Y_{i+2} - 2Y_{i+1} + Y_i}{h^2} = 0 \qquad (22)$$

There are also two position constraints specified above that we have to take into account:

$Y_0 = 10$ and

$Y_2 = 5$

All that remains is to go through the setup of the Hessian and Jacobian and then solve the equations described above. Assuming a sampling rate of 1 sample per second (an extremely impractical thing to do, but it is done here to illustrate the method) and assuming the indices for S_j are in the following order: $\{y_0, y_1, y_2, f_0, f_1, f_2\}$ gives the following Hessian:

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	2	0	0
0	0	0	0	2	0
0	0	0	0	0	2

and Jacobian (with boundary constraints given last):

$-m/h^2$	$2m/h^2$	$-m/h^2$	0	1	0
0	$-m/h^2$	m/h^2	0	0	1
1	0	0	0	0	0
0	0	1	0	0	0

The second line in this Jacobian is the solution to a tricky implementation problem. Since we are using the finite difference formula to express acceleration ($\ddot{y}$) in terms of position (y), we have to sample the positions two steps ahead of our current step in order to know the acceleration. In our example, this means the physics constraint at time 1 will be: $P_1 = f_1 - mg + m\frac{Y_3 - 2Y_2 + Y_1}{h^2} = 0$. However, there is no y_3 in our example, since we only are looking at three time steps: 0, 1, and 2. To handle this problem, simply set $y_3 = y_2$, which gives $P_1 = f_1 - mg + m\frac{-Y_2 + Y_1}{h^2} = 0$.

The gradient vector needed in equation 13 is then:

$$\frac{\partial R}{\partial \mathbf{S}_j} = 2 * f_j \text{ or } 0 \text{ if } S_j = y_i$$

The solution then comes from solving equations 13 and 16 using the pseudo-inverses of the Hessian and Jacobian described above for $\hat{\mathbf{S}}$ and $\tilde{\mathbf{S}}$. Then set the new $\mathbf{S} = \mathbf{S}' + \hat{\mathbf{S}} + \tilde{\mathbf{S}}$ and repeatedly solve equations 13 and 16 until there is no improvement. At this point, the **S** vector will contain the location of the ball at times 0, 1, and 2 and how much force the jetpack will use at times 0, 1, and 2 in order to minimize the total force while following the constraints. For more examples of how spacetime constraints work and how SQP can be used consult [2] and [9].

3.2.3 Difference from Original Spacetime Constraints Paper

The above logic actually differs from Witkin and Kass in that there is a different treatment given of the finite difference formula. Witkin and Kass give the finite difference formula as follows:

$$\dot{y} = \frac{Y_i - Y_{i-1}}{h}$$

and

$$\ddot{y} = \frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2}$$

which are both perfectly valid ways to express the finite differences formulas. The problem with this is that in physics, instantaneous velocity is defined as:

$$\dot{y} = \frac{Y_{i+1} - Y_i}{h}$$

as I described in the previous section. So while both descriptions are accurate depictions of the finite difference formula (algebra doesn't care where you start the indexing), the Witkin and Kass example will result in a set of physics constraints that don't make logical sense. Using their description, one arrives at the following description of the physics constraints:

$$P_i = f_i + mg - m\frac{Y_{i+1} - 2Y_i + Y_{i-1}}{h^2} = 0$$

This is again just the information I have given in the previous section with altered indexing. However, this equation implies that applying a force at time I will result in a change in the position of an object at time i –1. Since it does not make intuitive sense that a force applied in the present can effect the past, and since it is inconsistent with the description of instantaneous velocity given in physics texts, I decided that the indexing scheme I used makes more sense.

One other important difference between the Witkin and Kass paper and my own lies is that we have different sign representations. I start from the equation $m\ddot{y} = f - mg$, and Witkin and Kass use the equation $m\ddot{y} = f + mg$. They are both technically good representations of the physics involved, but I am just explicitly showing that gravity acts down on the object. This is usually the way you see gravity dealt with in physics texts because constants, such as g, are generally considered positive.

Chapter 4: Creating Digital Music Videos From Musical Scores

Now that we have a better understanding of spacetime constraints, we can examine how the method can be used to build the system that addresses the “performer problem” in computer music. This chapter describes the syntax used in the RTcmix score files that will allow users to specify where objects are located in physical space. It then details how spacetime constraints is applied to the system that will automatically generate the timing data used in the system.

4.1 RTcmix .score File Description

In order to talk about how the mapping between audio and visual instruments will work, one needs to understand a few basic RTcmix commands. The basic design of a score in RTcmix involves loading an arbitrary number of instruments, and then calling functions with specific parameters that specify how the note will play. For example the following could be the relevant structure of a typical score file:

```
load( “FMINST”);
…..
FMINST($start, $dur, $amp, $freq, $mod_freq, $spread);
```

The FMINST call specifies that RTcmix should play a note using FM synthesis. (A common computer music technique that is not really relevant to this discussion, for more information consult [10]). The parameters to the function specify key information about the note such as the time the note should sound, its duration, its pitch, and its amplitude. This brings us to how virtual instruments can be specified in a score file. The function “make_virtual_inst(…)” allows the user to specify where the instrument will be located in the virtual world. The statement shown below can be inserted after the load(Instrument Name) call to associate that instrument with the given sound:

```
make_virtual_inst("FMINST",-1,-1,1.0, 1.0, 1.0);
```

This call specifies that for all notes of type "FMINST" that instrument is located at position 1,1,1 in the virtual world. The –1 flags say that all instances of "FMINST" are located at that position. If the call looked like:

```
make_virtual_inst("FMINST", 3, 440, 1.0, 1.0, 1.0);
```

then this call would say that whenever the third parameter of FMINST (in this case $freq) is equal to 440 (which specifies the A above middle C) the location of that note is 1,1,1 in the virtual world. This allows the user to specify the exact location of different notes on an instrument. For example, if the virtual instrument were a piano, each of the keys would be located in different positions, so the user would want to specify different locations for each key. Luckily, when the animator is creating the visual representation of the instrument, the computer graphics program he is using will usually involve placing objects at different coordinates in space. This information can just be taken from the animation and put in the score file, so there is no real work associated with setting up the locations of the virtual instruments.

4.2 Application of Constraints to System

The idea is to have the system play the virtual instruments specified above through the use of movable cannons. These cannons can translate and rotate along a track in the virtual world, and they will fire balls at the appropriate times to play the notes specified by the score file. The fact that there will be fewer ball launchers than notes on the instruments will necessitate the use of the optimization power of spacetime constraints. Since there will not be enough launchers to play every note without moving, the cannons will have to move along their tracks at appropriate times to take care of playing all of the notes. If there was no notion of minimizing the force these cannons use to achieve this motion, then there would be a great deal of extraneous motion along the tracks which would look illogical to the person watching the animation.

The system fits quite well into the spacetime constraints paradigm. Since RTcmix scores contain start time information for each note, each note will know at exactly what time it should sound. Or, more usefully, this contains the information for exactly when and where a ball should strike a certain virtual instrument to play in synch with the music. The laws of physics and the score file itself therefore contain all of the information needed to produce a convincing animation.

So we now know that the system can conceptually handle all of the timing issues necessary to create the animation. The question now is how does the system actually use this data to create the animation. The system begins with the specification of virtual instruments in the score file. The spacetime constraints system will then parse (read and understand) the file, and run the SQP algorithm to determine where each object should be located. The position data for each object in the animation will be contained as position variables in the **S** vector, which is the ultimate result of the SQP step. This position data can then be used by the computer graphics program to specify where each object in the virtual world should be at each point in time. And the result, a digital music video that appears to "play" the music specified in the score file. The "performance problem" in computer music can be solved without the artists involved having too great a dependence on technical knowledge.

Chapter 5: Conclusion

Now that I have given a complete description of the system as it stands, this chapter summarizes the ideas of the system, interprets the results, and gives recommendations for the future of the project.

5.1 Summary

Once completed, the system presented here will allow artists to create digital music videos without worrying about technical details. The system uses Spacetime Constraints to remove all of the timing issues involved in making the movie. Notes in a score file represent position constraints for the objects that "play" the virtual instruments. Minimizing the total force used by the various cannons used to play the music will produce a visually satisfying result once the system is fully functional. The main challenge remaining is to simply scale the problem to three dimensions, and use this information to implement the system to automatically generate the timing information needed to play the instruments in the virtual world.

5.2 Interpretation

Even though the system does not currently work, I am confident that it will be functional within the next few weeks. It is my hope that, if nothing else, the presentation of spacetime constraints presented in chapter three will be easy to read and understand by future computer science researchers. I feel that spacetime constraints is a very powerful tool that is currently underused in animation. One possible reason for this problem is that it is a very difficult method to implement. My hope is that the presentation of the method in this report will clear up some of the confusion and ambiguity that exists in some of the earlier spacetime constraints papers.

Naturally it is my hope that artists will be interested in using this system. One possible criticism of this project is that the application is too narrow. The system will produce compelling visualization of musical pieces with a lot rhythmic complexity, but more subtle aspects of music that computer musicians like to explore will not really be captured by the visuals of this system. It is also limited by its dependence on percussive instruments. In addition it still involves collaboration between visual and aural artists, which can frequently be hard to manage. However, I believe that once the system is working, it will have significant practical use for today's artists.

5.3 Recommendations

Clearly the most obvious shortcoming of this project is that it is not yet fully functional. As mentioned above, the SQP algorithm is working well, i.e. the system will minimize the objective function and still satisfy the constraints given. However, the description of the physics of the system is not quite rigorous enough. Although the equations given to the system are satisfied, the answer that the SQP system produces is not physically accurate. Describing the physics of the system accurately is therefore a top priority.

There are several other tasks that could be done to make the system more useful for artists. Ideally, interested programmers could add plugins to 3D Studio Max, Maya, and Renderman that would take the data specified by the system and map these numbers to specific objects in the 3D world. Having an easy-to-use system associated with these popular rendering tools should make the system much easier to use than it will be initially. A graphical interface for mapping the relationships between musical events and events in the virtual world would also make the system much easier to use. Currently RTcmix developers are working on graphical user interfaces for RTcmix, so an ultimate goal could be to integrate RTcmix with one of the graphical programs listed above to make the system that much easier to use.

Another interesting addition that researchers could look into would be extending the system to include more complex physical models than simply balls moving around in the scene. The physical objects modeled in this system can be arbitrarily complex. Integrating the system with a general purpose physics library could make the imagination of the artist the only limitation on the kinds of innovative animations they can create. This is the ultimate goal of a project of this type, so ideally a future researcher will be interested in tackling this problem. The potential for future work in this area is really quite extensive and hopefully other researchers will be as interested in this subject as I am.

Works Cited

[1] Wayne Lytle. *Pipe Dream*. 2001. film.

[2] Andrew Witkin and Michael Kass. *Spacetime Constraints*. Proc. SIGGRAPH,1988.

[3] Pixar. *Luxo, Jr.,* 1986. film.

[4] Andrew Witkin and William Welch. *Fast Animation and Control of Nonrigid Structures*. Proc. SIGGRAPH, 1990.

[5] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael Cohen. *Efficient Generation of Motion Transitions Using Spacetime Constraints*. Proc. SIGGRAPH, pages 147-154, 1996.

[6] Michael Gleicher. *Retargeting motion to new characters*. Proc. 25th annual conference on computer graphics, pages 33-42, 1998.

[7] Jovan Popovic, Steven M. Seitz, Michael Erdmann, Zoran Popovic, and Andrew Witkin. *Interactive manipulation of rigid body simulations*. Proc. conference on Computer graphics, Pages 209 – 217, 2000.

[8] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press. 1988-1992.

[9] Roger Fletcher. *Practical Methods of Optimization. Second Edition*. John Wiley & Sons. 2000.

[10] Charles Dodge and Thomas A. Jerse. *Computer Music: Synthesis, Composition, and Performance*. *Second Edition.* Schirmer publishing. 1997.

[11] *Dictionary.com*. April 23, 2002. Lexico, LLC. http://www.dictionary.com/search?q=degrees%20of%20freedom

[12] *Dictionary.com*. April 23, 2002. Lexico, LLC. http://www.dictionary.com/search?q=parse

Bibliography

Dictionary.com. April 23, 2002. Lexico, LLC.

Dodge, Charles and Jerse, Thomas A.. *Computer Music: Synthesis, Composition, and Performance*. *Second Edition.* Schirmer Publishing. 1997.

Fletcher, Roger. *Practical Methods of Optimization. Second Edition*. John Wiley & Sons. 2000.

Gleicher, Michael. *Retargeting motion to new characters*. Proc. 25th annual conference on computer graphics, pages 33-42, 1998.

Hecker, Chris. "Physics, The Next Frontier." *Game Developer.* November 1996.

Lunn, Mary. *A First Course in Mechanics*. Oxford University Press. 1991.

Lytle, Wayne. *Pipe Dream*. 2001. film.

Pixar. *Luxo, Jr.,* 1986. film.

Popovic, Jovan, Seitz, Steven M., Erdmann, Michael, Popovic, Zoran, and Witkin, Andrew. *Interactive manipulation of rigid body simulations*. Proc. conference on Computer graphics, Pages 209 – 217, 2000.

Press, William H., Teukolsky, Saul A., Vetterling, William T., Flannery, Brian P.. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press. 1988-1992.

Rose, Charles, Guenter, Brian, Bodenheimer, Bobby, and Cohen, Michael. *Efficient Generation of Motion Transitions Using Spacetime Constraints*. Proc. SIGGRAPH, pages 147-154, 1996.

Winzell, Par. *An Implementation of the Spacetime Constraints Approach to the Synthesis of Realistic Motion*. Master's Thesis. Linköping Institute of Technology, Sweden. 1998.

Witkin, Andrew and Baraff, David. *Differential Equation Basics*. Carnegie Mellon University. 1997.

Witkin, Andrew and Kass, Michael. *Spacetime Constraints*. Proc. SIGGRAPH,1988.

Witkin, Andrew and Welch, William. *Fast Animation and Control of Nonrigid Structures*. Proc. SIGGRAPH, 1990.