EFFICIENT ANIMATION TECHNIQUES BALANCING BOTH USER CONTROL AND PHYSICAL REALISM

Zicheng Liu

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

November 1996

To my wife and my parents

Abstract

Specifying the motion of an animated linked figure such that it achieves given tasks (e.g., throwing a ball into a basket) and performing the tasks in a realistic fashion (e.g., gracefully, and following physical laws such as gravity) has been an elusive goal for computer animators. The *spacetime constraints* paradigm has been shown to be a valuable approach to this problem, but it suffers from the computational complexity growth as creatures and tasks approach those one would like to animate. There are two sources which contribute to the complexity problem: one is the symbolic processing of the animator's constraints and the objective functions derived from the physical models and the second lies in the numerical optimization phase. This thesis reports on work to enhance the spacetime constraints techniques both symbolically and numerically to significantly speed up computations.

Our first contribution is to develop a new symbolic interface with a recursive evaluation scheme so that the time required for gradient computation which is needed by the numerical optimization is reduced from exponential growth to the optimal quadratic growth. Furthermore, with the new symbolic method, a language developed for the symbolic expressions can be easily input, thus it provides a more convenient interface.

Secondly, we develop a keyframe optimization system which allows the user to specify a few keyframes while letting the computer determine the speed and timing, parameters which may be less intuitive for the animator, by using optimization methods. The novelty of this approach is that the user-specified keyframes are used both to provide control over the motion and to reduce the complexity of the optimization problem. This method has the advantage of providing much of the user control of the

traditional animation system while providing the physical realism of the optimization based system. Due to the reduced complexity of the optimization problem, the computation time is nearly interactive for complex figures.

Finally we develop a hierarchical scheme to solve the nonlinear variational problem arising in the spacetime constraints formulation by using wavelets. In this scheme, the functions through time of the generalized degrees of freedom are reformulated in a hierarchical wavelet representation. This provides a means to automatically add detailed motion only where it is required, thus minimizing the number of unknowns. In addition, the optimization problem is better conditioned so that the convergence is faster.

Acknowledgements

First of all, I would like to thank my advisor, Professor Michael Cohen, for introducing me to the area of computer animation, and advising my thesis research even after he left Princeton. It is a pleasure and honor to work with him. He is both a great researcher and a nice person to work with. Without his inspiration and constant encouragement, this thesis would have been impossible.

I would like to thank Professor Pat Hanrahan for teaching me the basics of computer graphics and fascinating me with beautiful pictures and animations.

I would like to thank my thesis readers, Professor David Dobkin and Professor Andrew Yao, for their time and effort in reading my thesis and their helpful comments. Thanks to Professor Dobkin for helping me with the financial support after my advisor left Princeton. Thanks to Professor Andrew Yao for teaching me the techniques in theoretical computer science. His class is always inspiring.

I would like to thank Professor Jin-Yi Cai for supporting me during my first two years at Princeton.

I would like to thank Professor Ding-Zhu Du for encouraging me to pursue a PhD degree in computer science and recommanding me to Princeton University.

Many thanks to Steven Gortler for his help in clarifying the wavelet work used in the hierarchical spacetime control. I would also like to thank Chuck Rose for writing an interface for the hierarchical spacetime control system which is part of the thesis.

Thanks to Richard Alpert for his friendship and help to make my life in Princeton easier and a lot more fun. I would like to thank my officemate Stefanos Damianakis for teaching me the basics of computer systems and how to use X Window, UNIX, EMACS, and other hacking tools. His help made it possible for me to go through the difficult first months at Princeton without it being a disaster. I feel so fortunate to be his officemate.

Finally I would like to thank my wife Xue Li for her love and support.

This work was partially supported by National Science Foundation Grant CCR-9296146.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Animation originated from artistry. It is the process in which artists bring life to their *virtual* figures by adding motion. Walt Disney's Donald Duck and Mickey Mouse are two well known examples of animated figures. These figures are said to be virtual because they exist only in the form of images. When speaking of animation, people usually think of motion of lifelike figures, but animation also covers the motion of lifeless objects such as flowing water or a rolling ball. Even the change of a camera position can be considered animation since it generates a visual change.

With the development of computer technology in the past decade, *computer animation* (or simply animation) has become an active field for computer scientists. Techniques and tools have been developed in an attempt to automate the process of animation and alleviate some of the work of the artists.

Computer animation is widely used in entertainment, education, training and scientific research. Movies have increased their use of computer animation in creating special effects. The cartoon industry is utilizing animation software packages to produce more realistic and complex motion for their characters. The knowledge and ability to use animation software packages is becoming more important for a professional animator's work. In education, people use animation to demonstrate scientific concepts and visualize processes which can not be seen in the real world. Flight and driving simulators are used to safely train beginners for situations which otherwise could not be taught in the real world. Molecular simulation techniques have allowed scientists to view molecular interactions and to perform experiments in a virtual molecular world.

The work presented in this thesis, however, will focus primarily on animating linked figures representing animate creatures.

1.1 Conventional Animation and Computer Assistance

Conventional animation is a fairly routine process. Generally speaking, it consists of the following steps as described in [9, 22]. First, a story board is laid out, consisting of a sequence of skeletal drawings which outlines the whole story. Each sketch in the sequence represents a segment in the whole story. Then for each segment, certain *key frames* are drawn. Each key frame corresponds to a scene where the characters or other objects are at their extreme or characteristic positions so that the intermediate positions can be inferred.

For instance, if a segment represents the motion of a bouncing ball. Then those scenes where the ball is on the ground or at its maximal height can be used as key frames, and the other intermediate frames can be inferred from these positions based on the fact that the ball is in between these extreme positions.

Then the intermediate frames are filled in by the animator or an assistant (called inbetweening). Because of the use of key frames and inbetweening, this type of method is called *keyframe animation*.

Many of the tasks in conventional animation can be performed or assisted by the computer. Computers are good for drawing objects with regular geometry such as buildings, desks, pipes, etc. Computers can also be used for image composition such as blending the background and foreground.

Inbetweening is another task in which the computer can provide assistance. This task is particularly interesting for the topic of this thesis since it affects the appearance of the motion. The key frames only determine the skeleton of the motion, while details of the motion largely depend on the inbetweening. Inbetweening is essentially a multidimension *interpolation* problem.

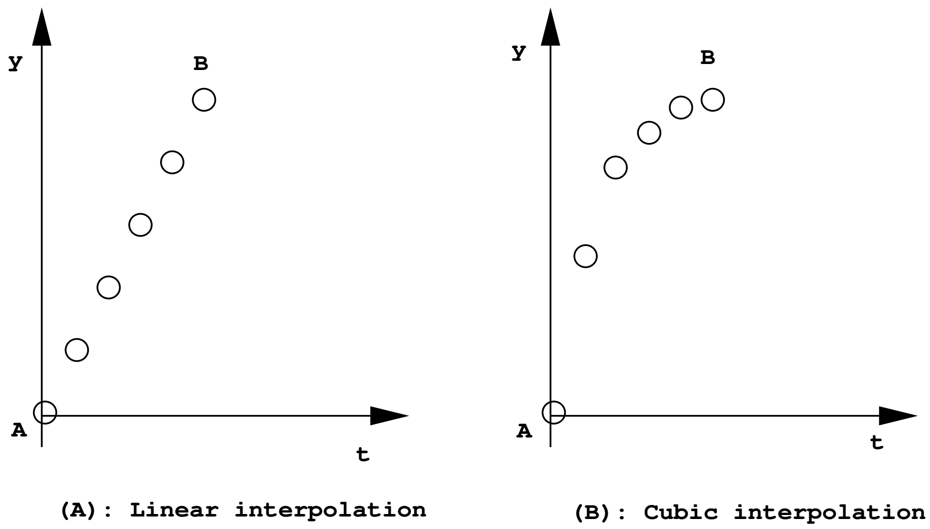


Figure 1: The bouncing ball trajectories under linear and cubic interpolation

1.2 Interpolation

To illustrate the idea of interpolation, let's consider the bouncing ball as an example. As in Figure 1, assuming the ball is on the floor $A = (x_0, y_0)$ at time time 0 and bounced to the highest position $B = (x_1, y_1)$ at time 1, we want to determine the position of the ball at any intermediate time t where $0 < t < 1$. The simplest solution would be to use a linear function to interpolate the two points A and B, that is,

$$\begin{aligned} x &= x_0 + t * (x_1 - x_0) \\ y &= y_0 + t * (y_1 - y_0) \end{aligned} \tag{1}$$

Figure 1 (A) shows the positions of the ball at time 0, 0.2, 0.4, 0.6, 0.8, and 1.0, respectively. The ball goes straight from A to B with a uniform velocity. The resulting motion does show the ball going upward after bouncing, but doesn't reflect the effect of the gravity, and as a result, the motion doesn't look realistic.

A remedy for this problem would be to take into consideration the velocities when performing the interpolation. We know that the vertical velocity of the ball is zero when the ball is at its highest position B. So we can try to find a function $y = y(t)$ such that $\dot{y}(t)$ (the derivative) is 0 when $t = 1$, as well as $y(0) = y_0$ and $y(1) = y_1$. It turns out that given $y(0)$, $y(1)$, $\dot{y}(0)$ and $\dot{y}(1)$, there is a unique cubic polynomial $y = a_0 + a_1 t + a_2 t^2 + a_3 t^3$ which satisfies all of the 4 conditions. (Such polynomials are called Hermite polynomials, and its construction is described in Section 2.8.1).

In this example, we do not know $\dot{y}(0)$, but given the height the ball flies plus gravity this can also be solved for. Figure 1 (B) shows the positions of ball at time 0, 0.2, 0.4, 0.6, 0.8 and 1.0 by using the cubic polynomial as the interpolation function. We can see that the motion looks more realistic.

There are still two problems in general with the cubic polynomial interpolation. First the user has to supply the velocities at the key frames which may not be available. Second, the time interval between two consecutive keyframes has to be small so that the motion in between the keyframes does not vary too much. For example, if we choose the two key frames where the ball is at its highest positions and skip the one where the ball is on the floor, then in the interpolated motion, the ball would fly straight forward horizontally and there would be no bouncing motion at all. In practice, more key frames means more work for the animator, since he (she) has to determine the configurations and perhaps velocities of the objects or figures at these key frames which may require a lot of trial and error.

1.3 Linked Figures

A linked figure consists of rigid links connected by joints. Human beings and animals are examples of linked figures. The generalized *degrees of freedom* of a linked figure are the independent parameters which completely determine its configuration. For example, a planar three link arm as shown in Figure 2 (A) has 3 degrees of freedom, since the 3 angles θ_0, θ_1, and θ_2 completely determine the configuration of the linkage.

For linked figures, the task of determining the configuration often involves determining the rotational angles given a desired location for the end of the linkage. Such a problem is called *inverse kinematics* problem. For example, suppose we want to determine the configuration of the three link arm such that the end effector reaches point A (see Figure 2 (B)). We need to find the three angles θ_0, θ_1, and θ_2 so that the coordinate (x,y) of the end effector is equal to that of A. To solve such an inverse kinematics problem, we can write down the equations which relate end effector positions to rotational angles. These equations are called *kinematics equations* and are described in Section 2.4.

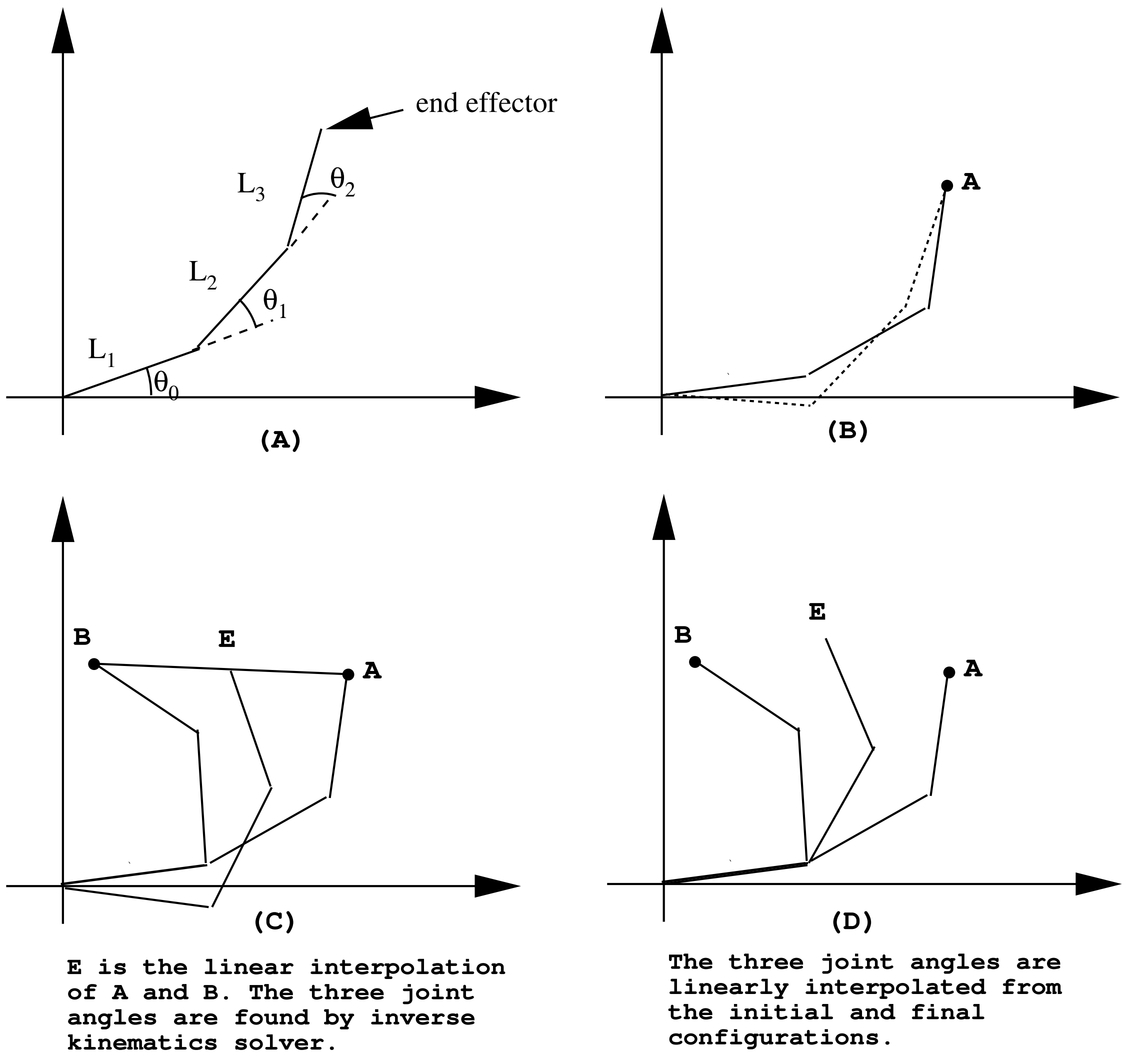


Figure 2: Inverse kinematics and interpolation of linked figures

For this example, we can write down equations of the form $x = x(\theta_0, \theta_1, \theta_2)$ and $y = y(\theta_0, \theta_1, \theta_2)$. Then the problem becomes solving the system of nonlinear equations. In general, there are more unknowns than the number of equations, so that the solution is not unique (Figure 2 (B) shows two solutions). In general, an arbitrary solution may not be what we want. For example, if we are simulating a human being's arm, we may want to put some limits on the three joint angles (such as the elbow should not bend backward). Joint limit constraints are inequality constraints, thus, the inverse kinematics problem with joint limit constraints becomes a system of nonlinear equations and linear inequalities.

Zhao and Badler [57] used Rosen's projected gradient method [21] to solve the system of nonliner equations and linear inequalities. They showed that their inverse kinematics solver achieves nearly interactive speed (in a few seconds) for fairly large figures (with upto 30 degree of freedoms). Their techniques have been used in commercial animation systems such as those from SoftImage, Alias, and Wavefront.

For linked figures, interpolation becomes more complex too. As an example, consider a planar three link arm moving an object from position A to position B (see Figure 2 (C)). In the first approach, to find the configuration at a given time point, one can find the position of the end effector by linear interpolation. Then one can apply the inverse kinematics solver to find the joint angles (see Figure 2 (C)). One problem with this approach is that the joint angles may not be a continuous function of time. To see why, notice that the inverse kinematics system has multiple solutions so that the solutions found by the inverse kinematics solver at consecutive time points may not be consistent. For example, in Figure 2 (B), if at position A, the inverse kinematics solver gives the solution as drawn with the solid line, but at the next position which is close to A, gives a solution close to the other one as drawn with the dashed line, then the two solutions are not consistent. As a result of such inconsistent solutions, the resulting motion would have sudden changes (*jumps*). Another problem is that in some cases, a realistic trajectory of the end effector may be difficult to find. For example, if instead we want the three link arm to throw a ball into a basket, it is not clear how to find a trajectory of the end effector first so that the motion looks realistic.

Another approach for interpolation would be to interpolate the joint angles directly. One can find a few key configurations of the motion as key frames, then interpolate the joint angles to obtain the motion. (In Figure 2 (D), the joint angles of the middle configuration are linearly interpolated from the joints of the initial and final configurations). This method has the advantage that the angles will be continuous, thus the motion has no jumps. But, the difficult issues remain of how to achieve motion realism as was discussed in Section 1.2.

1.4 Dynamic Simulation

Not surprisingly, dynamic simulation is an effective approach to achieve motion realism. For the bouncing ball example, the motion of the ball after bouncing can be easily computed by using Newton's Law. In general, a simulation problem is an *initial value problem* ([19]) where the differential equations describe the physical laws and the initial configuration and velocities are the initial conditions. The solution to the initial value problem is the motion. The advantage of this approach is that the resulting motion looks realistic since the motion obeys physical laws. So far, this approach has been used to successfully creat motions of chains, bowling, pool, waves, snakes, automobiles, etc [2, 3, 6, 27, 30, 31, 36, 41, 44, 55, 10, 42].

One problem with the simulation approach is that the user has no *control* over the motion once the initial conditions are specified. Therefore it is difficult to create motions with some desired goals. For example, one can use dynamic simulation to generate a physically correct rotation motion of a figure skater in the air if the initial (at takeoff) velocities are given (both rotational velocity and the velocity of center of gravity). But to guarantee that the skater completes the desired number of rotations in the air, the animator has to find the correct initial velocities. The mathematical model of such a control problem is a *boundary-value problem* [19] which usually requires more elaborate solution methods than the forward simulation approach for solving initial value problems [48].

For animate linked figures, there is another problem which the simulation approach does not address, namely, how to determine the internal forces (which are functions over time). Linked figures are usually autonomous, that is, they can exert forces or torques. The internal forces affect the appearance of the motion and determine whether the goal is reached or not. Thus, for linked figures, the motion control problem becomes finding the internal forces so that the goal is reached and the motion looks realistic.

How to gain control over a dynamic system has been the focus of many researchers including the work reported in this thesis. In particular, this thesis concentrates on the problem of how to generate goal directed motion for linked figures. Before we describe our work, let's briefly review some of the work which has been done in this direction.

1.5 Control

One approach to gain control over a dynamic system is to design *controllers*. A controller is a specification of forces or torques as functions of the positions and velocities (called the *state*) of an object or figure. Given a controller and the initial state, the motion can be computed by integrating over time. For example, consider a particle with unit mass at initial position $x = x_0$ and velocity $\dot{x} = 0$. We want to apply a force f to bring it to rest at position $x = 0$. Intuitively, we would like to apply a force whose direction is opposite to the direction of the displacement and velocity, which leads to a controller of the following form:

$$f = -k_1\theta - k_2\dot{\theta} \tag{2}$$

where k_1 and k_2 are positive constants to be determined. Now the motion equation becomes

$$\ddot{x} = -k_1\theta - k_2\dot{\theta} \tag{3}$$

We can then solve this differential equation (the solution contains k_1 and k_2 as parameters) and see if we can find the desired k_1 and k_2 so that the goal is satisfied. A detailed discussion is omitted here. We just want to point out that if we choose k_1 an k_2 so that $k_2^2 > 4k_1$, the solution will have the form:

$$x = C_1 e^{-w_1 t} + C_2 e^{-w_2 t}$$

where $C_, C_2, w_1, w_2$ are some constants with $w_1 > 0$, $w_2 > 0$. Clearly both x and $\dot{x}$ tend to 0 as t goes to infinity. Although in practice, t can never become infinity, but when t is big enough, both x and $\dot{x}$ will become very small so that they are acceptable solutions.

Brotman and Netravali [7] use an *optimal control* formulation to define their controllers, and the motion is the solutions to a sequence of optimal control problems.

Raibert and his colleagues have designed controllers to produce running motions of a variety of simulated and real creatures [51]. More recently, Hodgins and her colleagues have developed controllers to produce a variety of athletic motions [28]. Controllers are used to regulate speed and gait, maintain balance, etc. However, each new motion (e.g. a different number of rotations in diving) would require designing a new controller and the motion has to be tuned manually in order to create the desired behavior. Both the design of a controller and the fine-tuning are tedious tasks. Systematic ways need to be developed for such method to be useful in practice.

To automate the process of finding controllers, techniques have been developed to use parameterized controllers to represent motion while the parameters are found through an optimization process [54, 43]. Ngo and Marks use a stimulus-response controller to represent motion with an array of stimulus-response (SR) parameters as unknowns [54] . Van de Panne and Fiume use a sensor-actuator network with weighted connections as the controller to represent motion with the weights as unknown parameters [43]. Given the values of the SR parameters or weights in either of these two methods, the motion is generated by forward dynamics simulation. The control over the motion is realized by specifying a performance function which is a mixture of some objective (e.g., maximizing the distance to be traveled) and some constraints (e.g., the figure doesn't fall) in a penalty function form. The parameters or weights must be found by using some numeric search algorithms so that the resulting motion has the maximum score. The advantage of such methods is that the resulting motion is always physically plausible and it can generate any motions which we can simulate. A good example is that such system can easily generate motions with contact and collision which the trajectory based approach, as discussed in the next section, has difficulty to deal with. The disadvantage of both methods is the difficulty of search. Since it is not clear how to compute gradients, the searchs usually have a bruce-force nature. With such a rudimentary search strategy solving a constrained optimization problem is expensive. In practice, usually only simple objectives are set such as to maximize the distance traveled. The resulting motions usually have convincing contacts and collisions but may not have gracefulness and a goal directed nature.

1.6 Spacetime Constraints

Another approach to gain control is to represent the user's goals as constraints. For example, suppose a point with unit mass, initially at rest at position $x = 0$, is required

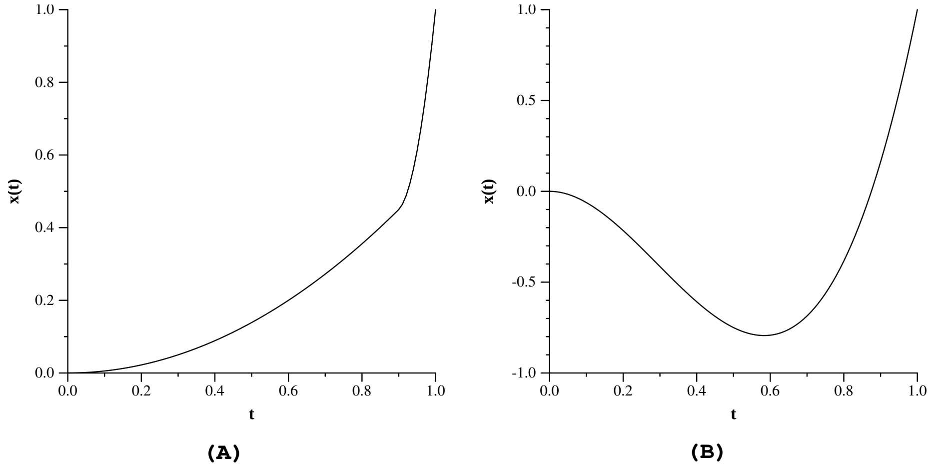


Figure 3: The paths of the point mass

to reach position $x = 1$ with velocity $\dot{x} = 10$ at time $t = 1$. The initial and final conditions can be represented as the following constraint equations:

$$\begin{aligned} x(0) &= 0 \\ x(1) &= 1 \\ \dot{x}(0) &= 0 \\ \dot{x}(1) &= 10 \end{aligned} \tag{4}$$

Together with the dynamic equation $f = \ddot{x}$, we obtain a system of equations:

$$\begin{aligned} f &= \ddot{x} \\ x(0) &= 0 \\ \dot{x}(0) &= 0 \\ x(1) &= 1 \\ \dot{x}(1) &= 10 \end{aligned} \tag{5}$$

Intuitively, one can easily think of a solution such that the point accelerates directly to the position $x = 1$ with the desired velocity. For instance,

$$f = \begin{cases} 10/9, & t \in [0, 9/10] \\ 90, & t \in [9/10, 1] \end{cases} \tag{6}$$

is such a solution. Figure 3 (A) is a plot of the trajectory $x = x(t)$.

But this solution is not the most naturally looking solution. Why? If we think of this point as a hammer , then in order to hit something hard, a more natural and efficient way would be to swing the hammer backward first to gain enough space for acceleration. The reason why this way is more efficient is because it requires less power. This suggests us to find the solution with minimum amount of forces. Mathematically we can minimize the integral of the square of forces over time, that is , we have the following formulation:

$$\begin{array}{ll} Minimize & \int f^2 dt \\ s.t & \\ f & = \ddot{x} \\ x(0) & = 0 \\ x(1) & = 1 \\ \dot{x}(0) & = 0 \\ \dot{x}(1) & = 10 \end{array} \tag{7}$$

This type of optimization problem where the unknowns are functions is called a *variational problem* [19]. It turns out this particular problem has an analytical solution [19]:

$$x = -7t^2 + 8t^3. \tag{8}$$

Figure 3 (B) is a plot of the trajectory. We can see that the point goes backward first and then accelerates to the desired velocity, which is a smoother and more natural motion.

The idea of finding the motion which minimizes some smoothness criteria such as to minimize the total forces as in the above example was proposed by Witkin and Kass [56], and the method is called the *spacetime constraints* method.

In the spacetime constraints formulation, the user can set up constraints over the time course including the initial and final configuration, the goal constraints, as well as the dynamic equations. An objective function is optimized to control the style of the motion. For example, minimizing the integral of forces as in the above example results in a graceful motion, while a minimum time objective function would result in a rushed motion.

The unknowns of a spacetime constraints formulation are the degree of freedom (DOF) functions (*trajectories*). Numerical techniques can be used to search for the trajectories which solve the optimization problem, as described briefly in the next section and in the next chapter with more detail.

1.7 Interactive Spacetime Constraints

The spacetime constraints formulation leads to a nonlinear constrained variational problem, that in general, has no closed form solution. In practice, the solution is carried out by reducing the space of possible trajectories to those representable by a linear combination of basis functions such as cubic B-splines. This results in a related constrained optimization problem (i.e., optimizing the coefficients to create motion curves for the DOF that minimize the objective while satisfying the constraints). Unfortunately, general solutions to such a nonlinear optimization problem are also unknown.

Observing such a difficulty, Cohen developed an interactive spacetime constraints system using a hybrid symbolic and numerical processing techniques [13]. In this system, the user can interact with the iterative numerical optimization and can guide the optimization process to converge to an acceptable solution. One can also focus attention on subsets or windows in spacetime to perform local refinement. This method overcomes some problems in the previous spacetime constraint system as proposed by Witkin and Kass. But some computational difficulties still remain, most notably as the complexity of the creature or animation increases. Addressing these problems is the central focus of the research as reported in this thesis.

1.8 Scope of the Thesis

This thesis concentrates on the goal directed animation of linked figures. We believe that a practical animation system must have the following properties. First, it must generate realistic motions by incorporating dynamics into the system to save the animator from tedious trial and error. Secondly, it must provide control to the animator so that the animator can create motions which perform the desired task or achieve the desired goal. Finally, it achieves interactive computation speed so that the animation system is productive. While spacetime constraints system do provide physical realisms and some user control, they have been computationally expensive. How to improve the spacetime constraints method to achieve efficiency is the central focus of the thesis. Most of the results reported here are also found in papers by the author [40, 39, 38].

1.9 Contributions

Our first contribution is to develop a new symbolic interface with a recursive evaluation scheme so that the time required for gradient computation which is needed by the numerical optimization is reduced from exponential growth to the optimal quadratic growth. Furthermore, with the new symbolic method, a language developed for the symbolic expressions can be easily input, thus it provides a more convenient interface. Secondly, we develop a keyframe optimization system which allows the user to specify a few keyframes while letting the computer determine the speed and timing parameters, which may be less intuitive for the animator, by using optimization methods. The novelty of this approach is that the user-specified keyframes are used both to provide control over the motion and to reduce the complexity of the optimization problem. This method has the advantage of providing much of the user control of the traditional animation system while providing the physical realism of the optimization based system. Due to the reduced complexity of the optimization problem, the computation time is nearly interactive for complex figures. Finally we develop a hierarchical scheme to solve the nonlinear variational problem arising in the spacetime constraint formulation by using wavelets. The advantage of the wavelet representation is two fold. First, the optimization problem is better conditioned so that the convergence is faster. Secondly, the number of unknowns is minimized so that each iteration takes less time.

The rest of the thesis is organized as follows. We provide some necessary background and notation in chapter 2. Chapter 3 through 5 describe the symbolic processing, optimized keyframes, and hierarchical wavelet basis. Finally we conclude and discuss future research ideas in chapter 6.

Chapter 2

Preliminaries

In this chapter, we give the necessary background and notations which will be used throughout this thesis. For a more general treatment of linked figure modeling, kinematics, and dynamics, the reader is referred to [14, 20, 17]. Fletcher [21] gives an excellent introduction to various optimization methods.

2.1 Coordinate systems and transformation matrices

Each *coordinate system* consists of an origin, and three axis which are mutually perpendicular unit vectors. If we denote the origin to be O and the three axis to be X, Y and Z, then the coordinate system is denoted to be (X, Y, Z, O). Given a coordinate system $\Omega = (X, Y, Z, O)$, for any point P, there exist three real numbers p_x, p_y, and p_z so that

$$P = O + p_x X + p_y Y + p_z Z. \tag{9}$$

The 3D column vector $\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$ is called the *coordinate of the point* P with respect to the coordinate system Ω, and is denoted by P^{Ω}. Similarly, for each vector V, there exist three real numbers v_x, v_y, and v_z so that

$$V = v_x X + v_y Y + v_z Z. \tag{10}$$

The 3D column vector $\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$, denoted by V^Ω, is called the *coordinate of the vector* V with respect to the coordinate system Ω.

Given two coordinate systems $\Omega_1 = (X_1, Y_1, Z_1, O_1)$ and $\Omega_2 = (X_2, Y_2, Z_2, O_2)$. The 3×3 matrix $(X_1^{\Omega_2}, Y_1^{\Omega_2}, Z_1^{\Omega_2})$ is called the *rotation matrix* (also *orientation matrix*) of Ω_1 with respect to Ω_2, and is denoted by $R_{\Omega_1}^{\Omega_2}$. The 4×4 matrix, $W_{\Omega_1}^{\Omega_2}$,

$$\begin{pmatrix} R_{\Omega_1}^{\Omega_2} & O_{\Omega_1}^{\Omega_2} \\ 0 & 1 \end{pmatrix} \tag{11}$$

is called the *homogeneous transformation matrix* (or simply transformation matrix) of Ω_1 with respect to Ω_2. From (10), we have

$$V^{\Omega_2} = R_{\Omega_1}^{\Omega_2} V^{\Omega_1}, \tag{12}$$

and from (9), we have

$$\begin{pmatrix} P^{\Omega_2} \\ 1 \end{pmatrix} = W_{\Omega_1}^{\Omega_2} \begin{pmatrix} P^{\Omega_1} \\ 1 \end{pmatrix}. \tag{13}$$

For any coordinate system Ω, the vector

$$\begin{pmatrix} P^{\Omega} \\ 1 \end{pmatrix}$$

is called the homogeneous coordinate of the point P with respect to Ω. We will also use P^Ω to denote its homogeneous coordinate where there is no confusion.

For any coordinate system Ω, since

$$\begin{aligned} X_1^{\Omega} &= R_{\Omega_2}^{\Omega} X_1^{\Omega_2}, \\ Y_1^{\Omega} &= R_{\Omega_2}^{\Omega} Y_1^{\Omega_2}, \\ Z_1^{\Omega} &= R_{\Omega_2}^{\Omega} Z_1^{\Omega_2}, \end{aligned} \tag{14}$$

we have

$$R_{\Omega_1}^{\Omega} = R_{\Omega_2}^{\Omega} R_{\Omega_1}^{\Omega_2}. \tag{15}$$

Furthermore, from (9),

$$O_1^{\Omega} = O_2^{\Omega} + R_{\Omega_2}^{\Omega} O_1^{\Omega_2}. \tag{16}$$

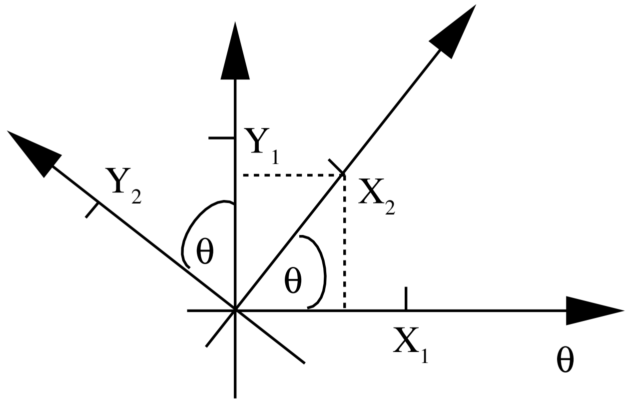


Figure 4: X_1 and Y_1 are rotated by θ counterclockwise

Therefore

$$W_{\Omega_1}^{\Omega} = W_{\Omega_2}^{\Omega} W_{\Omega_1}^{\Omega_2}. \tag{17}$$

Given any point P, since

$$\begin{aligned} P^{\Omega} &= W_{\Omega_1}^{\Omega} P^{\Omega_1} \\ &= W_{\Omega_2}^{\Omega} W_{\Omega_1}^{\Omega_2} P^{\Omega_1} \end{aligned} \tag{18}$$

and

$$P^{\Omega} = W_{\Omega_2}^{\Omega} P^{\Omega_2} \tag{19}$$

so

$$P^{\Omega_2} = W_{\Omega_1}^{\Omega_2} P^{\Omega_1} \tag{20}$$

2.2 Rotation about coordinate axis

Given a coordinate system $\Omega_1 = (X_1, Y_1, Z_1, O_1)$, let's rotate it θ around Z_1 and denote the resulting coordinate system as $\Omega_2 = (X_2, Y_2, Z_2, O_2)$. Notice that $Z_2 = Z_1$ and $O_2 = O_1$. To compute X_2 and Y_2, we can see from Figure 4 that

$$X_2 = cos(\theta)X_1 + sin(\theta)Y_1 \tag{21}$$

$$Y_2 = -sin(\theta)X_1 + cos(\theta)Y_1 \tag{22}$$

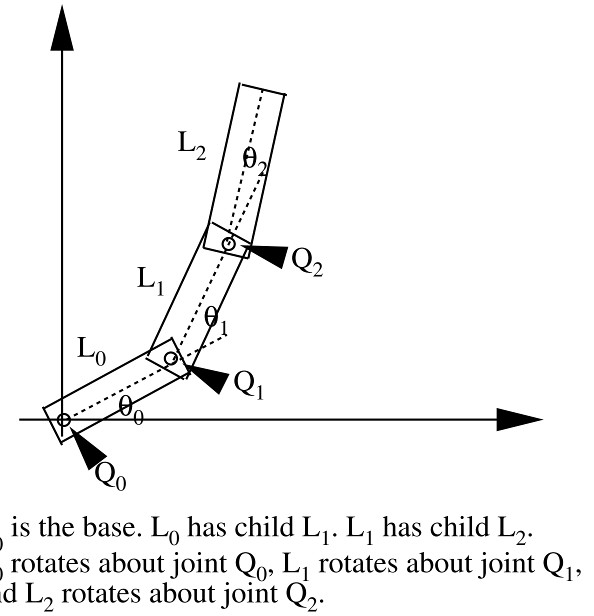


Figure 5: An example of linked figure structure

Therefore,

$$R_{\Omega_2}^{\Omega_1} = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{23}$$

We call this matrix *the rotation matrix around Z-axis*, and denote it by $\Re_z(\theta)$. Similarly we can derive the rotation matrices around X and Y-axis as

$$\Re_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{pmatrix}. \tag{24}$$

$$\Re_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix}, \tag{25}$$

2.3 Linked figure structure

Throughout this thesis, we assume the figure structure is a tree of rigid links (See Figure 5). Each link, except one, has a parent and and can rotate around its parent about a joint. The special link, which doesn't has a parent, is called a base. Each joint can have 1 to 3 degrees of freedom and we use Euler angles (rotated around Z, Y, and X successively) to represent the rotation angles at each joint. Given a figure, we label the links as L_0, L_1, ..., and L_{n-1} and the joint angles as θ_0, θ_1,..., and θ_{m-1}, where the labeling orders are arbitrary. For each link i, we use the $\wp_i$ to denote the label of its parent. And we use $D(i)$ to denote the set of the labels of all the children of link i. Denote Q_i to be the joint which connects link i with its parent.

2.4 Kinematics

Each link is attached to a local coordinate system which rotates with the link. The origin of this local coordinate system is at the joint which connects this link to its parent.

For the rest of this chapter, we use Ω_i to denote the local coordinate system of link i. We also assume there is a fixed world coordinate system, denoted by Ω. For any point P, we will use P to represent P^{Ω} when there is no confusion. And we use P^i to denote P^{Ω_i}. We use R_i to denote $R^{\Omega}_{\Omega_i}$, the rotation matrix of link i with respect to the world coordinate system. And we use R^j_i to denote $R^{\Omega_j}_{\Omega_i}$. We will use similar notations for homogeneous transformation matrices.

We assume that when all the rotational angles are 0's, all the coordinate systems have the same orientation as the world coordinate system. Now let's see how to compute the transformation matrices of these coordinate systems as functions of the rotational angles. To compute $R^{\wp_i}_i$, i.e, the transformation matrix of link i with respect to the coordinate system of its parent $\wp_i$, notice that R_i is obtained by the following process. Start with the same orientation as $\Omega_{\wp_i}$, then rotate Ω_i around its Z-axis by θ_{iz}, then rotate around Y-axis by θ_{iy}, and finally rotate around X-axis by θ_{ix}. So we have

$$R_i = R_{\wp_i}\Re_z(\theta_{iz})\Re_y(\theta_{iy})\Re_x(\theta_{ix}) \tag{26}$$

Therefore

$$R^{\wp_i}_i = \Re_z(\theta_{iz})\Re_y(\theta_{iy})\Re_x(\theta_{ix}) \tag{27}$$

```
void transMatrices(int i /*the current link*/)
{
        W_i^{℘_i} = ℜ_z(θ_iz)ℜ_y(θ_iy)ℜ_x(θ_ix)
        if (i is the base)
                W_i = W_i^{℘_i}
        else
                W_i = W_{℘_i} W_i^{℘_i}
        for all j ∈ D(i)
                transMatrices(j);
}
```

Figure 6: Compute transformation matrices

In the case when link i is a base, we simply have

$$R_i = \Re_z(\theta_{iz})\Re_y(\theta_{iy})\Re_x(\theta_{ix}). \tag{28}$$

The homogeneous transformation matrix is

$$W_i^{\wp_i} = \begin{pmatrix} R_i^{\wp_i} & Q_i^{\wp_i} \\ 0 & 1 \end{pmatrix}. \tag{29}$$

Notice that $Q_i^{\wp_i}$ is constant since Q_i is a fixed point on the link $\wp_i$. To compute the transformation matrices W_i for all the links, we can do a recursive tree traversal using the algorithm in Figure 6.

For any point P which is *fixed* on link i, its coordinate with respect to the world coordinate system can be computed from its local coordinate P^i as the following

$$P = W_i P^i. \tag{30}$$

Notice that P^i is constant since the link is rigid, thus the velocity of P is

$$\dot{P} = \dot{W}_i \dot{P}^i. \tag{31}$$

Since $W_i = W_{\wp_i} W_i^{\wp_i}$, we have

$$\dot{W}_i = \dot{W}_{\wp_i} W_i^{\wp_i} + W_{\wp_i} \dot{W}_i^{\wp_i}. \tag{32}$$

From (29) and recalling that $Q_i^{\wp_i}$ is constant, we have

$$\dot{W}_i^{\wp_i} = \begin{pmatrix} \dot{R}_i^{\wp_i} & 0 \\ 0 & 0 \end{pmatrix}. \tag{33}$$

Differentiating (27), we obtain

$$\dot{R}_i^{\wp_i} = \dot{\Re}_z(\theta_{iz})\Re_y(\theta_{iy})\Re_x(\theta_{ix}) + \Re_z(\theta_{iz})\dot{\Re}_y(\theta_{iy})\Re_x(\theta_{ix}) + \Re_z(\theta_{iz})\Re_y(\theta_{iy})\dot{\Re}_x(\theta_{ix}). \tag{34}$$

And the derivatives of $\Re_z$,$\Re_y$,and $\Re_x$ are

$$\dot{\Re}_z(\theta) = \begin{pmatrix} -sin(\theta) & -cos(\theta) & 0 \\ cos(\theta) & -sin(\theta) & 0 \\ 0 & 0 & 0 \end{pmatrix} \dot{\theta}, \tag{35}$$

$$\dot{\Re}_y(\theta) = \begin{pmatrix} -sin(\theta) & 0 & cos(\theta) \\ 0 & 0 & 0 \\ -cos(\theta) & 0 & -sin(\theta) \end{pmatrix} \dot{\theta}, \tag{36}$$

and

$$\dot{\Re}_x(\theta) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -sin(\theta) & -cos(\theta) \\ 0 & cos(\theta) & -sin(\theta) \end{pmatrix} \dot{\theta} \tag{37}$$

Similarly all the derivatives $\dot{W}_i$ can be computed by a recursive tree traversal (see Figure 7).

2.5 Dynamics

In this section, we show how to compute Lagrangian dynamics.

First let's see how to compute the kinetic energy of link i. Denote $\rho_i(x, y, z)$ to be the mass density of the link i which may be a function of the position in this link. For each point P with local coordinate $P^i = \begin{pmatrix} x \\ y \\ x \end{pmatrix}$, consider an infinitely small cube at this point with edge lengths dx, dy, and dz, respectively. Then the kinetic energy of this piece is approximately $0.5\rho_i(P^i)dxdydz\dot{P}^T\dot{P}$. By (31), $\dot{P} = \dot{W}_iP^i$. Denoting

```
void transMatricesDot(int i /*the current link*/)
{
        compute Ẇ_i^{℘_i} using ( 33- 37)
        if (i is the base)
                Ẇ_i = Ẇ_i^{℘_i}
        else
                Ẇ_i = Ẇ_{℘_i} W_i^{℘_i} + W_{℘_i} Ẇ_i^{℘_i}
        for all j ∈ D(i)
                transMatricesDot(j);
}
```

Figure 7: Compute the first order derivatives of the transformation matrices

$X = \begin{pmatrix} P^i \\ 1 \end{pmatrix}$, the energy becomes $0.5\rho_i(x, y, z)X\dot{W}_i^T\dot{W}_iXdxdydz$. Integrating this, we obtain the kinetic energy of link i

$$E_i = 0.5\int X^T\dot{W}_i^T\dot{W}_iX\rho_i dxdydz \tag{38}$$

where $X = (x, y, z, 1)^T$ ranges over the homogeneous coordinates of all the points in this link with respect to the local coordinate system.

Given a square matrix M, let $tr(M)$ denote the *trace* of M, i.e., the sum of all the diagonal elements of M. Notice that for any column vector Y, we have

$$Y^TY = tr(YY^T). \tag{39}$$

Therefore

$$\begin{aligned} E_i &= 0.5\int tr(\dot{W}_iXX^T\dot{W}_i^T)\rho_i dxdydz \\ &= 0.5tr(\dot{W}_i\int XX^T\rho_i dxdydz\dot{W}_i^T) \\ &= 0.5tr(\dot{W}_iJ_i\dot{W}_i^T), \end{aligned} \tag{40}$$

where $J_i = \int XX^T\rho_i dxdydz$ is the *inertial tensor* of the ith link with respect to its local coordinate system. Notice that all the components of J_i are constants since they only depend on the geometry and mass distribution of link i. And they can be computed once the linked figure is given with the local coordinate systems for each link being attached.

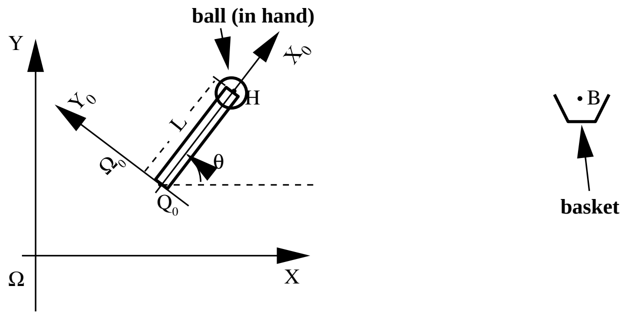


Figure 8: One-link arm

From (40), we have

$$\frac{\partial E_i}{\partial \theta_j} = 0.5tr(\frac{\partial \dot{W}_i}{\partial \theta_j} J_i \dot{W}_i^T + \dot{W}_i J_i \frac{\partial \dot{W}_i^T}{\partial \theta_j}) \tag{41}$$

$$\frac{\partial E_i}{\partial \dot{\theta}_j} = 0.5tr(\frac{\partial \dot{W}_i}{\partial \dot{\theta}_j} J_i \dot{W}_i^T + \dot{W}_i J_i \frac{\partial \dot{W}_i^T}{\partial \dot{\theta}_j}) = 0.5tr(\frac{\partial W_i}{\partial \theta_j} J_i \dot{W}_i^T + \dot{W}_i J_i \frac{\partial W_i^T}{\partial \theta_j}) \tag{42}$$

$$\frac{d}{dt}\frac{\partial E_i}{\partial \dot{\theta}_j} = 0.5tr(\frac{\partial \dot{W}_i}{\partial \theta_j} J_i \dot{W}_i^T + \frac{\partial W_i}{\partial \theta_j} J_i \ddot{W}_i^T + \ddot{W}_i J_i \frac{\partial W_i^T}{\partial \theta_j} + \dot{W}_i J_i \frac{\partial \dot{W}_i^T}{\partial \theta_j}) \tag{43}$$

So

$$\frac{d}{dt}\frac{\partial E_i}{\partial \dot{\theta}_j} - \frac{\partial E_i}{\partial \theta_j} = 0.5tr(\frac{\partial W_i}{\partial \theta_j} J_i \ddot{W}_i^T + \ddot{W}_i J_i \frac{\partial W_i^T}{\partial \theta_j}) = tr(\frac{\partial W_i}{\partial \theta_j} J_i \ddot{W}_i^T) \tag{44}$$

The potential energy due to gravity is $P_i = m_i G W_i r_i$, where r_i is the coordinate vector of the center of mass of the link i, and $G = (0, 9.8, 0)$ is the gravity vector. So

$$\frac{d}{dt}\frac{\partial P_i}{\partial \dot{\theta}_j} - \frac{\partial P_i}{\partial \theta_j} = -m_i G \frac{\partial W_i}{\partial \theta_j} r_i \tag{45}$$

Therefore by Lagrangian equation[32], the generalized force with respect to θ_j is

$$f_{\theta_j} = \frac{d}{dt}\frac{\partial (E_i - P_i)}{\partial \dot{\theta}_j} - \frac{\partial (E_i - P_i)}{\partial \theta_j} = \sum_{i=1}^{n} [tr(\frac{\partial W_i}{\partial \theta_j} J_i \ddot{W}_i^T) + m_i G \frac{\partial W_i}{\partial \theta_j} r_i] \tag{46}$$

2.6 Spacetime Constraints

To demonstrate the idea of spacetime constraints, let's see an example. Suppose we have a one link arm with one rotational degree of freedom around the Z axis (see

Figure 8). The rectangle represents the arm. The circle represents a ball. Let θ be the rotational angle of the arm with respect to the X axis in the counterclockwise direction.

Suppose we want to animate the following motion sequence. The arm starts in a straight down position ($\theta = -\frac{\pi}{2}$) at time t_0, throws a ball (of negligible mass) at time t_1, and finally comes back to the straight down position at time t_2. The requirement is that the ball is in the hand from t_0 to t_1, then is in free flight from t_1 to $t_1 + T$, and at time $t_1 + T$ the ball is coincident with a basket which is the position B as shown in Figure 8.

The spacetime constraint method starts by specifying *constraints* including start and ending conditions, goal requirement, and some limits on the physical parameters of the figure such as joint angles, torques, etc.

In this example, the start and ending conditions are that the ball is in the straight down position. So we have the following equations:

(a) the start condition:

$$\begin{aligned} \theta(0) &= -\frac{\pi}{2} \\ \dot{\theta}(0) &= 0. \end{aligned} \tag{47}$$

(b) the final condition:

$$\begin{aligned} \theta(t_2) &= -\frac{\pi}{2} \\ \dot{\theta}(t_2) &= 0. \end{aligned} \tag{48}$$

The goal constraint in this case is that the ball goes into basket at time $t_1 + T$ which corresponds to the following equation (notice that the ball is in free flight after leaving the hand at time t_1).

(c)

$$H(t_1) + T\dot{H}(t_1) - \frac{1}{2}G^T T^2 = B \tag{49}$$

where $G = (0, g, 0)$ is the gravity vector, B is the position of the basket, and H is the position of the end effector of the arm (see Figure 8) which is a function over time.

We may want to put some limits on the joint angle θ so that the arm only moves in a limited range. For example, a human can not move his (her) arm back and up. To simulate this, we can limit θ to be in the range $[-\pi, \pi/2]$. Thus we have the following inequality:

(d)

$$-\pi \leq \theta(t) \leq \pi/2, \forall t \in [0, t_2]. \tag{50}$$

After the constraints are specified, the user needs to supply an *objective* function such as to perform the tasks specified by the constraints with minimum energy or some other style consideration. In this example, we can use the generalized forces (see (46)) as the objective function to minimize. Notice that the generalized force is a function over time, so we need to integrate over time. Formally the objective function is the following:

$$\int \tau^2(\theta(t), \dot{\theta}(t), \ddot{\theta}(t)), \tag{51}$$

where τ is the generalized force.

After a serious of specifications of objective and constraints, we end up with a constrained optimization problem where the unknowns are a set of functions through time (*trajectories*) of each degree of freedom (DOF). In this example, the optimization problem is:

$$\begin{array}{ll} \text{minimize} & (\ 51) \\ s.t. & (\ 47) - (\ 50) \end{array} \tag{52}$$

The unknown is $\theta(t)$ which is a scalar function over time.

In order to solve the optimization problem, the expressions have to be represented in terms of the unknowns. In general, this can be performed by using the the kinematics and dynamics formulas as derived in the previous sections. In the following we will show in detail how to use the formulas in previous sections to derive the expressions in (52). For such a simple example, this may not be the simplest way, but the purpose is to demonstrate the general framework.

The world coordinate system is denoted Ω, and the local coordinate system of the link is denoted Ω_0. The origin of the link is denoted Q. Let $(q_x, q_y, 0, 1)$ denote the homogeneous coordinate of Q with respect to Ω. Let L denote the length of the link. The transformation matrix of Ω_0 is

$$W_0 = \Re_z(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{53}$$

Since the local coordinate of H is $H^0 = \begin{pmatrix} L \\ 0 \\ 0 \\ 1 \end{pmatrix}$, from (30),

$$H = W^0 H^0 = \begin{pmatrix} Lcos(\theta) \\ -Lsin(\theta) \\ 0 \\ 1 \end{pmatrix} \tag{54}$$

and

$$\dot{H} = W^0 H^0 = \begin{pmatrix} -Lsin(\theta)\dot{\theta} \\ -Lcos(\theta)\dot{\theta} \\ 0 \\ 0 \end{pmatrix} \tag{55}$$

So constraint (c) becomes

$$q_x + Lcos(\theta) - TLsin(\theta)\dot{\theta} = b_x \tag{56}$$

$$q_y + Lsin(\theta) + TLcos(\theta)\dot{\theta} - 0.5gT^2 = b_y \tag{57}$$

where b_x and b_y are the X and Y world coordinates of B. To use equation (46) to compute the torque, we need to compute the inertial tensor J. Assuming the mass density ρ of the link is constant, then $\int_{x=0}^{L} x^2 \rho dx = \frac{1}{3}L^3\rho = \frac{1}{3}L^2 m$ where m is the mass of this link. Therefore

$$J = \begin{pmatrix} \frac{1}{3}L^2 m & 0 & 0 & * \\ 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \\ * & * & * & * \end{pmatrix} \tag{58}$$

where $*$ means we don't care about the value. Finally the center of gravity $r_0 = (0, L/2, 0)^T$. Now we can apply equation (46), and after some algebraic manipulations (the details are omitted), one can obtain that

$$\tau = \frac{1}{3}mL^2\ddot{\theta} + \frac{1}{2}mgLcos(\theta). \tag{59}$$

In summary, we have the following optimization problem:

$$\begin{array}{ll} min & \int_0^{t_2}(\frac{1}{3}mL^2\ddot{\theta} + \frac{mgL}{2}cos(\theta))^2 dt \\ s.t. & \\ & \theta(0) = -\frac{\pi}{2} \\ & \dot{\theta}(0) = 0 \\ & \theta(t_2) = -\frac{\pi}{2} \\ & \dot{\theta}(t_2) = 0. \\ & q_x + Lcos(\theta(t_1)) - TLsin(\theta(t_1))\dot{\theta}(t_1) = b_x \\ & q_y + Lsin(\theta(t_1)) + TLcos(\theta(t_1))\dot{\theta}(t_1) - \frac{1}{2}gT^2 = b_y \\ & -\pi \leq \theta(t) \leq \pi/2, \forall t \in [0, t_2] \end{array} \tag{60}$$

Problems like this, in which the unknowns are functions (in this example, the unknown is the *trajectory* $\theta(t)$ over $[0, t_2]$), is called a variational problem [19]. In general, there is no closed form solution. Numerically we can solve this problem by using finite element methods which are the topic of the next section.

2.7 The Finite Element Method

Finite element methods convert an infinite dimensional problem into a finite dimensional problem by reducing the function space of possible trajectories to a finite dimensional function space. For example, we can choose spline spaces, which are spaces of piecewise polynomials, as the finite dimensional function spaces. After we have chosen a finite dimensional function space, we can choose a basis for this space. Suppose $B_1(t)$, $B_2(t)$, ..., $B_r(t)$ are the basis functions where r is the dimension of the function space. Then any degree of freedom function $\theta_i(t)$ can be represented as a linear combination of these basis functions:

$$\theta_i(t) = c_{i1}B_1(t) + c_{i2}B_2(t) + ... + c_{ir}B_r(t) \tag{61}$$

where c_{ij} are coefficients. The derivatives are simply

$$\dot{\theta}_i(t) = c_{i1}\dot{B}_1(t) + c_{i2}\dot{B}_2(t) + ... + c_{ir}\dot{B}_r(t), \tag{62}$$

and

$$\ddot{\theta}_i(t) = c_{i1}\ddot{B}_1(t) + c_{i2}\ddot{B}_2(t) + ... + c_{ir}\ddot{B}_r(t) \tag{63}$$

We can substitute these equations into the spacetime constraint formulation such as (60), replacing the unknown trajectory function with scalar unknowns c_{i1}, c_{i2}, ..., and c_{ir}. The problem thus becomes a finite dimensional nonlinear constrained optimization problem.

Given a function $F(\theta_i, \dot{\theta}_i, \ddot{\theta}_i)$, its gradient (partial derivatives with respect to the coefficients) can be computed by using chain rules

$$\begin{aligned} \frac{\partial F}{\partial c_{ik}} &= \frac{\partial F}{\partial \theta_i}\frac{\partial \theta_i}{\partial c_i k} + \frac{\partial F}{\partial \dot{\theta}_i}\frac{\partial \dot{\theta}_i}{\partial c_i k} + \frac{\partial F}{\partial \ddot{\theta}_i}\frac{\partial \ddot{\theta}_i}{\partial c_i k} \\ &= \frac{\partial F}{\partial \theta_i} B_k + \frac{\partial F}{\partial \dot{\theta}_i}\dot{B}_k + \frac{\partial F}{\partial \ddot{\theta}_i}\ddot{B}_k \end{aligned} \tag{64}$$

For an integral expression, we have the similar formula:

$$\int \frac{\partial F}{\partial c_{ik}} dt = \int \frac{\partial F}{\partial \theta_i} B_k dt + \int \frac{\partial F}{\partial \dot{\theta}_i}\dot{B}_k dt + \int \frac{\partial F}{\partial \ddot{\theta}_i}\ddot{B}_k dt \tag{65}$$

Each integral can be approximated by some quadrature formulas [48]. The simplest is to simply sum the integrand values at sample points over the whole integration interval. To save unnecessary computations, we actually only need to sample points where B_k is nonzero. The subinterval over which B_k is nonzero is called the *support* of B_k, and B_k is said to *cover* the support. If the support of B_k is a finite interval, B_k is called *compact*.

2.8 Cubic Splines

Splines are piecewise polynomials which are used to approximate functions in numerical computations. In this thesis, we mainly use two types of cubic splines: Hermite splines and Uniform B-splines. So we will only talk about these two types of splines here. The reader is referred to [22, 4] for descriptions of general splines.

2.8.1 Hermite Splines

Given an interval $[t_0, t_1]$, and four numbers A_0, P_0, A_1 and P_1, there is a unique cubic polynomial

$$h(t) = a + b(t - t_0) + c(t - t_0)^2 + d(t - t_0)^3 \tag{66}$$

so that

$$\begin{aligned} h(t_0) &= A_0, \\ h(t_1) &= A_1, \\ \tfrac{dh(t_0)}{dt} &= P_0, \\ \tfrac{dh(t_1)}{dt} &= P_1. \end{aligned} \tag{67}$$

This polynomial is called a cubic *Hermite polynomial* (or *Hermite spline*). Geometrically, we can think of the problem as interpolating two points (t_0, A_0) and (t_1, A_1) with the slopes P_0 and P_1, respectively. The corresponding curve is called a Hermite curve, and this interpolation method is called Hermite interpolation. Sometimes we denote it to be $h(A_0, A_1, P_0, P_1, t_0, t_1, , t)$ to show its dependence on $A_0, A_1, P_0, P_1, t_0, t_1$.

By solving equations from (67), one can find that

$$\begin{aligned} a &= A_0, \\ b &= P_0, \\ c &= \tfrac{(3A_1 - 3A_0 - P_1(t_1 - t_0) - 2P_0(t_1 - t_0))}{(t_1 - t_0)^2}, \\ d &= \tfrac{2A_0 - 2A_1 + P_1(t_1 - t_0) + P_0(t_1 - t_0)}{(t_1 - t_0)^3} \end{aligned} \tag{68}$$

Substituting (68) into (66), and collecting terms, we have the representation in terms of linear combination of basis functions:

$$h(t) = A_0 B_0(t) + A_1 B_1(t) + P_0 B_2(t) + P_1 B_3(t) \tag{69}$$

where the basis functions are

$$\begin{aligned} B_0(t) &= 1 + \tfrac{-3}{(t_1 - t_0)^2}(t - t_0)^2 + \tfrac{2}{(t_1 - t_0)^3}(t - t_0)^3 \\ B_1(t) &= \tfrac{3}{(t_1 - t_0)^2}(t - t_0)^2 + \tfrac{-2}{(t_1 - t_0)^3}(t - t_0)^3 \\ B_2(t) &= (t - t_0) + \tfrac{-2(t_1 - t_0)}{(t_1 - t_0)^2}(t - t_0)^2 + \tfrac{(t_1 - t_0)}{(t_1 - t_0)^3}(t - t_0)^3 \\ B_3(t) &= \tfrac{-(t_1 - t_0)}{(t_1 - t_0)^2}(t - t_0)^2 + \tfrac{(t_1 - t_0)}{(t_1 - t_0)^3}(t - t_0)^3 \end{aligned} \tag{70}$$

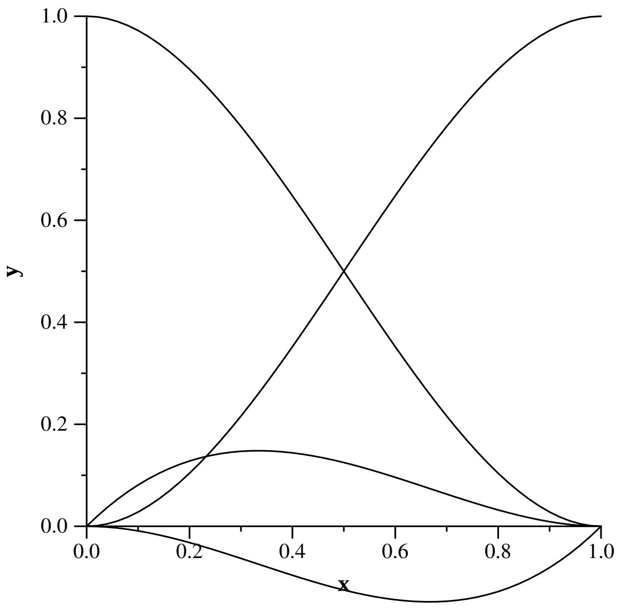


Figure 9: The Hermite spline basis functions over $[0, 1]$

Figure 9 shows the four basis functions with $t_0 = 0$ and $t_1 = 1$.

Suppose we are given a sequence of points $t_0 < t_1 < ... < t_n$. For any given $A_0, A_1, ..., A_n$ and $P_0, P_1, ..., P_n$, there is a unique function $H(t)$ which is a polynomial over each subinterval $[t_i, t_{i+1}]$, so that $H(t_i) = A_i$ and $\frac{dH(t_i)}{dt} = P_i$. In fact,

$$H(t) = h(A_i, A_{i+1}, P_i, P_{i+1}, t_i, t_{i+1}, t), t \in [t_i, t_{i+1}], 0 \le i \le n - 1. \tag{71}$$

$H(t)$ is called a Hermite cubic spline, or piecewise Hermite cubic curve, or Hermite interpolation.

We can also obtain the basis function representation by using (69). However, since the basis representation is rarely used, we will not give the details. It is not hard to verify that that Hermite splines have continuous first order derivatives, but not second order derivatives.

2.8.2 Uniform B-splines

There are rich theories about general B-splines [4], but we only describe cubic uniform B-spines here.

First we consider B-splines over the interval $[0, n]$ with n segments $[i, i + 1], i = 0, ..., n - 1$, where n is some positive integer. In other words, each spline is a (cubic)

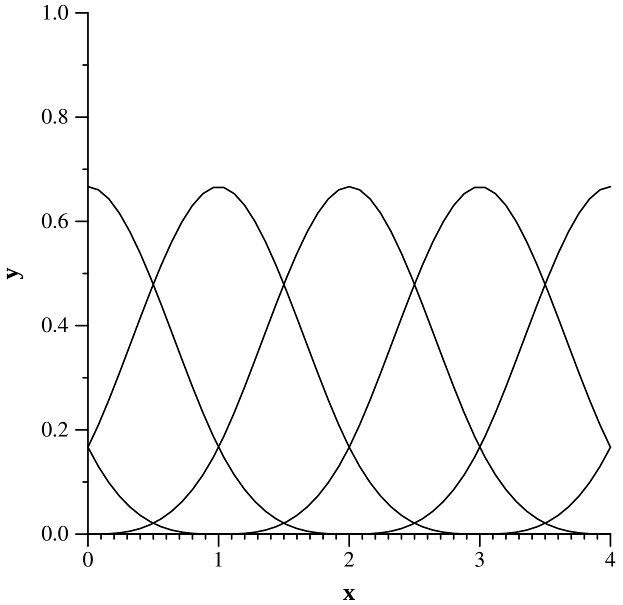


Figure 10: The uniform B-spline basis functions over $[0, 4]$

polynomial over each subinterval $[i, i+1]$, $i = 0, ..., n-1$. There are $n+3$ basis functions $B_i(t), i = -3, -2, -1, 0, 1, ..., n-1$, where

$$B_0(t) = \begin{cases} 0 & t < 0 \\ \frac{t^3}{6} & 0 \le t < 1 \\ \frac{-3(t-1)^3+3(t-1)^2+3(t-1)+1}{6} & 1 \le t < 2 \\ \frac{3(t-2)^3-6(t-2)^2+4}{6} & 2 \le t < 3 \\ \frac{(4-t)^3}{6} & 3 \le t < 4 \\ 0 & 4 \le t \end{cases} \tag{72}$$

and the other basis functions are all the translations of $B_0(t)$: $B_i(t) = B(t-i)$, $i = -3, -2, ..., n-1$. Figure 10 shows the 7 basis functions over $[0, 4]$: $B_{-3}, ..., B_3$.

Each B-spline curve is a linear combination of these basis functions

$$b(t) = \sum_{i=-3}^{n-1} c_i B_i(t), \tag{73}$$

where the c_i's are coefficients. Notice that for any $t \in [0, n]$, there are only four basis functions which are nonzero at t. If $t \in [i, i+1]$, these four basis functions are $B_{i-3}(t)$, $B_{i-2}(t)$, $B_{i-1}(t)$, and $B_i(t)$. Thus

$$b(t) = c_{i-3}B_{i-3}(t) + c_{i-2}B_{i-2}(t) + c_{i-1}B_{i-1}(t) + c_iB_i(t). \tag{74}$$

B-splines over the interval $[a, b]$ can be obtained from B-splines over $[0, n]$ using linear transformations. The B-spline basis functions over $[a, b]$ are

$$B_i^*(t) = B_i(\frac{t-a}{b-a}n), i = -3, -2, ..., n-1. \tag{75}$$

.

2.9 Wavelets

In this section, we give a brief introduction to wavelets. We will introduce some terminology and describe the construction of two wavelets: Haar wavelets and Chui-Wang B-wavelets. For a general and more theoretical treatment of wavelets, the reader is referred to the monographs by Chui [12] and Daubechies [16]. Part of the materials given in this section is extracted from [40].

Before we go to the introduction to wavelets, let's review a few concepts regarding function spaces. Given any two functions $f(t)$ and g(t), the (L_2) *inner product* of these two functions is defined as

$$\langle f(t), g(t) \rangle = \int_{-\infty}^{+\infty} f(t)g(t) \ dt \tag{76}$$

$(\langle f(t), f(t) \rangle)^{\frac{1}{2}}$ is called the (L_2) *norm* of $f(t)$, denoted by $||f||$. If $\langle f(t), g(t) \rangle = 0$, we say f and g are *orthogonal* to each other. Given any two function spaces V and W, their *sum* is a function space which consists of all the functions of the form $f(t)+g(t)$ where $f \in V$ and $g \in W$. If all functions in V are orthogonal to all functions in W, then V and W are *orthogonal* to each other, in which case, the sum of V and W is called a *direct sum*, and is denoted by $V \dot{+} W$.

If $U = V \dot{+} W$, then for any $h(t) \in U$, there exist $f(t) \in V$ and $g(t) \in W$ so that $h(t) = f(t) + g(t)$. In this case, $f(t)$ is called the *orthogonal projection* of $h(t)$ into V. It is an important fact that $f(t)$ is the *best approximation* of $h(t)$ in the space V, that is,

$$||f - h|| \leq ||a - h||, \forall a(t) \in V. \tag{77}$$

The proof is simply the following:

$$||a - h||^2 = ||a - f - g||^2 = ||a - f||^2 + ||g||^2 \geq ||g||^2 = ||f - h||^2. \tag{78}$$

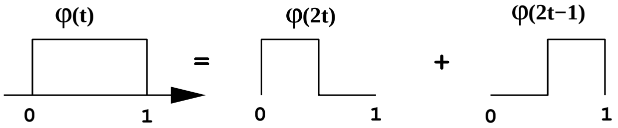


Figure 11: Two scale relationship of Haar basis

2.9.1 Haar Wavelets

To illustrate the basic concept of the Haar wavelet construction, let's consider the space V_L of all those functions which are defined over $[0,1]$ and are linear combinations of the 2^L basis functions $\phi_{L,j}(t) = \phi(2^L t - j)$, $j \in [0, 2^L - 1]$, where L is some given positive integer, and

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{79}$$

is called a *scale function*. For each $j \in [0, 2^L - 1]$, $\phi_{L,j}$ is a box function which is equal to 1 over $[2^{-L}j, 2^{-L}(j+1))$ and 0 elsewhere. So V_L consists of all those functions which are constants over each interval $[2^{-L}j, 2^{-L}(j+1))$, $j \in [0, 2^L - 1]$. Now let's consider V_{L-1} which is generated by the 2^{L-1} basis functions $\phi_{L-1,j}(t) = \phi(2^{L-1}t - j)$ where $j \in [0, 2^{L-1} - 1]$. Each basis function $\phi_{L-1,j}$ is also a box function, but is twice as wide as those in V_L. Clearly,

$$V_{L-1} \subset V_L. \tag{80}$$

To see how to represent $\phi_{L-1,j}$ as a linear combination of those basis functions in $V_{L,j}$, we need the following equality, called the *two scale relationship*,

$$\phi(t) = \phi(2t) + \phi(2t-1). \tag{81}$$

See Figure 11.

Substituting $2^{L-1}t - j$ for t in (81), we obtain

$$\phi(2^{L-1}t - j) = \phi(2^L t - 2j) + \phi(2^L t - 2j - 1), \tag{82}$$

that is,

$$\phi_{L-1,j} = \phi_{L,2j} + \phi_{L,2j+1}. \tag{83}$$

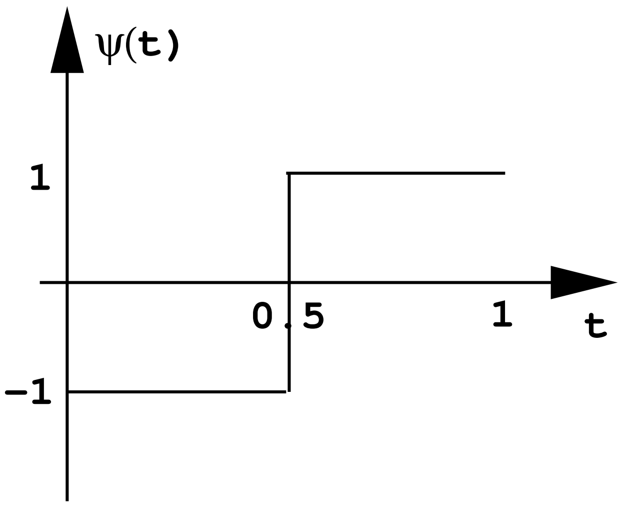


Figure 12: The Haar wavelet function $\psi(t)$

Since $V_{L-1} \subset V_L$, there must exist a subspace W_{L-1} of V_L orthogonal to V_{L-1} so that

$$V_{L-1} \dot{+} W_{L-1} = V_L. \tag{84}$$

This is called a *decomposition* of V_L. In fact, W_{L-1} is generated by the following basis functions:

$$\psi_{L-1,j}(t) = \psi(2^{L-1}t - j), j \in [0, 2^{L-1}], \tag{85}$$

where

$$\psi(t) = \begin{cases} -1 & 0 \leq t < 1/2 \\ 1 & 1/2 \leq t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{86}$$

as shown in Figure 12

In the following, we show why (84) is true. It is easy to see that

$$\psi(t) = -\phi(2t) + \phi(2t-1), \tag{87}$$

which is called the *two scale relationship* for wavelet functions. Substituting $2^{L-1}t - j$ for t in the above equality, we obtain

$$\psi(2^{L-1}t - j) = -\phi(2^L t - 2j) + \phi(2^L t - 2j - 1), \tag{88}$$

that is

$$\psi_{L-1,j} = -\phi_{L,2j} + \phi_{L,2j+1}. \tag{89}$$

This shows that

$$W_{L-1} \subset V_L. \tag{90}$$

The inverse of (83) and (89) is

$$\begin{array}{ll} \phi_{L,2j} & = \frac{1}{2}\phi_{L-1,j} - \frac{1}{2}\psi_{L-1,j} \\ \phi_{L,2j+1} & = \frac{1}{2}\phi_{L-1,j} + \frac{1}{2}\psi_{L-1,j} \end{array} \tag{91}$$

This shows that $V_L \subset V_{L-1}\dot{+}W_{L-1}$, It is a trivial verification to see why $\phi_{L-1,j}$ is orthogonal to $\psi_{L-1,i}$ for all i and j. Therefore (84) holds.

Now each function $f(t)$ in V_L has two representations:

$$f(t) = \sum_{j=0}^{2^L-1} c_{L,j}\phi_{L,j}, \tag{92}$$

and

$$f(t) = \sum_{j=0}^{2^{L-1}-1} c_{L-1,j}\phi_{L-1,j} + \sum_{j=0}^{2^{L-1}-1} w_{L-1,j}\psi_{L-1,j}. \tag{93}$$

As discussed in the beginning of this section, the term $\sum_j c_{L-1,j}\phi_{L-1,j}$ is the best approximation of $f(t)$ in the space V_{L-1}, and is called the *smooth part* of $f(t)$. The other term $\sum_j w_{L-1,j}\psi_{L-1,j}$ is the difference of this approximation from the original function $f(t)$, called the *detail part.* The coefficients $c_{L-1,j}$'s are called *smooth coefficients*, and the coefficients $w_{L-1,j}$'s are called *detail (or wavelet) coefficients.*

Now let's see how to transform between the two sets of coefficients, namely, $\{c_{L,j}\}_{j=0}^{2^L-1}$ and $\{c_{L-1,j}, w_{L-1,j}\}_{j=0}^{2^{L-1}-1}$. From (92), we have

$$\begin{array}{ll} f(t) & = \sum_{j=0}^{2^L} c_{L,j}\phi_{L,j} \\ & = \sum_{j=0}^{2^{L-1}-1}(c_{L,2j}\phi_{L,2j} + c_{L,2j+1}\phi_{L,2j+1}) \\ & = \sum_{j=0}^{2^{L-1}-1}(c_{L,2j}(\frac{1}{2}\phi_{L-1,j} - \frac{1}{2}\psi_{L-1,j}) + c_{L,2j+1}(\frac{1}{2}\phi_{L-1,j} + \frac{1}{2}\psi_{L-1,j})) \\ & = \sum_{j=0}^{2^{L-1}-1}((\frac{1}{2}c_{L,2j} + \frac{1}{2}c_{L,2j+1})\phi_{L-1,j} + (-\frac{1}{2}c_{L,2j} + \frac{1}{2}c_{L,2j+1})\psi_{L-1,j}) \end{array} \tag{94}$$

where the third equality is from (91) and the last one is by collecting terms. Compare this with (93), we have

$$\begin{aligned} c_{L-1,j} &= \tfrac{1}{2}c_{L,2j} + \tfrac{1}{2}c_{L,2j+1} \\ w_{L-1,j} &= -\tfrac{1}{2}c_{L,2j} + \tfrac{1}{2}c_{L,2j+1} \end{aligned} \tag{95}$$

We can immediately see the inverse of this relationship, namely,

$$\begin{aligned} c_{L,2j} &= c_{L-1,j} - w_{L-1,j} \\ c_{L,2j+1} &= c_{L-1,j} + w_{L-1,j} \end{aligned} \tag{96}$$

We can repeat this decomposition logic and replace V_{L-1} by $V_{L-2}\dot{+}W_{L-2}$. In this way, we have another decomposition of V_L

$$V_L = V_{L-2}\dot{+}W_{L-2}\dot{+}W_{L-1}, \tag{97}$$

and any function $f(t) \in V_L$ can be represented as

$$f(t) = \sum_{j=0}^{2^{L-2}-1} c_{L-2,j}\phi_{L-2,j} + \sum_{j=0}^{2^{L-1}-1} w_{L-2,j}\psi_{L-2,j} + \sum_{j=0}^{2^{L-1}-1} w_{L-1,j}\psi_{L-1,j}. \tag{98}$$

In general, for any $L_0 < L$, we have a decomposition of V_L:

$$V_L = V_{L-L_0}\dot{+}\sum_{i=L-L_0}^{L-1} W_i, \tag{99}$$

and $f(t)$ can be represented as

$$f(t) = \sum_{j=0}^{2^{L-L0}-1} c_{L-L0,j}\phi_{L-L_0,j} + \sum_{l=L-L_0}^{L-1}\sum_{j=0}^{2^l-1} w_{l,j}\psi_{l,j}. \tag{100}$$

When $L_0 = L$, we have

$$V_L = V_0\dot{+}\sum_{i=0}^{L-1} W_i, \tag{101}$$

and

$$f(t) = c_{0,0}\phi_{0,0} + \sum_{l=0}^{L-1}\sum_{j=0}^{2^l-1} w_{l,j}\psi_{l,j}. \tag{102}$$

The case when $L = 3$ is shown in Figure 13.

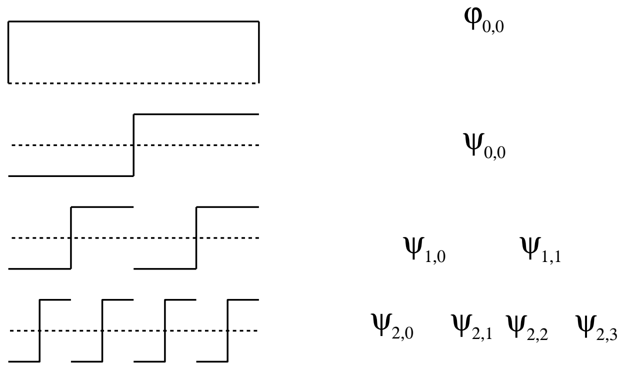


Figure 13: Haar basis functions when $L = 3$

This new basis $\{\phi_{0,0}\} \cup \{\psi_{l,j} | 0 \leq j \leq 2^l - 1, 0 \leq l \leq L - 1\}$ is called the Haar wavelet basis (or simply Haar basis). In this new basis, each $\psi_{l,j}$ is a scaling and translation of the single function $\psi(t)$. This special function $\psi(t)$ is called a wavelet (or a wavelet function). And all the functions $\psi_{l,j}$ are called wavelet functions.

This new basis also contains $1 + \sum_{l=0}^{L-1} 2^l = 2^L$ basis functions. But it represents functions hierarchically. The widest box function $\phi_{0,0}(t) = \phi(t)$ represents the overall average, and at each level going down, the wavelet basis functions represent the finer and finer details. If at some point, the function has no more detail (i.e., the function is constant over the support of the wavelet), the corresponding wavelet coefficient will be zero as will all finer wavelets below.

For example, consider the function (see Figure 14)

$$f(t) = \begin{cases} 2 & t \in [0, \frac{1}{4}) \\ 2 & t \in [\frac{1}{4}, \frac{2}{4}) \\ 4 & t \in [\frac{2}{4}, \frac{3}{4}) \\ 1 & t \in [\frac{3}{4}, 1) \end{cases} \tag{103}$$

We can use $\phi_{2,0}$,$\phi_{2,1}$, $\phi_{2,2}$, and $\phi_{2,3}$ to represent $f(t)$ by using coefficients $2, 2, 4, 1$, respectively. Alternatively, one can use the wavelet basis $\phi_{0,0}, \psi_{0,0}, \psi_{1,0}$, and $\psi_{1,1}$ with coefficients $2.25, 0.25, 0$, and -1.5, respectively (we will describe the algorithms to compute these coefficients later). Notice that the coefficient of $\psi_{1,0}$ is 0, because the function is smooth (constant) over interval $[0, 0.5]$ which $\psi_{1,0}$ covers. Also note that the coefficient of $\psi_{0,0}$ is 2.25, the average value over the the whole domain.

The transformation to and from the coefficients of the Haar wavelet basis functions is called a pyramid transformation. Given the coefficients $\{c_{L,j}\}_{j=0}^{2^L-1}$ of the original

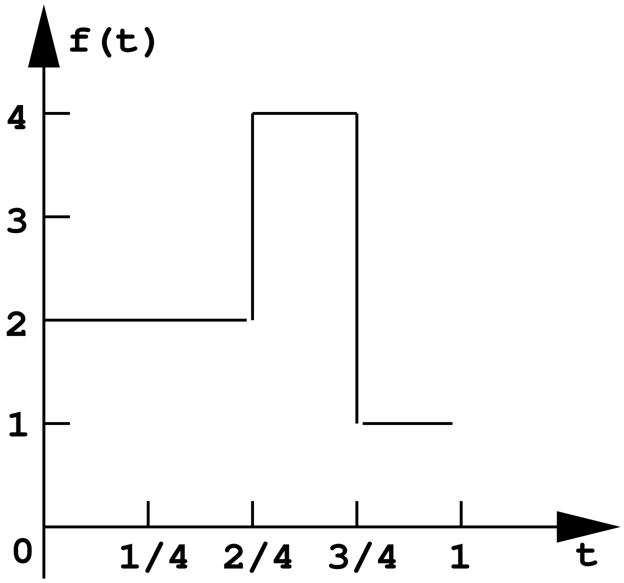


Figure 14: $f(t) = 2\phi_{2,0} + 2\phi_{2,1} + 4\phi_{2,2} + \phi_{2,3} = 2.25\phi_{0,0} + 0.25\psi_{0,0} - 1.5\psi_{1,1}$

basis functions, the coefficients of the Haar basis functions can be computed by using the formula (95) (see `haar_coef_xform_up`) L times, as shown in `haar_coef_pyrm_up`. Conversely, given the coefficients of the Haar basis, $c_{0,0}$ and $w_{l,j}$, $j \in [0, 2^l - 1]$ $l \in [0, L-1]$, `haar_coef_pyrm_down` computes $c_{L,j}, j \in [0, 2^L - 1]$, by applying (96) (see `haar_coef_xform_down`) L times.

haar_coef_pyrm_up(c_{in}[] , c_{out}[] , w_{out}[][] , L)

  $c_{\text{temp}}[L][] = c_{\text{in}}[]$;

  for($l = L$; $l \geq 1$; $l--$)

    haar_coef_xform_up($c_{\text{temp}}[l][]$, $c_{\text{temp}}[l-1][]$,

          $w_{\text{out}}[l-1][]$, l) ;

  $c_{\text{out}}[0] = c_{\text{temp}}[0][0]$;

haar_coef_pyrm_down(c_{in}[] , w_{in}[] [] , c_{out}[] , L)

$c_{\text{temp}}[0][0] = c_{\text{in}}[0]$;

for($l = 1$; $l \leq L$; $l + +$)

haar_coef_xform_down($c_{\text{temp}}[l-1][]$, $w_{\text{in}}[l-1][]$,
$c_{\text{temp}}[l][]$, l) ;

$c_{\text{out}}[] = c_{\text{temp}}[L][]$;

haar_coef_xform_up(c_{in}[] , c_{out}[] , $w_{\text{out}}[]$, L)

for($j = 0$; $j < 2^{L-1}$; $j + +$) {

$c_{\text{out}}[j] = \frac{1}{2}c_{\text{in}}[2j] + \frac{1}{2}c_{\text{in}}[2j+1]$;

$w_{\text{out}}[j] = -\frac{1}{2}c_{\text{in}}[2j] + \frac{1}{2}c_{\text{in}}[2j+1]$;

}

haar_coef_xform_down(c_{in}[] , w_{in}[] , $c_{\text{out}}[]$, L)

for($j = 0$; $j < 2^{L-1}$; $j + +$) {

$c_{\text{out}}[2j] = c_{\text{in}}[j] - w_{\text{in}}[j]$;

$c_{\text{out}}[2j+1] = c_{\text{in}}[j] + w_{\text{in}}[j]$;

}

Both haar_coef_pyrm_up and haar_coef_pyrm_down take time $2^L + 2^{L-1} + ... + 1 = 2^{L+1} - 1$, and hence they run in linear time (in terms of the number of basis functions, which is 2^L in this case).

2.9.2 B-spline Wavelets

In this section, we consider the wavelets on the whole real line $\mathcal{R}$ first, and we will then consider wavelets on the interval in the next section.

Unlike the Haar wavelet basis which is an alternate basis for the function space generated by box functions (zero order B-splines), B-spline wavelet basis is an alternate basis for the function space generated by cubic B-spline functions (in this section, we will only consider uniform cubic B-splines). The scale function is the cubic B-spline basis function:

$$\phi(t) = B_0(t) \tag{104}$$

where B_0 is defined in (72). Similarly define

$$\phi_{L,j} = \phi(2^L t - j), \tag{105}$$

and let V_L be the function space generated by $\{\phi_{L,j} | j \in \mathcal{Z}\}$ where $\mathcal{Z}$ is set of all integers. Again we have

$$V_{L-1} \subset V_L. \tag{106}$$

The two scale relationship for the B-spline scale function can be obtained from the well known B-spline knot insertion algorithm [5, 12], which is

$$\phi(t) = \sum_{i=0}^{4} h_i \phi(2t - i). \tag{107}$$

where the nonzero elements of h is defined in the following:

$$h[0..4] = \frac{1}{8}\{1, 4, 6, 4, 1\}. \tag{108}$$

Substituting $2^{L-1}t - j$ for t, and replacing i by $k - 2j$ on the right hand side, we obtain

$$\phi_{L-1,j} \;=\; \sum_{k=2j}^{2j+4} h_{k-2j}\ \phi_{L,k} \tag{109}$$

See Figure 15.

Similarly there exists a W_{L-1} so that $V_L = V_{L-1} \dot{+} W_{L-1}$, and there exists a wavelet function so that its scaling and translations form a basis of W_{L-1}. Actually there are

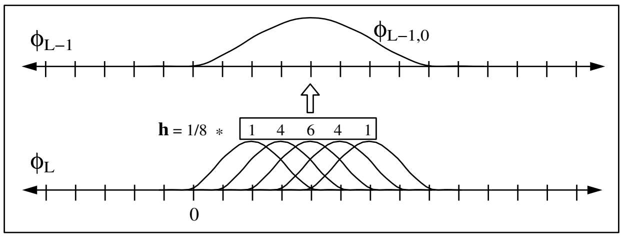


Figure 15: Five B-splines $\phi_{L,j}$ may be combined using the weights h to construct the double width B-spline $\phi_{L-1,0}$

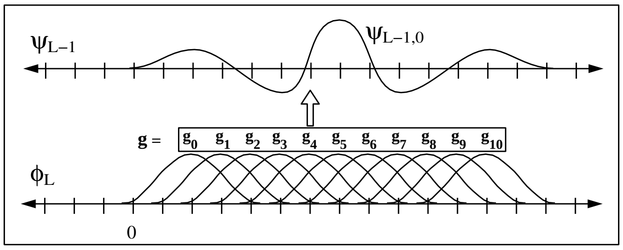


Figure 16: Eleven B-splines $\phi_{L,j}$ may be combined using the weights g to construct the wavelet function $\psi_{L-1,0}$

more than one such wavelet function. The one we give below is called Chui-Wang B-wavelet [12]. The two scale relationship for Chui-Wang B-wavelet function is

$$\psi(t) = \sum_{i=0}^{10} g_i \phi(2t - i). \tag{110}$$

where the sequence g is given in the following (see [40, 50]):

$$g[0..10] = \tfrac{1}{8!} * \{ \begin{array}{l} -1, 124, -1677, 7904, -18482, 24264, -18482, 7904, \\ -1677, 124, -1\}. \end{array} \tag{111}$$

Substituting $2^{L-1}t - j$ for t, and replacing i by $k - 2j$ on the right hand side, we obtain

$$\psi_{L-1,j} \;=\; \sum_{k=2j}^{2j+10} g_{k-2j}\ \phi_{L,k} \tag{112}$$

See Figure 16.

Similar to (91) in Haar wavelet case, one can also represent $\phi_{L,j}$ as a combination of $\phi_{L-1,j}$ and $\psi_{L-1,j}$ as the following:

$$\phi_{L,j} = \sum_k \tilde{h}_{j-2k}\ \phi_{L-1,k} + \sum_k \tilde{g}_{j-2k}\ \psi_{L-1,k} \tag{113}$$

To see how to compute $\tilde{h}$ and $\tilde{g}$, we need to introduce the concept of dual scaling functions and dual wavelets.

The functions $\tilde{\phi}_{L,j}$ are called *dual* to $\phi_{L,k}$ (or *dual scaling functions*) if

$$\langle \tilde{\phi}_{L,j}, \phi_{L,k} \rangle = \delta_{j,k}, j, k \in Z \tag{114}$$

where

$$\delta_{j,k} = \begin{cases} 1 & j = k \\ 0 & j \neq k \end{cases}$$

is called the *Kronecker delta.*

The function $\tilde{\psi}_{L,j}$ are called *dual* to $\psi_{L,k}$ (or *dual wavelets*) if

$$\langle \tilde{\psi}_{L,j}, \psi_{L,k} \rangle = \delta_{j,k}, j, k \in Z \tag{115}$$

Taking inner products in (113) with $\tilde{\phi}_{L-1,k}$ and $\tilde{\psi}_{L-1,k}$ respectively, we obtain

$$\tilde{h}_{j-2k} = \langle \phi_{L,j}, \tilde{\phi}_{L-1,k} \rangle, \tag{116}$$

and

$$\tilde{g}_{j-2k} = \langle \phi_{L,j}, \tilde{\psi}_{L-1,k} \rangle, \tag{117}$$

The derivation of the dual scaling functions and the dual wavelets are described in [12]. The sequences $\tilde{h}$ and $\tilde{g}$ have infinite length but decay quickly from their centers.

The pyramid transformation between the B-spline coefficients and the wavelet coefficients can be derived in the same way as in the Haar wavelets. Similar to (92) and (93), a function in V_L has two representations:

$$f(t) = \sum_j c_{L,j} \phi_{L,j}, \tag{118}$$

and

$$f(t) = \sum_j c_{L-1,j} \phi_{L-1,j} + \sum_j w_{L-1,j} \psi_{L-1,j}. \tag{119}$$

Comparing (118) and (119), and substituting (109) and (112) into (119), we obtain

$$\sum_j c_{L,j} \phi_{L,j} = \sum_j c_{L-1,j} \sum_{k=2j}^{2j+4} h_{k-2j} \ \phi_{L,k} + \sum_j w_{L-1,j} \sum_{k=2j}^{2j+10} g_{k-2j} \ \phi_{L,k} \tag{120}$$

Exchanging the summation orders, we obtain

$$\sum_j c_{L,j}\phi_{L,j} = \sum_k \sum_{k/2-2\le j\le k/2} c_{L-1,j}h_{k-2j}\phi_{L,k} + \sum_k \sum_{k/2-5\le j\le k/2} w_{L-1,j}g_{k-2j}\phi_{L,k} \tag{121}$$

Therefore

$$c_{L,k} = \sum_{k/2-2\le j\le k/2} c_{L-1,j}h_{k-2j} + \sum_{k/2-5\le j\le k/2} w_{L-1,j}g_{k-2j} \tag{122}$$

Similarly if we substitute (113) into (119), and comparing (118) and (119), we obtain

$$c_{L-1,j} = \sum_j c_{L,j}\tilde{h}_{j-2k}, \tag{123}$$

and

$$w_{L-1,j} = \sum_j c_{L,j}\tilde{g}_{j-2k}. \tag{124}$$

2.9.3 B-spline Wavelets on a Bounded Interval

The B-spline wavelets described in the last section are used to represent functions which are defined over the whole real line $\mathcal{R}$. But in an animation context, only functions over some fixed finite interval of time need to be expressed, and, from the efficiency point of view, it is important to only deal with a finite number of basis functions.

Without loss of generality, we assume all the functions are defined over $[0,1]$. For a function $f(t)$ defined over a general interval $[a,b]$, one can define $g(t) = f((t-a)/(b-a))$ so that $g(t)$ is defined over $[0,1]$.

Since we are only interested in the interval $[0,1]$, those basis functions which are zero's in $[0,1]$ can be ignored. Let $V_L|_{[0,1]}$ denote the space of the functions of V_L truncated on $[0,1]$. It is easy to check that only for $j \in [-3, 2^L-1]$, $\phi_{L,j}(t)$'s support intersects $[0,1]$. And these functions are linearly independent. So the basis functions of $V_L|_{[0,1]}$ are $\{\phi_{L,j} | j = -3, ..., 2^L-1\}$, and hence the dimension of $V_L|_{[0,1]}$ is 2^L+3. Among this basis, there are 2^L-3 functions, $\phi_{L,j}, j \in [0, 2^L-4]$, whose supports are entirely inside $[0,1]$. They are called *inner* B-spline basis functions. And there are 3 functions truncated on each of the two boundaries, which are called *left* and *right boundary* B-spline basis functions. For the boundary basis functions, an alternative way is to use special boundary basis functions that arise from placing quadruple knots at the boundaries [5]. This complete set of basis functions will be denoted $\phi_{L,j}$ with j in $[-3, 2^L - 1]$, where it is understood that the first and last three basis functions are the special boundary B-spline basis functions.

The wavelet basis for $W_{L-1}|_{[0,1]}$ also contains *inner* wavelet basis functions and *boundary* wavelet basis functions. The inner basis functions are $\psi_{L-1,j}$, $j = 0, ..., 2^{L-1} - 7$, as defined in (112). These inner basis functions lie completely inside $[0, 1]$. There are three special boundary wavelet basis functions on each side. We denote these basis functions as $\psi_{L-1,j}$ $j \in [-3, 2^{L-1} - 4]$, with the understanding that the first and last three basis functions are the special boundary wavelet basis functions which will be described below. A full description of this construction is given in [11, 50].

The two scale relationships for the boundary B-spline basis functions are

$$\begin{aligned} \phi_{L-1,j} &= \textstyle\sum_{k=j}^{2j+4} h_{j,k}\phi_{L,k}, j \in [-3, -1] \\ \phi_{L-1,2^{L-1}-(4+j)} &= \textstyle\sum_{k=j}^{2j+4} h_{j,k}\phi_{L,2^L-4-k}, j \in [-3, -1] \end{aligned} \tag{125}$$

where $h_{j,k}$ are given below:

$$\begin{aligned} h_{-3,[-3,-2]} = &\quad \tfrac{1}{16} * \{16, 8\}, \\ h_{-2,[-2,0]} = &\quad \tfrac{1}{16} * \{8, 12, 3\}, \\ h_{-1,[-1,2]} = &\quad \tfrac{1}{16} * \{4, 11, 8, 2\}. \end{aligned} \tag{126}$$

The two scale relationships for the boundary wavelet basis functions are

$$\begin{aligned} \psi_{L-1,j} &= \textstyle\sum_{k=-3}^{10+2j} g_{j,k}\phi_{L,k}, j \in [-3, -1] \\ \psi_{L-1,2^{L-1}-(7+j)} &= \textstyle\sum_{k=-3}^{10+2j} g_{j,k}\phi_{L,2^L-4-k}, j \in [-3, -1] \end{aligned} \tag{127}$$

where $g_{j,k}$ are given below (see [40, 50]):

$$\frac{1}{8!} * \begin{array}{c} \begin{array}{ccc} \quad g_{-3} \quad & \quad g_{-2} \quad & \quad g_{-1} \quad \end{array} \\ \begin{pmatrix} \frac{1136914560}{27877} & -\frac{2387315040}{195139} & \frac{123066720}{1365937} \\ -\frac{1655323200}{27877} & \frac{2141121840}{195139} & -\frac{2226000}{1365937} \\ \frac{1321223960}{27877} & \frac{878161880}{195139} & \frac{188417600}{1365937} \\ -\frac{633094403}{27877} & -\frac{498772701}{27877} & -\frac{2293862247}{1365937} \\ \frac{229000092}{27877} & \frac{4726413628}{195139} & \frac{10796596516}{1365937} \\ -\frac{46819570}{27877} & -\frac{3606490941}{195139} & -\frac{25245248833}{1365937} \\ 124 & 7904 & 24264 \\ -1 & -1677 & -18482 \\ 0 & 124 & 7904 \\ . & -1 & -1677 \\ . & 0 & 124 \\ . & . & -1 \\ . & . & 0 \\ . & . & . \end{pmatrix} \end{array} \tag{128}$$

Similar to the equation (101), one can obtain the decomposition of V_L as

$$V_L = V_3 \dot{+} \sum_{l=3}^{L-1} W_l \tag{129}$$

Here the starting smooth function space is V_3 instead of V_0 as in the case of Haar wavelet basis. The reason is because we need to make sure the left boundary wavelet basis functions don't intersect the right boundary functions. The corresponding basis consists of the functions $\{\phi_{3,k} | k \in [-3, 7]\} \cup \cup_{l=3}^{L-1} \{\psi_{l,k} | k \in [-3, 2^l - 4]\}$.

Given the coefficients $\{c_{L-1,j} | j \in [-3, 2^{L-1} - 1]\}$ and $\{w_{L-1,j} | j \in [-3, 2^{L-1} - 4]\}$, the procedure `coef_xfrom_down` as defined below computes the coefficients $\{c_{L,j} | j \in [-3, 2^L - 1]\}$ by using (109), (112), (125) and (127):

```
coef_xform_down( c_in[], w_in[], c_out[], L )
    c_out = 0 ; /* zero vector */
    for(k = 0; k ≤ 2^{L-1} - 4; k++)
        for(j = 2k; j ≤ (2k + 4); j++)
            c_out[j] += h[j - 2k] * c_in[k] ;
    for(k = 0; k ≤ 2^{L-1} - 7; k++)
        for(j = 2k; j ≤ (2k + 10); j++)
            c_out[j] += g[j - 2k] * w_in[k] ;
    for(k in [-3,-2,-1,2^{L-1} - 3,2^{L-1} - 2,2^{L-1} - 1] )
            c_out    += h_k * c_in[k] ; /*vector addition*/
    for(k in [-3,-2,-1,2^{L-1} - 4,2^{L-1} - 5,2^{L-1} - 6] )
            c_out    += g_k * w_in[k] ;
```

where

$h_{2^{L-1}-j,k} = h_{j-4,2^L-4-k}$, $2^L - 2j \leq k \leq 2^L - j$, $j = 1, 2, 3$,

$g_{2^{L-1}-j,k} = g_{j-7,2^L-4-k}$, $2^L - 2j \leq k \leq 2^L - 1$, $j = 1, 2, 3$.

The above procedure can be expressed as multiplication by a banded matrix, and so the inverse procedure `coef_xform_up(` c_{in}`[]`, c_{out}`[]`, w_{out}`[]`, L `)` can be obtained by solving this banded linear system.

The transformation from the B-spine coefficients to the full wavelet basis coefficients can be done by using the procedure `coef_pyrm_up`, that makes $L - 3$ calls to `coef_xform_up` each time with an input vector of $\frac{1}{2}$ the length. (Note since this transforms an $2^L + 3$-vector to an $2^L + 3$-vector, it can be implemented with proper indexing using linear storage).

```
coef_pyrm_up( c_in[], c_out[], w_out[][], L )
    c_temp[L][] = c_in[] ;
    for( i = L; i >= 4; i-- )
        coef_xform_up( c_temp[i][], c_temp[i-1][],
                       w_out[i-1][], i ) ;
    c_out[] = c_temp[3][] ;
```

The inverse transformation is

```
coef_pyrm_down( c_in[], w_in[][] , c_out[], L )
    c_temp[3][] = c_in[] ;
    for( i = 4; i <= L; i++ )
        coef_xform_down( c_temp[i-1][], w_in[i-1][],
                         c_temp[i][], i ) ;
    c_out[] = c_temp[L][] ;
```

The running time of these pyramid transformations is governed by the geometric series $(2^L + \frac{2^L}{2} + \frac{2^L}{4} + \ldots + 1) = O(2^L)$, so they run in linear time.

2.10 Nonlinear Optimization

2.10.1 Unconstrained Optimization

An unconstrained minimization problem has the form

$$\text{minimize } F(\mathbf{c}), \mathbf{c} \in \mathcal{R}^n \tag{130}$$

where $F(\mathbf{c}) = F(c_1, c_2, ..., c_n)$ is a multivariate function, and $\mathcal{R}^n$ is the n-dimensional Euclidean space. A point $\mathbf{x}^*$ is called a *local minimum point* if $f(\mathbf{x}^*)$ has the smallest value in some neighborhood of $\mathbf{x}^*$. $\mathbf{x}^*$ is called a *global minimum point* if $f(\mathbf{x}^*)$ has the smallest value in the whole space $\mathcal{R}^n$.

We will only discuss the method of finding local minimum points. The problem of finding the global minimum point is considerably more difficult and no practical methods have been found so far. The interested reader is referred to [18].

The necessary conditions for $\mathbf{x}^*$ to be a local minimum point is that $g(\mathbf{x}^*) = 0$, where $g(\mathbf{x})$ is the gradient of F at $\mathbf{x}$ [21]. Such a point, at which the gradient is zero, is called a *stable point.* A local minimum point must be a stable point, but the stable point may not be a local minimum point. However, almost all the known nonlinear optimization methods can only guarantee to find a stable point.

The simplest method is the gradient decent method:

Initialization: let $\mathbf{c_0}$ be the initial guess. and $k = 0$.

Step 1: compute the gradient g_k of $f(c)$ at c_k. If $g_k = 0$, terminate (the stable point has been found). Otherwise, let $s_k = -g_k$.

Step 2: find $\alpha_{(k)}$ to minimize $F(c_k + \alpha s_k)$.

Step 3: set $c_{k+1} = c_k + \alpha_k s_k$.

Step 4: Stop if termination condition holds. Otherwise, let k := k+1 and go to Step1.

Notice that $\frac{dF(c_k+\alpha s_k)}{d\alpha}|_{\alpha=0} = g_k^T s_k = -g_k^T g_k < 0$. So $F(c_k+\alpha s_k)$ is decreasing when α is close to zero. Such a direction s_k is called a *decent direction.* From step 2 and step 3,

$$F(c_{k+1}) < F(c_k). \tag{131}$$

This ensures that the function value strictly decreases at each step. When $g_k = 0$, then we have reached a stable point, so we are done. In practice, we usually want to terminate when the solution is close to a minimum point. The choice of termination conditions depend on the applications. Some possibilities are (1) $|F(c_{k+1})-F(c_k)| < \epsilon$, or (2) $||g_k|| < \epsilon$, or (3) $||c_{k+1} - c_k|| < \epsilon$. The reader is referred to [21] for other possibilities and discussions.

It can be proved that, under minor condition, this method converges for any chosen $\mathbf{c_0}$ [21]. Such property is called *global convergence.* Another advantage of this method is that it only requires computing the first order derivatives. Compared to other methods which require second order derivatives, this method is a lot cheaper.

The rate of convergence, however, is only linear. Especially when the iterative solutions get close to the final solution, the solution may tend to bounce back and force. This "zigzag" phenomenon can become serious and result in poor convergence.

A faster method, which uses second order derivatives, is Newton's method. Newton's method is essentially a root finding method applied to the gradient function. The scheme is the same as the gradient decent method except s_k is different. Here $s_k = -G_k^{-1} g_k$ where G_k is the Hessian matrix. This method has quadratic convergence, but it may not converge for some initial guesses. The reason is the following. To ensure convergence, the objective value has to decrease at each iteration, i.e., the search direction $s_k = -G_k^{-1} g_k$ has to be a decent direction. This requires G_k to be positive definite. Therefore, in order to ensure the convergence, the initial guess $\mathbf{c}_0$ has to be chosen such that the Hessian matrix at $\mathbf{c}_0$ and all the subsequent iterative points are all positive definite. For simplicity, let's assume that the Hessian matrix at the optimal solution is positive definite. By continuity (assuming that the objective function has second continuous derivatives), there exists a neighborhood of the optimal solution, such that the Hessian matrix at any point in this region is also positive definite. If we pick up $\mathbf{c}_0$ from this region, then it can be shown that all the subsequent iterative points are also in this region. Therefore the convergence is guaranteed. For a more general discussion on the convergence criteria, the reader is referred to [21]. Since Newton's method convergences only when c_0 is in some neighborhood of the optimal solution, we say it *converges locally*.

Even though Newton's method can be modified to ensure global convergence, its major disadvantage is that it requires computing second order derivatives which may be expensive in practice.

Quasi-Newton methods are a class of methods with convergence rate close to Newton's method but only require first order derivatives. This type of method is like Newton's method, except that G_k^{-1} is approximated by a symmetric positive definite matrix H_k which is corrected or updated from iteration to iteration. One such method, which has been used successfully in practice is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. In the BFGS method, H_0 is chosen by the user which should be a positive definite matrix (e.g. the identity) to ensure that $s_0 = -H_0 g_0$ is a decent direction. At each iteration, H_k is updated by using the formula

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \left(1 + \frac{\gamma^T \mathbf{H}_k \gamma}{\delta^T \gamma}\right) \frac{\delta \delta^T}{\delta^T \gamma} - \left(\frac{\delta \gamma^T \mathbf{H}_k + \mathbf{H}_k \gamma \delta^T}{\delta^T \gamma}\right) \tag{132}$$

where $\gamma = \nabla F_{k+1} - \nabla F_k$ (the change in the gradient of the cost function) and $\delta = \mathbf{c}_{k+1} - \mathbf{c}_k$ (the change in the coefficients).

It can be shown that as long as H_0 is positive definite, then all the subsequent H_k's are also positive definite, and as a result, s_k's are all decent directions. Therefore when H_0 is chosen to be positive definite such as the identity, BFGS method converges for any chosen $\mathbf{c_0}$.

2.10.2 Line Search

No matter whether we use gradient decent method or quasi-Newton method, at step 2 of each iteration, we need to find $\alpha_k > 0$ which minimizes $F(\mathbf{c_k} + \alpha s_k)$. This step is called a *line search*. There are a variety of line search methods as discussed in Fletcher's book [21]. In our application, we don't want to compute the gradients during the line search since gradient computation is expensive. Therefore we will only discuss methods which do not need gradient computation. We perform an approximate line search, that is, we seek an α so that $F(\mathbf{c_k} + \alpha s_k)$ sufficiently decreases (from $F(\mathbf{c_k})$). We will describe two such methods: bisection and quadratic interpolation.

For convenience, denote $f(\alpha) = F(\mathbf{c_k} + \alpha s_k)$. For any line search method, the user has to choose an initial guess $\alpha_0 > 0$. To make sure that α_0 is not too small, we can test to see if both

$$f(\alpha_0) < f(0) \tag{133}$$

and

$$f(\alpha_0) < f(\alpha_0/2). \tag{134}$$

If one of them is not true, then we know that there is a minimum point in the interval $[0, \alpha_0]$. Otherwise, we can double α_0 to see if the new α_0 (after doubled) satisfies (133) and (134). Repeat this process until we find a α_0 which doesn't satisfy both of

```
float bisectLineSearch(float α0)
{
        float a, b;
        a=0;
        b=α0;
        while(termination condition is not true)
        {
            if(f(a)<f(b))
                b = (a+b)/2;
            else
                a = (a+b)/2;
        }
        return (f(a)<f(b)? a:b);
}
```

Figure 17: Line search: bisection method

them. Notice that theoretically for some functions it may take forever to find such α_0, but in our experiments, it can always be found easily.

The simplest line search method is the bisection method. Bisection method works by starting with $[a, b]$, where $a = 0$ and $b = \alpha_0$, and compare $f(a)$ and $f(b)$. If $f(a) < f(b)$ then let the new interval be $[a, (a+b)/2]$, otherwise let the new interval be $[(a+b)/2, b]$, and repeat until some prespecified termination condition is satisfied (Figure 17).

The pseudo code for the quadratic interpolation method is given in Figure 18.

This method seeks an α^k so that $f(\alpha^k) \leq f^0 + 0.5\alpha^k \dot{f}$ where $f^0 := f(0)$, and $\dot{f} := \frac{df(0)}{d\alpha}$. Notice that such α^k must exist since when α approaches 0, $f(\alpha) \approx f^0 + \alpha\dot{f} < f^0 + 0.5\alpha\dot{f}$. If $f(\alpha^k) \leq f^0 + 0.5\alpha^k\dot{f}$, then $f(\alpha^k)$ has decreased sufficiently, so the search stops. Otherwise, we compute the minimum point of the quadratic function interpolated from f^0, f^k and $\dot{f}$. The interpolated quadratic function is $f^0 + f^1\alpha + \frac{f^1 - f^0 - \dot{f}\alpha_1}{\alpha_1^2}\alpha^2$. The minimum point is $-\frac{\dot{f}}{2\frac{f^1 - f^0 - \dot{f}\alpha_1}{\alpha_1^2}}$ which is used as α_{k+1} in the algorithm.

Now we show that $0 < \alpha_{k+1} < \alpha^k$ so that we can repeat this process.

float quadraticInterpolationLineSearch

{

  float $\dot{f} := \frac{df(0)}{d\alpha}$;

  float $f^0 := f(0)$;

  int $k := 1$;

  α^i $(i \geq 1)$ is an array storing the α values in the iterations

  f^i $(i \geq 1)$ is an array to denote $f(\alpha^i)$.

  $\alpha^1 := \alpha_0$;

  while(1)

  {

    $f^k := f(\alpha^k)$

    if($f^k \leq f^0 + \frac{\alpha^k}{2}\dot{f}$)

      return α^k;

    else

    {

      $\alpha^{k+1} := \frac{-\dot{f}}{2\frac{f^k - f^0 - \dot{f}\alpha^k}{(\alpha^k)^2}}$.

      $k++$;

    }

  }

}

Figure 18: Line search: quadratic interpolation

In fact, since $f(\alpha^k) > f^0 + 0.5\alpha^k \dot{f}$, from the definition of α_k we have

$$\alpha^{k+1} > 0, \tag{135}$$

and by arranging terms, one obtains

$$\dot{f} < \frac{2(f^k - f^0)}{\alpha^k}. \tag{136}$$

Adding $-2\dot{f}$ to both sides, one obtains

$$\begin{aligned} -\dot{f} &< \tfrac{2(f^k - f^0)}{\alpha^k} - 2\dot{f} \\ &= 2\alpha^k \tfrac{f^k - f^0 - \alpha^k}{(\alpha^k)^2}. \end{aligned} \tag{137}$$

Dividing both sides by $2\frac{f^k - f^0 - \alpha^k}{(\alpha^k)^2}$, we have

$$\alpha^{k+1} < \alpha^k. \tag{138}$$

2.10.3 Constrained Optimization

A constrained minimization problem has the form

$$\begin{aligned} &\text{minimize} \quad F(\mathbf{c}) \\ &\text{subject to} \\ &\qquad\qquad C_i(\mathbf{c}) = 0, i = 1, 2, ..., m_1 \\ &\qquad\qquad D_i(\mathbf{c}) \geq 0, i = 1, 2, ..., m_2 \end{aligned} \tag{139}$$

where $F(\mathbf{c})$ is the objective function to minimize, the C_i's are called *equality constraints*, and the D_i's are called *inequality constraints.*

A constrained optimization problem can be transformed into an unconstrained one using penalty functions. such as

$$\phi = F + \sum_{i=1}^{m_1} w_{1i} C_i^2 + \sum_{i=1}^{m_2} w_{2i} (min(D_i, 0))^2,$$

where w_{1i} and w_{2i} are positive numbers, called *weights.* It can be shown that, as w_{1i} and w_{2i} tend to ∞, the stable points of ϕ approach the stable points of the problem 139 (for the constrained optimization problem, the meaning of stable point has to be modified. This leads to the definition of Kuhn-Tucker conditions [21] which is not discussed in this thesis).

After the transformation, we can apply the unconstrained optimization method as discussed previously.

Another more direct method is called *sequential quadratic programming*. This consists of a sequence of quadratic subproblems (which are second order approximations of the original problem) making a series of Newton steps towards a solution. The attraction of the method is its quadratic convergence. But there are two disadvantages. First the convergence is local, meaning that this method converges only when the initial guesses are sufficiently close the optimum solution. When the initial guesses are not close to the optimal solution, this method may diverge. However, there are some ways to modify this method to ensure global convergence (with the sacrifice of the convergence rate [21]). The second and more serious disadvantage is that this method requires computing explicit Hessian, that is, second order derivatives. In practice, this is often expensive. In our applications as described in the later chapters, computing Hessians is considerably more expensive than computing gradients. However, this method has been used to solve spacetime constraint problems [56, 13]. When the figure becomes more complex, the Hessian computation alone can become computationally forbidding. Thus, we use methods which only need gradients, such as the quasi-Newton methods.

There are other search methods which do not require gradients such as stochastic search [54], genetic search [43], etc. At each iteration of these search methods, the next solution is found with some randomness (the search direction may be randomly chosen or the new solution is generated through some random process). Because of the lack of gradients, these methods do not have a decent nature meaning that the objective value can go up and down during the solution process. Thus the convergence can not be guaranteed, and these methods usually require more iterations than the methods using gradient information.

2.11 Interactive Spacetime Constraint System

The interactive spacetime constraint system, introduced by Cohen [13] in 1992, has the structure as shown in Figure 19. In this system, the user can interact with the

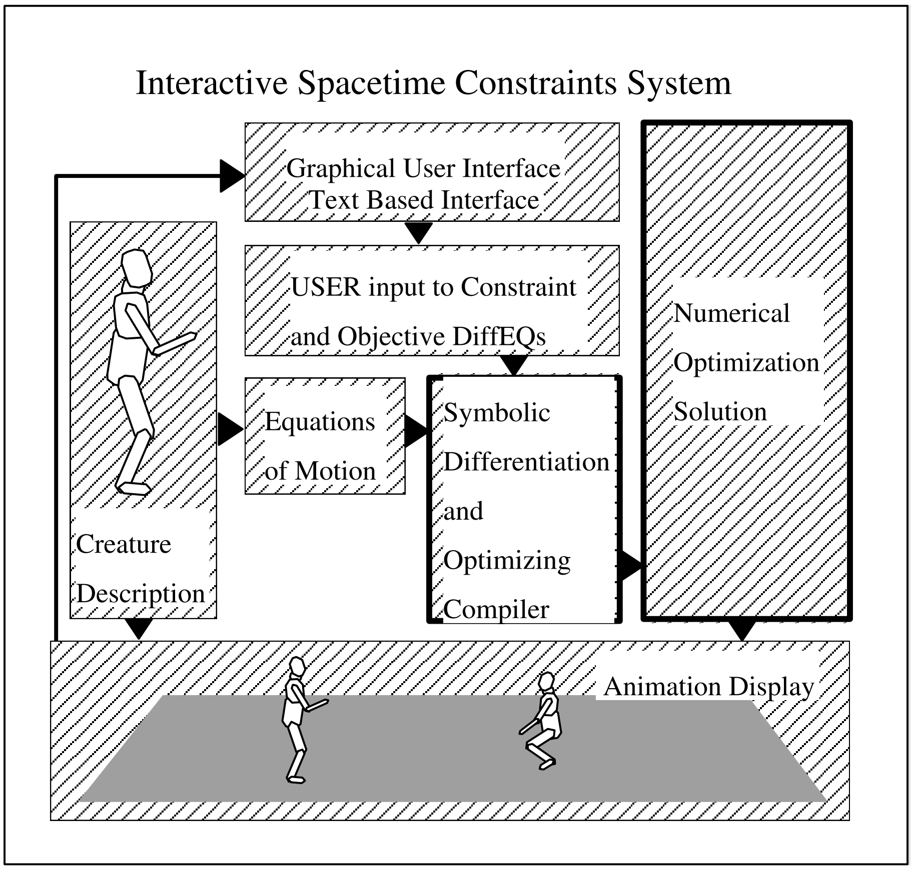
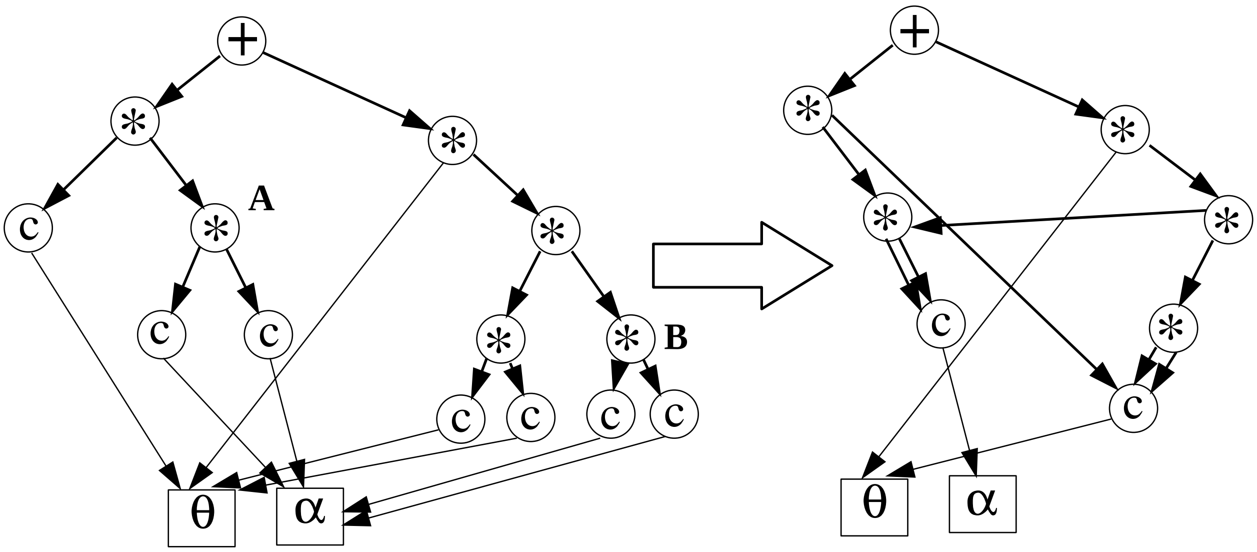


Figure 19: The Interactive Spacetime Constraints System

$$\cos(\theta) * \cos^2(\alpha) + \theta * \cos^2(\theta) * \cos^2(\alpha)$$



(a) (b)

Figure 20: The number of nodes to be evaluated is greatly decreased by common subexpression extraction. The "c" nodes represent the cosine operator.

iterative numerical optimization and can guide the optimization process to converge to an acceptable solution.

The user specified constraints can be typed in by the user (or read from an input file), and physical equations of motion can be generated automatically. The symbolic expressions, which formulate an optimization problem, are parsed and the evaluation trees generated. A symbolic differentiation is applied to these evaluation trees and generates new evaluation trees (called *differentiation trees*) for gradient (and Hessian) computation.

After the preprocessing stage is done, the user can provide an initial guess by using a keyframing system and start the optimization process. At each iteration, the user can look at the resulting motion, and can modify the current solution, change the optimization parameters such as weights in the penalty function, and delete or insert constraints.

2.12 Compilation, Common Subexpression Elimination, and Symbolic Differentiation

The compiler is responsible for parsing the expressions of the constraints and objective and generating evaluation trees for the numerical optimization phase. It begins with a bottom up parser that reads the expressions (objective and constraints) and produces an evaluation tree for each expression (Figure 20 (a)). Formally speaking, an evaluation tree is a directed acyclic graph in which each leaf represents either a constant or a variable (DOF variables and there derivatives), and each inner node represents a function which can be binary operators `+, -, *, or /`, or the unary `minus, sin, cos, tan, cot, exp, log`, and `∧a` (power) where `a` is any real number. For each node T, `L(`T`)` and `R(`T`)` represent the two children of T.

The evaluation is carried out by inserting the DOF values in the leaves of the tree and recursively evaluating nodes upward until the value of the expression is contained at the topmost node.

One bottleneck in the spacetime constraint system is that the symbolic expressions are usually very long resulting large evaluation trees. For example, even a planar three-link arm can result in evaluation trees containing over 145,000 nodes that must be evaluated multiple times during the optimization process. To reduce the size of the evaluation trees, the compiler optimization technique of *common subexpression elimination* (CSE) can be used. More specifically, common subexpressions are extracted by recursively moving from the leaves to the top node, making a list of unique nodes and checking each new node against the list. In the case of leaf nodes that are always variable ID's, this is simple, for internal function nodes their children must be checked, and in the case of commutative operations both possibilities `((left==left && right==right) || (left==right && right==left))` must be checked. If the nodes are determined to be equivalent, one node is used for both (Figure 20 (b)).

Given an evaluation tree T and a variable x , the derivative of T with respect to x, $\partial T/\partial x$ is recursively defined as shown in Figure 21. Simply by enumerating each case, it can be seen that any single differentiation generates at most 5 output nodes for each input node, thus the growth due to differentiation is $O(n)$ with a constant

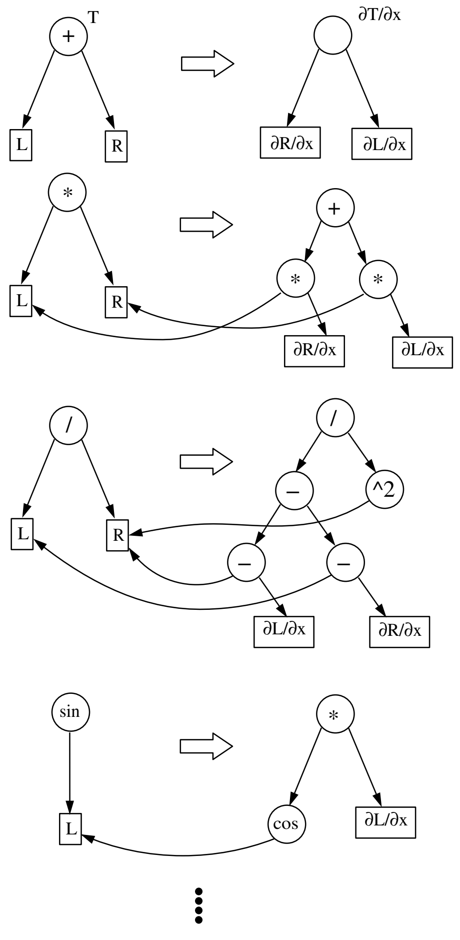


Figure 21: Differentiation Rules (the unary operators are not all enumerated as many are similar).

factor of 5. In fact, since in the animation application the divide is rare, the average growth is much less than a factor of 5.

As an example of the power of the common subexpression elimination, the evaluation trees for a planar three-link arm without CSE contained 145,584 nodes, compared to 2,932 nodes after CSE!

2.13 Summary

We have introduced the basic idea of the interactive spacetime constraints system and the necessary background for the numerical and symbolic computations. While the interactive spacetime constraints systems do provide physical realism and certain user control, they have been computationally expensive. In the symbolic phase, even with common subexpression elimination, the sizes of the evaluation trees still grow exponentially in terms of the number of degrees of freedoms as we will show in the next chapter. The numerical optimization phase is another bottleneck because the number of unknowns in the resulting nonlinear optimization problem is usually large and each iteration usually requires numerical integrations. We will present both symbolic and numerical techniques to address these issues in the next three chapters.

Chapter 3

Efficient Symbolic Interface

In this chapter, we describe a new symbolic interface for optimization based animation system. The materials are from the paper [38].

3.1 Introduction

As described in Chapter 2, symbolic methods have been used to represent constraints and objectives in many optimization based animation systems [56, 13]. The major advantage of symbolic methods is that they are general enough to represent various kinds of constraints and objectives and the gradients and Hessians can be obtained automatically by symbolic differentiation. For example, without symbolic methods, all possible types of constraints must be hard coded as in Zhao and Badler's [57] inverse kinematics system. In addition, gradients have to be derived manually which is very tedious even when the expressions are small, and almost impossible when the expressions become complex. Another advantage is that the user can modify the expressions such as the constraint and objection functions at runtime.

However, one major disadvantage of previous symbolic systems is that the length of the resulting symbolic expressions are exponential in the depth of the tree of degrees of freedom (DOFs). This problem has been noticed and subexpression elimination techniques have been used to reduce the sizes of the evaluation trees as described in Section 2.12 ([33] also gives similar techniques). However, this does not fully solve the problem since the size of the evaluation trees after subexpression elimination are

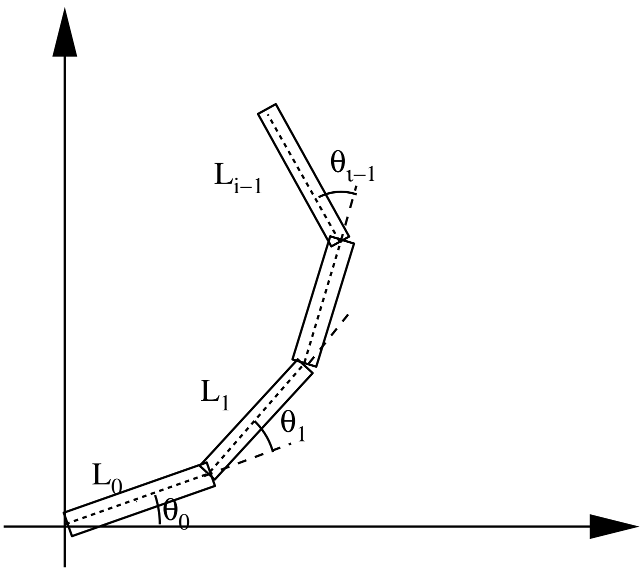


Figure 22: The planar i-link chain

still exponential in general. To see why, let's consider a simple example. Suppose we have a planar i-link chain as shown in Figure 22. Using the terminology in Chapter 2, R_i^{i-1}, the orientation matrix of link i with respect to the local coordinate system of link $i-1$, is equal to

$$\begin{pmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{pmatrix} \tag{140}$$

Therefore the orientation matrix of link i with respect to world coordinate system is

$$R_i = R_0 R_1^0 R_2^1 ... R_i^{i-1} \tag{141}$$

We claim that the elements of R_i contain at least 2^i many different summation terms. For example,

$$R_2 = \begin{pmatrix} cos(\theta_0)cos(\theta_1) - sin(\theta_0)sin(\theta_1) & -cos(\theta_0)sin(\theta_1) - sin(\theta_0)cos(\theta_1) \\ sin(\theta_0)cos(\theta_1) + cos(\theta_0)sin(\theta_1) & -sin(\theta_0)sin(\theta_1) + cos(\theta_0)cos(\theta_1) \end{pmatrix} \tag{142}$$

contains the four different terms $cos(\theta_0)cos(\theta_1)$, $cos(\theta_0)sin(\theta_1)$, $sin(\theta_0)cos(\theta_1)$,and $sin(\theta_0)sin(\theta_1)$. In general, it is easy to show by induction that R_i contains all of the terms with the form $f_0(\theta_0)f_1(\theta_1)...f_{i-1}(\theta_{i-1})$, where each f_j can be either a sin or cos function. In total there are 2^i such terms. Notice that the common subexpression elimination would not eliminate any of these 2^i terms since no two terms are the same.

Therefore even after common subexpression elimination, R_i will contain at least 2^i terms. However, if we don't expand R_i symbolically but instead evaluate matrices $R_0, R_1^0, ..., R_i^{i-1}$ first and then do matrix multiplication numerically we can compute R_i in linear time.

This suggests that expanding the matrices as explicit symbolic functions of DOFs is not a good idea unless the numerical matrix multiplication is more expensive than expanding and evaluating the full expressions. We will see that this is not the case except in the most trivial examples. We propose to use position vectors and orientation matrices as first class symbolic variables in the symbolic expressions, (i.e., we don't expand them). Together with a few special operators that act on matrices and vectors which will be defined later in this chapter, we will show that with this paradigm common kinematic and dynamic expressions can be represented as a small number of operations on the joint positions and link orientations. These expressions are usually small enough to be simply typed in, and more importantly, there are very efficient ways to evaluate these expressions.

A similar insight has been used by Hollerbach [29] in developing a fast dynamics algorithm. Our contribution is that we extend the idea to the constrained optimization arena and integrate the numerical techniques into a symbolic interface to achieve an efficient and general symbolic interface to the optimization based animation system.

3.2 The Language

The full syntax of an expression is shown below by using YACC notations. The parser is generated with LEX and YACC [1]. It generates an evaluation tree for each expression. The structure of an evaluation tree is similar to what we have seen in Section 2.12 except that there are new types of variables and operations.

```
expr     :  expr '+' expr
         | expr '-' expr
         | expr '*' expr
         | expr '/' expr
```

```
          | '(' expr ')'
          | '-' expr %prec UMINUS
          | fun '(' expr ')'
          | SUMA '[' ID ']' '(' expr ')' /*sum over all joint angles*/
          | SUML '[' ID ']' '(' expr ')' /*sum over all links*/
          | ITG '[' const ',' const ']' /*integral*/
          | VDOT '(' ID ',' expr ',' expr ')'
          | VCROSS '(' ID ',' expr ',' expr ')'
          | variable
          | const
          | ROWV '[' const ',' const ',' const ']' /*row vector*/
          | COLV '[' const ',' const ',' const ']' /*column vector*/
          ;
fun       :  SIN
          | COS
          | TG
          | CTG
          | EXP
          | LN
          | '^ ' NUMBER /*exponent*/
          | SQR /*square*/
          | TRANS /*transpose*/
          ;
variable :  dofVar
          | posVar
          | oriVar
          | torqueVar
          ;
const     :  ID
          | NUMBER
          | ID '_' ID
          ;
```

```
dofVar    :  DOF '_' NUMBER '[' order ',' time ']'
          ;
posVar    :  POS '_' ID '[' der ',' time ',' NUMBER ',' NUMBER ',' NUMBER ']'
          | POS '_' ID '[' der ',' time ']'
          ;
oriVar    :  ORI '_' ID '[' der ',' time ']'
torqueVar:  TORQUE '_' ID '[' time ']'
          ;
torqueVar:  TORQUE '_' NUMBER '[' time ']'
          ;
order     :  NUMBER
          ;
time      :  ID
          | NUMBER
          ;
der       :  der '|' idOrNum
          | idOrNum
          ;
idOrNum   :  ID
          | NUMBER
          ;
```

In the rest of this section, we will explain in detail the interesting extensions that provide the efficiencies discussed above.

3.2.1 Constants

There are four types of constants: real numbers, 3D row vectors, 3D column vectors, and 3×3 matrices. A row vector has the form **rowv**[x,y,z], a column vector has the form **colv**[x,y,z] representing column vector $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$, and a 3×3 matrix has the form **mat**$[a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}]$.

3.2.2 Variables

Normal scalar variables representing DOF functions and their time derivatives are represented by **dof_***label[d,t]* where *label* is the label (an integer) of the DOF (recall that each DOF has a label, see Section 2.3), d is $0, 1$ or 2 representing the order of the derivative (with respect to time), and t is the time. In addition, three new types of variables are introduced: position variables, orientation variables, and torque variables. A position variable has the form: **pos_***name[der-info, t , x, y, z]* where *name* is the label of the link at which this position is located, t is either a real number representing a specific time or simply the symbol **T** indicating "at all times", (*x,y,z*) is the coordinate of this position in the local coordinate system, and finally *der-info* is a sequence of integers of the form $(i_1, i_2, ..., i_p, j)$ which store the derivative information representing the operator $\frac{\partial^p}{\partial\theta_{i_1}\partial\theta_{i_2}...\partial\theta_{i_p}}\frac{d^j}{dt^j}$. When (x, y, z) represents the center of gravity, it can be omitted. That is, we can simply write **pos_***name[der-info, t]*.

For example, suppose we have three links labeled $L1$, $L2$, and $L3$. If we use $E(t)$ to denote the world coordinate of the end effector at time t. Then $E(t)$, $\dot{E}(t)$, and $\ddot{E}(t)$ can be represented as **pos_**$L3[0, t, ex, ey, ez]$, **pos_**$L3[1, t, ex, ey, ez]$, and **pos_**$L3[2, t, ex, ey, ez]$, respectively, where (ex, ey, ez) is the coordinates of the end effector relative to the local coordinate system of $L3$. The partial derivative $\frac{\partial\dot{E}(t)}{\partial\theta_2}$ can be represented as **pos_**$L3[(2, 1), t, ex, ey, ez]$, $\frac{\partial^2\dot{E}(t)}{\theta_2(t)\theta_3(t)}$ can be represented as **pos_**$L3[(2, 3, 1), t, ex, ey, ez]$. How do we represent the partial derivatives with respect to the time derivatives of angles such as $\frac{\partial\dot{E}(t)}{\partial\dot{\theta}_2(t)}$? Actually $\frac{\partial\dot{E}(t)}{\partial\dot{\theta}_2(t)} = \frac{\partial E(t)}{\partial\theta_2(t)}$. In general, any partial derivatives with respect to time derivatives of angles can always be reduced to partial derivatives with respect to angles (without time derivatives). In most cases where only up to second order derivatives are under consideration, the following formulas are sufficient for the reductions:

$$\frac{\partial E(t)}{\partial\dot{\theta}(t)} = 0, \tag{143}$$

$$\frac{\partial E(t)}{\partial\ddot{\theta}(t)} = 0, \tag{144}$$

$$\frac{\partial\dot{E}(t)}{\partial\dot{\theta}(t)} = \frac{\partial E(t)}{\partial\theta}, \tag{145}$$

$$\frac{\partial \dot{E}(t)}{\partial \ddot{\theta}(t)} = 0, \tag{146}$$

$$\frac{\partial \ddot{E}(t)}{\partial \dot{\theta}(t)} = 2 * \frac{\partial \dot{E}(t)}{\partial \theta(t)}, \tag{147}$$

$$\frac{\partial \ddot{E}(t)}{\partial \ddot{\theta}(t)} = \frac{\partial E(t)}{\partial \theta(t)}. \tag{148}$$

Similarly, an orientation variable has the form **ori_***name[der-info, t]* which denotes the orientation matrix of the link *name* or its derivatives including mixed partial and time derivatives.

Notice that an orientation variable is a 3×3 matrix, while a position variable is a 3D vector. For example, if the orientation matrix of link $L1$ in the world coordinate system is $M_1(t)$, then

$$\frac{dM_1(t)}{dt} = \mathbf{ori\_}L1[1, t]. \tag{149}$$

A torque variable has the form $\mathbf{tor\_}id[t]$, which represents the generalized force with respect to the rotational angle named id at time t. Notice that torque variables are not essential to this symbolic system since torques (generalized forces) can be represented as functions of positions. The reason to introduce the torque variables is to gain the optimum efficiency for computing torques and their gradients since torques are the most expensive quantities. We will show how to extend Hollerbach's technique [29] of computing Lagrangian inverse dynamics to evaluate the gradients of all the torques in $O(m^2)$ time, which is the optimum since we have to evaluate $O(m^2)$ entries. Without such special treatment, it would take $O(m^3)$ time.

3.2.3 Operations

Many operations on scalar variables and constants are defined with the expected syntax and differentiation rules. Similarly, common matrix and vector operations are defined. In addition to normal matrix operations, we need two special operations to represent quantities like kinetic energy and angular momentum of a rigid link. Given two 3×3 matrices M_1 and M_2, and a link L, we define operators **Vdot** and **Vcross** as the following:

$$\mathbf{Vdot}(L, M_1, M_2) = \int (M_1(X - C)) \cdot (M_2(X - C))\rho(X)dX \tag{150}$$

and

$$\mathbf{Vcross}(L, M_1, M_2) = \int (M_1(X-C)) \times (M_2(X-C))\rho(X)dX \tag{151}$$

where X ranges over the coordinates of all the points on the link L in its local coordinate system, C is the local coordinate of its center of mass, and $\rho(X)$ is the mass density at X.

Given a link L with a local coordinate system and assuming its orientation matrix is M, then the rotational energy of L is

$$0.5 \int (\dot{M}(X-C)) \cdot (\dot{M}(X-C))\rho(X)dX = 0.5\mathbf{Vdot}(L, \dot{M}, \dot{M}), \tag{152}$$

and its angular momentum about its center of mass is

$$\int (M(X-C)) \times (\dot{M}(X-C))\rho(X)dX = \mathbf{Vcross}(L, M, \dot{M}). \tag{153}$$

It should be pointed out that the computation of both integrals can be reduced to the computation of inertial tensors as defined in Section 2.5. To see why, denote the three columns of M as M_1, M_2 and M_3, and the three columns of $\dot{M}$ as $\dot{M}_1, \dot{M}_2$ and $\dot{M}_3$. And denote $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ and $C = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$. Then

$$\begin{array}{l} \int (\dot{M}(X-C)) \cdot (\dot{M}(X-C))\tau(X)dX \\ = \int \sum_{i,j=1}^{3} M_i \cdot M_j (x_i - c_i)(x_j - c_j)\rho dx_1 dx_2 dx_3 \\ = \sum_{i,j=1}^{3} M_i \cdot M_j \int (x_i - c_i)(x_j - c_j)\rho dx_1 dx_2 dx_3 \end{array} \tag{154}$$

and

$$\begin{array}{l} \int (\dot{M}(X-C)) \times (\dot{M}(X-C))\tau(X)dX \\ = \int \sum_{i,j=1}^{3} M_i \times M_j (x_i - c_i)(x_j - c_j)\rho dx_1 dx_2 dx_3 \\ = \sum_{i,j=1}^{3} M_i \times M_j \int (x_i - c_i)(x_j - c_j)\rho dx_1 dx_2 dx_3 \end{array} \tag{155}$$

In order to compute $\int (x_i - c_i)(x_j - c_j)\rho dx_1 dx_2 dx_3$ for all $1 \leq i, j \leq 3$, all we need is the matrix

$$\int \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \begin{pmatrix} x_1 x_2 x_3 1 \end{pmatrix} dx_1 dx_2 dx_3 \tag{156}$$

which is the inertial tensor matrix of this link as defined in Section 2.5.

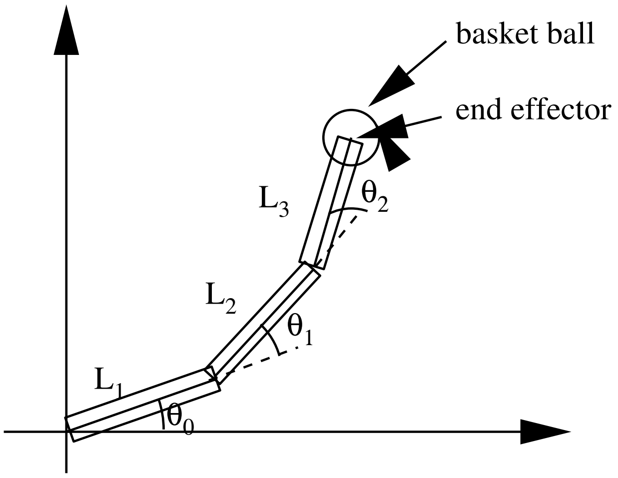


Figure 23: The planar 3-link chain throwing a basket ball

3.2.4 Representing Kinetic Energy

We have derived formulas to represent kinetic energy of a link by using the homogeneous transformation in Section 2.5. In our symbolic language, we need to represent the kinetic energy in terms of the centers of gravity and orientation matrices. Since the kinetic energy of a rigid body is the sum of rotational energy and the kinetic energy of the center of gravity, the kinetic energy of link L is

$$\frac{1}{2}\mathbf{trans}(\mathbf{pos\_}L[(1),t]) * \mathbf{trans}(\mathbf{pos\_}L[(1),t])$$
$$+\frac{1}{2}\mathbf{Vdot}(L, \mathbf{ori\_}L[(1),t], \mathbf{ori\_}L[(1),t]).$$

3.3 An Example

We give an example to show all the expressions in a spacetime constraint formulation in this new language. The example is extracted from [40]. Suppose we want to animate a planar three link arm (see Figure 23) which starts by holding a ball at the end effector (or "hand") in its rest position at time $t0$, throws the ball at time $t1$, and comes back to its rest position at time $t2$. The additional requirement is that the ball has to go into basket.

Assume the basket location is (Bx, By, Bz), the local coordinate of the end of the hand is (Hx, Hy, Hz). And let $g = 9.8$ be the gravity constant. Assume T is the time period during which the ball flies from leaving the hand to getting into the basket.

The objective function is the integral of the sum of squares of the three torques. There are three types of constraints:

(1). The ball should go into the basket.

(2). At time $t0$ and tf, the arm is still and in its rest position.

(3). Each joint angle has a limited range.

The third constraint is a built-in constraint, (i.e., each rotational angle has an upper and lower bound which are stored in the figure data structure), so the user doesn't have to input joint limit constraints.

Using our language, the objective function, the integral of the sum of the squared torques, can be written as

$$\mathbf{itg}[t0, tf](\mathbf{sqr}(\mathbf{tor\_}0[\mathbf{T}]) + \mathbf{sqr}(\mathbf{tor\_}1[\mathbf{T}]) + \mathbf{sqr}(\mathbf{tor\_}2[\mathbf{T}])), \tag{157}$$

constraint (1) is

$$\begin{aligned}&\mathbf{pos\_}L3[(0), t1, Hx, Hy, Hz] + \mathbf{pos\_}L3[(1), t1, Hx, Hy, Hz] * T + \\ &0.5 * \mathbf{colv}(0, -g, 0) * T * T = \mathbf{colv}(Bx, By, Bz),\end{aligned} \tag{158}$$

and constraint (2) is represented by twelve equations,

$$\mathbf{dof\_}0[0, t0] = 0, \mathbf{dof\_}1[0, t0] = 0, \mathbf{dof\_}2[0, t0] = 0, \tag{159}$$

$$\mathbf{dof\_}0[1, t0] = 0, \mathbf{dof\_}1[1, t0] = 0, \mathbf{dof\_}2[1, t0] = 0, \tag{160}$$

$$\mathbf{dof\_}0[0, tf] = 0, \mathbf{dof\_}1[0, tf] = 0, \mathbf{dof\_}2[0, tf] = 0, \tag{161}$$

$$\mathbf{dof\_}0[1, tf] = 0, \mathbf{dof\_}1[1, tf] = 0, \mathbf{dof\_}2[1, tf] = 0. \tag{162}$$

If we instead use the change of energy as the objective function (for example, see [39]), we can represent the objective function as

$$\begin{aligned}&\mathbf{itg}[t0, tf](\mathbf{sqr}(m_1 * \mathbf{trans}(\mathbf{pos\_}L1[1, \mathbf{T}]) * \mathbf{pos\_}L1[2, \mathbf{T}] \\ + \ &\mathbf{Vdot}(L1, \mathbf{ori\_}L1[1, \mathbf{T}], \mathbf{ori\_}L1[2, \mathbf{T}])) \\ + \ &\mathbf{sqr}(m_2 * \mathbf{trans}(\mathbf{pos\_}L2[1, \mathbf{T}]) * \mathbf{pos\_}L2[2, \mathbf{T}] \\ + \ &\mathbf{Vdot}(L2, \mathbf{ori\_}L2[1, \mathbf{T}], \mathbf{ori\_}L2[2, \mathbf{T}])) \\ + \ &\mathbf{sqr}(m_2 * \mathbf{trans}(\mathbf{pos\_}L3[1, \mathbf{T}]) * \mathbf{pos\_}L3[2, \mathbf{T}] \\ + \ &\mathbf{Vdot}(L3, \mathbf{ori\_}L3[1, \mathbf{T}], \mathbf{ori\_}L3[2, \mathbf{T}])))\end{aligned} \tag{163}$$

3.4 Symbolic Differentiation

The ability to automate the gradient computation using symbolic differentiation is one of the major advantages of a symbolic language. The symbolic differentiation can be carried out essentially the same way as the previous method as discussed in Section 2.12 except that while the leaves represent either a position or a matrix variable, *der-info* should be updated using the reductions (3)-(8) if necessary. For the torque variables, the partial derivative information is stored in the node, and we will show how to compute the torques and their partial derivatives efficiently at evaluation time in the next section.

One should also notice that there are two special operators: **Vdot** and **Vcross**. The differentiation formulae of these two operators are simply

$$\frac{\partial \mathbf{Vdot}(L, M_1, M_2)}{\partial \theta} = \mathbf{Vdot}(L, \frac{\partial M_1}{\partial \theta}, M_2) + \mathbf{Vdot}(L, M_1, \frac{\partial M_2}{\partial \theta}), \tag{164}$$

and

$$\frac{\partial \mathbf{Vcross}(L, M_1, M_2)}{\partial \theta} = \mathbf{Vcross}(L, \frac{\partial M_1}{\partial \theta}, M_2) + \mathbf{Vcross}(L, M_1, \frac{\partial M_2}{\partial \theta}). \tag{165}$$

3.5 Torque Variables

In this section, we show how to extend Hollerbach's technique [29] to compute the gradients of torques with optimum efficiency.

Let W_i represent the 4×4 transformation matrix of link i. Let r_i be its center of gravity. Let $G = (0, g, 0)$ be the gravity vector. Denote u_j to be the index of the link where θ_j is located (i.e., θ_j is one of the degrees of freedom allowing this link to rotate around its parent). Let $K(l)$ be the set of indices of the children of link l. Let $D(l)$ be the set of indices of all the descendents of link l. Let $\bar{D}(l) = D(l) \cup \{l\}$. Let W_v^u be the orientation matrix of link v in the coordinate system of link u, (i.e., $W_v = W_u * W_v^u$). For $k \in D(l)$, we use R_k to denote W_k^l. We use $u_k \preceq u_j$ to denote the relationship that u_k is either an ancestor of u_j or $u_k = u_j$. Given a matrix M, let $tr(M)$ be the trace of M. As shown in (46) of Section 2.5, the generalized force with respect to θ_j is

$$f_{\theta_j} = \sum_{i \in \bar{D}(u_j)} (tr\{\frac{\partial W_i}{\partial \theta_j} J_i (\ddot{W}_i)^T\} + m_i G \frac{\partial W_i}{\partial \theta_j} r_i) \tag{166}$$

In order to derive the recursive formulas for computing $f(\theta_j)$ and its partial derivatives, we need to use the following intermediate variables:

$$A_l = \sum_{i \in \bar{D}(l)} W_i^l J_i \ddot{W}_i^T, \tag{167}$$

$$b_l = \sum_{i \in \bar{D}(l)} m_i W_i^l r_i, \tag{168}$$

$$B_{l,k} = \sum_{i \in \bar{D}(l)} W_i^l J_i \frac{\partial \ddot{W}_i^T}{\partial \theta_k}, \tag{169}$$

$$C_{l,k} = \sum_{i \in \bar{D}(l)} 2 W_i^l J_i \frac{\partial \dot{W}_i^T}{\partial \theta_k}, \tag{170}$$

and

$$D_{l,k} = \sum_{i \in \bar{D}(l)} W_i^l J_i \frac{\partial W_i^T}{\partial \theta_k}. \tag{171}$$

First of all, we claim that

$$f_{\theta_j} = tr(\frac{\partial W_{u_j}}{\partial \theta_j} A_{u_j}) + G \frac{\partial W_{u_j}}{\partial \theta_j} b_{u_j} \tag{172}$$

Since

$$f_{\theta_j} = \sum_{i \in \bar{D}(u_j)} (tr\{\frac{\partial W_i}{\partial \theta_j} J_i \ddot{W}_i^T\} + m_i G \frac{\partial W_i}{\partial \theta_j} r_i) \tag{173}$$

$$= \sum_{i \in \bar{D}(u_j)} (tr\{\frac{\partial (W_{u_j} W_i^{u_j})}{\partial \theta_j} J_i (\ddot{W}_i)^T\} + m_i G \frac{\partial (W_{u_j} W_i^{u_j})}{\partial \theta_j} r_i) \tag{174}$$

$$= tr\{(\frac{\partial W_{u_j}}{\partial \theta_j}) \sum_{i \in \bar{D}(u_j)} (W_i^{u_j} J_i \ddot{W}_i^T)\} + G \frac{\partial W_{u_j}}{\partial \theta_j} \sum_{i \in \bar{D}(u_j)} (m_i W_i^{u_j} r_i) \tag{175}$$

$$= tr(\frac{\partial W_{u_j}}{\partial \theta_j} A_{u_j}) + G \frac{\partial W_{u_j}}{\partial \theta_j} b_{u_j}, \tag{176}$$

which is equation (172).

The formula for computing partial derivatives with respect to a DOF variable and its first and second order derivatives are

$$\frac{\partial f_{\theta_j}}{\partial \theta_k} = \begin{cases} tr\{\frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} A_{u_j} + \frac{\partial W_{u_j}}{\partial \theta_j} B_{u_j,k}\} + G \frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} b_{u_j}, & u_k \preceq u_j \\ tr\{\frac{\partial^2 W_{u_k}}{\partial \theta_j \partial \theta_k} A_{u_k} + \frac{\partial W_{u_k}}{\partial \theta_j} B_{u_k,k}\} + G \frac{\partial^2 W_{u_k}}{\partial \theta_j \partial \theta_k} b_{u_k}, & \text{otherwise} \end{cases} \tag{177}$$

$$\frac{\partial f_{\theta_j}}{\partial \dot{\theta}_k} = tr\{\frac{\partial W_{u_j}}{\partial \theta_j} C_{u_j,k}\}, \tag{178}$$

and

$$\frac{\partial f_{\theta_j}}{\partial \ddot{\theta}_k} = tr\{\frac{\partial W_{u_j}}{\partial \theta_j} D_{u_j,k}\}. \tag{179}$$

To derive (177), when $u_k \preceq u_j$ we have

$$\frac{\partial f_{\theta_j}}{\partial \theta_k} = \sum_{i \in \bar{D}(u_j) \cap \bar{D}(u_k)} (tr\{\frac{\partial^2 W_i}{\partial \theta_j \partial \theta_k} J_i (\ddot{W}_i)^T + \frac{\partial W_i}{\partial \theta_j} J_i \frac{\partial \ddot{W}_i^T}{\partial \theta_k}\}$$

$$+ m_i G \frac{\partial^2 W_i}{\partial \theta_j \partial \theta_k} r_i) \tag{180}$$

$$= \sum_{i \in \bar{D}(u_j)} (tr\{\frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} W_i^{u_j} J_i (\ddot{W}_i)^T + \frac{\partial W_{u_j}}{\partial \theta_j} W_i^{u_j} J_i \frac{\partial \ddot{W}_i^T}{\partial \theta_k}\}$$

$$+ m_i G \frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} W_i^{u_j} r_i) \tag{181}$$

$$= tr\{\frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} \sum_{i \in \bar{D}(u_j)} W_i^{u_j} J_i (\ddot{W}_i)^T + \frac{\partial W_{u_j}}{\partial \theta_j} \sum_{i \in \bar{D}(u_j)} W_i^{u_j} J_i \frac{\partial \ddot{W}_i^T}{\partial \theta_k}\}$$

$$+ G \frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} \sum_{i \in \bar{D}(u_j)} m_i W_i^{u_j} r_i \tag{182}$$

$$= tr\{\frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} A_{u_j} + \frac{\partial W_{u_j}}{\partial \theta_j} B_{u_j,k}\} + G \frac{\partial^2 W_{u_j}}{\partial \theta_j \partial \theta_k} b_{u_j}. \tag{183}$$

The other case when u_k is u_j's descendent is similar. What follows are the derivations of (178) and (179). From (173), we have

$$\frac{\partial f_{\theta_j}}{\partial \dot{\theta}_k} = \sum_{i \in \bar{D}(u_j)} tr\{\frac{\partial W_i}{\partial \theta_j} J_i \frac{\partial \ddot{W}_i^T}{\partial \dot{\theta}_k}\} \tag{184}$$

$$= tr\{\sum_{i \in \bar{D}(u_j)} 2 \frac{\partial W_{u_j}}{\partial \theta_j} W_i^{u_j} J_i \frac{\partial \dot{W}_i^T}{\partial \theta_k}\} \tag{185}$$

$$= tr\{\frac{\partial W_{u_j}}{\partial \theta_j} \sum_{i \in \bar{D}(u_j)} 2 W_i^{u_j} J_i \frac{\partial \dot{W}_i^T}{\partial \theta_k}\} \tag{186}$$

$$= tr\{\frac{\partial W_{u_j}}{\partial \theta_j} C_{u_j,k}\}, \tag{187}$$

$$\frac{\partial f_{\theta_j}}{\partial \ddot{\theta}_k} = \sum_{i \in \bar{D}(u_j)} tr\{\frac{\partial W_i}{\partial \theta_j} J_i \frac{\partial \ddot{W}_i^T}{\partial \ddot{\theta}_k}\} \tag{188}$$

$$= tr\{\sum_{i \in \bar{D}(u_j)} \frac{\partial W_{u_j}}{\partial \theta_j} W_i^{u_j} J_i \frac{\partial W_i^T}{\partial \theta_k}\} \tag{189}$$

$$= tr\{\frac{\partial W_{u_j}}{\partial \theta_j} \sum_{i \in \bar{D}(u_j)} W_i^{u_j} J_i \frac{\partial W_i^T}{\partial \theta_k}\} \tag{190}$$

$$= tr\{\frac{\partial W_{u_j}}{\partial \theta_j} D_{u_j,k}\}. \tag{191}$$

The recursive formulas for computing A_l, b_l, $B_{l,k}$, $C_{l,k}$ and $D_{l,k}$ are

$$A_l = J_l \ddot{W}_l^T + \sum_{u \in K(l)} R_u A_u, \tag{192}$$

$$b_l = m_l r_l + \sum_{u \in K(l)} R_u b_u, \tag{193}$$

$$B_{l,k} = J_l \frac{\partial \ddot{W}_l^T}{\partial \theta_k} + \sum_{u \in K(l)} R_u B_{u,k}, \tag{194}$$

$$C_{l,k} = 2 * J_l \frac{\partial \dot{W}_l^T}{\partial \theta_k} + \sum_{u \in K(l)} R_u C_{u,k}, \tag{195}$$

$$D_{l,k} = J_l \frac{\partial W_l^T}{\partial \theta_k} + \sum_{u \in K(l)} R_u D_{u,k}, \tag{196}$$

The derivations of (192)-(196) are similar to each other. So we only give the one for (192).

$$A_l = J_l \ddot{W}_l^T + \sum_{i \in D(l)} (W_i^l J_i \ddot{W}_i^T) \tag{197}$$

$$= J_l \ddot{W}_l^T + \sum_{k \in K(l)} \sum_{i \in \bar{D}(k)} (W_i^l J_i \ddot{W}_i^T) \tag{198}$$

$$= J_l \ddot{W}_l^T + \sum_{k \in K(l)} \sum_{i \in \bar{D}(k)} (W_k^l W_i^k J_i \ddot{W}_i^T) \tag{199}$$

$$= J_l \ddot{W}_l^T + \sum_{k \in K(l)} R_k \sum_{i \in \bar{D}(k)} (W_i^k J_i \ddot{W}_i^T) \tag{200}$$

$$= J_l \ddot{W}_l^T + \sum_{k \in K(l)} R_k A_k \tag{201}$$

To compute the gradients, for each k, we traverse the figure tree to compute A_l, B_l, $B_{l,k}$, $C_{l,k}$, and $D_{l,k}$ for all l by using (192)-(196). Then (172)-(179) are applied to compute the torques and all of their partial derivatives. So in total, the gradient computation takes $O(m^2)$ time. Notice that we have m^2 components to evaluate (the gradient of each torque contains m components), so in average each component takes constant time. Clearly this is the optimum.

3.6 Evaluation

The evaluation of the evaluation trees (including differentiation trees) begins with the the variables at the leaves. The DOF variables are evaluated directly from their representations (e.g., B-splines, B-spline wavelets, piecewise Hermite splines, etc.). From the last section we can see that the evaluation of torque variables reduce to the evaluation of orientation matrices, their first and second order derivatives, and their partial derivatives. In addition, it is easy to see that the evaluation of positions and their time or partial derivatives can be computed recursively from the orientation matrices. Thus, all that is needed is to recursively compute all the orientation matrices, their first and second order time derivatives, and their partial derivatives. Formally we need to evaluate W_i, $\dot{W}_i$, $\ddot{W}_i$, $\frac{\partial W_i}{\partial \theta_k}$ $\frac{\partial \dot{W}_i}{\partial \theta_k}$ $\frac{\partial \ddot{W}_i}{\partial \theta_k}$ for all i and k.

We have given algorithms to compute W_i and $\dot{W}_i$ in Section 2.4. The case for $\ddot{W}_i$ is similar, The recursion formula is

$$\ddot{W}_i = \ddot{W}_{\wp_i} W_i^{\wp_i} + 2\dot{W}_{\wp_i} \dot{W}_i^{\wp_i} + W_{\wp_i} \ddot{W}_i^{\wp_i}. \tag{202}$$

where the formula for $\ddot{W}_i^{\wp_i}$ can be obtained by differentiating (33) in Section 2.4 , the details are omitted.

The recursion formulae for computing partial derivatives with respect to DOF variables are

$$\frac{\partial \dot{W}_i}{\partial \theta_j} = \frac{\partial \dot{W}_{\wp_i}}{\partial \theta_j} W_i^{\wp_i} + \dot{W}_i \frac{\partial W_i^{\wp_i}}{\partial \theta_j} + \frac{\partial W_{\wp_i}}{\partial \theta_j} \dot{W}_i^{\wp_i} + W_{\wp_i} \frac{\partial \dot{W}_i^{\wp_i}}{\partial \theta_j} \tag{203}$$

$$\begin{aligned} \frac{\partial \ddot{W}_i}{\partial \theta_j} = \ & \frac{\partial \ddot{W}_{\wp_i}}{\partial \theta_j} W_i^{\wp_i} + \ddot{W}_{\wp_i} \frac{\partial W_i^{\wp_i}}{\partial \theta_j} + 2 \frac{\partial \dot{W}_{\wp_i}}{\partial \theta_j} \dot{W}_i^{\wp_i} + 2\dot{W}_{\wp_i} \frac{\partial \dot{W}_i^{\wp_i}}{\partial \theta_j} \\ & + \frac{W_{\wp_i}}{\partial \theta_j} \ddot{W}_i^{\wp_i} + W_{\wp_i} \frac{\partial \ddot{W}_i^{\wp_i}}{\partial \theta_j} \end{aligned} \tag{204}$$

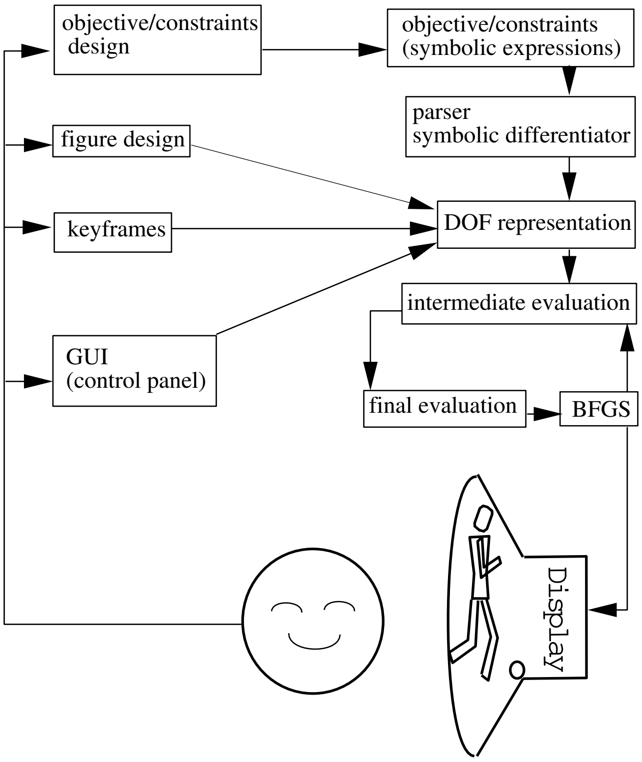


Figure 24: The new interactive spacetime constraints system

and the partial derivatives with respect to the time derivatives of the DOF variables can be computed using the reductions (143) through (148).

In general, an optimization formulation may contain multiple expressions, and some expressions may need to be evaluated at multiple sample time points to evaluate integral expression. To avoid redundant recursive matrix evaluations, for each time sample point one can do a recursive matrix evaluation once and evaluate all the expressions at this time.

3.7 Interactive Spacetime Constraints System with the New Symbolic Interface

With the new symbolic interface in place, the interactive spacetime constraints system can be modified as shown in the Figure 24. The intermediate step evaluates variables W_i, $\dot{W}_i$, $\ddot{W}_i$, $\frac{\partial \dot{W}_i}{\partial \theta_j}$, $\frac{\partial \ddot{W}_i}{\partial \theta_j}$, A_l, b_l, $B_{l,k}$, $C_{l,k}$, and $D_{l,k}$ as defined in Section 3.5. This step takes m^2 time where m is the number of DOFs. After this intermediate evaluation, all the variables (including the differentiation trees) can be evaluated immediately (in constant time).

3.8 Experiments

Tests to compare the computation time of the previous method and this new method were conducted. The test cases are linear chains with the number of links ranging from 9 to 16. Each joint has one degree of freedom. The expression is the position of the end effector. The results are shown in Figure 25. Figure 25 (A) shows the number of nodes of the evaluation trees vs. the number of links using previous symbolic method before common subexpression elimination. Figure 25 (B) shows the number of nodes of the evaluation trees vs. the number of links after common subexpression elimination as described in [40]. Figure 25 (C) plots the evaluation time vs. the number of links using the previous symbolic method with common subexpression elimination. Figure 25 (D) shows the evaluation time vs. the number of links using our new symbolic method. Note the changes in scales of the horizontal and vertical axes.

We can see that in the previous method, even after subexpression elimination, both the number of nodes and the evaluation time (Figure 25 (C)) are growing exponentially as the number of links increases. But if we use the new symbolic method, the evaluation time grows linearly as expected. Figure 25 (E) zooms in on the lower left corner of graphs (C) and (D) to show a direct comparison using the same axes. The linear growth is critical to allowing the interactive spacetime constraint methods to extend to complex figures.

3.9 Conclusion

In this chapter, we have described a new symbolic method and showed that this method is general enough to represent the kinematic and dynamic quantities that arise in the spacetime constraint method for animating linked figures. The evaluation of the resulting symbolic expressions and their gradients are in complexity, the same as the

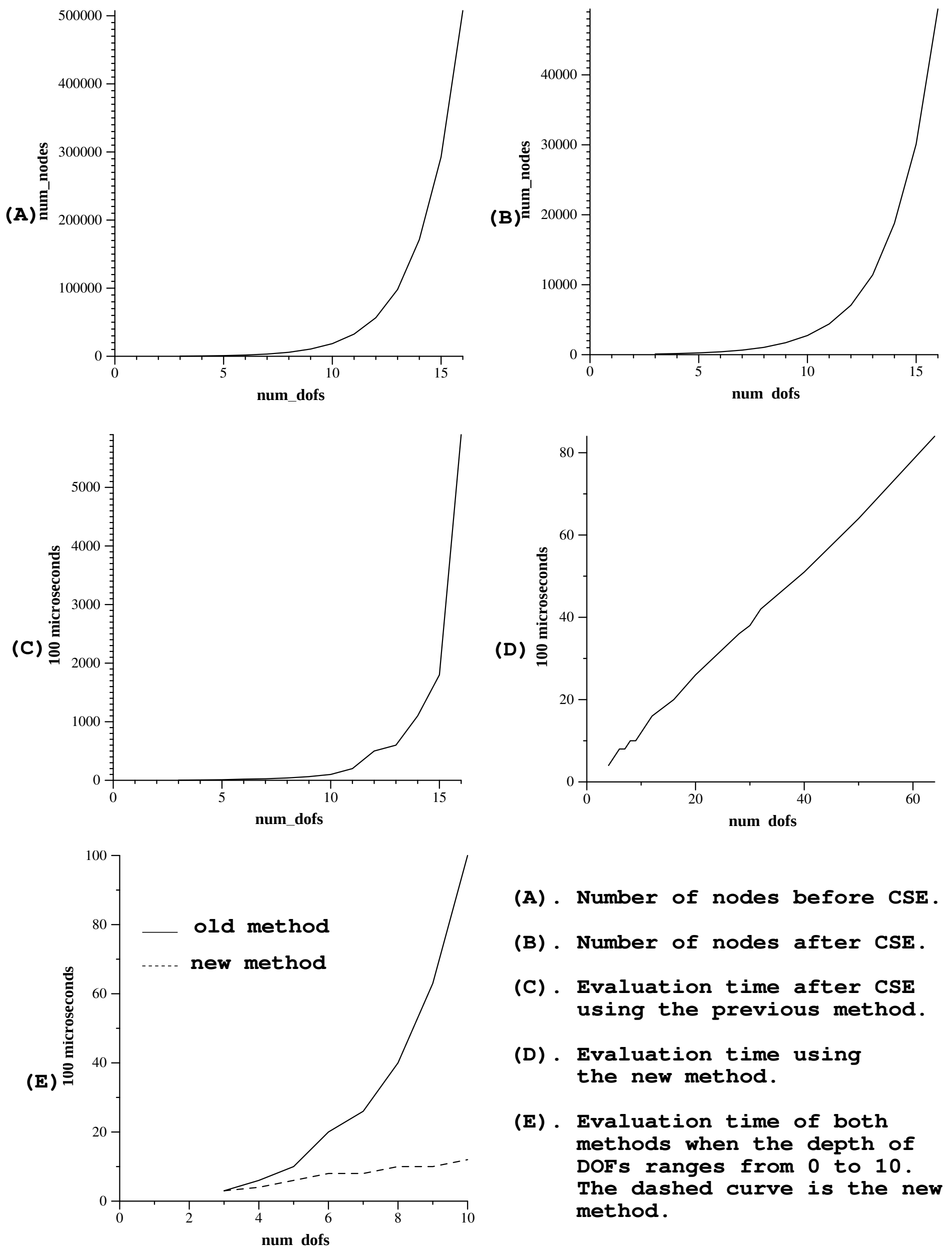


Figure 25: Comparisons of previous CSE method and the new symbolic method

equivalent numerical methods. Furthermore, the expressions are usually very small, so this method makes a general, easy to use, and efficient interface to optimization based linked figure animation systems. Currently the symbolic expressions are read from a file which is created by hand. Figure 26 is a snapshot of the interface where the button "Load Exps" at the bottom left is used to select a file and read in the symbolic expressions. Clearly, work needs to be done to provide more graphically based and/or intuitive means to generate, modify and inspect the symbolic expressions.

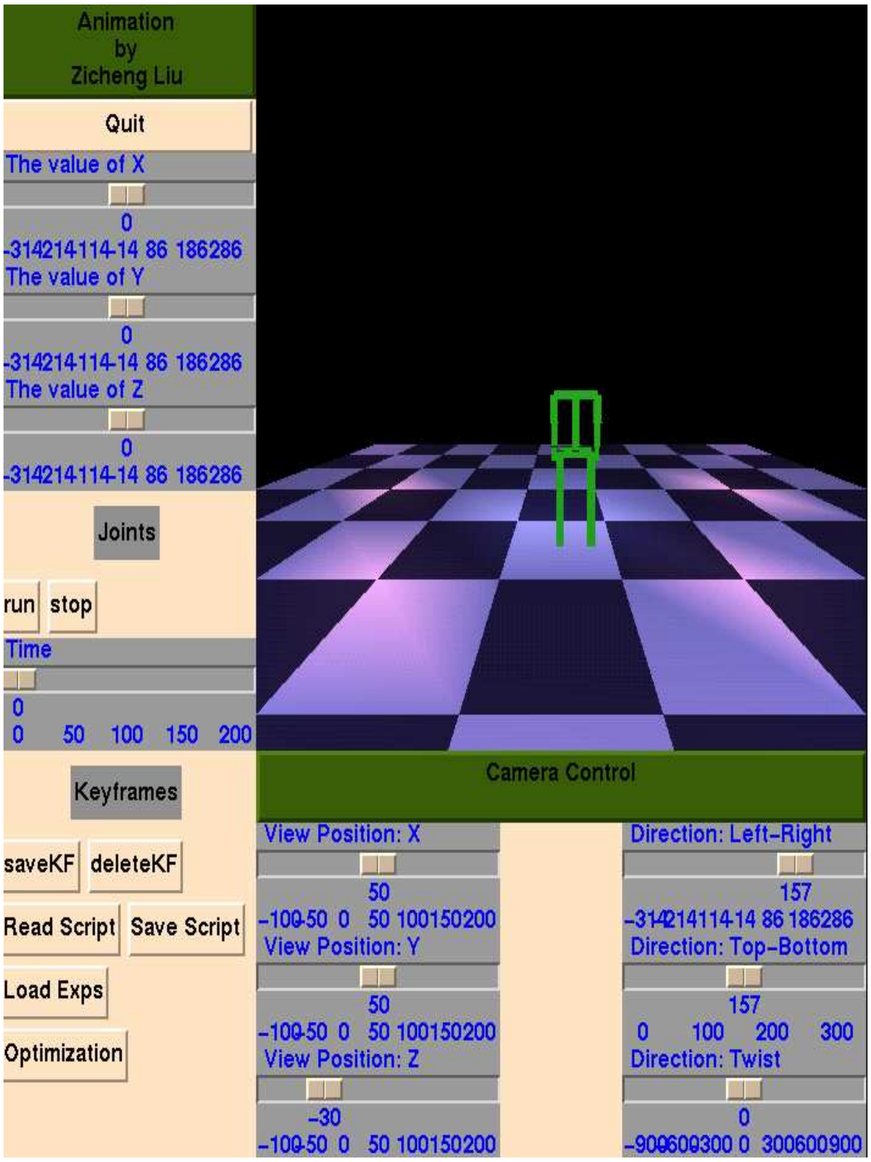


Figure 26: The graphical interface

Chapter 4

Keyframe Optimization

4.1 Introduction

In addition to symbolic processing, another bottleneck of the spacetime constraints system is the numerical optimization. The number of unknowns and the number of constraints tend to be large. In general, there are no efficient numerical methods available to solve such nonlinear constrained optimization problem. This chapter describes a variation of the full spacetime constraints method which uses user supplied information to reduce the complexity of the optimization problem.

4.1.1 The Idea

Figure 27 lists choices of animation paradigms outlined in Chapter 1 by their extent of control versus automation, (the portion of the work which the computer does to generate a motion). The approach presented in this chapter (labeled "keyframe optimization") fills a gap between simple keyframe systems and other optimization based systems. The central idea in the work presented here is to maintain as much of the semantics and control offered by keyframe systems as possible, while still providing some of the benefits of optimization based systems.

We also wish to provide results at interactive speeds (i.e., a few seconds) to encourage the iterative modification and evaluation of the animation design process. Reducing the complexity of the numerical process is achieved in a number of ways: by reducing the dimensionality of the space of possible solutions, by creating a more

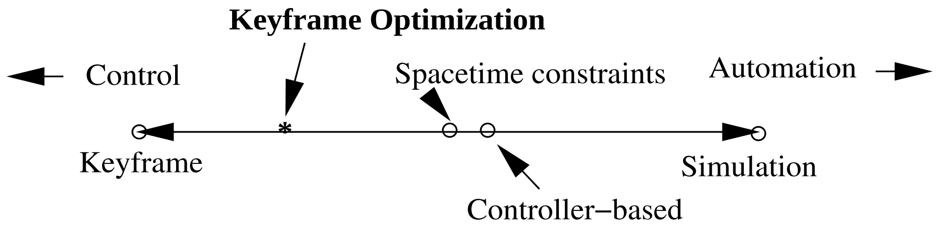


Figure 27: Control vs. Automation

linear problem, by constraining the problem more tightly (implicitly reducing the space as well), and by providing a better starting guess for solutions.

The observation we make in reformulating the problem to achieve these goals is that the animator may have a good sense of specific key positions a character should pass through (the keyframes). But in general, the inexperienced animator may have less intuition about the precise timing of the keyframes and the velocity of the individual DOF as they pass these unknown points in time. Beyond the implicit constraints of the keyframes, the animator may also know specific instances of higher level constraints on the motion. For example, given a desire to have the figure jump to a certain height, the vertical velocity of the center of gravity (COG) at the beginning of a jump can be derived.

Based on these observations, the solution we propose is to

1. fix the user specified keyframed positions as constraints, as well as a few key time points selected by the animator,

2. allow specification of other higher level constraints, for example the velocity of the center of gravity for a jump, or the velocity of an end effector to achieve a physically realistic throw, and

3. select a piecewise cubic Hermite interpolation between keyframes as the underlying representation of the trajectories, but

4. leave the time of most keyframes and the first derivative of each DOF needed to fully specify the Hermite segments as unknowns to be solved for by a simplified optimization process.

These choices satisfy the criteria set out above in that they "trust" the animator to position the character at keyframes. This leaves intact any user interface for positioning the keyframes themselves. The animator is also implicitly fixing the amount of freedom in each DOF function through the number of keyframes chosen (i.e., additional keyframes inserted automatically increase the freedom in the resulting trajectory).

The optimization problem is greatly simplified as there are only time marks and velocity unknowns per DOF per keyframe. This contrasts with previous systems which solve for an unknown number of position values for each DOF. In addition, in this approach, the velocity terms appear as linear terms in an energy functional as opposed to the quadratic position terms. The choice of piecewise Hermite cubic curves for the trajectories matches the mix of known and unknown quantities in the system (i.e., known position, unknown velocities and timing).

4.1.2 Comparison to Standard Keyframing and Constrained Optimization

Compared to a pure keyframe system, the method outlined here uses optimization only to decide velocities and those time marks not explicitly specified by the animator. These constitute arguably the most difficult part of a keyframe system to produce a graceful and natural looking motion. In essence, these ideas primarily replace the interpolation step (and the parameter-to-time, i.e. "ease-in, ease-out" function) in traditional keyframe systems. Since the optimization is fast, the method is well suited to be implemented in any keyframe system.

This method differs from the optimal control interpolation method of [7] which solves the complete animation from a sequential solution of a series of two-point boundary value problems. In contrast, we perform a unified solution over as many time intervals as desired.

Another feature which is different from previous work is that the timing between keyframes is relaxed. In the spacetime constraint setting [56, 13, 40], the user had to specify when each constraint was to hold [1]. Our experiments suggest that leaving the timing to the optimization process results in significantly better results at a minimal cost. It is inexpensive since there is only one time value per keyframe, that is, it is not tied to the figure's complexity.

[1]It appears that a similar concept may have been employed in one of the examples in [56], however, it is not reported as such.

There are, of course, disadvantages in the proposed system. The reduction in the space of possible solutions requires more specification from the animator (i.e., this is not a system for creating autonomous creatures). The results are also not guaranteed to be physically accurate, as the space of solutions is highly constrained by the choice of keyframes. Furthermore, beyond what is provided in standard keyframe systems, objective functions such as minimization of energy, or other physically based objectives such as maintaining balance must be specified, and be able to be evaluated and differentiated. Finally, it can also be argued that the system being proposed really does very little for the animator, since the keyframes provide the backbone of the animation sequence. However, we will show that the subtleties of timing and velocity add significantly to the lifelike feeling of an animation. Although a highly skilled animator may have the ability to directly specify this also, many animators cannot.

4.2 System Overview

The animation system can be thought of either as a keyframe system relying on a spacetime constraint paradigm for interpolation between keyframes, or as a spacetime constraint system taking keyframes as constraints. The basic structure is described in Figure 24 where the DOF representation is the Hermite representation, the keyframes are used as constraints, and a scheme is provided for the user to fix/relax the variables.

The user interactively defines figures and manipulates them by changing the angles of individual joints or through the use of any inverse kinematics method[2] The time points for a few keyframes may also be fixed, as may be velocities, if so desired. Higher level constraints can be specified as well[3]. For example, the COG of the figure or an end effector can be constrained to have a specific velocity at some keyframe.

Given the user specified keyframes, and an objective function (currently a minimum energy objective), the optimization process is run, and the resulting motion is

[2]The interactive system used to test the ideas in this work is quite rudimentary compared to commercial systems and is not discussed here.

[3]There is no high level interface developed for the experiments reported in this paper, but the symbolic method as developed in [38] should provide a good interface for such animation system.

displayed. At this point, the user may go back and modify or add keyframes and other constraints, and rerun the optimization procedure.

The optimization process involves iterative gradient computation and an optimization step. The optimization step is made either with a line search along the negative gradient or along a gradient modified by the pseudo-inverse Hessian constructed with the Broyden-Fletcher-Goldfarb-Shanno (BFGS) nonlinear optimization algorithm [49, 40] discussed in Chapter 2. Typically 3 to 10 iterations of from 1 to 5 seconds are required.

4.2.1 Hermite Interpolation as DOF Representation

Piecewise cubic Hermite splines (see Section 2.8.1) were selected as the underlying representation for the DOF trajectories due to their good match between the animator specification and optimization parameters, and the Hermite basis. Each Hermite segment is defined by endpoint positions (the keyframes), endpoint velocities (unknowns to be determined through optimization), and endpoint parameter values (in this case time, also to be determined). Thus, for m time segments, there are $2n(m+1)+m$ parameters in total (the position and velocity of each of n DOF at each of $m+1$ keyframes plus the m time values.

Since the animator's keyframes are taken as explicit solutions for the positions, there are only half of these, or $n(m+1)+m$ remaining as unknowns. Some of the remaining velocities are also prescribed by the animator, such as perhaps starting and ending at rest, as are some time points, further reducing the problem size. The problem, then is to find these $n(m+1)+m$ (or fewer) parameters so that the objective is minimized.

4.2.2 Relaxing Speed and Timing

A simple soccer player serves as an example to demonstrate how this idea works. Suppose the soccer player, starting from an initial standing position, is to kick a ball into the goal, and return to a standing position (see the middle of Figure 32). The animator has some intuitive notion about how to kick a ball, for example, the player swings his leg backward to prepare for the kick, then swings it forward to kick to

Figure 28: One-link arm throwing the ball into the basket

ball. Thus, four keyframes are defined with the leg swung backward, with the leg about to kick the ball (in this keyframe, we make sure that the toe is at the ball's position), plus one keyframe each for the beginning and final standing positions. The timing of and velocities through the middle two keyframes is left unspecified. The first and last keyframes are fixed both in time and with zero velocity. One additional constraint requires the foot to have sufficient velocity to kick the ball at the time of the third keyframe. With the figure constrained to start and stop at rest, we are left with a very small optimization problem with only the second and third velocities, and the three time intervals as unknowns. These $(2 * n + 3)$ unknowns represent a much smaller, albeit more restrictive, problem than those that arise in general spacetime constraint problems. Although the solution space is quite restrictive, it is generally still big enough to contain good solutions (given reasonable keyframes of course). In experiments below, a smooth and realistic kicking motion of a 3D figure with 15 degree of freedoms was found in 3 seconds on an SGI R4000 processor.

Relaxing Timing

Let's use a simple example to demonstrate the fact that relaxing timing can generate better motions.

Suppose there is a one link arm with one rotational degree of freedom, as shown in Figure 28. Recall that we have derived the spacetime constraint formulation in (3.5). Assume the length of the link, L, is equal to 2. The task is to start with the rest position (the arm is straight down, $\theta = -\pi/2$) at time $t_0 = 0$ and throw the ball

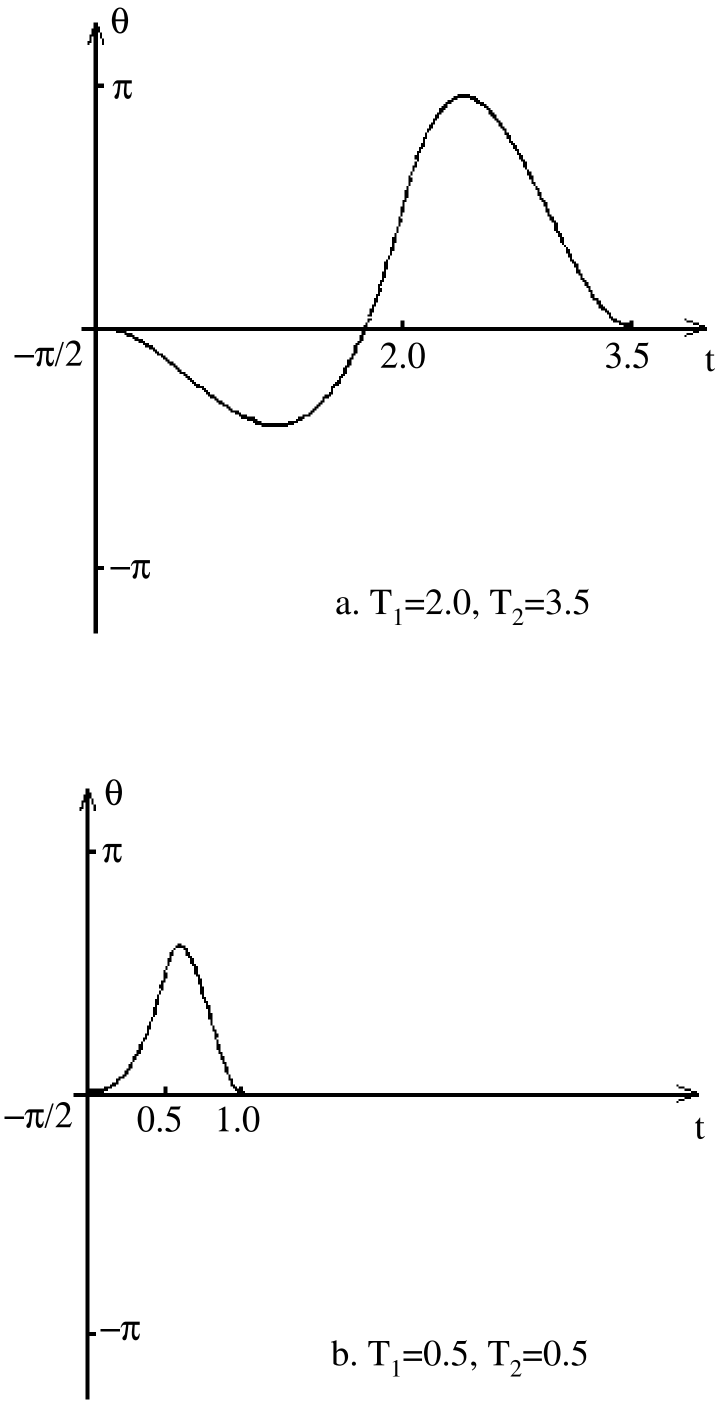


Figure 29: $\theta = \theta(t)$

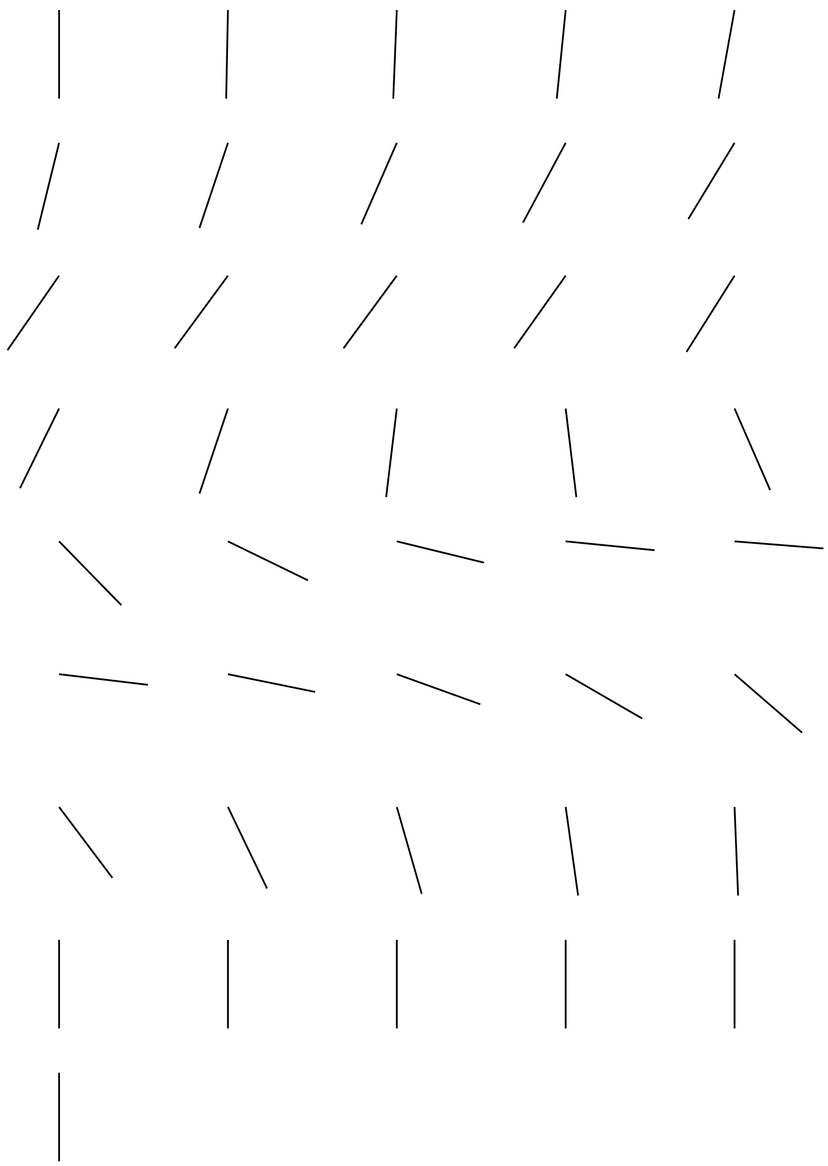


Figure 30: Motion sequence when $T_1 = 2.0$ and $T_2 = 1.5$

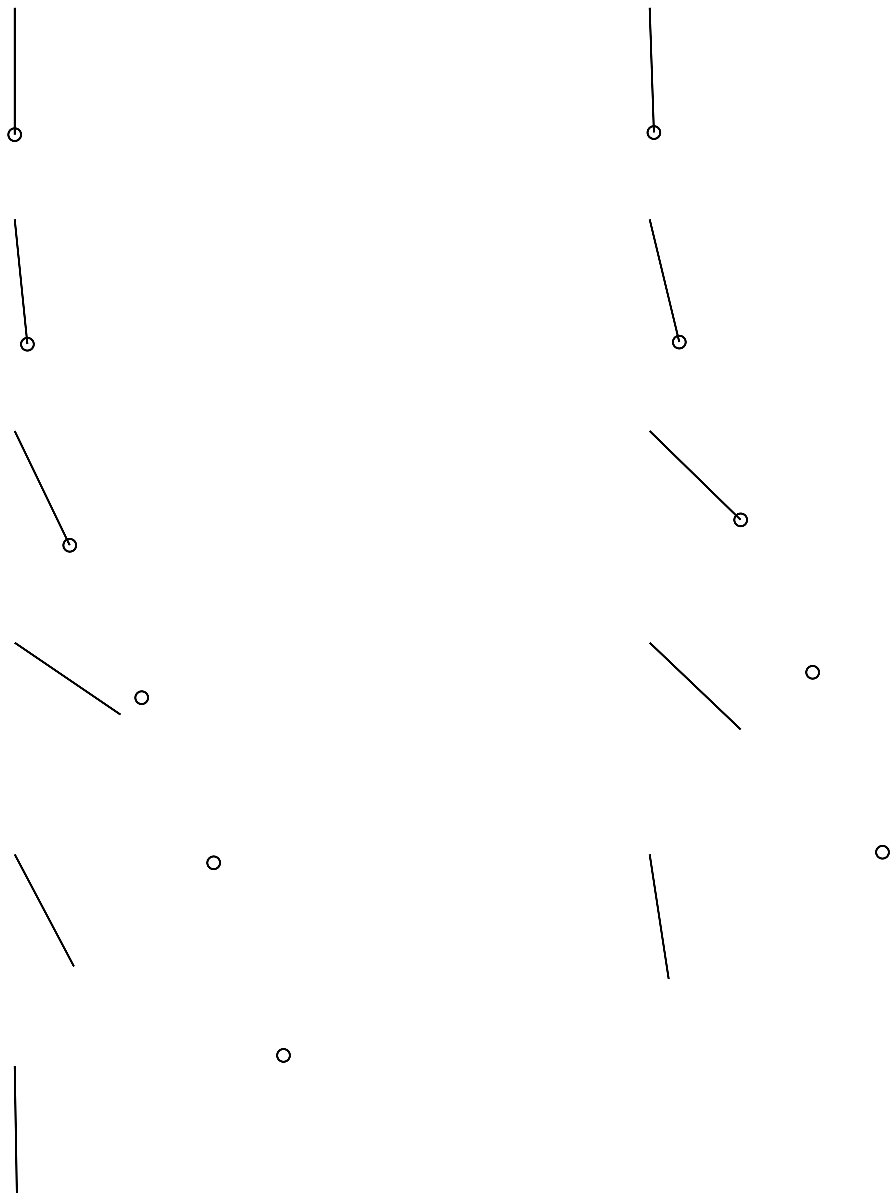


Figure 31: Motion sequence when $T_1 = 0.5$ and $T_2 = 0.5$

at time t_1 and come back to its rest position at time t_2. Suppose $\theta(t_1) = -\pi/2 + \pi/4$, which has been fixed by the user. Suppose B_x and B_y and T are chosen such that $\dot{\theta}(t_1)$ must be equal to 4.0 in order for the ball to make the goal. Denote $T_1 = t_1 - t_0$, and $T_2 = t_2 - t_1$. Notice that over $[t_0, t_1]$, θ is determined by T_1, and over $[t_1, t_2]$, θ is determined by T_2.

If we choose $T_1 = 0.5$ and $T_2 = 0.5$. Using Hermite interpolation, $\theta(t)$ over $[t_0, t_2]$ is shown in Figure 29 (b). The corresponding motion sequence is shown in Figure 31. The time interval between each frame is 0.1. For each frame, we also draw the ball so that we can compare the velocity of the ball with that of the arm around the moment of throwing. We can see that the arm moves forward immediately, throws the ball, quickly decelerates, and moves backward to its rest position. The motion is very rigid. But if we choose $T_1 = 2.0$ and $T_2 = 1.5$, at which the objective function achieves its minimum, the resulting trajectory of $\theta(t)$ is shown in Figure 29 (a). The motion sequence is shown in Figure 30 (the length of the arm is scaled down and the ball is not drawn to save space). Again, the time interval between each frame is 0.1. In this case, the arm firstly moves backward slowly to obtain some potential, and then accelerates forward to achieve the speed and height to throw the ball. After throwing, it continues to move forward and slow down gradually and finally moves backward to its rest position. This motion is much more natural than the previous one.

4.3 Keyframe Optimization

The optimization used in this work is a standard BFGS quasi-Newton solver [49]. BFGS defaults to a simple gradient descent for the first iteration. In fact, for small problems, we have found a pure gradient descent is all that is required. Each iteration consists of a local gradient determination followed by a bisection line search (see, for example, [49]) along the negative gradient. The ability to use such a simple system is due in part to the fact that a simplistic interpolation of the keyframes already places the solution near a local minimum, and thus in a well behaved region of the solution space. In addition, the unknown velocities of the DOF appear as linear terms in the goal constraint (see the next paragraph for the reason) as opposed to the positions that appear as nonlinear terms in a general system. The time values and higher level constraints are nonlinear, however, the initial values implied by the animators keyframes provide a sufficient starting point for a local downhill search.

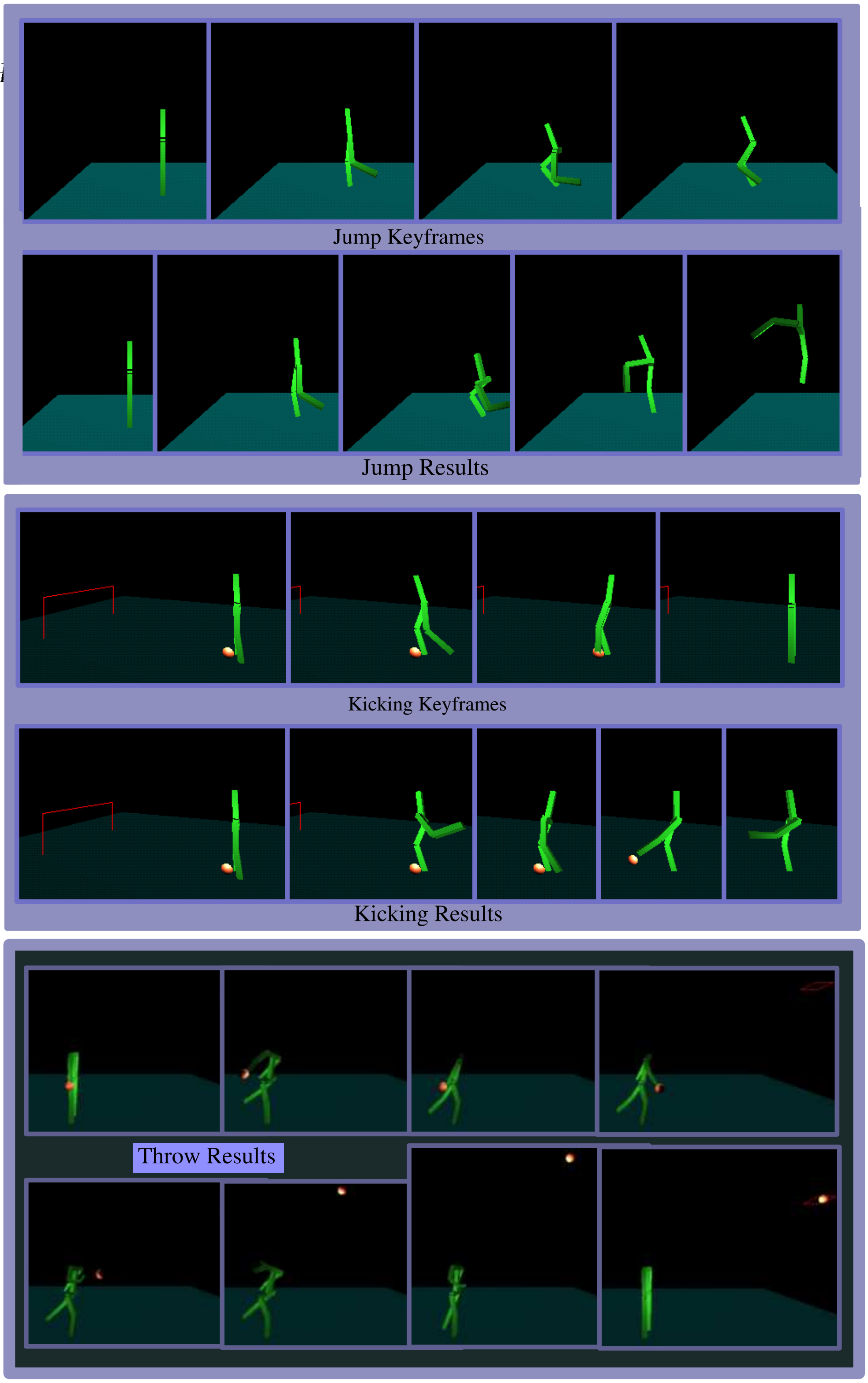
The reason that the unknown velocities of the DOF appear as linear terms in the goal constraint is the following. After the configuration is fixed, the goal constraint is a constraint on the velocity of the end effector or the center of gravity with the form $\dot{P}(t) = V$. Notice that $\dot{P}(t) = J(t)\dot{\Theta}(t)$ where J is the well-known Jacobi matrix, the constraint becomes $J(t)\dot{\Theta}(t) = V$. Since $J(t)$ is a constant matrix due to the fixed configuration, the left hand side is a linear function of $\dot{\Theta}(t)$.

4.4 Results

A series of experiments were conducted on two figures. The first is a truncated 3D human-like figure with 2 legs (jointed at the knees), a pelvis and a body, with a total of 15 rotational degrees of freedom plus translation and rotation within the environment. The second figure is a more complete humanlike creature with 29 internal DOF. There is a ground plane and a ball with which the figures interact. The experiments involved jumping, landing from a jump, walking, running, and motion throwing and kicking a ball. In each case, short animation segments were explored. Transitions between segments were also explored with the same techniques, for example transitioning from walking to running to kicking a ball.

A first guess interpolation can be found by fixing the times of the keyframes at evenly spaced intervals. The velocities for this interpolation are taken from the central finite differences between previous and future keyframes (or forward or backward differences at end frames). One would expect a skilled animator to do better than this on a first pass, however, it serves well as a starting point for the optimization.

In general, each optimization iteration took approximately 1 to 2 seconds for the simple figure and 4 to 5 seconds for the more complete figure on an R4000 SGI machine. The dominant portion of the time was taken by gradient computation and objective evaluation during the line search. Solutions for short segments were often found in only two iterations, however, some continued to improve for 10 iterations.



Color Plate: Jumping, Kicking, Throwing
From: Keyframe Motion Optimization by Relaxing Speed and Timing
by: Zicheng Liu and Michael F. Cohen

Figure 32: Motion sequences

The first example is a jumper. There are four keyframes: an initial relaxed position, swing backward, swing forward, takeoff. The center of gravity is constrained to have a desired upward velocity at take-off. The position of the COG is based on free flight dynamics after leaving the ground. The optimization process determines the velocities at the second, third, and fourth frames and the three time segment lengths are relaxed (48 unknowns). Six iterations (about 10 seconds), result in a realistic jumping motion with desired takeoff velocity results (see top of Figure 32).

A similar landing sequence was also determined. Given the landing velocity, the figure is required to land, absorb the energy of the fall, and then stand up. There are three frames: landing, follow through, and standing up. Four iterations result in a natural looking motion.

The second experiment looks at a soccer player. Initially in a standing at rest position, the player kicks the ball (hopefully) into the goal and comes back to his rest position. The ball's motion is computed from simple Newtonian physics assuming it simply takes on the velocity of the foot at impact. There are four keyframes: the first and fourth are trivial rest configuration. The second one is the prekicking configuration with the foot drawn back, and the third one represents when the foot strikes the ball. A constraint is set for the velocity of the foot at the third keyframe so as to kick the ball into the goal.

The total of 33 variables are determined in two iterations resulting in a kick with an anticipatory bend of the knee and a follow through that projects the ball with desired velocity (see middle of Figure 32).

Further experiments were run on a throwing motion with the larger figure (see bottom of Figure 32). As in the kicking motion there are 4 keyframes, and a constraint for the velocity of the hand. Experiments were run both holding the time points constant and with the full model. The ability to relax the time values shows a marked improvement in finding a lower energy (and more natural looking) solution. The roughly doubled number of DOF led to approximately 5 second iterations, with the solution taking 20 iterations. The gradient determination is at worst quadratic in the size of the tree describing the figure.

A multisegment motion was constructed starting from a standing position, the soccer player walks, then runs and finally kicks the ball into the goal to score. The run, walk and kick motions are created first. Then the three transition motions are created: from initial standing to a periodic walk, from walking to running, and from running to the kick. Transition phases may take their first and last keyframes from other segments and may at the animator's discretion have intermediate keyframes specified. Each piece of the motion including transitions is individually optimized, and the results are spliced together. Each individual part took from 2 to 6 iterations, or about 3 to 10 seconds of CPU time.

4.5 Conclusions

The goal of combining intuitive user control with automated solutions for animation sequences is a delicate balancing act. This chapter has presented a new animation paradigm to achieve a good balance in this continuum. The animator is required to specify keyframes for which he intends the figure to pass through, as well as any additional key constraints on the figure's motion. The velocities and timing are then computed by an optimization process. The inclusion of the time points within the optimization is distinct from other systems and has proven to be an important addition. The DOF functions are represented by piecewise hermite splines. Initial results of the use of such a system were reported.

This paradigm effectively unburdens the optimization process so that it runs fast enough for use in an interactive setting. It has been shown to be able to effectively create graceful and realistic 3D complex figure animations. In addition, since the Hermite representation and the gradient computation are easy to implement, this method can be integrated in any keyframe system.

Chapter 5

Hierarchical Spacetime Control

5.1 Introduction

In this chapter, we describe a hierarchical scheme to solve the general spacetime constraint problem more efficiently by using a wavelet representation. As we will discuss, this method achieves both faster convergence and faster iterations than previous methods. This work has been reported in the paper[40].

5.2 Hierarchical B-splines

As described in Section 2.7, solving the spacetime constraint problem requires restricting the solution to some finite dimensional function space, leading to a finite number of scalar unknowns. The possible trajectories a particular DOF can take are thus restricted to be a linear combination of basis functions chosen to represent the DOF motion curve. In the previous chapter, we restricted the solution to a combination of a small number of Hermite basis functions indirectly indicated by the user. In this chapter, we remove this restriction by returning to the more general problem using n basis functions,

$$\theta(t) = \sum_{i=1}^{n} c_i \, \phi_i(t) \tag{205}$$

where the coefficients c_i scale the basis functions $\phi_i(t)$. In the spacetime constraint systems to date, Witkin and Kass [56] used discretized functions consisting of evenly spaced points in time from which derivatives were approximated by finite differencing. Cohen represented the DOF functions as uniform cubic B-spline curves, with some provision to change the resolution of the B-splines within specified regions of the curve.

The more basis functions and corresponding coefficients that are used, the larger the space of possible solutions. Unfortunately, for two reasons, one pays a high cost in terms of computational resources for this extra freedom. The extra unknown coefficients translate into larger subproblems at each iteration of the solution. In addition, discretizations of this type also lead to ill-conditioned systems requiring more iterations to solve [53].

Ideally, one would like to select a function space with just enough freedom to allow an almost optimal answer. In smooth portions of the trajectories, basis functions can be wider and in regions where the trajectory varies quickly, there should be more, narrower bases. Unfortunately, the optimal trajectories are not known in advance and thus a more flexible basis must be developed that can *adapt* to the local detail in the trajectories as the iterative solution proceeds.

Hierarchical systems of basis functions offer just this type of adaptivity. Hierarchical B-splines have been used in the context of shape design [23] to allow modification of curves and surfaces at levels of detail selected by the user. The hierarchical B-spline basis consists of a pyramid of translations and dilations of B-splines (the rows labeled V in Figure 33) ranging from very wide B-splines at the top to finer scaled basis functions below.

Each level going down contains twice as many basis functions per unit length. This hierarchical basis has attractive properties for use in describing the trajectories in the current application. However, this is a *redundant* basis, since any function realizable at one level can also be created from the finer basis functions below. In addition, how to achieve the desired adaptivity is not immediately apparent.

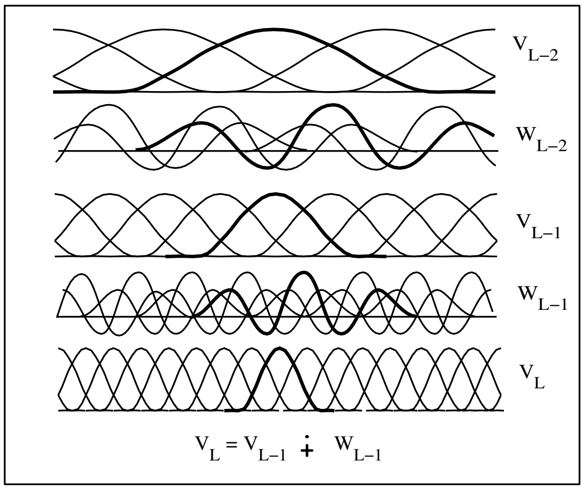


Figure 33: Hierarchy of B-spline and Wavelet Bases.

5.3 Wavelets

A more elegant and concise hierarchical basis, and one that leads naturally to an adaptive basis, is offered by a *wavelet* construction. This section concentrates on the advantages of wavelets and wavelet formulations in the spacetime animation problem.

The primary difference between wavelets and hierarchical B-splines is that the wavelet coefficients at each level represent *differences* from the levels above as opposed to directly representing the local function value. Also, unlike hierarchical B-splines, each new layer is not redundant with those above but rather adds only *local detail* in the result at some resolution.

5.3.1 Advantages of Wavelets to Spacetime Animation

The wavelet construction results in a non-redundant basis that provides the means to begin with a low resolution basis and then *adaptively refine* the representation layer by layer when necessary without changing the representation above. If refinements are required in only part of the interval, then only those coefficients whose bases have support in this region need to be added.

Since the wavelet coefficients encode differences, in smooth portions of the trajectory the coefficients encoding finer scale detail will be nearly zero. Thus, only those basis functions with resulting coefficients greater than some ϵ will have a significant influence on the curve and the rest can be ignored. In other words, given an *oracle* function [26, 25], (discussed later) that can predict which coefficients will be above a threshold, only the corresponding subset of wavelets needs to be included.

Solutions to the non-linear spacetime problem, as discussed in more detail below, involve a series of quadratic subproblems for which the computational complexity depends on the number of unknown coefficients. The smaller number of significant unknown coefficients in the wavelet basis provide faster iterations. In addition, the wavelet basis provides a better conditioned system of equations than the uniform B-spline basis, and thus requires less iterations. The intuition for this lies in the fact that there is no single basis in the original B-spline basis that provides a global estimate of the final trajectory (i.e., the locality of the B-spline basis is, in this case, a detriment). Thus, if the constraints and objective do not cause interactions across points in time, then information about changes in one coefficient travels very slowly (in $O(n)$ iterations) to other parts of the trajectory. In contrast, the hierarchical wavelet basis provides a shorter ($O(\log(n))$) "communication" distance between any two basis functions. This is the basic insight leading to *multigrid* methods [53], and the related hierarchical methods discussed here. In the next section, we will report some test results on quadratic functions showing that using a wavelet representation is usually faster than non-hierarchical representations when both the solution and the initial guess are smooth functions. For nonsmooth functions, especially when only the finest detail coefficients are significant (i.e., there is no reasonable global estimate), the wavelet representation will not perform well. But in practice, functions are usually smooth so that wavelet representation achieves faster convergence.

An additional benefit involves the integration of the objective. Since a lower resolution results, by definition, in a smoother trajectory, less samples must be taken during the numerical quadrature. As wavelets are added in particular regions, the sampling density only needs to be increased in these areas.

Finally, the wavelet representation allows the user to easily lock in the coarser level solution and only work on details simply by removing the coarser level basis functions from the optimization. This provides the means to create small systems that solve very rapidly to refine the finest details in the trajectories.

5.3.2 Quadratic Function

In this section, we report some test results on a simple quadratic function to show that the wavelet representation leads to faster convergence.

The optimization problem considered is simply to

$$\text{minimize } \int_0^1 (\theta(t) - \theta^*(t))^2 f(t)dt \tag{206}$$

where $\theta^*(t)$ and $f(t)$ are known functions, and $f(t) > 0$ for all $t \in [0,1]$. The optimum solution is clearly $\theta = \theta^*$.

Suppose

$$\theta^*(t) = \sum_{j=0}^{2^L-1} c_j^* \phi_{L,j} \tag{207}$$

where ϕ is the scale function used in the Haar wavelet construction as described in Section 2.9.1, and L is some given integer. If we also represent $\theta(t)$ as a linear combination of these basis functions, that is,

$$\theta(t) = \sum_{j=0}^{2^L-1} c_j \phi_{L,j}, \tag{208}$$

then

$$\begin{aligned}
\textstyle\int_0^1 (\theta(t) - \theta^*(t))^2 f(t)dt &= \textstyle\int_0^1 \sum_{j=0}^{2^L-1} (c_j\phi_{L,j} - c_j^*\phi_{L,j})^2 f(t)dt \\
&= \textstyle\int_0^1 \sum_{0\le i,j\le 2^L-1} (c_i - c_i^*)(c_j - c_j^*)\phi_{L,i}\phi_{L,j} f(t)dt \\
&= \textstyle\int_0^1 \sum_{j=0}^{2^L-1} (c_j - c_j^*)^2 \phi_{L,j} f(t)dt \\
&= \textstyle\int_0^1 \sum_{j=0}^{2^L-1} (c_j - c_j^*)^2 \int_{\frac{j}{2^L}}^{\frac{j+1}{2^L}} f(t)dt
\end{aligned} \tag{209}$$

where the third equality holds because for all $t \in [0,1]$

$$\phi_{L,i}(t) * \phi_{L,j}(t) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{210}$$

Let $c = (c_0, c_1, ..., c_{2^L-1})^T$, $c^* = (c_0^*, c_1^*, ..., c_{2^L-1}^*)^T$, and

$$M = \begin{pmatrix} f_0 & & & \\ & f_1 & & \\ & & \ddots & \\ & & & f_{2^L-1} \end{pmatrix} \tag{211}$$

where $f_j = \int_{\frac{j}{2^L}}^{\frac{j+1}{2^L}} f(t)dt$. Then

$$\int_0^1 (\theta(t) - \theta^*(t))^2 f(t)dt = (c - c^*)^T M(c - c^*) \tag{212}$$

So the problem 206 becomes a quadratic minimization problem:

$$\text{minimize } (c - c^*)^T M(c - c^*). \tag{213}$$

We denote this problem by $\aleph(M, c^*)$.

As shown in Section 2.9.1, we can also use Haar basis functions $\{\phi_{0,0}\} \cup \{\psi_{l,j} | 0 \leq j \leq 2^l - 1, 0 \leq l \leq L - 1\}$ (see Section 2.9.1) to represent $\theta^*(t)$. Denote d^* to be the coefficients of these basis functions. Section 2.9.1 showed that there is a linear transformation (pyramid transformation) between c and d. Denote U to be the matrix such that $c^* = Ud^*$. Let d be the Haar wavelet basis coefficients of θ, that is, $c = Ud$, then

$$(c - c^*)^T M(c - c^*) = (d - d^*)^T U^T MU(d - d^*) \tag{214}$$

So the problem $\aleph(M, c^*)$ is transformed into the problem $\aleph(U^T MU, U^{-1}c^*)$.

In our experiments, $L = 4$. for each $k, l \in \{1, 3, 5, 7, 9, 11\}$, we randomly pick numbers from $[10 - k * 0.1, 10 + k * 0.1]$ to form 100 c^*'s, and then for each c^*, we randomly pick numbers from $[20 - l * 0.1, 20 + l * 0.1]$ to form a vector c^0 which is used as the initial guess for $\aleph(M, c^*)$. The initial guess for $\aleph(U^T MU, U^{-1}c^*)$ is $U^{-1}c^0$.

Then we run the gradient decent optimization algorithm (see Section 2.10.1) on both problems for 10 iterations, and compare the objective function values. The one with the smaller objective value is considered to be faster. Among the 100 tests, we use $w(l, k)$ to denote the number of times that $\aleph(U^T MU, U^{-1}c^*)$ is faster than $\aleph(M, c^*)$. The results are in the table 1.

From the table we can see that using wavelet basis is usually faster than using box basis when k is small. Since each component of c^* is randomly chosen from $[10 - k * 0.1, 10 + k * 0.1]$, the smaller the k , the smoother the function θ^*. Therefore, this table basically shows that when θ^* is smooth, the wavelet representation is usually faster than a flat basis.

This table also shows that when the optimum solution is smooth, we should choose smooth functions as initial guesses to achieve fast convergence.

l\k	1	3	5	7	9	11
1	90	80	63	37	17	5
3	87	75	63	41	23	6
5	79	72	61	44	27	14
7	69	62	51	40	21	13
9	63	59	48	35	25	12
11	53	47	40	32	20	15

Table 1: Comparison of the convergence speeds of using Haar bwavelet basis vs. using box basis. Each entry $w(l,k)$ represents the number of times among 100 tests that the wavelet basis is faster than the box basis.

5.3.3 Choice of Wavelets

The wavelet basis we choose is Chui-Wang B-wavelet basis on bounded interval as described in section 2.9.3. We choose wavelets on bounded interval because animation problems are considered over some finite time interval. We choose Chui-Wang B-wavelets because of their smoothness (they have continuous second order derivatives), semi-orthogonality and compactness. The requirement for smoothness is obvious since we need to compute second order derivatives. The semi-orthogonality property results in best approximations of functions. The desire for compactness is for locality.

5.3.4 Scaling

One final issue is the scaling ratio between the basis functions. Traditionally [12] the wavelet functions are defined with the following scaling:

$$\begin{aligned} \phi_{i,j}(t) &= 2^{(i-L)/2}\ \phi(2^{(i-L)}t - j) \\ \psi_{i,j}(t) &= 2^{(i-L)/2}\ \psi(2^{(i-L)}t - j) \end{aligned} \tag{215}$$

This means that at each level up, the basis functions become twice as wide, and are scaled $\frac{1}{\sqrt{2}}$ times as tall. While in many contexts this normalizing may be desirable, for optimization purposes it is counter productive. For the optimization procedure to be well conditioned [15] it is advantageous to emphasize the coarser levels and hence use the scaling defined by

$$\begin{aligned}\phi_{i,j}(t) &= 2^{L-i}\ \phi(2^{(i-L)}t - j)\\ \psi_{i,j}(t) &= 2^{L-i}\ \psi(2^{(i-L)}t - j)\end{aligned} \tag{216}$$

where the wider functions are also taller. In the pyramid code, this is achieved by multiplying all of the h and g entries by 2.

5.4 Implementation

The input to the wavelet spacetime problem includes the creature description, the objective function (i.e., symbolic expressions of joint torques generated from the creature description), and user defined constraints specifying desired actions (throw, catch, etc.), and inequality constraints such as joint limits on the elbow. For small figures, one can use either the symbolic system as described in Figure 19 or the one as described in Figure 24. For large figures, the later one is the only feasible choice. Since our experiments only use small figures, we have chosen to use the first symbolic method. As discussed in Section 2.12, the symbolic expressions are differentiated and compiled into DAGs.

At this point, a constrained variational problem is defined

$$\begin{aligned}&\text{minimize} && f(\Theta, \dot{\Theta}, \ddot{\Theta})\\ &\text{subject to} && C_i(\Theta, \dot{\Theta}, \ddot{\Theta}) = 0,\ i \in n_{eq}\\ & && C_i(\Theta, \dot{\Theta}, \ddot{\Theta}) \geq 0,\ i \in n_{ineq}\end{aligned} \tag{217}$$

where Θ is the vector of trajectories of the degrees of freedom of the creature.

Each trajectory, $\theta(t)$, is represented in the uniform cubic B-spline basis. The unknowns are then the B-spline coefficients, $\mathbf{c}$, or the equivalent wavelet coefficients, $\mathbf{w}$, scaling the individual basis functions. This finite set of coefficients provide the information to evaluate the $\theta(t)$, $\dot{\theta(t)}$, and $\ddot{\theta(t)}$ at any time t, that comprise the leaves of the DAGs. This finite representation transforms the variational problem into a constrained non-linear optimization problem.

The penalty method as described in Section 2.10.3 is used to transform the constrained optimization problem into an unconstrained optimization problem. The BFGS method is used to solve the resulted unconstrained optimization problem.

BFGS iterations begin with a user provided initial guess of wavelet coefficients $\mathbf{c}_0$ (that can be derived from B-spline coefficients using `coef_pyrm_up`) and a guess $\mathbf{H}_0$ of the inverse of the Hessian (usually an identity matrix leading to the first iteration being a simple gradient descent).

At each iteration, the gradient with respect to the wavelet coefficients are computed and the new (better) solution $\mathbf{w}_k$ is obtained. This solution can be transformed back to B-spline coefficients by using `coef_pyrm_down` for display.

If the initial function space is restricted to a coarse representation consisting of the broad B-splines and a single level of wavelets, after each iteration a simple *oracle* function (see the next paragraph) may be called to add wavelets at finer levels only when the wavelet coefficient above exceeds some tolerance. This procedure quickly approximates the optimal trajectory and smoothly converges to a final answer with sufficient detail in those regions that require it.

Each time when the oracle is called, it adds wavelet coefficients at finer levels to refine the DOF representation. To decide where to add coefficients, for each wavelet coefficient $w_{l,j}$ (of the basis function $\psi_{l,j}$) whose absolute value is bigger than some ϵ, the oracle adds all the coefficients $w_{l+1,k}$ of the finer level wavelet basis functions $\psi_{l+1,j}$ whose supports are subsets of the support of $\psi_{l,j}$. When to call the oracle can be decided by checking the progress of the optimization process. If the the difference of the objective function at two consecutive iterations is below some threshold, it can be assumed that the local minimum at the current resolution has been nearly achieved, thus the oracle is called so that the optimization process continues at a finer representation.

An important feature of the system discussed in [13] is also available in the current implementation. The user can directly modify the current solution with a simple key frame system to help *guide* the numerical process. This is critical to allow the user, for example, to move the solution from an underhand to an overhand throw, both of which represent local minima in the same optimization problem. The next iteration then begins with these new trajectories as the current guess.

5.5 Results

A set of experiments was run on the problem of a three-link arm and a ball (see Figure 35). The goal of the arm is to begin and end in a rest position hanging straight down, and to throw the ball into a basket. The objective function is to minimize energy, where energy is defined as the integral of the sum of the squares of the joint torques. Gravity is active.

The four graphs in Figure 34 show the convergence of four different test runs of the arm and ball example. The short dashed line, solid line and long dashed line represent the convergence with B-spline representation, full wavelet representation and adaptive wavelet representation, respectively. Each plot differs only in the starting trajectories of the arm DOF. Each run converged to either an underhand or overhand throw into the basket. The full B-spline basis contained 67 basis functions for each of the three DOF, thus there were 201 unknown coefficients to solve for. Iterations took approximately 7 seconds each on an SGI workstation with an R4000 processor. Convergence was achieved on each, but only after many iterations due to the ill-conditioning of the B-spline formulation.

The full wavelet basis also contained 67 basis function per DOF (11 B-splines at the top level and 56 wavelets below), thus iterations also took approximately the same 7 seconds. Figure 34 clearly shows the improved convergence rates of the wavelet formulations over the B-spline basis, due to better conditioned linear systems. The adaptive wavelet method with the oracle was the fastest since the number of unknowns was small in early iterations, leading to a very fast approximation of the final trajectories, in addition to the better conditioning provided by the hierarchical basis. The final few iterations involved more wavelets inserted by the oracle to complete the process. Note that in each case, a good approximation to the complete animation was achieved in less than a minute of computation.

A short sequence involving two basketball players with six degrees of freedom each (see Figures 35,36) was animated. The task was a "give and go" play. Player A passes the ball to player B, then moves towards the basket. Player B passes it pack to A who makes the shot. This animation was created in stages: first player A throws the ball to a location set by the user, then player B's actions are animated to catch the ball, then player B's throw is animated followed by player A catching this throw and

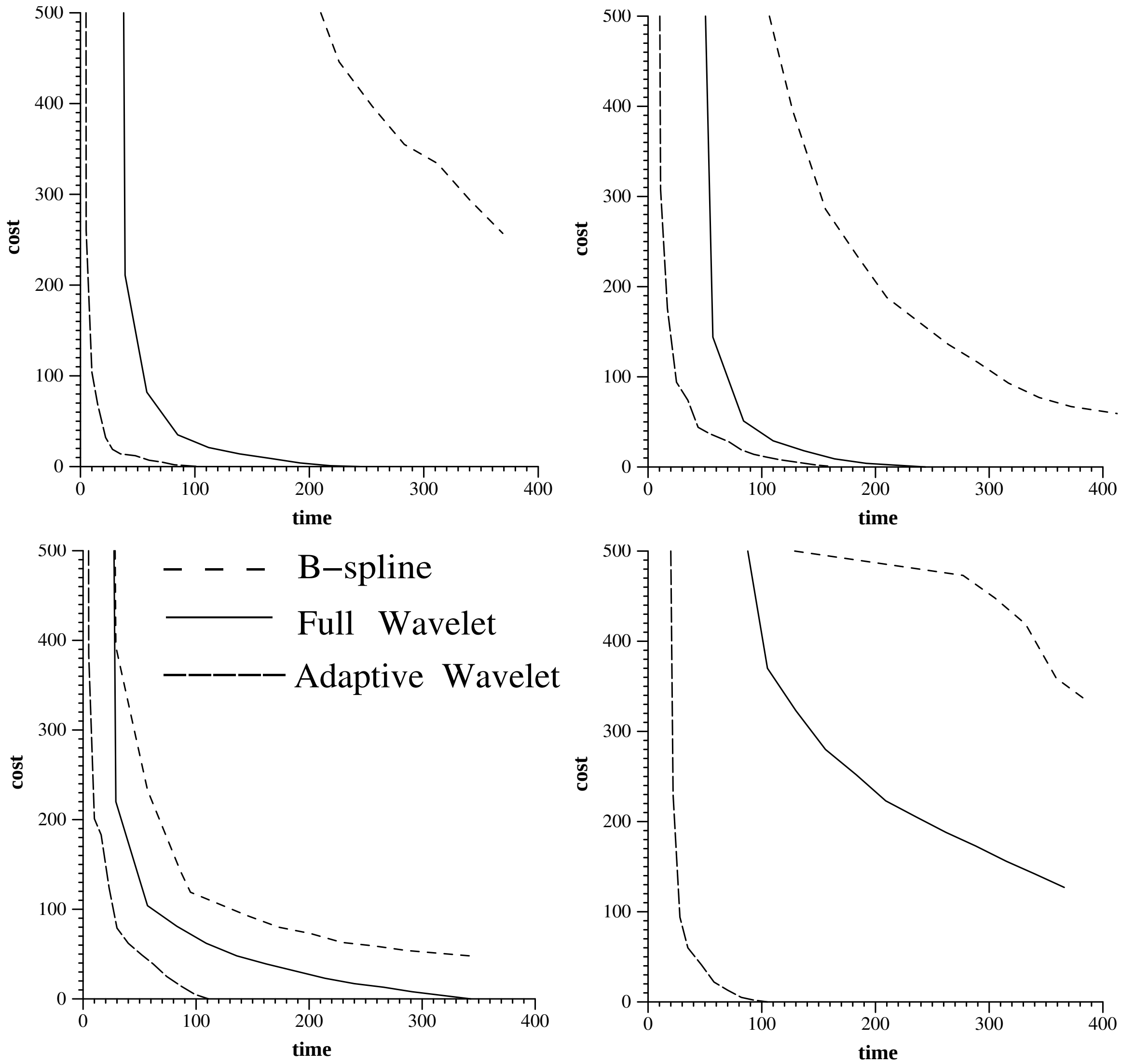


Figure 34: Convergence of Arm and Ball example for 4 different starting trajectories. The first and fourth examples resulted in underhand throws, and the rest overhand. Time is in seconds, and the cost is a weighted sum of constraint violations and energy above the local minimum.

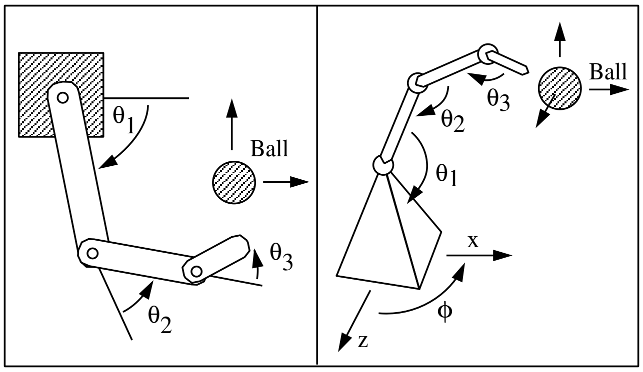


Figure 35: A planar three-link arm and a 6 DOF basketball player.

making the basket. Each stage of the animation took between 10 and 25 iterations of approximately 6-10 seconds each. The longer iteration times are due to the 6 DOF of each creature leading to twice the number of unknowns. Figure 37 shows the motion sequence.

5.6 Conclusion

The spacetime constraint system first suggested by Witkin and Kass [56] for animating linked figures has been shown to be an effective means of generating goal based motion. Cohen enhanced this work by demonstrating how to focus the optimization step on *windows* of spacetime to do local refinement. This chapter has extended this idea to a general hierarchical scheme.

The improvement lies in the representation of the trajectories of the DOF in a wavelet basis. This resulted in faster optimization iterations due to less unknown coefficients needed in smooth regions of the trajectory. In addition, even with the same number of coefficients, the systems become better conditioned and thus less iterations are required to settle to a local minimum. Results are shown for a planar three-link arm and two six DOF “basketball players”.

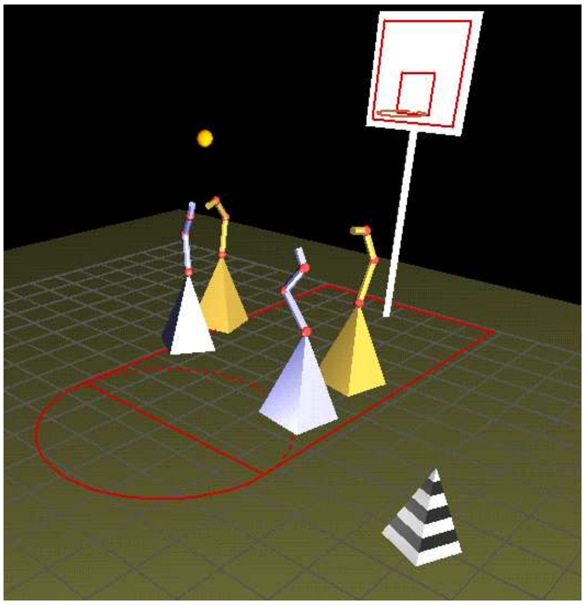


Figure 36: Scene from a basketball game.

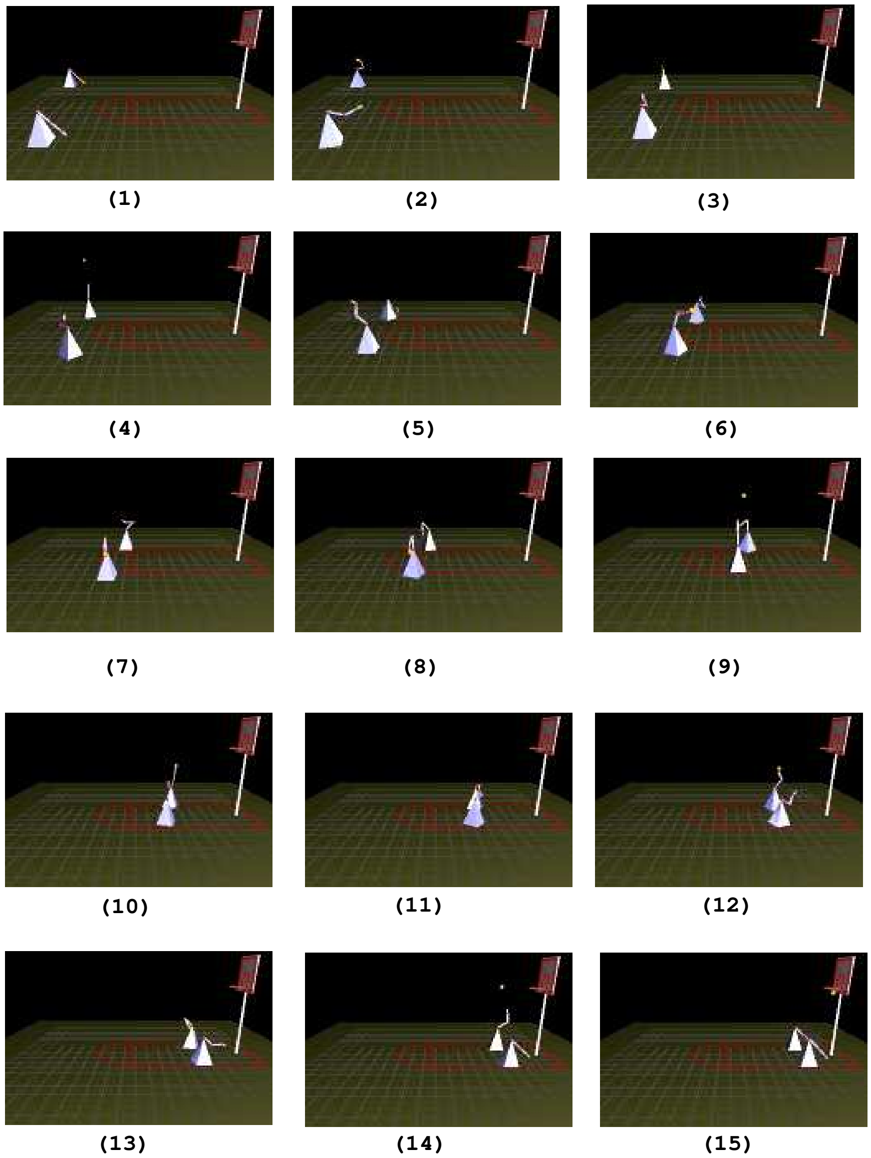


Figure 37: The two basketball players

Chapter 6

Conclusions and Future Research

Despite continued research on physically based animation approaches, keyframing systems are still the predominant tool for animation. This is due to the control provided to the animator and the fast computation of inbetween frames. How to provide control over a dynamic system while achieving efficiency is essential for physically based animation systems to become useful in practice.

6.1 Contributions

This thesis has presented the progress we have made in this direction. In particular, we made improvements to the spacetime constraint system both symbolically and numerically. The symbolic method as described in Chapter 3 provides a convenient and general interface with fast dynamics and gradient computations. This symbolic method makes it possible to apply spacetime constraint methods to complex figures. The keyframe optimization system as described in Chapter 4 integrates dynamics into a keyframe system to achieve good user control while taking advantage of the optimization system. The notion of allowing the user to specify key frames while letting the computer decide speed and timing is valuable, and the method has been shown to be able to animate complex human figures at nearly interactive computation speeds. The hierarchical spacetime constraint method with its wavelet representation of the DOF functions has been shown to significantly speed up computations in the more general spacetime constraint system. The hierarchical refinement scheme with local details added layer by layer wherever necessary is shown to be effective for generating high quality goal directed motion of linked figures.

6.2 Future Work

One limitation with our system is that it is difficult to deal with contacts and collisions due to the resulting discontinuities in the trajectories. Contacts and collisions are important behaviors, for example, when a soccer player kicks a ball, there is an impact force on him (her) which results in extra motions to respond to the impact force. How to incorporate contacts and collisions into a trajectory based system is still an open problem.

Another practical issue that needs to be addressed is the user interface. In particular, we need to provide more graphically based and/or intuitive means to generate, modify and inspect the expressions of constraints and objective in the language as described in Chapter 3, so that the system is easier to use and more productive.

Reusability is another interesting problem. Each trajectory corresponds to just one motion. If we change the environment or the figure, the solution has to change. Ideally, we could somehow modify the previous solution without having to run the optimization process all over again. For example, consider the basketball player. If we change the position of the basket, then the same throw won't make the goal any more, but if the change of the basket position is not too large, an accurate throw to the new basket position may be very similar to, though not exactly the same as, the previous one. The previous solution can be used as the initial guess for the new solution to save the number of iterations. An interesting question is: can we totally avoid runing the optimization process? One possible approach would be to modify the trajectory of the end point (the hand) in such a way that the difference is minimum (based on some sort of measurement) and the ball goes into the basket. Then we may find the DOF trajectories simply using inverse kinematics.

A solution to the reusability problem would also have applications in motion capture. How to generate new motions from the recorded motion is a central problem to make motion capture more productive. A solution to reusability problem may provide one way of generating new motions.

Task hierarchy is another possible direction to speed up computations. Again let's consider an arm throwing a ball into a basket. In our experiments so far, the hand does not have fingers. If we want to consider all the fingers, the resulting optimization problem would become extremely large. However, in real life, the fingers of a real basketball player move significantly only at the moment when the ball leaves the hand to adjust the speed of the ball so that it can make the goal. In order words, the shoulder, elbow and wrist are controlling the motion in a large scale, while the fingers are making finer adjustments. Therefore it may be reasonable to compute the rough motion first by only considering the significant joints (shoulder, elbow and wrist) and not having to require the ball make the goal exactly. The resulting motion, though not an accurate throw, would roughly determine the style of the motion. Then we can compute the trajectories of those insignificant joints (fingers) so that the throw is actuate. Since the fingers move significantly only at the moment of throwing, we can limit the time interval to be a small neighborhood of the throwing moment to save computations.

Bibliography

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Priciples, Techniques and Tools.* Addison Wesley, 1986.

[2] Bill Armstrong and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. In *Proceedings of Graphics Interface*, pages 407–415. Computer Graphics Society, May 1986.

[3] D. Baraff. Analytical models for dynamic simulation of non-penetrating rigid bodies. *ACM Computer Graphics*, 23(3):223–232, July 1989.

[4] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, CA, 1990.

[5] Richard Bartels, John Beatty, and Brian Barsky. *An Introduction to Splines for Use in Computer Graphics and Modeling.* Morgan Kaufmann, 1987.

[6] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In *Proceedings of SIGGRAPH'88 (Atlanta, Georgia, August 1–5, 1988)*, pages 179–188. ACM, August 1988.

[7] Lynn Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. In *Proceedings of SIGGRAPH'88 (Atlanta, Georgia, August 1–5, 1988)*, volume 22, pages 309–315. ACM, August 1988.

[8] A. Bruderlin and T. Calvert. Goal-directed, dynamic animation of human walking. *ACM Computer Graphics*, 23(3):233–422, July 1989.

[9] E. Catmull. The problems of computer-assisted animation. In *SIGGRAPH*, pages 348–353, 1978.

[10] Jim X. Chen, Niels da Vitoria Lobo, Charles E. Hughes, and I. Michael Moshell. Simulation and synchronization of fluids in a dis. In *The First Workshop on Simulation and Interaction in Virtual Enviroments*, pages 159–167, University of Iowa, July 13-15 1995.

[11] Charles Chui and Ewald Quak. Wavelets on a bounded interval. *Numerical Methods of Approximation Theory*, 9:53–75, 1992.

[12] Charles K. Chui. *An Introduction to Wavelets*, volume 1 of *Wavelet Analysis and its Applications*. Academic Press Inc., 1992.

[13] Michael F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[14] John J. Craig. *Introduction to Robotics*. Addison-Wesley, Reading, MA, 1986.

[15] Wolfgang Dahmen and Angel Kunoth. Multilevel preconditioning. *Numerische Mathematik*, 63:315–344, 1992.

[16] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF Series in Appl. Math., SIAM publications, Philadelphia., 1992.

[17] Garcia de Jalon Javier. *Kinematic and dynamic simulation of multibody systems: the real-time challenge*. New York: Springer-Verlag, 1994.

[18] L. C. W. Dixon and G. P. Szego. *Towards Global Optimization*. North-Holland, Amsterdam, 1975.

[19] D.N.Burghes and A.M.Downs. *Modern Introduction to Classical Mechanics and Control*. Ellis Horwood, 1975.

[20] Roy Featherstone. *Robot dynamics algorithms*. Boston:Kluwer, 1987.

[21] R. Fletcher. *Practical Methods of Optimization, Vol. 1 and 2*. John Wiley and Sons, 1980.

[22] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, Reading, Massachusetts, 2 edition, 1990.

[23] David Forsey and Richard Bartels. Hierarchical b-spline refinement. *Computer Graphics*, 22(4):205–212, August 1988.

[24] Michael Girard and A.A. Macielewski. Computational modeling for the computer animation of legged figures. In *Proceedings of SIGGRAPH'85 (San Francisco, California, July 22–26, 1985).* ACM, July 1985.

[25] Steven Gortler and Michael F. Cohen. Variational modeling with wavelets. Technical Report CS-TR-456-94, Department of Computer Science, Princeton University, 1994.

[26] Steven Gortler, Peter Schröder, Michael Cohen, and Pat Hanrahan. Wavelet radiosity. In *Computer Graphics, Annual Conference Series, 1993*, pages 221–230. Siggraph, August 1993.

[27] James Hahn. Realistic animation of rigid bodies. In *Proceedings of SIGGRAPH'88 (Atlanta, Georgia, August 1–5, 1988)*, pages 299–308. ACM, August 1988.

[28] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of Siggraph*, Los Angeles, CA, August 6-11 1995.

[29] John M. Hollerbach. A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):730–736, November 1980.

[30] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions, and inverse dynamics. In *Proceedings of SIGGRAPH'87 (Anaheim, California, July 27–31, 1987)*, pages 215–224. ACM, July 1987.

[31] Paul M. Isaacs and Michael F. Cohen. Mixed methods for kinematic constraints in dynamic figure animation. *The Visual Computer*, 4, 1988.

[32] Thomas R. Kane. *Dynamics*. Stanford University, 1972.

[33] Michael Kass. Condor: Constraint-based dataflow. In *Proceedings of SIGGRAPH'92 (Chicago, July 26–31, 1992)*, pages 321–330. ACM, July 1992.

[34] James U. Korein and Norman I. Badler. Techniques for generating goal-directed motion of articulated structures. *IEEE Computer Graphics and Applications*, 2(6):71–81, November 1982.

[35] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH*, pages 35–44, 1987.

[36] Philip Lee, Susanna Wei, Jianmin Zhao, and Norman I. Badler. Strength guided motion. In *Proceedings of SIGGRAPH'90 (Dallas, Texas, August 6–10, 1990)*, pages 253–262. ACM, August 1990.

[37] Zicheng Liu and Michael F. Cohen. Decomposition of linked figure motion: Diving. In *Proceedings of 5th EuroGraphics Workshop on Animation and Simulation*, (Oslo, Norway, September2-3 1994).

[38] Zicheng Liu and Michael F. Cohen. An efficient symbolic interface to constraint based animation systems. In *Proceedings of 6th EuroGraphics Workshop on Animation and Simulation*, (Maastricht, The Netherlands, September2-3 1995). Springer-Verlag.

[39] Zicheng Liu and Michael F. Cohen. Keyframe motion opimization by relaxing speed and timing. In *Proceedings of 6th EuroGraphics Workshop on Animation and Simulation*, (Maastricht, The Netherlands, September2-3 1995). Springer-Verlag.

[40] Zicheng Liu, Steven Gortler, and Michael F. Cohen. Hierarchical spacetime control. *Computer Graphics*, pages 35–42, July 1994.

[41] Gavin S. P. Miller. The motion dynamics of snakes and worms. In *Proceedings of SIGGRAPH'88 (Atlanta, Georgia, August 1–5, 1988)*, pages 169–173. ACM, August 1988.

[42] Brian Mirtich and John Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, 1995.

[43] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In *Computer Graphics, Annual Conference Series, 1993*, pages 343–350. Siggraph, August 1993.

[44] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. In *Proceedings of SIGGRAPH'89 (Boston, Mass., July 31–Aug 4, 1989)*, pages 215–222. ACM, July 1989.

[45] Cary B. Phillips, Jianmin Zhao, and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. In *Proceedings of Symposium on Interactive 3D Graphics (Snowbird, Utah, March, 1990)*, volume 24, pages 245–250. ACM, March 1990.

[46] Gary B. Phillips and Norman I. Badler. Interactive behaviors for bipedal articulated figures. *Computer Graphics*, 25(4):359–362, July 1991.

[47] W. H. Press and B. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[48] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipies*. Cambridge University Press, 1986.

[49] Willaim Press, S. Teukolski, W Vetterling, and B Flannery. *Numerical Recipies in C, The Art of Scientific Computing*. Cambridge University Press, 2 edition, 1992.

[50] Ewald Quak and Norman Weyrich. Decomposition and reconstruction algorithms for spline wavelets on a bounded interval. Technical Report 294, Center for Approximation Theory, Texas A&M, 1993.

[51] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *siggraph91*, 4(25):349–358, 1991.

[52] S. M. Roberts and J. S. Shipman. *Two-Point Value Problems: Shooting Methods.* American Elsevier, 1972.

[53] Demetri Terzopoulos. Image analysis using multigrid relaxation methods. *IEEE PAMI*, 8(2):129–139, March 1986.

[54] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In *Computer Graphics, Annual Conference Series, 1993*, pages 335–342. Siggraph, August 1993.

[55] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[56] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

[57] Jianmin Zhao and Norman I. Badler. Real time inverse kinematics with joint limits and spatial constraints. *Technical Report MS-CIS-89-09, Department of Computer and Information Science, University of Pennsylvania*, 1989.