# Automating gait generation

Harold C. Sun[1] Dimitris N. Metaxas[2]

University of Pennsylvania

## Abstract

One of the most routine actions humans perform is walking. To date, however, an automated tool for generating human gait is not available. This paper addresses the gait generation problem through three modular components. We present ElevWalker, a new low-level gait generator based on *sagittal elevation angles*, which allows curved locomotion - walking along a curved path - to be created easily; ElevInterp, which uses a new *inverse motion interpolation* algorithm to handle uneven terrain locomotion; and MetaGait, a high-level control module which allows an animator to control a figure's walking simply by specifying a path. The synthesis of these components is an easy-to-use, real-time, fully automated animation tool suitable for off-line animation, virtual environments and simulation.

**Keywords:** Animation, animation systems, animation w/ constraints, human body simulation

## 1 Introduction

One of the most routine actions humans perform is walking. It is also one of the most important actions to animate [29, 4, 3, 19, 5, 16, 17, 22, 27]. To date, however, an automated tool for generating human gait is not available. The difficulty arises on two levels: at the low level, most methods are not able to generate the many gait variations necessary to fulfill the functional obligations of walking: walking along an arbitrary path on possibly uneven terrain. At the high level, previous methods require the animator to control the walking by manipulating low-level parameters; this is a difficult and tedious task.

We believe that in order to be a useful tool, a gait animation system should be powerful enough to create curved locomotion (walking along a curved path) on uneven terrain. Furthermore, animators should not need to set low-level parameters of the gait system; the walking system should compute these itself, so that the animator is freed to engage in more creative and stimulating tasks. We have developed a multi-layer system which addresses these issues.

Our system is shown in Figure 1. At the base of our system is a motion generator, consisting of two parts: ElevWalker, which is capable of animating curved path human locomotion; and ElevInterp, a motion data transformation component, which allows our system to handle walking on uneven terrain. Above these, we have built a user-level control module, MetaGait, which automates the decisions of how to follow paths and terrain, how to compute the next footstep, and how to compute the values of the motion generator's parameters. Our interface allows a user to direct a human figure to "follow this path to that point", and the system takes care of the details. Other systems' interfaces often require the user to specify individual footsteps; this can be useful for the control it gives, but it is not an option for many applications, such as real-time simulation and games. Moreover, many application developers do not want to have to specify such detail - our system provides a "black box" which does the right thing, with a minimum of interaction necessary.

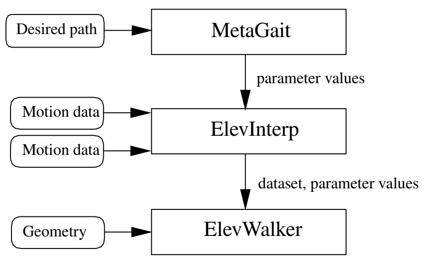


Figure 1: Our gait system components

Our system accomplishes these goals by integrating high-level knowledge with a low-level walking motion generator through a simple interface. Our ElevWalker and ElevInterp modules have a small set of high-level, intuitive parameters - such as step length, step height, heading direction and toe-out - which makes it easy for MetaGait to control the walking pattern. Within the MetaGait controller, we have embedded walking-specific rules and gait data about how these variables are interrelated, which allow it to compute their values automatically.

Our ElevWalker component introduces a new algorithm using the *sagittal elevation angle* motion representation [2, 24, 25, 23], which allows curved locomotion to be generated extremely easily. Our ElevInterp component utilizes a new technique for solving the *inverse motion interpolation* [28] problem, calculating low-level interpolation parameters so that users only work with the high-level parameters step height and step length. Our MetaGait component is the first high-level control module for human gait animation which combines control of many gait variables to ensure curved path following on uneven terrain.

The rest of the paper proceeds as follows. Section 2 describes previous work on animation of gait. Section 3 describes the kinematic model we use in this work. Section 4 describes ElevWalker, our walking motion generator based on sagittal elevation angles. Section 5 describes ElevInterp, our motion data transformation engine. We introduce MetaGait in Section 6, describing how to generate curved path and uneven surface gait. Section 7 describes results obtained with our model, and Section 8 presents a discussion and future work.

[1]University of Pennsylvania, Philadelphia, PA 19104, hsun@graphics.cis.upenn.edu
[2]University of Pennsylvania, Philadelphia, PA 19104, dnm@graphics.cis.upenn.edu

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA
© 2001 ACM 1-58113-374-X/01/08...$5.00

## 2 Related work

There has been much work on generating gait for animation. An excellent survey of these efforts can be found in [20]. There are two general approaches to generating gait: physically based or kinematics.

Modelling walking and other forms of locomotion using a physically based approach [19, 21, 10, 17, 13] has the seemingly important advantage that any generated walking solution must be physically realistic, because gravity, muscle torques and the various surface reaction forces have been accounted for.

However, upon closer inspection, this is not the guarantee it appears to be. Dynamics alone does not make a motion look like a human performed it. The set of walking motions an observer might accept as "normal human walks" is a strict subset of the physically realistic walking motions the human body can produce. Therefore, in addition to physical realism, there must be another discriminant which captures the essence of "human-ness" to a walk. The dynamics problem turns into a search for this control criterion. While biomechanics research has shown some success [1], this solution is extremely computationally expensive, and real-time animation is not currently possible.

The alternative is to use a kinematics approach. While this technique does not guarantee physical realism, there has still been good success using kinematics. This is largely due to the talents of skilled animators, as well as the increasing ease of capturing kinematic data and using these to drive animation.

The issue with using kinematics for generating walking motions [7, 3, 4, 16] has been how to generalize these kinematic data to different figures and different situations: walking style, path and terrain. Bruderlin and Calvert [4] used a dynamic model, which computed a generic walking pattern, with inverse kinematics to ensure that the swing ankle moved along a specified trajectory. Their system also allowed the user to interact with high-level parameters such as our ElevInterp module, and modelled some interactions between high-level parameters such as our MetaGait module. However, their models were not extended to the general problem of curved locomotion on uneven terrain. Ko and Badler [14, 15] used an analytic method for generalizing gait data across different stride lengths, along with an inverse dynamics post-processing step to check that the motion was feasible. Their system allowed curved path locomotion but not on uneven terrain.

These previous systems, and others, used biomechanical knowledge, such as the movement of the foot and pelvis in each of the phases of gait, along with some gait data, to produce their generalizations. Another approach is to use larger amounts of gait data to compensate for less biomechanical knowledge. Wiley and Hahn [28] used linear interpolation across a single parameter, slope angle, to generate walking on a sloped surface. This work solved the inverse interpolation problem, but assumed a linear relationship between interpolation parameter and slope parameter, and used more data than our ElevInterp approach. Rose *et al.* [22] used radial-basis function interpolation to generate curved locomotion on uneven terrain, with different emotional adverbs. They did not address how to compute interpolation coefficients given desired physical constraints.

We combine the strengths of both approaches, incorporating the use of multiple data sets as well as biomechanical knowledge. Using data sets allows us to generate uneven terrain locomotion which looks realistic. Furthermore, we embed biomechanical knowledge so that our MetaGait model can modify gait parameters in a biomechanically principled way. This allows our system to operate automatically, by knowing how step length or toe-out will affect turning radius, or how step height and step length interact.

Motion editing systems [8, 18] are invaluable tools for general-purpose motion work. However, by their nature, they require and encourage user interaction. Hence they do not fit well with the paradigm of automatic, non-interactive motion generation. Furthermore, these systems do not have the specialized knowledge about how walking parameters interact. Therefore, additional structure such as our MetaGait model is still necessary to supply this knowledge.

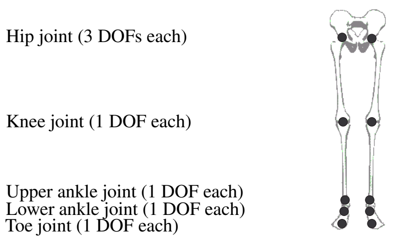


Figure 2: Kinematic model structure

## 3 Kinematic model

Our gait generator computes the motion for the lower body of a human figure, therefore our kinematic chain only incorporates the legs, feet and pelvis. Our model contains 14 joint degrees, which are more than sufficient to generate many stylistic variations in walking. The location and degrees of freedom (DOF) at each joint is illustrated in Figure 2.

We compute the joint trajectories for 14 joint DOFs. We also calculate the position and orientation of the kinematic root, which has 3 translational and 3 rotational DOFs. Our model therefore has a total of 20 DOFs.

## 4 ElevWalker: gait motion generation

In this section, we describe our gait motion generator, ElevWalker, and introduce a new algorithm using the *sagittal elevation angle* motion representation [2, 23]. This data representation, described below, makes generating curved locomotion extremely simple. ElevWalker uses a combination of motion data and procedural animation techniques to produce realistic, curved path walking motion in real-time.

ElevWalker takes motion data in the form of sagittal elevation angles. The sagittal plane is a vertical plane which bisects the figure into left and right halves. We define a limb segment $\mathbf{v}$ between two points $\mathbf{a}$ and $\mathbf{b}$ on the limb: $\mathbf{v} = \mathbf{a} - \mathbf{b}$. Typically $\mathbf{a}$ and $\mathbf{b}$ are points at opposite ends of a limb, e.g. the knee center and the ankle center are used to measure the elevation of the lower leg.

We project $\mathbf{v}$ onto the sagittal plane to form $\mathbf{v}^{\mathbf{sag}}$. The angle between $\mathbf{v}^{\mathbf{sag}}$ and the negative y axis is its sagittal elevation angle, $\phi$. If we consider the $XY$ plane to be the sagittal plane, then:

$$\tan\phi = \left(\frac{\mathbf{v}_x^{sag}}{-\mathbf{v}_y^{sag}}\right) \tag{1}$$

In addition to allowing curved locomotion to be generated, sagittal elevation angles are interesting for biomechanical reasons [2, 9, 24, 23] as well, which are beyond the scope of the current paper.

An example of marker placement and sagittal elevation angle measurement is shown in Figure 3. We have followed the definition of elevation angles and placement of markers as used in [2], with the addition of a heel marker. In our model, we use the elevation angles of four limb segments of the lower body: the foot, the lower leg, the upper leg and the pelvis. At each instant, the elevation angles for the four limb segments define a point in 4D. A walking motion dataset is thus represented concisely as a curve $c(t)$, a set

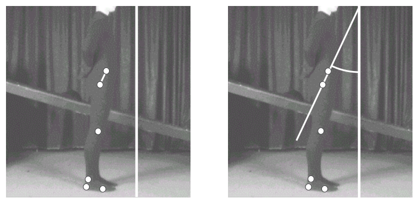


Figure 3: Measuring the pelvis sagittal elevation angle

of 4D points over time. We normalize the domain of $c(t)$ so that $t \in [0, 1)$.

Points in the range $t \in [0, .5)$ represent the elevation angles during the stance phase of gait, while points in the range $t \in [.5, 1.0)$ represent the elevation angles during swing phase. At each frame, we select two points, $c(t_i)$ and $c(t_i + .50)$, where $t_i \in [0, .5)$. The information contained in these two points can be thought of as a "silhouette" of the legs of the walking figure. ElevWalker uses this information to compute the joint angles of the lower body.

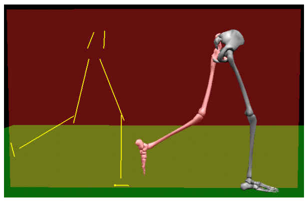


Figure 4: The limb segments' elevations must equal the elevation angle data

Each elevation angle implies the orientation of its corresponding limb segment's projection onto the sagittal plane. Therefore, we compute the figure's joint angles so that its silhouette matches the sagittal elevation angle data silhouette. This is depicted in Figure 4. We also compute the kinematic (or figure) root based on these constraints.

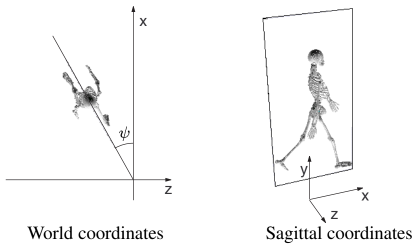


Figure 5: The sagittal plane

As a matter of definition, we parameterize the sagittal plane orientation by a single angle, $\psi$, which describes the rotation of the sagittal plane about the global y-axis. This is described by Figure 5; the figure also shows the sagittal plane coordinate system. We will use $\psi$ later to perform curved locomotion.

ElevWalker's algorithm for computing the kinematic (figure) root transformation and joint angles begins at the current stance foot. The figure root transformation is computed first. We then compute the figure's joint angles, working up the stance leg, across the pelvis, and back down the swing leg; the order of these computations is shown in Figure 6.

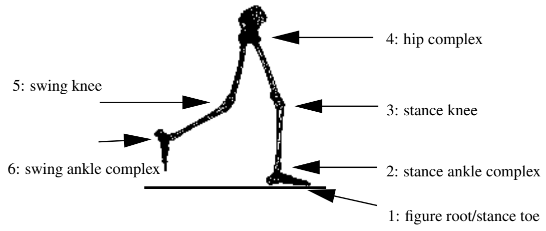


Figure 6: The joint angles and their order of computation

### 4.1 Computing the figure root

The kinematic figure root is calculated first at each time step. At all times, the figure root is located within the current stance foot. We have developed a foot-floor interaction model which maps the foot elevation angle to the kinematic root transformation, while producing realistic-looking foot and floor contact.

The foot-floor interaction model solves for a root transformation so that the foot segment satisfies the foot elevation angle constraint: the foot segment's elevation angle must equal the dataset elevation angle. The interaction model also keeps the foot in contact with the ground without penetrating the ground. Furthermore, the foot is not allowed to slide across the ground. These properties are obtained by calculating the instantaneous point of contact of the foot with the ground. The root is then updated by rotating the foot around this point to satisfy the elevation constraint. This is depicted in Figure 7. In our implementation, we compute the instantaneous point of contact using a modified version of the collision detection package V-COLLIDE [11]. Since the contact point continuously moves forward along the bottom of the foot, we must recompute the contact point at each frame.

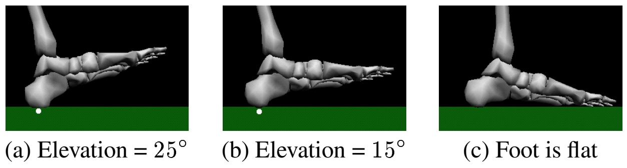


Figure 7: The foot rotates about the point of contact (in white) until it is flat on the ground

Once we have the point of contact, we rotate the foot segment so that its orientation equals the dataset foot elevation angle. The rotation axis is a vector normal to the sagittal plane, and through the point of contact. By rotating about this axis, and by recomputing the point of contact at each frame, the foot stays on the surface of the ground and does not penetrate it.

We define the foot being flat on the ground as occurring when a point in the forefoot (toe segments) and a point in the hindfoot (heel segment) are both in contact with the ground. When this condition is met, the root is moved to the toes, and the elevation of the foot is then satisfied by the first toe joint (allowing the heel to rise from the ground). The root transformation no longer changes until the swing side and stance side are swapped.

### 4.2 Computing the joint angles

After computing the figure root, we proceed to compute the joint angles, working up the stance leg and then down the swing leg. At many of these joints, we are concerned with only a single DOF and a single elevation angle constraint. For example, the stance knee joint (depicted in Figure 6) has only a single DOF. At this joint and others like it - the stance toe, the swing knee, and the swing ankle - we can compute the joint angle analytically.

We wish to compute the joint angle $\theta$ such that the sagittal elevation angle of the limb segment $\mathbf{v}$ yields the desired dataset elevation angle $\phi$. Let the joint axis $\hat{\mathbf{q}}$ pass through the joint center $\mathbf{c}$. In general, the joint axis $\hat{\mathbf{q}}$ will not be perpendicular to the sagittal plane, but skew[1].

The left-hand diagram in Figure 8 illustrates how $\phi$ changes as $\mathbf{v}$ rotates about the joint axis.

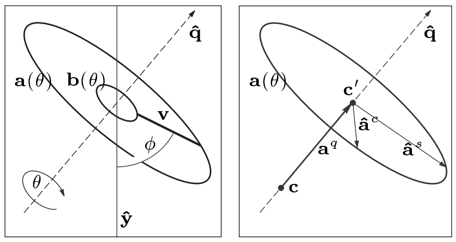


Figure 8: Solving elevation angle constraint for a single joint

Recall from Section 4 that each limb segment has two sites associated with it: $\mathbf{a}$ and $\mathbf{b}$. As $\theta$ changes, the sites $\mathbf{a}$ and $\mathbf{b}$ will trace circles about $\hat{\mathbf{q}}$. Figure 8 depicts these circles projected onto the sagittal plane; in general, the joint axis is skew to the sagittal plane, thus the projections are ellipses. Parameterizing the circle formed by $\mathbf{a}$ yields:

$$\mathbf{a}(\theta) = \mathbf{a}^c \cos(\theta) + \mathbf{a}^s \sin(\theta) + \mathbf{c}'$$

where the quantities used are defined as:

|  |  |  |  |  |
|---|---|---|---|---|
| $\mathbf{a}^q$ | $=$ | $(\mathbf{a} \cdot \hat{\mathbf{q}})\hat{\mathbf{q}}$ | $=$ | component of $\mathbf{a}$ along the axis |
| $\mathbf{c}'$ | $=$ | $\mathbf{c} + \mathbf{a}^q$ | $=$ | center of circle in space |
| $\mathbf{a}^c$ | $=$ | $(\mathbf{a} - \mathbf{a}^q)$ | $=$ | radial component of the circle in space |
| $\mathbf{a}^s$ | $=$ | $\hat{\mathbf{q}} \times \mathbf{a}^c$ | $=$ | other radial component of circle |

These quantities are shown on the right-hand diagram in Figure 8. A similar expression for $\mathbf{b}(\theta)$ arises, with $\mathbf{c}''$ as its center.

In order to compute the sagittal elevation angle of $\mathbf{v}$, we project $\mathbf{a}$ and $\mathbf{b}$ onto the sagittal plane. $\mathbf{a}$ is projected onto the sagittal plane by $\mathbf{a}^{sag} = R(\psi)^{-1}\mathbf{a}$, where $\psi$ describes the orientation of the sagittal plane (see Figure 5) and $R(\psi)$ is a rotation about the y-axis. This yields:

$$\begin{aligned} \mathbf{a}^{sag}(\theta) &= R(\psi)^{-1}\mathbf{a}(\theta) \qquad (2)\\ &= [R(\psi)_x \cdot \mathbf{a}(\theta), R(\psi)_y \cdot \mathbf{a}(\theta), R(\psi)_z \cdot \mathbf{a}(\theta)] \end{aligned}$$

where $R(\psi)_x$, $R(\psi)_y$ and $R(\psi)_z$ are the first, second and third column vectors of $R(\psi)$, respectively. $\mathbf{b}^{sag}(\theta)$ has a similar expression.

The elevation angle constraint (Equation 1) relates $\mathbf{a}^{sag}$ and $\mathbf{b}^{sag}$ to the elevation angle $\phi$:

$$\frac{\mathbf{a}_x^{sag} - \mathbf{b}_x^{sag}}{\mathbf{a}_y^{sag} - \mathbf{b}_y^{sag}} = -\tan\phi \qquad (3)$$

Substitute $\mathbf{a}^{sag}(\theta)$ and $\mathbf{b}^{sag}(\theta)$ into Equation 3. Letting $\mathbf{d} = \mathbf{a}^c - \mathbf{b}^c$, $\mathbf{e} = \mathbf{a}^s - \mathbf{b}^s$, and $\mathbf{f} = \mathbf{c}' - \mathbf{c}''$ yields:

$$A\cos\theta + B\sin\theta + C = 0 \qquad (4)$$

[1]This is particularly true at the ankle joints [12].

where

$$\begin{aligned} A &= (R(\psi)_x \cdot \mathbf{d}) + \tan\phi(R(\psi)_y \cdot \mathbf{d}) \qquad (5)\\ B &= (R(\psi)_x \cdot \mathbf{e}) + \tan\phi(R(\psi)_y \cdot \mathbf{e}) \qquad (6)\\ C &= (R(\psi)_x \cdot \mathbf{f}) + \tan\phi(R(\psi)_y \cdot \mathbf{f}) \qquad (7) \end{aligned}$$

This equation can be solved analytically, and yields 0, 1 or 2 solutions. The case where 0 solutions are returned occurs when no value of the joint angle $\theta$ achieves the desired elevation angle $\phi$. This situation has never occured in practice, and is unlikely to occur unless the joint axis $\hat{\mathbf{q}}$ is very close to lying in the sagittal plane. In the case of 1 solution, the returned value of $\theta$ is used as the joint angle. In the case of 2 solutions, only one solution will cause $\mathbf{v}$ to achieve $\phi$. The second solution causes $\mathbf{v}$ to achieve $-\phi$; we test which is correct and use the proper one.

## 4.3 Stance ankle and hip joint complexes

The previous section detailed how to compute joint angles if only a single DOF and a single elevation constraint existed. However, at the stance ankle complex, we have two DOFs, the upper and lower ankle joints, and only a single elevation constraint on the stance lower leg. At the hip complex, things are similar: we have six DOFs, three from each side, but only three elevation angle constraints. At these areas, we must do something different.

Our solution is to add constraints until we have as many constraint as DOFs. We can then solve these problems numerically. The benefit of adding constraints is that we can parameterize them with high-level, intuitive variables such as stance width and toe-out. These parameters will later allow us to modify the gait while keeping the motion dataset invariant.

**Stance ankle complex** At the stance ankle complex, we define stance width $W_{st}(t)$ to be the distance from the stance foot to the pelvis, in the direction of the sagittal plane normal. At the stance ankle complex, we simultaneously solve two constraints: the lower leg elevation angle constraint, and the following:

$$(\mathbf{a}_{st} - \mathbf{h}_{st}) \cdot \hat{\mathbf{n}} - W_{st}(t) = 0 \qquad (8)$$

where $\hat{\mathbf{n}}$ is a unit vector normal to the sagittal plane, $\mathbf{a}_{st}$ is a site on the stance ankle and $\mathbf{h}_{st}$ is a site at the stance hip. By manipulating $W_{st}(t)$, we can change the stance side width during gait.

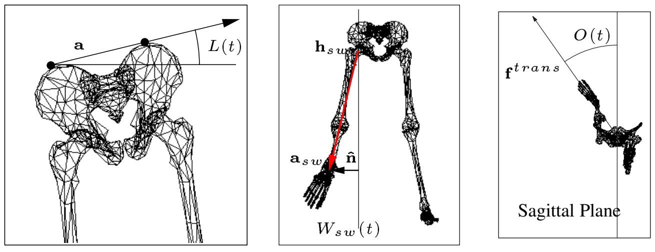
These constraints are solved simultaneously to yield the upper and lower ankle joint angles. The joint angles are computed using a numerical search method; we have found Gauss-Newton's method to be extremely effective, using the previous joint angles as the search starting point.

**Hip joint complex** The hip complex consists of the stance hip and swing hip joints. Each joint has 3 degrees of freedom, so the entire complex has a total of 6 DOFs. Three elevation angle constraints apply to the hip joints: stance pelvis, swing pelvis, and swing upper leg elevation angles. We require 3 more equations to determine a unique solution. As with the stance ankle, we add constraints which have high-level parameters.

The first parameterized constraint is for *pelvic list*. This is the amount of rotation in the frontal plane and manifests as one hip being higher than the other. Pelvic list is shown in Figure 9(a). List is defined to be the angle made between $\mathbf{a}$ and a horizontal plane. The pelvic list parameter is $L(t)$ and its associated constraint is the following:

$$\tan^{-1}(a_y, a_z) - L(t) = 0$$

By manipulating $L(t)$, we can change the amount of hip "waggle" during gait.



(a) (b) (c)

Figure 9: The hip model controls pelvic list, swing width and toe-out

The second parameterized constraint is *swing width*, similar to the stance width $W_{st}(t)$ defined above. This parameter is depicted in Figure 9(b). The parameter for swing width is $W_{sw}(t)$ and the constraint is the following:

$$(\mathbf{a}_{sw} - \mathbf{h}_{sw}) \cdot \hat{\mathbf{n}} - W_{sw}(t) = 0$$

where $\mathbf{a}_{sw}$ is a site on the swing ankle and $\mathbf{h}_{sw}$ is a site at the swing hip.

The third parameterized constraint is *toe-out*, shown in 9(c). Toe-out is the amount of inward/outward rotation of the leg; we define toe-out for the swing leg only, as the stance leg is fixed on the ground. Let $\mathbf{f}$ be a vector affixed to the swing leg; $\mathbf{f}$ should point forward when toe-out is $0°$. Let $\mathbf{f}^{trans}$ be the projection of $\mathbf{f}$ onto the transverse, or horizontal, plane. Toe-out is defined as the angle between $\mathbf{f}^{trans}$ and the forward direction of the sagittal plane, $\mathbf{x}^{sag}$. The parameter for toe-out is $O(t)$, and its associated constraint is the following:

$$\cos^{-1}(\mathbf{f}^{trans} \cdot \mathbf{x}^{sag}) - O(t) = 0$$

By manipulating $O(t)$, we can change the direction the swing foot points, in the transverse plane. Later we will show how toe-out is used in producing curved locomotion.

As in the stance ankle model, we use a numerical search method to find the hip joint angles which satisfy these six equations. We again use the solution from the previous time step as the starting point and find the method usually converges rapidly, within 4 Gauss-Newton iterations.

### 4.4 Curved locomotion

The heading direction $\psi$ describes the orientation of the sagittal plane (see Figure 5) and the direction of walking. By leaving it fixed, the figure walks in a straight line. However, if we vary $\psi$, curved locomotion results. This is extremely important for reducing the number of necessary datasets. Generating curved locomotion does not require collecting any more motion data, only modifying the $\psi$ parameter. In Section 6, we describe how MetaGait updates the heading direction $\psi$.

## 5 ElevInterp: motion data transformation

The previous section described how ElevWalker generates realistic-looking, real-time curved locomotion. However, given a fixed dataset, ElevWalker is not able to modify the the step length and the step height of its resulting gait. Step length and height are features which can be measured by looking at the sagittal plane, hence they are fixed for a given sagittal elevation angle dataset. The procedurally controlled parameters, such as heading direction, stance/swing width, toe-out and list, are not observable in the sagittal plane, hence they were independent of the dataset.

ElevInterp uses multiple datasets and interpolates between them to generate new datasets which satisfy step length and height requirements. These are then fed to ElevWalker; ElevInterp thus controls the step length and height. In its use of interpolation between motion datasets, ElevInterp is similar to the interpolation-based motion systems of [28, 22].

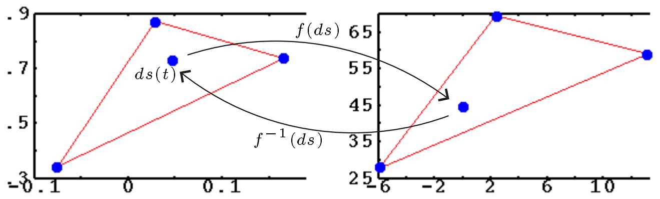
However, ElevInterp's interface does not take interpolation parameters as input; instead, it lets high-level controllers use step length and step height as input parameters. ElevInterp takes care of the background work, computing which datasets to interpolate, and how to blend them, to achieve the specified length and height, eliminating the need for the controller (or animator) to specify these low-level and often unintuitive parameters. For this task, we have developed a new, efficient solution to this *inverse motion interpolation problem* which requires less data than the technique of [28].

### 5.1 Inverse motion interpolation

We first think about datasets as motions. In order to characterize a motion, we will measure some features when it is applied to a specific figure, such as step length and height. Let $f(ds)$, which we define below, be a function which measures the step height and step length achieved by a motion (a dataset). This produces a 2-dimensional space, a Height/Length space, in which each dataset can be located.

We also think of our motion datasets as points in an abstract "dataset space". To be consistent with the two features we are measuring, the dataset space is also two-dimensional. Coordinates in this space will later serve as interpolation parameters, so we have set them to be approximately in the range [0, 1] for step length and [-1:1] for step height[2].

Figure 10 illustrates these two spaces. The space on the left is the "dataset space". The space on the right is the Height/Length space, showing the actual step height and length achieved by a particular figure using that dataset; these coordinates represent real distances in centimeters. $f(ds)$ takes a dataset from dataset (interpolation) space to physical space.



Dataset space (unitless dimensions) Height/Length space (in cm)

Figure 10: Mapping from the dataset space to the physical space

When an animator, or MetaGait, requests that ElevInterp generate a step with length $SL$ and height $SH$, ElevInterp must determine which datasets to interpolate, and what the interpolation weights should be. This is the *inverse* motion interpolation problem, because we must invert both the measurement function $f(ds)$ as well as the interpolation function.

ElevInterp performs interpolation using a simple yet general framework: triangulations and barycentric coordinates. Triangulations allow us to work with fewer data points than $n$-dimensional linear interpolation, as the input samples do not have to bound the desired point in all $n$ dimensions. While the technique of [22] is superior in terms of the number of datasets for forward interpolation, a cell decomposition using triangulations is useful for inverse interpolation because it allows us to rapidly compute which datasets are

[2]However, the actual values of these coordinates are arbitrary and different ranges could be used.

used in the inversion process. In our system, we use the vertices of the containing triangle.

Figure 10 shows three datasets forming the vertices of a triangle, both in dataset space and in Height/Length space. The figure also depicts a dataset which is within the interior of these triangles. Let the barycentric coordinates of this dataset with respect to the vertices in Height/Length space be $(a, b, c)$. A barycentric interpolation can be defined as the following:

$$ds(t) = a * ds_1(t) + b * ds_2(t) + c * ds_3(t) \tag{9}$$

where $ds(t)$ is the new dataset and $ds_1(t), ds_2(t)$ and $ds_3(t)$ are datasets associated with the vertices.

Given barycentric coordinates, it is simple to compute a new dataset by via barycentric interpolation. Likewise, if we are given a 2D point and a set of vertices which contain that point, it is simple to compute its barycentric coordinates. Inversion of the interpolation function is therefore straightforward.

Inverting $f(ds)$, the function which maps datasets to real physical step lengths/heights, takes a little more effort. First we will give our definition of $f(ds)$. In order to estimate the step length and height at the end of the gait cycle, we use a planar model of the figure. Let the stance side elevation angles be $ds(0.5) = \{\phi_1^{st}, \phi_2^{st}, \phi_3^{st}, \phi_4^{st}\}$, and the swing side elevation angles be $ds(1.0) = \{\phi_1^{sw}, \phi_2^{sw}, \phi_3^{sw}, \phi_4^{sw}\}$.

We also require the lengths of the limbs. Let $L$ be the lengths of the limbs from foot to pelvis: $L = (l_1, l_2, l_3, l_4)$, respectively. Then the estimated position of the swing ankle $\tilde{\mathbf{p}}_{sw}$ is the following:

$$\begin{aligned} f(ds; L) &= \tilde{\mathbf{p}}_{sw} \\ &= \sum_{i=1}^{4} l_i R(\phi_i^{st})(0,1,0) + \\ & \quad \sum_{i=2}^{4} l_i R(\phi_i^{sw})(0,-1,0) \end{aligned} \tag{10}$$

where $R(\phi)$ is a 2-by-2 rotation matrix.

This calculation produces $\tilde{\mathbf{p}}_{sw} = (l, h)$. The components of the computation are illustrated in Figure 11. The first summation represents the stance leg segments (up arrow), and the second summation represents the swing leg segments (down arrow). Inverting $f(ds)$ means finding the dataset which produces the correct step length and step height. Since the desired point $(SL, SH)$ exists in the Height/Length space, we can determine its barycentric coordinates with respect to its containing triangle. We then use these barycentric coordinates in the dataset space to compute the new dataset $ds(t)$.

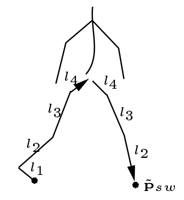


Figure 11: Computing the height/length of the swing foot

Unfortunately, $f(ds)$ is non-linear, so the computed dataset $ds(t)$ may not satisfy the requirement $f(ds(t)) = (SL, SH)$. If the triangle is small, then linearity will approximately hold, and $ds(t) \approx (l, h)$.

On the other hand, if the triangle is large, then the non-linearity of $f(ds)$ will likely cause $f(ds)$ not to equal $(SL, SH)$, although it will be close. In this case, we use $ds(t)$ as a good starting estimate from which to begin a numerical search for the correct interpolation parameters. The complete algorithm for finding the interpolation parameters for a desired height/length combination $(SL, SH)$ is as follows:

1. (Height/Length space) Locate the triangle $T$ which contains $(SL, SH)$ in Height/Length space;
2. (Height/Length space) Compute the barycentric coordinates of $(SL, SH)$ with respect to $T$;
3. (Dataset space) Compute a new dataset $ds(t)$ according to Equation 9 using the barycentric coordinates from Step 2;
4. If $f(ds) = (SL, SH)$, then the dataset satisfies the height/ length requirement. Go to 6.
5. Otherwise, use $ds(t)$ and $f(ds)$ as the starting point in the numerical search;
6. (Height/Length and Dataset space) Insert the dataset into both triangulations using $(SL, SH)$ for the physical coordinates, and ($SL$/*leg_length*, $SH$/*leg_length*) for the dataset coordinates.

We find that by exploiting temporal coherence and using the last located triangle, the effort to find the containing triangle in Step 1 remains small, even when the number of datasets grows large. In Step 5, we use a Gauss-Newton search to find the dataset which maps to $(SL, SH)$. In Step 6, after we have found the desired dataset, we insert it into both triangulations, splitting the triangles which contain these points. Since we wish the triangles to have as short sides as possible, we use a Delaunay triangulation [6]. This splitting decreases the triangles' sizes and improves the linear approximation, increasing the likelihood that in the future we will be able to skip the numerical search of Step 5.

## 6 MetaGait

The previous sections described the low-level modules we use to generate gait. In this section, we describe MetaGait, a high-level controller which manipulates the parameters of these modules so that an animator can direct a figure simply by specifying its path. At each frame, MetaGait performs the following computation:

1. Compute the parameter values for curved locomotion.
2. Compute the parameter values for uneven terrain, based on the values set for curved locomotion.
3. Notify the motion generator to use the new parameter values.

### 6.1 Curved locomotion

Following a curved path specified by the animator involves changing the heading direction parameter $\psi$, as well as possibly modifying the toe-out and step length parameters $O(t)$ and $SL(t)$ at each frame. Furthermore, if the path contains a very sharp turn, we may require a spin turn to augment the normal step turning (see Section 4.4) that ElevWalker uses.

Let $\mathbf{p}(t)$ be a path specified by the user. The first task is to compute the heading direction $\psi$. One possibility is simply to use the gradient $\nabla\mathbf{p}(t)$. However, this value can change very rapidly, particularly at sharp corners. We instead use a vector difference which varies less wildly.

At each frame, we estimate two points: $\mathbf{p}_0$, the point on the path closest to where the figure is, and $\mathbf{p}_1$, the point on the path where the figure will be one step from now. Given these two points, we form the vector $\mathbf{d} = \mathbf{p}_1 - \mathbf{p}_0$; $\mathbf{d}$ is the direction in which we wish to

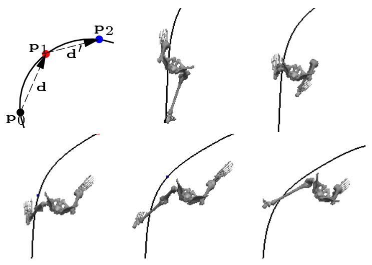


Figure 12: The estimated points used in path following, and some example frames

head. Figure 12 shows an example of these quantities. We then calculate the orientation of the sagittal plane as $\psi = \text{atan2}(-\mathbf{d}_z, \mathbf{d}_x)$.

We may also modify the toe-out parameter $O(t)$ during curved walking. Since toe-out affects only the swing leg, manipulating it does not change the gait during the current step. Instead, toe-out is modified in preparation for the *next* step.

A figure's orientation $\psi$ is dependent on the orientation of the current stance foot, because the amount of rotation possible at the hips is limited. Since the orientation of the stance foot depends on its orientation during the previous swing phase, we can manipulate its toe-out angle during swing phase to make a subsequent step easier.

The algorithm for this is as follows: at each frame, we estimate the orientation of our body, $\mathbf{d}$ above. We also estimate the orientation of our body during the step after this one, $\mathbf{d}'$ shown in Figure 12. If the angular difference $\Delta$ between these two orientations is greater than a threshold $T$ ($45°$, by default), it means we should adjust the toe-out parameter $O(t)$ during this step so that the next step will be easier to perform. We add the difference $\Delta - T$ to $O(t)$, pointing the swing foot in the direction of curvature.

In the presence of very sharp turns, large adjustments of toe-out may look unnatural. In these situations, we rotate the character on the stance foot as well. Let $T_2$ be a threshold on how much we can adjust toe-out ($25°$ by default). If $\Delta - T > T_2$, then we set the toe-out to $O(t) + T_2$. When the next step begins, we use a spin turn to assist the step turn, rotating $(\Delta - T - T_2)°$ by spinning on the stance foot.

Finally, step length $SL$ may also change as a result of curved locomotion: one does not take the same length steps when navigating sharp turns as when walking a straight path. We define $\Delta\psi_{thresh}$ to be the maximum angle that $\psi$ can change in a single gait cycle.

We compute a maximum step length so that the change in $\psi$ is less than $\Delta\psi_{thresh}$. To calculate the maximum step length $SL_{max}$, we iteratively estimate the change in $\psi$ using the current step length, and reduce the step length (currently by a fixed amount) until the change is less than $\Delta\psi_{thresh}$. This value of $SL_{max}$ is used later, to compute the actual step length.

## 6.2 Uneven terrain locomotion

After the parameters for curved locomotion have been computed, we compute the step height and step length for uneven terrain locomotion. MetaGait computes the step length and step height in physical coordinates, and lets ElevInterp handle the problem of calculating interpolation coefficients to satisfy these parameters.

Our uneven terrain model is an iterative process for computing $SH$ and $SL$, the height and length, of the current step. The process consists of three steps:

1. Predict where the swing foot will land.
2. Find the height of the ground at that point.
3. Make sure the step length at that height is acceptable, otherwise compute the new length and return to Step 1.

For Step 1, we use the current value of $SH$ and $SL$ as initial estimates. However, $SH$ and $SL$ measure the step length and height in sagittal plane coordinates, and what we need is the global location where the swing foot will land (for Step 2). The following equation transforms these into global coordinates:

$$\mathbf{p}_{sw} = R(\psi)\left[[SL, SH] + \text{offset}_{\text{pelvis}}\right] + \mathbf{p}_{st}$$

The term offset$_{\text{pelvis}}$ is the lateral distance between the hips, and helps to account for the lateral position of the swing foot. This is added to the vector $[SL, SH]$, and we multiply the resulting vector by $R(\psi)$ to transform into global coordinates. Finally, we add $\mathbf{p}_{st}$, the current stance foot position, to get the global location.

In Step 2, we find the height of the ground by intersecting a vertical line passing through $\mathbf{p}_{sw}$ with the ground geometry. This is subtracted from the current stance foot y-position to yield the height difference $h$.

In Step 3, we must test the step height and the step length for compatibility. In [26], Sun et al. present a linear relationship between ground slope angle and step length. Figure 13 shows how step length varies as a function of step height for an average sized person, using the equations presented in [26]. This curve is the "preferred step length for height" relationship, because it reveals how people naturally change their step length to accomodate up and down slope walking (step length decreases when not walking on flat terrain). Step height $SH$ and step length $SL$ are determined with respect to ground slope angle, $\rho$, measured in degrees. The downhill equations (in meters) are the following:

$$\begin{aligned} SL(\rho) &= 0.627 + 0.0075\rho \text{ (downhill)} && (11) \\ &= 0.623 - 0.0019\rho \text{ (uphill)} \\ SH(\rho) &= SL(\rho) * \tan(\rho) && (12) \end{aligned}$$

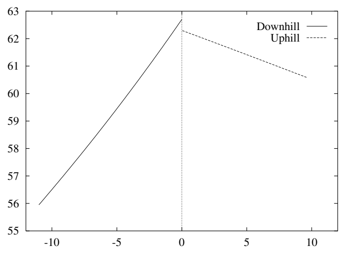


Figure 13: Preferred stride length (y) for stride height (x), in cm [26]

Due to the existence of preferred step length for height, we cannot simply pick a step length and compute the corresponding height. The step length might need to be lengthened or shortened to accomodate the preferred step length for that height. We should also ensure that our step length is not greater than the maximum step length $SL_{max}$ calculated for curved locomotion.

MetaGait uses Equations 11 and 12 to compute the preferred step length $SL_{pref}$ for the height $h$. We set $SL$ to $\min(SL_{pref}, SL_{max})$, and return to Step 1. At the end of Step 3, the value of $SH$ and $SL$ for the next step have been determined.

## 6.3 Setting new parameter values

After the parameters have been calculated for curved locomotion and uneven terrain walking, we notify the motion generator to use the new parameter values. ElevInterp handles step length and step height, while ElevWalker handles heading direction and toe-out.

MetaGait is activated at every frame, so these parameters are constantly changing. The underlying motion generator allows parameters to be modified at every frame, allowing it to be more responsive than systems which can modify parameters only at the beginning of a new step.

## 7 Results

Our system was designed for automated, real-time motion generation for virtual environment applications and simulations. We believe we have achieved this goal: across a variety of environments, our system can maintain a frame rate of 50+ frames per second (no graphics output). These tests were made on an 800 MHz Pentium 3 with 128 Mb memory.

We have generated several examples of walking on uneven terrain, using five datasets: walking normally, walking uphill, walking downhill, walking with short strides (level terrain), walking with long strides (level terrain). These show our system's ability to handle curved locomotion on a continuously changing surface using a very small amount of data.

We have used our system to generate animation on several different figures, including our primary skeleton figure, a smaller, female skeleton figure, and a larger, human figure. Due to the sagittal elevation angle datasets' invariance with respect to size, the same datasets were used for all three figures with no modification or preprocessing.

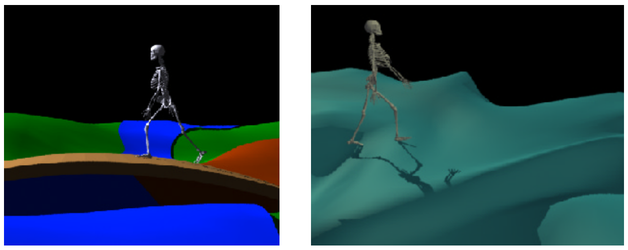


Figure 14: Examples of walking on uneven, sloped terrain

We also compare our generated motions against those of real humans. Below, we compare actual hip, knee and ankle forward trajectories with our generated trajectories. Since our model does not control these positions, e.g. through the use of a spatial spline curve, this is a test that we can recover position using the elevation angles. Figure 15 shows a graph of real hip, knee and ankle positions plotted against normalized gait cycle time [12], as well as the hip, knee and ankle positions generated by our model. The trajectories are very similar.

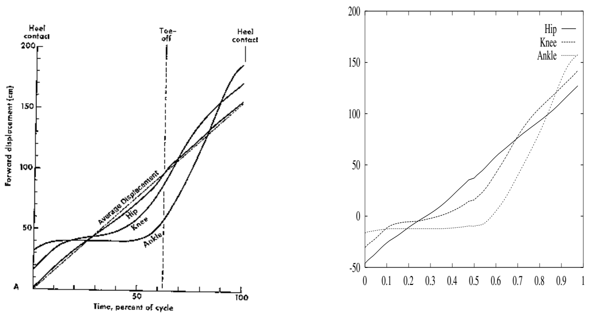


Figure 15: Positions of hip, knee, and ankle from data [12] vs generated data

We also compare the relative forward position of the pelvis. This is the position of the pelvis relative to a point travelling at the average velocity. If the pelvis always moved at the average velocity, the curve would be identically 0. When the curve is below 0, the pelvis is behind a point travelling at the average velocity; when above 0, the pelvis is forward of such a point. Figure 16 shows a graph of pelvis position from [12]. Figure 17 shows our computed pelvis position. Although our magnitude is larger, the overall shapes of the trajectories are very similar.

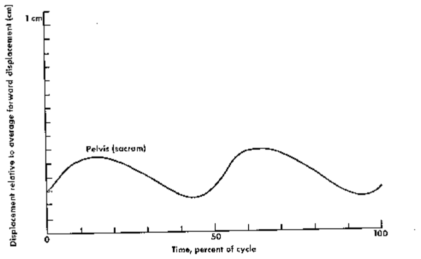


Figure 16: Pelvis position relative to avg. velocity from data [12]

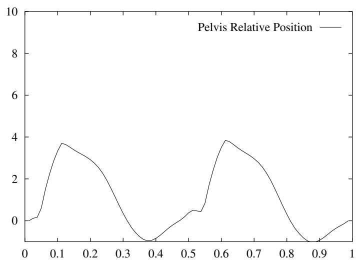


Figure 17: Generated pelvis position relative to avg. velocity

## 8 Discussion and Future work

One of the most interesting results to come out of our research is how well an alternate representation like sagittal elevation angles works for gait animation. We hypothesize that there are other interesting motion representations (other than joint angles) which are yet to be discovered for other animation tasks.

The constraints used at the hip joint complex, controlling pelvic list, toe-out and swing width, were chosen because these quantities are most directly related to the unused joint DOFs, that is, the joint DOFs not significantly involved in satisfying the elevation angle constraints. The stance width constraint used at the stance ankle joint complex was chosen for its analogy to the swing width constraint.

Five datasets were used in generating all of our examples, on all of the figures. We were able to use so few datasets because of the sagittal elevation angle representation's invariance with respect to figure size, and the ability to generate curved locomotion by changing the sagittal plane direction. This meant we did not need data for each curvature and inclination combination.

MetaGait modifies high-level parameters, such as step length, step height, toe out and heading direction, based on the terrain and the path it is following. However, there are more interrelationships which we do not model. For example, we do not modify the stride width parameters during curved locomotion and we do not have an explicit relationship between path curvature and step length, unlike our height/length curves. Research into these interrelationships will be crucial to the proper and realistic modelling of gait.

Our model generates movement for the lower body only. Currently we generate upper body motion by having the arms mirror the opposite leg for arm swing, and bending the spine to compensate for the pelvis' motion, so that the head stays relatively still. However, research in natural looking upper body motion is sorely needed in order to improve the realism of walking animation.

We are interested in extending our research to running motions in addition to walking. This would require the addition of some dynamics to our model, which we believe could also improve the veracity of our foot-floor interaction model, including the addition of a dynamic double-stance phase. Our model also does not compensate for objects in the path of the swing foot; additional research into how to modify the walking motion to avoid low obstacles is needed.

## 9 Conclusion

Our goal was to build a tool which relieves the burden of animating walking by handling the low-level details of motion generation as well as the higher-level details of correctly setting the parameters. The system comprises three components. ElevWalker, based on the sagittal elevation angle representation, computes low-level gait and can generate curved locomotion. ElevInterp, introducing a new solution to the inverse motion interpolation problem, performs dataset interpolation to control the step length and step height. MetaGait uses embedded biomechanical data and rules to control the parameters of the former components. Our system can be used to perform both traditional off-line animation of human walking as well as on-line animation of an autonomous or user-controlled agent in an interactive application.

## References

[1] F. Anderson and M. Pandy. A Dynamic Optimization Solution for One Complete Cycle of Human Gait. In *Proc. International Society of Biomechanics XVII Congress*, page 381, Calgary, Canada, 1999.

[2] A. Borghese, L. Bianchi, and F. Lacquaniti. Kinematic Determinants of Human Locomotion. *J. Physiology*, (494):863–879, 1996.

[3] R. Boulic, N. Thalmann, and D. Thalmann. A Global Human Walking Model with Real-time Kinematic Personification. *The Visual Computer*, (6):344–358, 1990.

[4] A. Bruderlin and T. Calvert. Goal-directed, Dynamic Animation of Human Walking. In *Computer Graphics (SIGGRAPH 89 Conference Proceedings)*, pages 233–242, 1989.

[5] A. Bruderlin and T. Calvert. Interactive Animation of Personalized Human Locomotion. In *Proc. of Graphics Interface 93*, pages 17–23, 1993.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry Algorithms and Applications*. Springer-Verlag Berlin, 1997.

[7] M. Girard and A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. In *Computer Graphics (SIGGRAPH 85 Conference Proceedings)*, pages 263–270, 1985.

[8] M. Gleicher. Motion Editing with Spacetime Constraints. In *SIGGRAPH 97 Conference Proceedings*, pages 139–148, 1997.

[9] R. Grasso, L. Bianchi, and F. Lacquaniti. Motor Patterns for Human Gait: Backward versus Forward Locomotion. *J. Neurophysiology*, 80:1868–1885, 1998.

[10] J. Hodgins, W. Wooten, D. Brogan, and J. O'Brien. Animating Human Athletics. In *SIGGRAPH 95 Conference Proceedings*, pages 71–78, 1995.

[11] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. In *Proc. of VRML '97*, 1997.

[12] V. Inman, H. Ralston, and F. Todd. *Human Walking*. Williams and Wilkins, Baltimore/London, 1981.

[13] Y-M. Kang, H-G. Cho, and E-T. Lee. 'an Efficient Control over Human Running Animation with Extension of Planar Hopper Model. *Journal of Visualization and Computer Animation*, 10:215–224, 1999.

[14] H. Ko. *Kinematic and Dynamic Techniques for Analyzing, Predicting and Animating Human Locomotion*. PhD thesis, Dept. CIS, University of Pennsylvania, 1984. MS-CIS-94-31.

[15] H. Ko and N. Badler. Animating Human Locomotion in Real-time using Inverse Dynamics, Balance and Comfort Control. *IEEE Computer Graphics and Applications*, 16(2):50–59, March 1996.

[16] H. Ko and J. Cremer. VRLOCO: Real-time Human Locomotion from Positional Input Streams. *Presence*, 5(4):367–380, 1996.

[17] J. Laszlo, M. van de Panne, and E. Fiume. Limit Cycle Control and Its Application to the Animation of Balancing and Walking. In *SIGGRAPH 96 Conference Proceedings*, pages 155–162, 1996.

[18] J. Lee and S.-Y. Shin. A Hierarchical Approach to Interactive Motion Editing for Human-like Figures. In *SIGGRAPH 99 Conference Proceedings*, pages 39–47, 1999.

[19] M. McKenna and D. Zeltzer. Dynamic Simulation of Autonomous Legged Locomotion. In *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, pages 29–38, 1990.

[20] F. Multon, L. France, M-P. Cani-Gascuel, and G. Debunne. Computer Animation of Human Walking: a Survey. *Journal of Visualization and Computer Animation*, 10:39–54, 1999.

[21] M. Raibert and J. Hodgins. Animation of Dynamic Legged Locomotion. In *Computer Graphics (SIGGRAPH 91 Conference Proceedings)*, pages 349–358, 1991.

[22] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[23] H. Sun. *Curved Path Human Locomotion on Uneven Terrain*. PhD thesis, Dept. of Computer and Information Sciences, University of Pennsylvania, December 2000.

[24] H. Sun, A. Goswami, D. Metaxas, and J. Bruckner. Cyclogram planarity is preserved in upward slope walking. In *Proc. International Society of Biomechanics XVII Congress*, page 514, Calgary, Canada, 1999.

[25] H. Sun and D. Metaxas. Animation of Human Locomotion Using Sagittal Elevation Angles. *Proceedings of Pacific Graphics 2000*, 2000.

[26] J. Sun, M. Walters, N. Svensson, and D. Lloyd. The Influence of Surface Slope on Human Gait Characteristics: a Study of Urban Pedestrians Walking on an Inclined Surface. *Ergonomics*, 39(4):677–692, 1996.

[27] K. Tsutsuguchi, S. Shimada, Y. Suenaga, N. Sonehara, and S. Ohtsuka. Human Walking Animation Based on Foot Reaction Force in the Three-dimensional Virtual World. *J. Visualization and Computer Animation*, 11(1):3–16, 2000.

[28] D. Wiley and J. Hahn. Interpolation Synthesis of Articulated Figure Motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, 1997.

[29] D. Zeltzer. Motor Control Techniques for Figure Animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, 1982.