Towards Generalised Motion Dynamics for Animation

Charles Herr and Brian Wyvill

Department of Computer Science, University of Calgary.
2500 University Drive N.W.
Calgary, Alberta, Canada, T2N 1N4

Abstract

Dynamic simulation has the advantage that the motion is produced from a model of the real world. However, including such physical descriptions in an animation system in a general way is difficult due to the problem of describing the simulation in high level terms. The lowest level primitives are forces, torques and masses and a set of differential equations that relate these primitives to the motion. We describe here an approach to integrating a high level set of primitives into a general purpose animation system via an intermediate language called CHARLI. Interactive user interfaces can be built on top of CHARLI to allow a user to specify motion dynamics for a fairly complex set of interacting bodies.

Key words: Computer Graphics, animation, motion control, dynamics.

Introduction

The use of dynamics for motion control in animation has attained some popularity in research systems. However, before such techniques can become standard a number of outstanding problems must be overcome. Barzel and Barr [Barzel & Barr 1988] recently stated these problems as follows:

- **Simulations are hard to implement** Many dynamical simulation problems are solved by special purpose software. A program which defines an articulated figure may not be easily generalisable to solve for a deformable object under various external forces.
- **Simulations are hard to control** With a purely dynamic approach it is often difficult for an animator to produce a desired motion.
- **Simulations are slow** Computationally intensive simulations may compete with rendering in the overal cost of an animation production.

In this work we have two main goals:

1. **Generality** As with earlier work in this area we wish to achieve a generalised way of simulating the dynamics of a broad class of mechanical systems. This includes the geometry, forces and constraints on the system.
2. **Integration with an animation system** In addition we are concerned with the integration of such simulation techniques with our existing object oriented animation system in a natural and usable way. We wish to provide simultaneous access to dynamic and traditional kinematic animation techiques.

Previous Work

A considerable volume of literature exists on techniques for performing dynamic simulations to solve various problems. Many of these techniques are discussed in the mechanical engineering literature, particularly for the design of mechanisms. However, they are also applicable for use in computer animation systems.

Haug Of interest to computer animation system designers is a survey paper by Haug in [Haug 1984], who discusses the derivation of differential equations using a minimum set of independent generalised coordinates vs. a much larger set of dependent coordinates and constraint equations relating them. The use of such a minimum set results in a small number of complex equations. The use of a large set of coordinates produces a large number of very simple equations. The latter approach amounts to constructing parts of the model separately and assembling the parts using constraints.

Nikravesh Also from the mechanical engineering literature is a survey paper by Nikravesh in [Nikravesh 1984]. In particular he discusses the control of numerical error in the integration of the differential equations. Violations in the constraint equations are used as feedback terms to correct the violations in the next time step.

Wilhelms A good example of dynamics for human body animation was by Wilhelms in her PhD work. She used the Gibbs-Appel formulation for the dynamical analysis [Wilhelms 1985].

Armstrong A recursive method for fast solution of the dynamics equations for tree structured (i.e. open loop) models is in [Armstrong & Green 1985].

Isaacs Two papers by Isaacs and Cohen, [Isaacs & Cohen 1987] and [Isaacs & Cohen 1988] discuss the problems of generalising dynamic simulation by allowing users to construct models by selecting joints from a joint library and specifying their connectivity. The complex kinematic constraints they provide facilitate kinematic driving and the construction of models with closed loops. The dynamics are formulated using Lagrange's equations.

Barzel At the same time related work was developed by Barzel and Barr [Barzel & Barr 1988] who are also concerned with generality and usability. A wide variety of models can be constructed by choosing and combining approriate elements from body, applied force, and constraint libraries.

Haumann
Haumann and Parent in [Haumann & Parent 1988] focus on the simulation of flexible objects such as bouncing blobs and waving flags using masses connected by springs and calculations of aerodynamic drag.

Wilhelms In later work by Wilhelms, Moore and Skinner [Wilhelms *et al* 1988], further low-level control issues are discussed such as collision detection and response, friction, and joint limits. Their system uses the fast recursive dynamics formulation of Armstrong and Green.

Miller A more specialised but interesting use of dynamics was in work by Miller [Miller 1988] to drive the animation of a snake model.

Terzopoulous Terzopoulous, Platt, and Fleischer in [Terzopoulo *et al* 1989] have also worked on deformable solids called goop and fluids called glop. These are animated using a variety of techniques. The deformable solids are modeled with partial differential equations derived using elasticity theory. The fluids are modeled as discrete particles with attractive and repulsive forces between them.

van Overveld In this work, [Overveld 1989] a very fast approximate but interactive solution of differential equations is offered for an arbitrary mesh of masses connected by springs.

Pentland A very fast solution of dynamics equations is given in [Pentland & Williams 1989]. Dynamic models are constructed using finite element analysis. Diagonalization of the mass matrix for their systems of differential equations and elimination of high frequency vibrations permit the extremely fast solution.

The CHARLI Modelling and Animation Language

In many areas where the computer has been used to aid an exisiting task such as draughting, systems have been designed to keep the spirit of earlier methods so that the designer does not have to cope with an entirely new way of thinking. Computer animation of three dimensional models does not lend itself to the same approach. The animator is working within the 3D world and not with sequences of 2D drawings. Although interactive graphical tools can often be used to specify animation sequences, sometimes essentially algorithmic animation is required and a language description is more suitable.

In our system we have taken the view that all animation and model descriptions should be language based. Interactive interfaces may also be used to build or modify the data structures but each session is saved in the form of a language script which may be further edited later using a text editor. While the current trend in the design of computer graphics systems is to break down the process of specifying and generating images into a number of separate parts, we are exploring a system which closely couples these separate parts, allowing intense interaction between them (for details see [Chmilar & Wyvill 1989]). The system integrates the specification of model geometry, model and scene structure, and animation. The system is based on a kernel which controls a modelling and animation data-structure. Although interactive programs may communicate directly with the kernel, the universal form of data communication in the system is by means of a powerful modelling and animation language called CHARLI. Such a scripting language can be used to describe models or motions which are algorithmic in nature and awkward to describe interactively. We have extended this language to allow the description of dynamical models. The language also allows a kinematic description of motion and thus an animator can achieve a spectrum of control from pure kinematic to pure dynamic control with an inbetween stage where certain quantities are kinematically controlled and others are driven dynamically. The language has variables, scoping, operations, casting and animation control; a full description is to be found in [Chmilar & Wyvill 1989].

Adding Dynamics to CHARLI

The following are the key elements of the CHARLI language:

```
# Comments begin with '#' and continue to end of line
        var r, theta_initial, theta_dot_initial;
        dynamic theta;

#assign animation track to r
        r = {0 at 0 sec slowinout 1.8 at 2 sec };

# set initial position and velocity of theta
        theta_initial = pi;
        theta_dot_initial = -1;
        theta = initial(theta_initial, theta_dot_initial);
```

Figure 1: Valid declarations and assignments

Variables There are two data types; variables which can take constant, kinematic animation track, or path values; and dynamic variables which take dynamic initialisations.

Primitives Geometric primitives; line, polygon (others may be added). Dynamic Primitives; masses and labels. Masses affect the simulation whereas labels are merely convenient attachment points.

Transformations Geometric transformations can be used to change the location and orientation of the primitives.

Constraints Geometric transformations provide the basic model construction mechanism. Constraints can be added to facilitate the building of closed loop models and also to provide alternative and often more convenient ways of constructing models. Dynamic primitives can be constrained to a path or to maintain a constant or time varying distance from another dynamic primitive.

We illustrate the use of the dynamical elements in CHARLI with a series of script excerpts.

Example - Declarations and assignments

Figure 1 shows some valid CHARLI declarations and assignments.

Example - Transformations

Figure 2 shows a triple pendulum. The CHARLI script in figure 3 illustrates how it might be constructed and animated using geometric transformations.

Example - Constraints

The use of constraints often results in shorter and clearer model specifications. The script in figure 4 produces the same animation as the script in figure 3.

Kinematic Control

For dynamics to be useful to the animator, it is important to be able to vary the degree of control that the simulation has over the models. In CHARLI this is done by allowing parts of the model to be animated

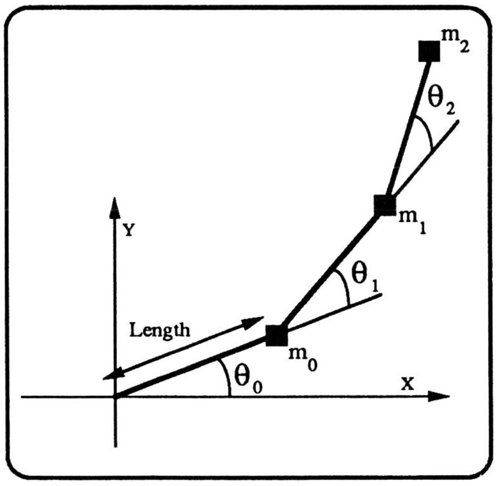


Figure 2: Triple Pendulum

kinematically, whilst other parts are animated dynamically. The dynamic parts react to the motions of the kinematic parts.

Animator control through geometric transformations

Arguments to transformations can be **tracks**, or **import** or **dynamic** variables. **Tracks**, see [Gomez 1985], are explicit functions of time built into the CHARLI system. **Import** variables are external functions written in a programming language. They allow the specification of custom time and force functions not available within the system. If the argument to a transformation is an **import** variable or **track**, the part of the model specified by the transformation changes as an explicit function of time. We say the degree of freedom is driven. If the argument is a **dynamic** variable, the part of the model changes according to the influences of applied forces and the motions of other objects in the system.

Animator control through constraints

A disadvantage of using geometric transformations is that only tree structured models can be constructed. The distance and path constraints allow the construction of models with closed loops. A **dist_constraint** forces a **mass** or **label** to be a certain distance away from another **mass** or **label**. A path constraint forces the motion of a **mass** or **label** to lie on a point, line or spline path, much like a bead sliding along a wire. The arguments to the point, line and spline generation functions are **import** variables or **tracks**.

An example: The *balloon* pendulum

We want to produce an animation of the mechanical system in figure 5. The top end of the pendulum arm moves on a smoothly curved wire. The pendulum arm

```
var length;
dynamic theta0, theta1, theta2;

#The pendulum bobs are to be drawn as boxes
def square
        # square contains a polygon primitive. The
        # parameters are vertex triplets in x,y,z order.
        polygon( 0,0,0, 1,0,0, 1,1,0, 0,1,0 ) scale(0.1);
end;

#define the pendulum bobs as mass objects
def m0 mass(1.0); square; end;
def m1 mass(2.0); square; end;
def m2 mass(5.0); square; end;

length = 1;
theta0 = initial(0, 0);
theta1 = initial(0, 0);
theta2 = initial(0, 0);

def p2
        m2 translatex(length) rotatez(theta2);
        m1;
        #_x, _y and _z give position of a defined object
        line(m1_x, m1_y, 0, m2_x, m2_y, 0);
end;
def p1
        p2 translatex(length) rotatez(theta1);
        m0;
        line(m0_x, m0_y, 0, m1_x, m1_y, 0);
end;
def p0
        p1 translatex(length) rotatez(theta0);
        line(0, 0, 0, m1_x, m1_y, 0);
end;

#Instantiate the pendulum at the origin
p0;
```

Figure 3: Triple pendulum using transformations

```
var length;
dynamic xroot, yroot, x0, y0, x1, y1, x2, y2;

#square, m0, m1 and m2 definitions are
# as in the previous script

def mroot mass(1.0); end;

length = 1;
x0 = initial(length, 0);
y0 = initial(0, 0);
x1 = initial(2 * length, 0);
y1 = initial(0, 0);
x2 = initial(3 * length, 0);
y2 = initial(0, 0);

#instantiating the mass at the origin
# with no transformation list
#fixes it there
mroot;
m0 translatex(x0) translatey(y0);
m1 translatex(x1) translatey(y1);
m2 translatex(x2) translatey(y2);

dist_constraint(mroot, m0, length);
dist_constraint(m0, m1, length);
dist_constraint(m1, m2, length);
line(mroot_x, mroot_y, 0, m0_x, m0_y, 0);
line(m0_x, m0_y, 0, m1_x, m1_y, 0);
line(m1_x, m1_y, 0, m2_x, m2_y, 0);
```

Figure 4: Triple pend defined using constraints

is a piston whose length we can change at will. The bottom end is attached to an odd springy bob.

m0 is constrained to always lie on the smoothly curved wire. To construct the pendulum arm, we must specify how it changes length over time. The bob is built by forcing the distances dist(m1, m2), dist(m2, m3), dist(m3, m4), dist(m4, m5) and dist(m5, m1) to remain constant, forming a pentagon. Spring forces are added between all pentagon masses that are not immediate neighbours. Attaching m1 to label1 completes the mechanism. Figure 6 shows the balloon pendulum CHARLI script. Figure 7 shows the motion of a many sided pendulum bob.

Implementation of CHARLI dynamics

Lagrange's Equations

CHARLI derives a system of differential equations (DE's) from the geometric information in a CHARLI script. It generates data that is passed to a standard numerical DE solver. Lagrange's equations are at the heart of CHARLI dynamics. They allow automatic generation of the DE's from the kinetic and potential energy

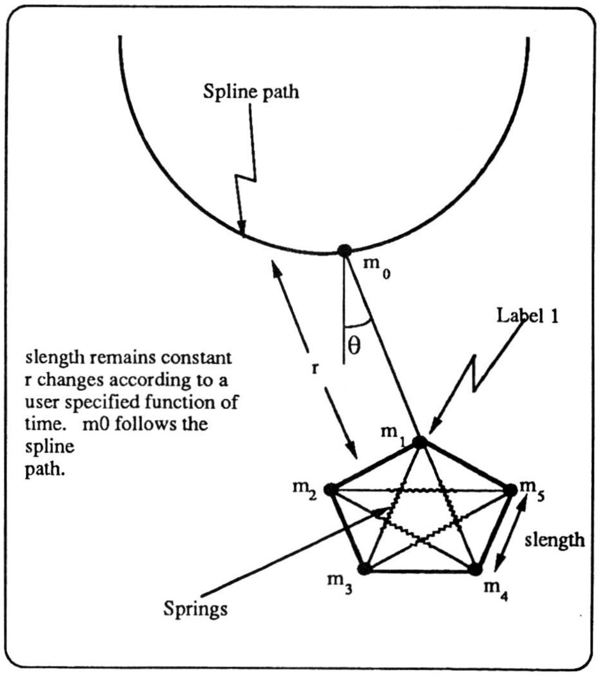


Figure 5: A balloon pendulum

```
var r, side_length, mass_const, p1;
var equil_length, spring_constant, damping_constant;
dynamic theta, x0, y0, x1, y1, x2, y2;
dynamic x3, y3, x4, y4, x5, y5;

        length = 1;
        mass_const = 1.0;
        side_length = 5.0;
        equil_length = 8.090170;
        spring_constant = 1.0;
        damping_constant = 3.0;

        theta = initial(-pi / 2, 0);
        x0 = initial(-10.000000, 0);
        y0 = initial(10.000000, 0);
        x1 = initial(-10.000000, 0);
        y1 = initial(0.000000, 0);
        x2 = initial(-14.045085, 0);
        y2 = initial(-2.938926, 0);
        x3 = initial(-12.500000, 0);
        y3 = initial(-7.694209, 0);
        x4 = initial(-7.500000, 0);
        y4 = initial(-7.694209, 0);
        x5 = initial(-5.954915, 0);
        y5 = initial(-2.938926, 0);

# p1 is a spline path
        p1 = spline(-10, 10, 0, -5, 0, 0,
                5, 0, 0, 10, 10, 0);
        r = {10 at 0 sec linear 0 at 10 sec};

        def label1 label(); end;

# m0 to m5 are defined as masses
        def m0 mass(mass_const); end;
        def m1 mass(mass_const); end;
        def m2 mass(mass_const); end;
        def m3 mass(mass_const); end;
        def m4 mass(mass_const); end;
        def m5 mass(mass_const); end;
```

```
        def pend_arm
                m0;
                label1 translatey(r) rotatez(theta);
        end;

        pend_arm translatex(x0) translatey(y0);
        dist_constraint(label1, m1, 0);
        path_constraint(m0, p1);

        m1 translatex(x1) translatey(y1);
        m2 translatex(x2) translatey(y2);
        m3 translatex(x3) translatey(y3);
        m4 translatex(x4) translatey(y4);
        m5 translatex(x5) translatey(y5);

        dist_constraint(m1, m2, side_length);
        dist_constraint(m2, m3, side_length);
        dist_constraint(m3, m4, side_length);
        dist_constraint(m4, m5, side_length);
        dist_constraint(m5, m1, side_length);

# Specify spring and damping forces between masses
        spring_and_damper(m1, m3, equil_length,
                spring_constant, damping_constant);
        spring_and_damper(m2, m4, equil_length,
                spring_constant, damping_constant);
        spring_and_damper(m3, m5, equil_length,
                spring_constant, damping_constant);
        spring_and_damper(m4, m1, equil_length,
                spring_constant, damping_constant);
        spring_and_damper(m5, m2, equil_length,
                spring_constant, damping_constant);

        line(m0_x, m0_y, 0, m1_x, m1_y, 0);
        polygon(m1_x, m1_y, 0, m2_x, m2_y, 0,
                m3_x, m3_y, 0, m4_x, m4_y, 0, m5_x, m5_y, 0);
# Draw the spline path
        path(p1);
```

Figure 6: Balloon pendulum script

expressions for a mechanical system. They also allow for the easy introduction of additional coordinates and constraint equations.

$$\frac{\partial L}{\partial q_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} + \sum_{k=1}^{m} \lambda_k \frac{\partial f_k}{\partial q_i} = -Q_i$$

$i = 1, \ldots, n$ where

$L = T - U$ is the Lagrangian (the difference between the kinetic and potential energies of the system).

$q_1 \ldots q_n$ are called generalized coordinates. They completely specify the position of the mechanical system.

$\dot{q}_1 \ldots \dot{q}_n$ are called generalized velocities. They completely specify the velocity of the mechanical system.

Q_i is the force applied to the i'th coordinate. This can be any sort of force, e.g. a linear force if q_i is a translation, or a torque if q_i is a rotation.

$f_1 \ldots f_m$ are constraint equations involving the coordinates.

λ_k is the force required to maintain the k'th constraint.

We can introduce new coordinates as long as this relationship is maintained:

number degrees of freedom = number coordinates - number constraint equations.

The number of degrees of freedom in a mechanical system is the minimum number of coordinates required to completely describe the state of the system.

Generating the Differential Equations

By introducing additional coordinates and constraints, we can simplify the derivation of the equations so that we don't require the use of complex symbolic mathematics software. For example, consider the mechanical system shown in figure 8.

We have a mass m attached to one end of a spring. The other end of the spring is pulled in the direction

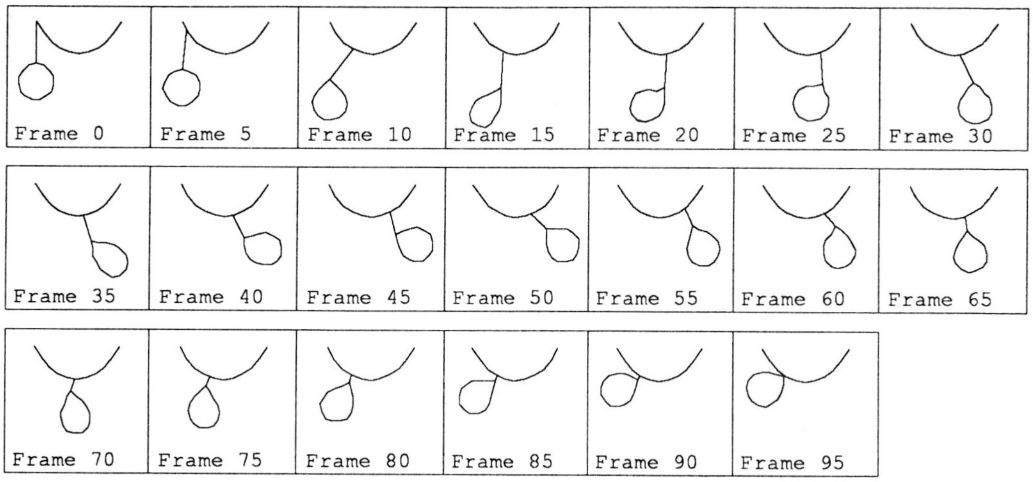


Figure 7: Frames from the balloon pendulum sequence

indicated in figure 8. The CHARLI script in figure 9 describes the action.

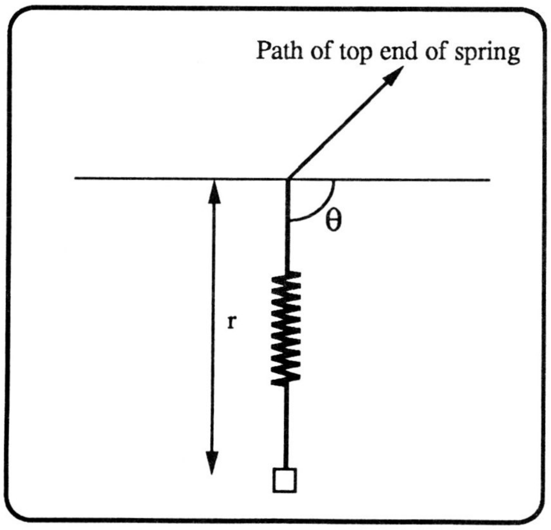


Figure 8: Pulling a spring

```
dynamic r, theta;
var x, y, equil_length, spring_constant,
        damping_constant;

equil_length = 5;
spring_constant = 3;
damping_constant = 1.5;

def square
        polygon(-0.1, -0.1, 0, 0.1, -0.1, 0,
                0.1, 0.1, 0, -0.1, 0.1, 0);
end;
def bob
        mass(10);
        square;
end;

x = {0 at 0 sec slowin 7.2 at 10 sec};
y = {0 at 0 sec slowin 7.2 at 10 sec};
r = initial(equil_length, 0);
theta = initial(-pi / 2, 0);

spring_and_damper(r, equil_length,
        spring_constant, damping_constant);

m0      translatex(r)
        rotatez(theta)
        translatex(x)
        translatey(y);

line(bob_x, bob_y, 0, x, y, 0);
```

Figure 9: Script for the pulled spring

The kinetic and potential energies of the mechanical system are

$$T = \frac{1}{2}m(\dot{m}_x{}^2 + \dot{m}_y{}^2 + \dot{m}_z{}^2)$$

and

$$U = mgh = mgm_y$$

where

m is the mass of particle.

m_x, m_y and m_z specify its position.

$(\dot{m_x}^2 + \dot{m_y}^2 + \dot{m_z}^2)$ is its speed squared.

g is the acceleration due to gravity.

We can find m_x, m_y and m_z from the geometric transformations in the CHARLI script.

$$\begin{bmatrix} m_x & m_y & m_z & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} Tx\,(r) \\ Rz\,(\theta) \\ Tx\,(x) \\ Ty\,(y)\,, \end{array}$$

where $Tx(translatex)$, $Ty(translatey)$ and $Rz(rotatez)$ are standard four by four homogeneous transformation matrices. Multiplying the transformation matrices yields m_x, m_y, and m_z. Applying Lagrange's equations yields the DE's. Deriving the equations in this direct manner often gives a set of very complicated DE's. [Haug 1984] gives an example. We introduce additional coordinates α_1, β_1, α_2 and β_2 to break up the transformation matrix list (For illustrative purposes, this example is in two dimensions). For each transformation a new pair of coordinates is introduced:

$$\begin{bmatrix} \alpha_1 & \beta_1 & 1 \end{bmatrix} = \begin{bmatrix} r & 0 & 1 \end{bmatrix} \begin{bmatrix} \xi & \eta & 0 \\ -\eta & \xi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \alpha_2 & \beta_2 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 & \beta_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} m_x & m_y & 1 \end{bmatrix} = \begin{bmatrix} \alpha_2 & \beta_2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & y & 1 \end{bmatrix}$$

Now instead of just *theta* and r being dynamical variables, we have m_x, m_y, r, ξ, η, α_1, β_1, α_2 and β_2. A corresponding set of constraint equations are found by the system from the above matrix expressions:

$$\begin{aligned} f_1 &: \alpha_1 - r\xi = 0 \\ f_2 &: \beta_1 - r\eta = 0 \\ f_3 &: \alpha_2 - \alpha_1 - x = 0 \\ f_4 &: \beta_2 - \beta_1 = 0 \\ f_5 &: m_x - \alpha_2 = 0 \\ f_6 &: m_y - \beta_2 - y = 0 \\ f_7 &: \xi^2 + \eta^2 - 1 = 0 \end{aligned}$$

We use ξ, η, and constraint equation f_7 instead of θ because $\frac{d^2}{dt^2}\xi$ is a simpler expression than $\frac{d^2}{dt^2}\cos(\theta)$. The Lagrangian L is:

$$L = \frac{1}{2}m(\dot{m_x}^2 + \dot{m_y}^2) - mgm_y$$

Applying Lagrange's Equations to the above gives us nine DE's in the 16 variables m_x, m_y, r, ξ, η, α_1, β_1, α_2, β_2, λ_1, λ_2, λ_3, λ_4, λ_5, λ_6 and λ_7. To get the additional seven equations needed for a solvable system, we use the second time derivatives of $f_1 \dots f_7$ [Marion 1970]. Figure 10 shows the DE's expressed in matrix form (zero elements are represented by "."):

The coefficient matrix has the following desirable properties:

- Very simple, often constant entries
- Very sparse
- Symmetric

Solving the differential equations

There are two phases to solving the differential equations:

1. Solve the sparse linear system to find the accelerations.
2. Use the accelerations to compute the system state at the next time step.

Currently our system solves the linear system of accelerations using Gaussian elimination with partial pivoting. The integration of the DE's is done using fourth order Runge-Kutta.

Implementation — dist_constraints

A **dist_constraint** is specified in a CHARLI script by

dist_constraint$(m_0, m_1, f(t))$;

where

m_0, m_1 are masses or labels.

$f(t)$ is a constant or track.

If $f(t)$ is a non-zero constant, the following equation is used to implement the constraint:

$$(m_{1x} - m_{0x})^2 + (m_{1y} - m_{0y})^2 + (m_{1z} - m_{0z})^2 = f(t)^2$$

Differentiating twice with respect to time gives us the constraint equation. If $f(t)$ is constant zero, $m_{1x} = m_{0x}$, $m_{1y} = m_{0y}$, and $m_{1z} = m_{0z}$. The constraint evaluates to $0 = 0$. To implement a zero distance constraint, three equations are used:

$$\begin{aligned} m_{1x} &= m_{0x} \\ m_{1y} &= m_{0y} \\ m_{1z} &= m_{0z} \end{aligned}$$

If $f(t)$ is a track and therefore time-varying, CHARLI checks for zero distance while the simulation is running and uses the approriate equations.

$$\begin{bmatrix}
-m & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . & . \\
. & -m & . & . & . & . & . & . & . & . & . & . & . & . & 1. & . \\
. & . & . & . & . & . & . & . & . & -\xi & -\eta & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & -r & . & . & . & . & . & 2\xi \\
. & . & . & . & . & . & . & . & . & . & -r & . & . & . & . & 2\eta \\
. & . & . & . & . & . & . & . & . & 1 & . & -1 & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & 1 & . & -1 & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & -1 & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & 1 & . & -1 & . \\
. & . & -\xi & -r & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & -\eta & . & -r & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & -1 & . & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & -1 & . & 1 & . & . & . & . & . & . & . \\
1 & . & . & . & . & . & . & -1 & 1 & . & . & . & . & . & . & . \\
. & 1 & . & . & . & . & . & . & -1 & . & . & . & . & . & . & . \\
. & . & . & 2\xi & 2\eta & . & . & . & . & . & . & . & . & . & . & .
\end{bmatrix}
\begin{bmatrix}
\ddot{m}_x \\ \ddot{m}_y \\ \ddot{r} \\ \ddot{\xi} \\ \ddot{\eta} \\ \ddot{\alpha}_1 \\ \ddot{\beta}_1 \\ \ddot{\alpha}_2 \\ \ddot{\beta}_2 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \\ \lambda_7
\end{bmatrix}
=
\begin{bmatrix}
-Q_{m_x} \\ -Q_{m_y} + mg \\ -Q_r \\ -Q_\xi \\ -Q_\eta \\ 0 \\ 0 \\ 0 \\ 0 \\ 2\dot{r}\dot{\xi} \\ 2\dot{r}\dot{\eta} \\ \ddot{x} \\ 0 \\ 0 \\ \ddot{y} \\ -2(\dot{\xi}^2 + \dot{\eta}^2)
\end{bmatrix}$$

Figure 10: DE's in matrix form

Implementation — path_constraints

A **path_constraint** is specified by one of:

path_constraint(m_0, **point**(x_0, y_0, z_0))
path_constraint(m_0, **line**($x_0, y_0, z_0, x_1, y_1, z_1$))
path_constraint(m_0, **spline**($x_0, y_0, z_0, \ldots, x_n, y_n, z_n$))

where

m_0 is a mass or label.

x_0, y_0, z_0, $\ldots x_n$, y_n, z_n are constants or tracks, $n \geq 3$.

A **point path_constraint** is essentially handled like a zero distance **dist_constraint**.

A **line path_constraint** is implemented by introducing a new dynamical variable u and using the parametric equations of a line passing through the two points $p_0 = (x_0, y_0, z_0)$ and $p_1 = (x_1, y_1, z_1)$.

$$\begin{aligned}
m_{0x} &= (x_1 - x_0)u + x_0 \\
m_{0y} &= (y_1 - y_0)u + y_0 \\
m_{0z} &= (z_1 - z_0)u + z_0
\end{aligned}$$

When m_0 is positioned in the CHARLI script, it should lie on the line (If is doesn't, CHARLI has a mechanism for pulling it onto the line). Since p_0 and p_1 can be tracks and thus time varying, CHARLI checks throughout the simulation that they are not coincident. If they are, CHARLI handles their degeneration into a point constraint. Differentiating twice with respect to time gives (for m_{0x}):

$$\ddot{m}_{0x} - (x_1 - x_0)\ddot{u} = (\ddot{x}_1 - \ddot{x}_0)u + 2(\dot{x}_1 - \dot{x}_0)\dot{u} + \ddot{x}_0$$

If x_0 or x_1 are not time dependent, i.e., not tracks, first and second derivative terms involving them will vanish.

Spline path_constraints are implemented using cubic B-splines. As with lines, CHARLI introduces a new dynamical variable u and uses the parametric spline equations

$$P(u) = \sum_{i=0}^{n} p_i N_{i,4}(u)$$

where

$p_i = (x_i, y_i, z_i)$ is the i'th control point.

$N_{i,4}(u)$ is the i'th cubic B-spline basis function. [Newman & Sproull 1979]

Spline path_constraints are handled essentially the same as line constraints, with one difference. Since B-splines are piecewise continuous, $N_{i,4}(u)$ is not the same function for all values of u. The equations constraining m_0 change as m_0 moves. Thanks to the second order continuity provided by cubic B-splines, the transitions between spline segments are fairly smooth. The same mechanism that pulls a **mass** or **label** onto a **line** path also serves to keep a **mass** or **label** from deviating from a **spline** path.

Ensuring constraint satisfaction

It is well known in control systems and circuit theory that circuits described by second order differential equations such as

$$\ddot{y} = 0$$

are unstable. Outside noise (such as numerical integration error) can be amplified. Circuits such as

$$\ddot{y} + 2\gamma\dot{y} + \delta^2 y = 0$$

are stable. Writing our distance and path constraints in this form will prevent constraint violations. Practical experience has shown that, for most problems, γ and δ values between 5 and 50 are adequate. [Nikravesh 1984].

The case $\gamma = \delta$ corresponds to critical damping and results in the fastest integration error correction. This technique was used by [Isaacs & Cohen 1988] in their dynamic animation system. It is this mechanism that we use to initially satisfy **dist** and **path** constraints and ensure that they remain satisfied.

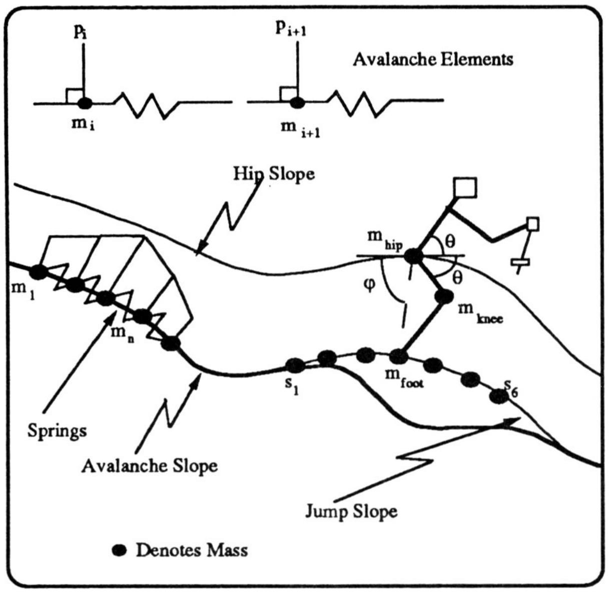


Figure 11: Skier chased by avalanche

A more complex example

Figure 11 depicts a crude skier and shows the parts that will be animated. Here's how CHARLI is used to animate the sequence:

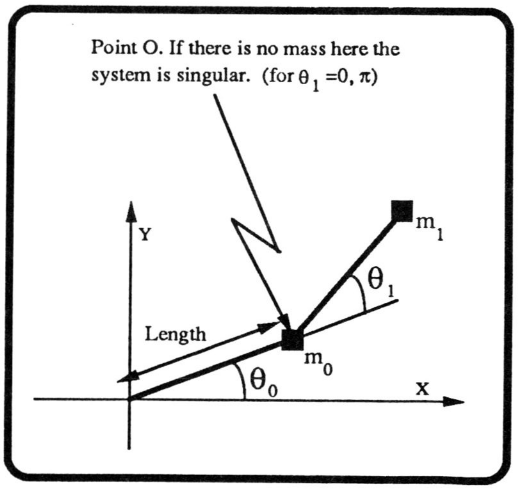
The avalanche The avalanche is constructed from a chain of avalanche elements. Each mass m_i is constrained to lie on the avalanche slope using a **path_constraint**. Spring forces act between adjacent m_i's. A damping force proportional to the speed of the avalanche acts on mass m_1 to stretch the avalanche out. In each avalanche element, the line from m_i to p_i remains perpendicular to the spring connecting m_i to m_{i+1}. Lines are drawn between adjacent p_i's to form the outline of the avalanche.

The skier The skier's lower body is constructed from three masses. m_{hip} is constrained to lie on the hip slope. m_{foot} is constrained to lie on the jump slope. m_{hip} and m_{foot} are placed such that their relative orientation (angle ϕ) remains constant. There are fixed **dist_constraints** between m_{knee} and m_{hip} and between m_{knee} and m_{foot}. As the jump slope and hip slope converge and diverge, the distance between m_{foot} and m_{hip} varies accordingly. This changing distance and the **dist_constraints** involving m_{knee} cause the skier's legs to pump up and down as he skis down the hill. The upper part of the body is driven kinematically.

The skis The skis are formed from a number of masses constrained to lie on the jump slope. **Dist_constraints** are specified between adjacent s_i's so that the skis remain the same length. Figure 12 shows the skier trying to outrun the avalanche.

Limitations of CHARLI dynamics

CHARLI allows the specification of dynamical systems that are singular in certain configurations (the linear system of accelerations has no solution). A sophisticated CHARLI user can generally avoid this problem through careful script writing. An unsophisticated CHARLI user will have no idea why his simulation is blowing up. It would be much better to notify the user when he has specified a dynamical system that has singularities. We lack the mathematical sophistication to know if it is possible to detect all such systems. We do, however, have some limited understanding of a certain class of singular dynamical systems. This class arises if not enough masses are in the system and they are not in the correct places. Consider the double pendulum illustrated in figure 13. It is specified by the script in figure 14.



Point O. If there is no mass here the system is singular. (for $\theta_1 = 0, \pi$)
Y
m_1
θ_1
Length
m_0
θ_0
X

Figure 13: Pendulum System with Singularities

Suppose there is no mass at point 0. If we analyse the acceleration matrix produced by CHARLI for this pendulum, we find that it is singular for $\theta_1 = 0, \pi$. Upon further consideration this seems quite reasonable. There are external forces (gravity) and internal forces (the tensions in the connecting rods) acting on point 0. According to Newton's second law $F = ma$, if a force is applied to a massless object, a singularity occurs (a is undefined). If $m \neq 0$, a is always defined. If we put

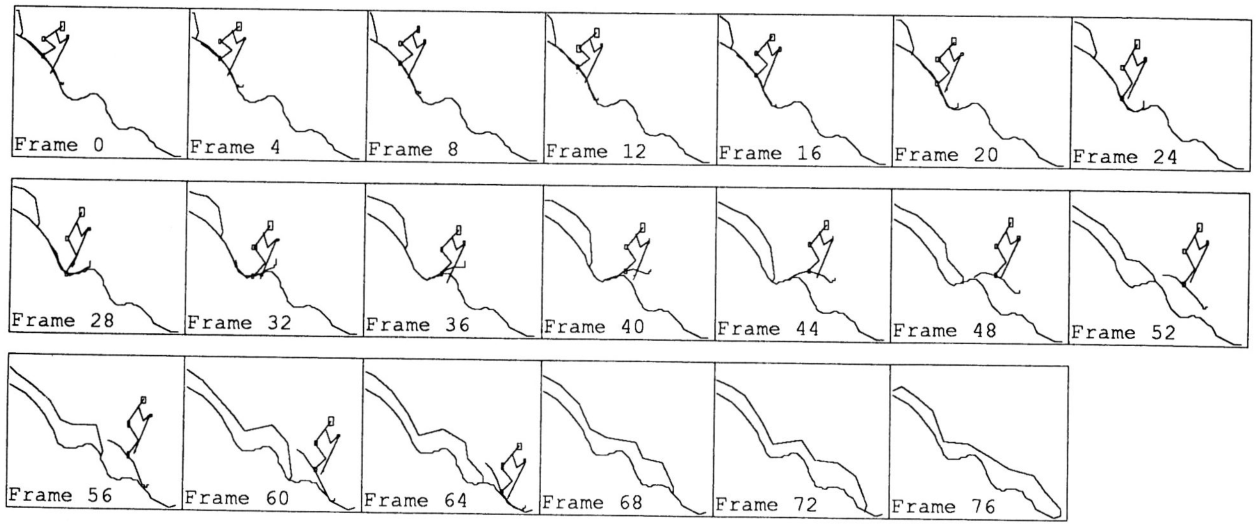


Figure 12: Frames from the ski chase sequence

```
var length;
dynamic theta0, theta1;

def square
        polygon( 0,0,0, 1,0,0, 1,1,0, 0,1,0 )
                scale(0.1);
end;
def m0
        #Include next line for mass at point 0
        mass(1.0);
        square;
end;
def m1 mass(2.0); square; end;

length = 1;
```

```
theta0 = initial(0, 0);
theta1 = initial(0, 0);

def p1
        m1 translatex(length) rotatez(theta1);
        m0;
        line(m0_x, m0_y, 0, m1_x, m1_y, 0);
end;
def p0
        p1 translatex(length) rotatez(theta0);
        line(0, 0, 0, m1_x, m1_y, 0);
end;

#Instantiate the pendulum at the origin
p0;
```

Figure 14: Script that generates pendulum system with singularities

a mass at point 0, the system should no longer have singularities. An analysis of the acceleration matrix for such a pendulum shows that this is indeed the case.

Future Work

Singularities The user is currently not informed if he has specified a dynamical system that has singularities. To make CHARLI dynamic animation available to the masses, this clearly needs to be done. We have some intuitive understanding of this problem, but we need to work from a more solid mathematical foundation.

Speed Solving the large, sparse acceleration matrix is expensive, and the system currently runs quite slowly. We are exploring two methods to speed this up:

Iterative methods The difference between the accelerations from the previous time step and the current time step is quite small. As such, the accelerations from the previous time step would provide a good initial guess for an iterative method. We are currently experimenting with NSPCG [Oppe *et al* 1988]. NSPCG is a package designed to help find the best iterative solution method for the problem at hand.

Preprocessing matrix diagonalization phase

Many of the acceleration matrix coefficients are constants. Of these, many are 1 or -1. Of the matrix coefficients that are not constant, we often know the bounds on their values. These properties might allow us to almost entirely diagonalize the coefficient matrix in a preprocessing stage, resulting in very fast runtime solution.

Conclusion

We have presented here a method of describing a dynamic simulation using a modelling and animation description language called CHARLI. We have used the language to generate a number of animated sequences that would have been very difficult to make without such a description language. The language, combined with some interactive tools, forms a very powerful method of describing complex animation sequences which include kinematic and dynamic descriptions.

Acknowledgements

We would like to acknowledge the help of the University of Calgary Graphics Lab. and graduate students past and present who have worked on the Graphicsland Animation system. This research is partially supported by grants from the Natural Sciences and Engineering Research Council of Canada. Chuck thanks Mike Chmilar, who developed kinematic CHARLI. He also thanks his wife Maggie, without whose encouragement this work would never have been completed.

References

[Armstrong & Green 1985] William Armstrong and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[Barzel & Barr 1988] Ronen Barzel and Alan H. Barr. A Modeling System Based On Dynamic Constraints. *Computer Graphics (Proc. SIGGRAPH 88)*, 22(4):179–188, August 1988.

[Chmilar & Wyvill 1989] Michael Chmilar and Brian Wyvill. A software architecture for integrated modelling and animation. *Proc. CG International 89*, pages 257–276, 1989.

[Gomez 1985] Julian E. Gomez. Twixt: A 3D Animation System. *Computers and Graphics*, 9(3):291–298, 1985.

[Haug 1984] Edward J. Haug. Elements and methods of computational dynamics. *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, f9:3–38, 1984.

[Haumann & Parent 1988] D. R. Haumann and R.E. Parent. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4(6):332–347, 1988.

[Isaacs & Cohen 1987] Paul M. Isaacs and Michael F. Cohen. Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4), July 1987.

[Isaacs & Cohen 1988] Paul M. Isaacs and Michael F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4(6):296–305, 1988.

[Marion 1970] Jerry B. Marion. *Classical Dynamics of Particles and Systems.* Academic Press, 111 Fifth Ave. New York, 1970.

[Miller 1988] Gavin S.P. Miller. The Motion Dynamics of Snakes and Worms. *Computer Graphics (Proc. SIGGRAPH 88)*, 22(4):169–178, August 1988.

[Newman & Sproull 1979] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, New York, Reading, Mass, 1979.

[Nikravesh 1984] Parviz E. Nikravesh. Some methods for dynamical analysis of constrained mechanical systems. *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, f9:351–368, 1984.

[Oppe *et al* 1988] Thomas C. Oppe, Wayne D. Joubert, and David R. Kincaid. Nspcg user's guide. Technical report, Center for Numerical Analysis, The University of Texas at Austin, April 1988.

[Overveld 1989] C.W.A.M van Overveld. A technique for motion specification in computer animation. *Proc. CG International 89*, 1989.

[Pentland & Williams 1989] Alex Pentland and John Williams. Good Vibrations: Modal Dynamics for Graphics and Animation. volume 23, pages 215–222, August 1989.

[Terzopoulo *et al* 1989] Demetri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models (from goop to glop). *Proc. Graphics Interface 1989*, pages 219–226, 1989.

[Wilhelms 1985] Jane Wilhelms. *Graphical Simulation of the motion of articulated bodies such as humans as humans and robots, with a particular emphasis on the use of dynamic analysis.* PhD thesis, University of California, Berkeley, Dept. of Computer Science, 1985.

[Wilhelms *et al* 1988] Jane Wilhelms, Matthew Moore, and Robert Skinner. Dynamic animation: interaction and control. *The Visual Computer*, 4(6):283–295, 1988.