Appeared in: Koza, J.R., et al. (editors), *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, pp. 139-146, July 13-16 1997, Stanford University. San Francisco: Morgan Kaufmann. (1997)

Genetic Programming Evolution of Controllers for 3-D Character Animation

Larry Gritz
Pixar Animation Studios
1001 W. Cutting Blvd.
Richmond, CA 94804
lg@pixar.com

James K. Hahn
The George Washington University
801 22nd St. NW
Washington, DC 20054
hahn@seas.gwu.edu

ABSTRACT

The dominant paradigm for 3-D character animation requires an animator to specify the values for all degrees of freedom of an articulated figure at key frames. Specifying motion that is physically believable and biologically plausible is a tedious practice requiring great skill.

We use evolutionary techniques (specifically Genetic Programming) as a means of controller synthesis for character animation. Controllers which drive a dynamic simulation of the character are evolved using the goals of the animation as an objective function, resulting in physically plausible motion. We discuss the development of objective functions used to guide the controller evolution, making reusable skill controllers, and comparisons of the convergence rates for different parameters of the evolutionary runs.

1. Introduction

In this investigation, we are concerned with a particular subset of computer animation: *character animation*. In an animation, many objects may be moving around. Characters are those objects whose movements contribute to telling the story and which express thinking, volition, and emotion.

An example of animation which does not fall into the category of character animation would be *effects animation*. This would include movement of objects other than characters, or which is not designed to convey emotion or intent. Examples of such animation include falling rain, leaves blowing in the wind, flying spaceships, rotating business logos, or a flock of bats flying around. Methods have been developed to successfully automate several specific cases of effects animation, largely due to the fact that many effects animation tasks can be solved by straightforward physical simulation.

As the art and craft of character animation was refined, particularly at Disney Studios, animators identified specific properties of superior animation. Thomas and Johnston [Thomas and Johntson, 1984], and later Lasseter [Lasseter, 1987], identified "principles of animation" which describe important aspects of realistic character animation. They included squash and stretch, anticipation, staging, straight ahead action and pose to pose, follow through and overlapping action, slow in and slow out, arcs, secondary action, timing, exaggeration, and appeal. Some of these principles are purely artistic (such as appeal), but many others describe the dynamic and behavioral properties of real materials and creatures.

The primary method for animating 3-D characters is by *key framing*. The technique takes its name from the practice in 2-D cel animation of a senior animator drawing the figure at particular frames in which significant events or extrema of motion occur. More junior animators then use the key frame drawings as a guide for the intermediate frames (doing a kind of human-powered interpolation of the poses). These techniques were extended into the realm of 3-D computer animation by specifying and interpolating positions and joint angles of the characters. Extrema for each degree of freedom (DOF) are specified by human animators, and the intermediate values are interpolated (frequently by cubic spline) by the animation software.

The motion resulting from interpolating the key frames is purely kinematic. If the motion is to be physically realistic (exhibiting inertia, obeying force laws, etc.), biologically plausible (obeying joint limits or other constraints of a particular creature), follow any of the principles of animation described earlier, or be entertaining, **it is entirely up to the human animator to choose key times and values which meet these constraints.** Virtually all production character animation is done using this method, in spite of the large amount of research into automating character animation. It is the principles mentioned earlier which make animation (of any

media) believable, entertaining, and able to convey the underlying story. It is the job of the animator to specify motion which adheres to these principles, to prevent bad motion from distracting the audience from the story. Being able to do this well is a rare talent, and viewers of animation are merciless critics who easily recognize unrealistic motion.

Given the extreme difficulty of key framing, yet our well-developed ability to recognize good or bad motion when we see it, we sought a method based on the following principle: *let the system generate the motion, and let the human provide the determination of what is "good."* Therefore, the approach we took was one of *controller synthesis*: our animated characters are dynamically simulated robots, and we use an optimization technique to generate controllers which satisfy the goals and constraints of the motion which are specified by the animator. Genetic Programming (GP) is used to synthesize the controllers, and often results in controllers for animation which exhibit the principles of animation and produce fluid, organic motion.

We previously described our early efforts in this area in [Gritz and Hahn, 1995]. In this paper, we both summarize those results for the GP community, and report on extensions to that work which include more robust action-based controllers and analysis of the rate of convergence of the GP process.

2. Related Work

Witkin and Kass [Witkin and Kass, 1988] proposed the *spacetime constraints* method, which generates kinematic motion which both satisfies high level goals (e.g. "jump from here to there") and also appears to be physically plausible. The resulting motion tends to exhibit many of the principles of traditional animation such as squash and stretch, anticipation, and follow-through. Spacetime constraints uses sequential quadratic programming (SQP) to optimize the kinematic positions of an articulated figure, using energy consumption as an objective function and constraining the solution in order to ensure that the motion is physically plausible. Unfortunately, this method generates solutions which depend on the initial guess provided to the optimizer, can easily find a local minimum, and the resulting kinematic motion is not particularly reusable. In addition, it is by no means clear that energy is the best criteria to optimize.

More recently, other researchers proposed methods of automatically generating walking motion using search and optimization techniques [Ngo and Marks, 1993, van de Panne and Fiume, 1993]. Both gave impressive results, yielding physically plausible motion and automatically finding a number of walking methods. However, the resulting motion had high *specialization* (i.e. they made walking gaits, not general motion), but low *specificity* (i.e. they both simply walked forward, rather than having more specific instructions, like "walk to position X"). Both of these methods were optimizing structures of fixed complexity (a network of fixed topology and a stimulus-response table, respectively). We believe that this is a disadvantage compared to a representation which may change its own complexity. We chose a different representation, namely a mathematical description (computer program) describing how the joint forces vary with time and changes in the state of the simulation. We believe that this representation offers many advantages, and is appropriately optimized using the GP technique.

Karl Sims used a GP-like process to evolve procedural textures, where the fitness metric was human evaluation [Sims, 1991]. Sims also described the use of evolutionary programming to design entire creatures [Sims, 1994], in which even the creature topology was evolved for walking, swimming, etc. In contrast, our work deals with figures of fixed topology and geometric structure, as one would expect for character animation.

3. GP Evolution of Controllers

3.1. System Overview

The conceptual layout of our system is shown in Figure 1. As input to the system, the user (animator) supplies both a detailed articulated figure model and a fitness metric. The figure model includes information on sizes and connectivity of the links, masses and inertias, joint limits, etc. The fitness metric (or objective function) implicitly encodes information about the motion which is desired from the character. The fitness metric rates the motion sequence by giving lower numbers to motion sequences which are close to what is desired by the user, and high numbers to sequences which deviate from the goals of the animation. The system itself consists of a Genetic Programming-based optimization module, and a dynamics simulation module.

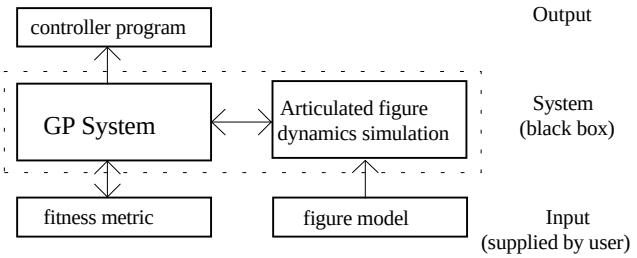


Figure 1: Overall Structure of the System

When a GP run has finished, the final output of the GP subsystem is a single controller program. This controller program is the one which, when used to govern the actions of a simulated agent, resulted in the best rated agent according to the user-supplied fitness metric. This controller program may then be used to generate the motion control needed for the animation.

3.2. Controller Structure and Simulation

The controllers themselves are LISP S-expressions [Steele, 1984], which give the "desired orientation" of each joint at the time when it is evaluated. The controller is a list of subexpressions for each degree of freedom (see Figure 2). Motion for animated characters is generated using dynamic simulation of articulated bodies using methods outlined in [Armstrong and Green, 1985, Hahn, 1988, Wilhelms, 1990]. Each joint is controlled internally by a damped angular spring, and has a "neutral" (or desired) orientation which at any time is given by evaluating the controller S-expression at each time step of the simulation. The spring pushes the joint toward the desired orientation with a torque proportional to the angle between the actual and desired orientations using a proportional derivative (PD) controller, which is a common technique used in both computer animation and robotic control. The system also accounts for joint limits and collision detection and response. Therefore, we are guaranteed to get motion which is physically plausible.

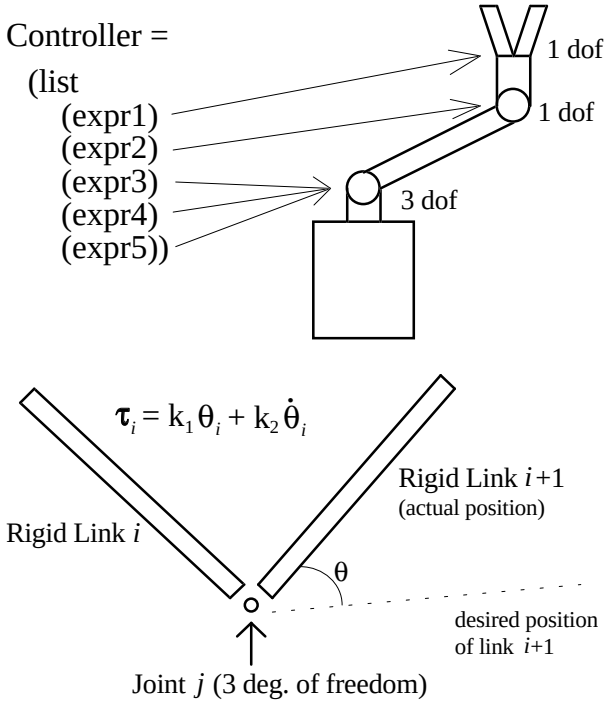


Figure 2: Controllers are expression lists which give desired joint angles for each degree of freedom. Torque at each joint is proportional to the angular difference between actual and desired orientations of the links.

3.3. Experimental Design

To test the idea of using GP controller synthesis for animation, we chose to animate locomotion of a desk lamp, after *Luxo, Jr.* [Pixar, 1986] (which seems appropriate since the film, though completely key framed, is still considered a high point of computer animation quality). Our lamp model (dubbed "L*xo") was a fully 3-dimensional articulated body with 4 links and 3 internally controllable degrees of freedom. All dynamics were simulated in 3-D. The geometry, masses, inertias, and so on were given to the system. The lamp had access to its position relative to the goal point, its velocity, and the amount of force being imparted to its base through contact with the floor. We wanted to teach the lamp to be able to move about. It would not have been considered a success to simply move forward—we wanted to generate a controller program to bring it to rest on a particular spot, to show that we could perform any intricate scripted motion we desired.

We previously reported (in [Gritz and Hahn, 1995]) generating locomotion controllers for L*xo for specific actions, for example, hopping forward 30 cm. This resulted in fairly realistic and organic looking motion, however this technique has a major shortcoming: the resulting controllers accomplish exactly one action. This leads to several conclusions about the deficiencies of this approach.

- The controllers are not reusable for other situations. Having trained L*xo to jump forward 30 cm, we must do a completely separate training if we later need L*xo to jump 50 cm.
- The controllers are "brittle." They are developed for a particular set of initial conditions, and would not be expected to do well under differing conditions.
- Though the motion generated by the controllers can be simulated in real time, the training process happens off-line, sometimes taking hours to derive the controllers. We could live with substantial off-line computation if we only need to do it once, resulting in a controller which is reusable in a variety of situations.

The solution to these problems is to evolve "robust" controllers which are reusable in a variety of related situations. Robust controllers are reusable, not brittle, and can handle a variety of initial conditions. If a character is used for multiple animations, the robust controllers evolved when the character was first created can continue to be used for subsequent animations involving that animated character. For example, a controller to "jump forward 30 cm" yields a specific, one time only *action*. In contrast, we might want a *skill*, such as "walk forward," which could accept parameters for distance traveled and time to spend traveling. This controller, if we could evolve it, could potentially be reused for all the locomotion tasks of an animation, or even several animations featuring the same character. To generate robust controllers, we used the method described in [Koza, 1992] of using a *training set* to evaluate the fitness function. Rather than have the fitness rating composed of some metric based on one particular set of initial conditions, a number of fitness cases are generated randomly, and the new aggregate fitness measure is given by the average of N_{fc} individual trials.

3.4. GP Subsystem

The GP subsystem runs the basic GP algorithm described by Koza [Koza, 1992]. Individuals are S-expressions giving desired joint angles, which generate torques at the joints of the simulated robot, as described in the previous subsection. To evaluate the fitness of an individual, the dynamics subsystem is invoked to simulate the situation, given the particular individual as the robot controlling program. When the simulation is over, the user-supplied fitness evaluation function rates the performance based on statistics collected by the dynamics program. GP run parameters are summarized in Table 1.

3.5. Constructing Fitness Measures

To rate a controller as a potential solution, the fitness function uses the dynamics system to simulate the motion using that particular controller program, then uses statistics supplied by the dynamics system to compute a fitness rating. We divided the fitness measure into a *main goal* and *style points*. The main goal is a simple metric of whether the primary task of that motion sequence has been fulfilled. For example, if the point of the motion sequence is to move the figure to the "X", then the main goal should simply be the distance between the figure and the "X" at the end of the time allotment. Since the motion is so grossly underconstrained, the GP system can often find outrageous ways of meeting such a simple fitness requirement. For example, it might somersault to the goal point instead of hopping. Because of this, we find it useful to add *style points*, which can be thought of as additional rewards or penalties granted to the individual's performance.

In the case of the L*xo lamp locomotion, we used the following fitness metric (described here informally):

1. Main goal: distance between base center and goal point "X" at the end of the time allotment.
2. Style points: a weighted sum of the following:
   a) bonus for completing the motion early (how much time did it take to get to the "X").
   b) penalty for excess movement after goal was met (this was to keep it still after it got done).
   c) penalty for hitting its head or falling over ("safety" considerations).
   d) bonus for ending with joints at neutral angles.

These fitness criteria were admittedly developed in an *ad hoc* manner, though we found out that they did surprisingly well for a variety of tasks (and even a variety of characters) with little modification.

We found that immediately placing all of the style considerations on the motion did not give the GP process the best opportunity to discover fit individuals, simply because it was so restrictive. We solved this problem by rating controller programs based on the main goal only for the first few generations, then slowly phased in the style considerations, generation by generation, yielding the aggregate fitness function:

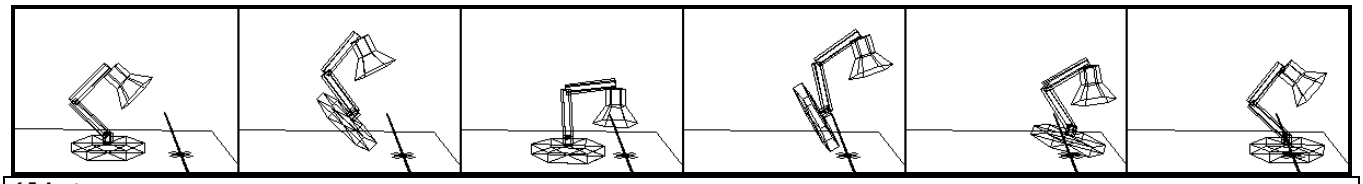
$$fitness = fit_{main} + \phi(fit_{style}), \qquad \phi = \sqrt{\frac{g}{G}}$$

where g is the current generation and G is the total number of generations for the run. The square root merely changes the shape of the phase curve, so that it increases more quickly in earlier generations than in later generations.

The net effect of the phase control was to teach it how to get to the "X" first, without regard to whether it fell over afterward or whether it ended in a neutral position. Once it had mastered that task, then we placed more and more restrictions on it. By the time we got to the last generation, the motion included the full set of style considerations. In this manner, we allowed the motion to start crudely and get progressively more stable over several generations. This

Table 1: The tableau for the GP run to construct a robust forward locomotion controller for the L*xo lamp.

Objective	To derive a robust controller for L*xo which can locomote forward any distance.
Terminal set	*t* (time) *a0, a1, a2* (L*xo's internal angle sensors) *s0* (sensor 0 - the current of the floor on the base) *px, py, pz, vx, vy, vz* (position and velocity of L*xo's base relative to the goal) *goal_met* (status flag - nonzero after the base is at the goal) *goal_dist* (distance between the base and the goal) ℜ (the ephemeral random constant on [-30,30])
Function set	**+, -, *, %, ifltz**
Fitness cases	*Nfc* = 6 distances to the goal position, chosen over [5 cm, 60 cm].
Fitness metric	The average, over *Nfc* fitness cases, of the fitness function described in § 3.5. Using phase, *fitness* = (*main goal fit*) + ϕ(*style point fit*)
Hits	The number of cases in which L*xo came within 1.5 cm of the goal position within the time limit.
Wrapper	The individual is a list of 3 subprograms which map to the "desired angles" of L*xo's three internally controllable degrees of freedom.
Misc. Parameters	Population size *M* = 400, Number of generations *G* = 50 Individuals chosen using tournament selection with tournament size of 6. *Pc* = 90% of individuals constructed using crossover, remainder using reproduction.



```
(list
  (- (ifltz a0 pz a2) (% a1 t))
  (ifltz (- (- (ifltz a0  pz a2) (% a1t)) (ifltz (+ a2 15.4963) (ifltz a0   pz a2) (% a1
    a2))) (ifltz (% (- s0  vx) (+ a2 s0)) (% (- s0  vx) (+ a2 s0)) (+ (- a1  a1) (- a1 t)) (%
    (+ a1 vz) (ifltz (*  pz a1) (% a0  pz) (% px a0))))
  (- (ifltz s0 (ifltz (- (%  vx a0) (% vx vx)) (-(% vx a0) (- (+  -28.4382 t) (% a1 t))) (*
    16.5266 s0)) vz) (+ a1 vz)))
```

Figure 3: Lamp's Jumping motion and a controller program which generates it.

was easier for the system than requiring optimal motion at the very start.

3.6. Fitness test case generation

The goal of this particular test was to generate a general locomotion controller for the lamp which could be used for forward locomotion of any distance. For practicality, it was assumed that useful distances to move would be between 10 and 60 cm, though it was hoped that a general controller could be used for indeterminate distances. A total of *Nfc* fitness test cases were generated with goal distances on [.1,.6] (measured in meters). A regular distribution (e.g. .1, .2, .3) was not used, for fear that it would generate adequate controllers only for multiples of the step size.

The solution we used was to generate sample goal distances using a Poisson disk distribution. A total of *Nfc* goal distances were chosen, but as each sample was generated, it was discarded if it was too close to a previously chosen sample. This resulted in a selection of goal distances which were well spread out across the interval, yet were not multiples of some basic step size.

3.7. Animation Results

The resulting controller program produced a hopping motion which brought the lamp to the exact spot we desired (see Figure 3). These "robust" controllers were able to be reused for locomotion of any distance. The motion appeared smooth, physically realistic, efficient, and surprisingly organic. Note that for the example in the figure, the solution involved two hops. A solution with such severe discontinuities (such as collisions) would not be found by a local gradient-based optimization method such as spacetime constraints. Figure 3 also shows a sample controller program which made a successful jump. The controller programs are usually opaque to human interpretation, but we are generally not concerned with the robot's internal mechanisms for movement. Figure 4 shows more robust controllers which were used for more constrained motion, and where used to make the short video "L*xo Learns to Limbo." Using a single additional style term—a large penalty for hitting his head on the limbo pole—we were able to produce a series of motions for various heights of the limbo pole. The resulting motion has slight imperfections and at times incorporates seemingly "clever" strategies, which gives it an organic rather than robotic appearance. Runs of this type tended to take several minutes per generation, on an SGI R4000 processor. Generating a single robust controller therefore took on the order of a few hours. But once a controller was generated, it could be used for many animation sequences, and the simulations with an existing controller could be performed in roughly real time.

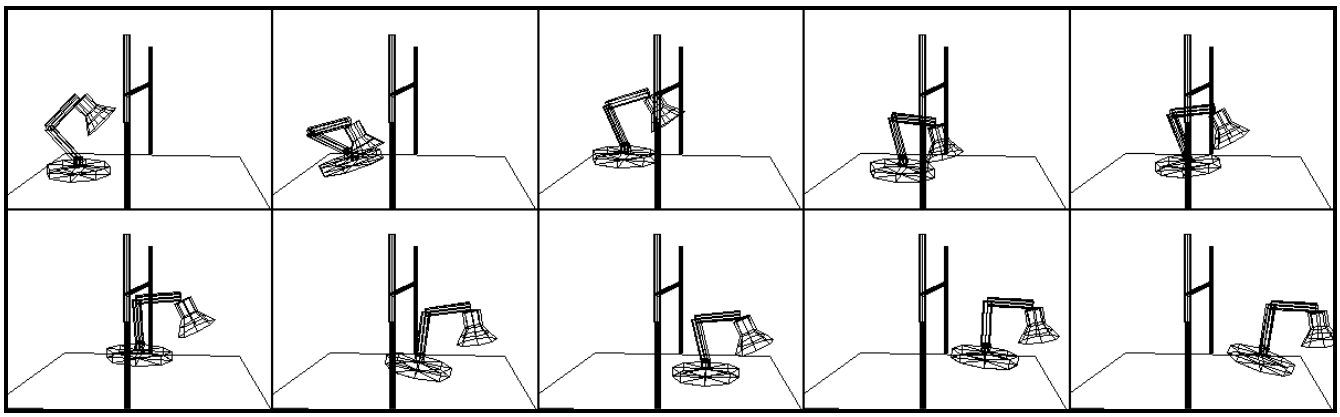


Figure 4: The lamp can learn to avoid obstacles when additional constraints are added to the system.

Appeared in: Koza, J.R., et al. (editors), *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, pp. 139-146, July 13-16 1997, Stanford University. San Francisco: Morgan Kaufmann. (1997)

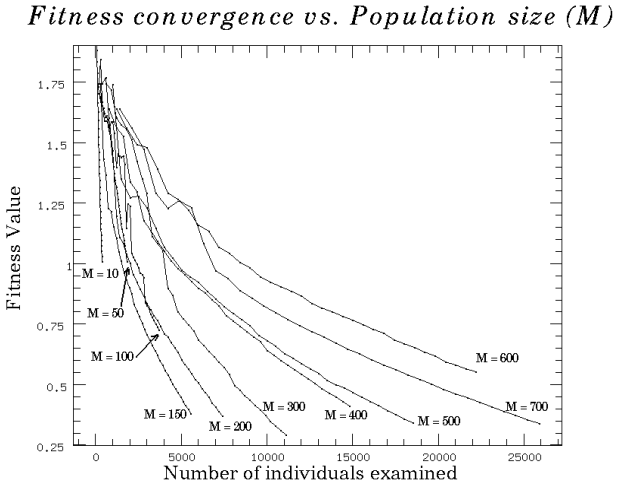


Figure 5: Graph of fitness vs. number of individuals examined for runs of different population sizes.

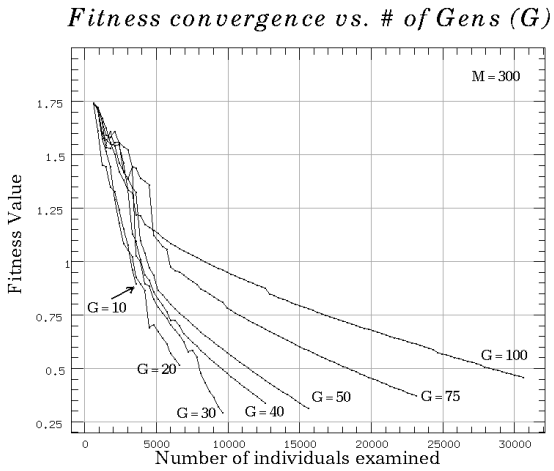


Figure 6: Graph of fitness vs. number of individuals examined for runs of different number of generations.

4. Convergence Analysis

A disturbing feature of the GP method of controller synthesis is that the GP run parameters are chosen so arbitrarily. It would be much more satisfying if we could point to empirical evidence for choosing the parameters. In addition, since the learning process is so time consuming, understanding how to make GP runs more efficient could be very beneficial in the effort to make this entire process more interactive. Choosing the various parameters of the GP runs is in itself an optimization problem. In this section, we report our experiments to determine optimal GP run parameters. The time it takes to derive a controller (the length of the run) is proportional to *M*, the population size; *G*, the number of generations; *Nfc*, and the number of fitness cases. Therefore, we concentrated on experiments involving variation of these parameters.

In all of the experiments, the most important metric is the total number of individuals examined, defined as $M \times G$. This is the basic measure of work done by the GP process. The other important quantity is of course fitness—lower values are better. The goal here is to find the values of the parameters which yield individuals with very low fitness values, by examining (i.e. simulating) as few individuals as possible. Note: all graphs are of the fitness of the best-of-generation for runs with the same random seed, and differing only by varying the parameter under investigation. Raw fitness values below 0.5 tended to correspond to good-looking motion when animated.

Population size — how many individuals do we actually need in each generation? Clearly there is some minimum population size for any given problem; with smaller populations, there will not be enough genetic diversity to converge to an adequate solution. Also there is some population size, above which only increases computation time without yielding better controller programs. To find this critical number, we ran several training runs for the robust L*xo locomotion described earlier. All of the trials used *G*=35, *Nfc*=3, and *Pc*=90%. Runs were performed with population sizes of 10, 50, 100, 150, 200, 300, 400, 500, 600, and 700 individuals. Figure 5 graphs fitness value vs. number of individuals examined for these different population sizes. Population sizes under 200 did not converge to good fitness values. However, for population sizes of 200 or more, the population size made little difference in the fitness of the best-of-run individual. Thus, it was most cost-effective to do runs with 200-300 individuals, and not beneficial to waste computation with population sizes of 500 or more.

Number of generations to run — we found that a population size of around 300 was optimal, but how many generations do we need to find a good solution? Again, some minimum probably exists. Fewer generations will simply not provide enough "time" to evolve fit controllers. But also there is likely some maximum, after which the controllers will not continue to improve. Using *M*=300, as suggested by the last experiment, *Nfc*=3, and *Pc*=90%, runs were performed with *G* = 10, 20, 30, 40, 50, 75, and 100 generations. Figure 6 graphs fitness value vs. number of individuals examined for runs of different number of generations. Remember that the graphs do not coincide because the phase function causes the fitness ratings to be dependent on the total number of generations to be run. The optimal value of *G* turned out to be 30. Presumably, fewer than 30 generations do not allow the controllers to evolve to good fitness. More than 30 generations not only does not produce more fit individuals, but in fact as *G* gets very large, the runs do not produce individuals as fit as *G*=30! This is not easy to explain, but one hypothesis is that the "selection pressure" of the phase function with

short runs forces fit individuals to flourish. When *G* is very large, there is less pressure to get rid of individuals with low style, and other individuals with potentially important (for style points) genetic material eventually are eliminated prematurely.

Type of selection for individuals — a variety of methods may be used to actually select the individuals for the next generation. Most of the examples from Koza [Koza, 1992] use fitness-proportionate selection, though other authors have suggested that tournament selection is more stable. We ran several trials which derived a robust hopping controller for L*xo. All of the trials used *M*=300, *G*=30, *Nfc*=3, and *Pc*=90%. Six trials were run: one using fitness-proportionate selection of individuals, and five using tournament selection using tournament sizes of 2, 4, 6, 10, and 20. Figure 7 shows a chart graphing fitness value vs. number of individuals examined for these individual selection methods. Tournament selection, regardless of the tournament size, clearly outperformed fitness-proportionate selection. But the size of the tournament seems to matter little; a tournament size of 6 appears to be optimal, but not by very much.

Crossover rate — Koza suggests that each generation should be constructed from the previous generation using the reproduction operator 10% of the time, and the crossover operator the remaining 90% of the time. However, little evidence is given to support this particular number. To verify these values, we ran several using *M*=300, *G*=30, *Nfc*=3, and using tournament selection (size = 6). Seven trials were run with *Pc* taking on the values 0% (indicating that only reproduction should be used), 10%, 50%, 70%, 80%, 90%, and 100% (indicating no reproduction).

Figure 8 shows fitness value vs. number of individuals examined for various crossover rates. When only reproduction was used, fitness values never improved, as expected (the *Pc*=0% curve appears to indicate that fitness is improving very slightly, but is an artifact of calculating the fitness using the phase value described earlier). A value of *Pc*=10% actually converges surprisingly quickly, though it is clearly not as good as when even higher values of *Pc* are used. The empirical evidence from this experiment indicates that the exact value of *Pc* does not seem to have a great effect on the convergence rate as long as it is at least 50%. A value of *Pc*=80% seemed to be the optimum, but was only slightly better than either 90% or 100%.

Summary: Optimal Run Parameters. In the previous section, when deriving robust locomotion controllers for L*xo, we ran runs of 400 individuals for 50 generations, using tournament selection and 90% crossover as suggested by Koza. The experiments of this chapter demonstrate that optimal fitness convergence is achieved with runs of 250 individuals for 30 generations, with individuals selected by tournaments of size 6, with 80% of individuals produced by crossover. These new parameters allow the derivation of controllers just as fit as the ones described in the previous chapter, using only half the computational resources as when we used naive values for these parameters.

5. Discussion

By treating animated characters as dynamically simulated robots, and using a GP process to generate control programs for those robots, we have successfully generated motion which can adhere to specific goals and constraints. The resulting motion is fluid, physically and biologically believable; obeys many of the important "principles of animation"; and often appears very organic.

We were able to generate robust controllers for more general skills, rather than simple actions. We constructed fitness metrics with several terms, phasing in some of the more constraining terms as the evolution continued, which appears to generate fit individuals with more reliability then with a static fitness function. We also tried to empirically find optimal parameters to the GP process itself, which yielded interesting data for our problem, though it is exceedingly hard to generalize to other problems.

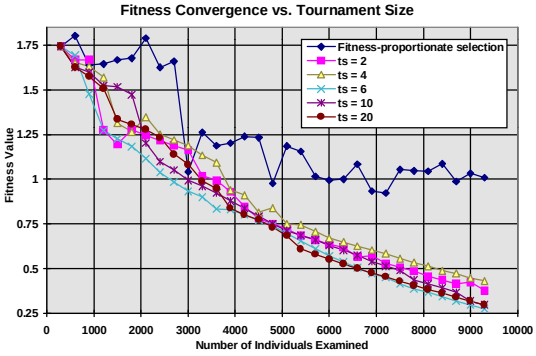


Figure 7: Graph of fitness vs. number of individuals examined for various methods of selecting individuals.

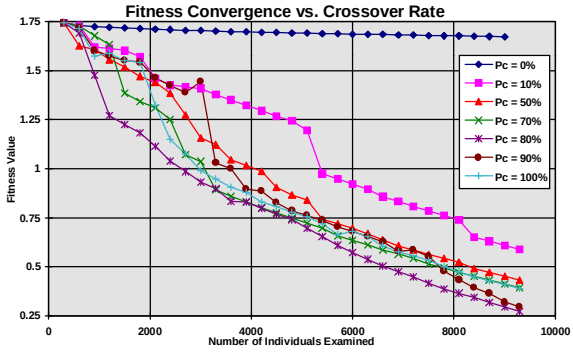


Figure 8: Graph of fitness vs. number of individuals examined for various crossover rates.

The motion resulting from our technique was specified implicitly through the fitness functions, rather than explicitly through key framing. In this case, it was easier to use the implicit method (we assure the reader that neither author is a skilled animator and could never have generated motion this good by key framing). However, the general problem exists that real-world computer animators are familiar and comfortable with the key framing paradigm, and would be reluctant to program fitness functions. We are continuing research on how to combine the two paradigms in ways which leverage both the automation and physical reality of our methods, and the animator-friendly interfaces of more traditional methods.

Bibliography

Armstrong, William W. and Mark W. Green. 1985. The dynamics of articulated rigid bodies for purposes of animation. Visual Computer. Springer-Verlag, 1985, pp. 231-240.

Gritz, Larry and James K. Hahn. 1995. Genetic Programming for Articulated Figure Motion. *Journal of Visualization and Computer Animation*. 6(3): 129-142.

Hahn, James K. 1988. Realistic animation of rigid bodies. *Computer Graphics*, 22(4) (Proceedings of SIGGRAPH '87), pp. 299-308.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Lasseter, John. 1987. Principles of traditional animation applied to 3D computer animation. *Computer Graphics*, 21 (Proceedings of Siggraph '87), pp. 35-44.

Ngo, J. Thomas and Joe Marks. 1993. Spacetime Constraints Revisited. *Computer Graphics Proceedings, Annual Conference Series 1993* (Proceedings of SIGGRAPH '93), pp. 343-350.

Pixar. 1986. *Luxo, Jr.* (computer animated film), 1986.

Sims, Karl. 1991. Artificial evolution for computer graphics. *Computer Graphics*, 25(4) (Proceedings of SIGGRAPH '91), pp. 319-328.

Sims, Karl. 1994. Evolving Virtual Creatures, *Computer Graphics Proceedings, Annual Conference Series 1994* (Proceedings of SIGGRAPH '94), pp. 15-22.

Steele, G. 1984. *Common Lisp, The Language*. Digital Press.

Thomas and Johnston. 1984. *Disney Animation: The Illusion of Life*, New York: Abbeville Press.

van de Panne, Michiel and Eugene Fiume. 1993. Sensor-actuator networks, *Computer Graphics Proceedings, Annual Conference Series 1993* (Proceedings of SIGGRAPH '93), pp. 335-342.

Wilhelms, Jane. 1990. "Dynamics for Computer Graphics: a Tutorial," ACM SIGGRAPH '90 Course notes #8 (Human Figure Animation: Approaches and Applications), pp. 85-115.

Witkin, Andrew and Michael Kass. 1988. Spacetime Constraints, *Computer Graphics*, 22(4) (Proceedings of SIGGRAPH '88), pp. 159-168.