Evolving Virtual Creatures

Karl Sims

Thinking Machines Corporation
245 First Street, Cambridge, MA 02142

Abstract

This paper describes a novel system for creating virtual creatures that move and behave in simulated three-dimensional physical worlds. The morphologies of creatures and the neural systems for controlling their muscle forces are both generated automatically using genetic algorithms. Different fitness evaluation functions are used to direct simulated evolutions towards specific behaviors such as swimming, walking, jumping, and following.

A genetic language is presented that uses nodes and connections as its primitive elements to represent directed graphs, which are used to describe both the morphology and the neural circuitry of these creatures. This genetic language defines a hyperspace containing an indefinite number of possible creatures with behaviors, and when it is searched using optimization techniques, a variety of successful and interesting locomotion strategies emerge, some of which would be difficult to invent or build by design.

1 Introduction

A classic trade-off in the field of computer graphics and animation is that of complexity vs. control. It is often difficult to build interesting or realistic virtual entities and still maintain control over them. Sometimes it is difficult to build a complex virtual world at all, if it is necessary to conceive, design, and assemble each component. An example of this trade-off is that of kinematic control vs. dynamic simulation. If we directly provide the positions and angles for moving objects, we can control each detail of their behavior, but it might be difficult to achieve physically plausible motions. If we instead provide forces and torques and simulate the resulting dynamics, the result will probably look correct, but then it can be very difficult to achieve the desired behavior, especially as the objects we want to control become more complex. Methods have been developed for dynamically controlling specific objects to successfully crawl, walk, or even run [11,12,16], but a new control algorithm must be carefully designed each time a new behavior or morphology is desired.

Optimization techniques offer possibilities for the automatic generation of complexity. The genetic algorithm is a form of artificial evolution, and is a commonly used method for optimization. A Darwinian "survival of the fittest" approach is employed to search for optima in large multidimensional spaces [5,7]. Genetic algorithms permit virtual entities to be created without requiring an understanding of the procedures or parameters used to generate them. The measure of success, or *fitness*, of each individual can be calculated automatically, or it can instead be provided interactively by a user. Interactive evolution allows procedurally generated results to be explored by simply choosing those that are the most aesthetically desirable for each generation [2,18,19,21].

The user sacrifices some control when using these methods, especially when the fitness is procedurally defined. However, the potential gain in automating the creation of complexity can often compensate for this loss of control, and a higher level of user influence is still maintained by the fitness criteria specification.

In several cases, optimization has been used to automatically generate dynamic control systems for given articulated structures: de Garis has evolved weight values for neural networks [4], Ngo and Marks have performed genetic algorithms on stimulus-response pairs [14], and van de Panne and Fiume have optimized sensor-actuator networks [15]. Each of these methods has resulted in successful locomotion of two-dimensional stick figures.

The work presented here is related to these projects, but differs in several respects. In previous work, control systems were generated for fixed structures that were user-designed, but here entire creatures are evolved: the optimization determines the creature morphologies as well as their control systems. Also, here the creatures' bodies are three-dimensional and fully physically based. The three-dimensional physical structure of a creature can adapt to its control system, and vice versa, as they evolve together. The "nervous systems" of creatures are also completely determined by the optimization: the number of internal nodes, the connectivity, and the type of function each neural node performs are included in the genetic description of each creature, and can grow in complexity as an evolution proceeds. Together, these remove the necessity for a user to provide any specific creature information such as shape, size, joint constraints, sensors, actuators, or internal neural parameters. Finally, here a developmental process is used to generate the creatures and their control systems, and allows similar components including their local neural circuitry to be defined once and then replicated, instead of requiring each to be separately specified. This approach is related to L-systems, graftal grammars, and object instancing techniques [6,8,10,13,20].

It is convenient to use the biological terms *genotype* and *phenotype* when discussing artificial evolution. A *genotype* is a coded representation of a possible individual or problem solution. In biological systems, a genotype is usually composed of DNA and contains the instructions for the development of an organism. Genetic algorithms typically use populations of genotypes consisting of strings of binary digits or parameters. These are read to produce *phenotypes* which are then evaluated according to some fitness criteria and selectively reproduced. New genotypes are generated by copying, mutating, and/or combining the genotypes of the most fit individuals, and as the cycle repeats the population should ascend to higher and higher levels of fitness.

Variable length genotypes such as hierarchical Lisp expressions

Published in:
Computer Graphics, Annual Conference Series,
(SIGGRAPH '94 Proceedings), July 1994, pp.15-22.

or other computer programs can be useful in expanding the set of possible results beyond a predefined genetic space of fixed dimensions. Genetic languages such as these allow new parameters and new dimensions to be added to the genetic space as an evolution proceeds, and therefore define rather a *hyperspace* of possible results. This approach has been used to genetically program solutions to a variety of problems [1,9], as well as to explore procedurally generated images and dynamical systems [18,19].

In the spirit of unbounded genetic languages, directed graphs are presented here as an appropriate basis for a grammar that can be used to describe both the morphology and nervous systems of virtual creatures. New features and functions can be added to creatures, or existing ones removed, so the levels of complexity can also evolve.

The next two sections explain how virtual creatures can be represented by directed graphs. The system used for physical simulation is summarized in section 4, and section 5 describes how specific behaviors can be selected. Section 6 explains how evolutions are performed with directed graph genotypes, and finally a range of resulting creatures is shown.

2 Creature Morphology

In this work, the phenotype embodiment of a virtual creature is a hierarchy of articulated three-dimensional rigid parts. The genetic representation of this morphology is a directed graph of nodes and connections. Each graph contains the developmental instructions for growing a creature, and provides a way of reusing instructions to make similar or recursive components within the creature. A phenotype hierarchy of parts is made from a graph by starting at a defined *root-node* and synthesizing parts from the node information while tracing through the connections of the graph. The graph can be recurrent. Nodes can connect to themselves or in cycles to form recursive or fractal like structures. They can also connect to the same child multiple times to make duplicate instances of the same appendage.

Each node in the graph contains information describing a rigid part. The *dimensions* determine the physical shape of the part. A *joint-type* determines the constraints on the relative motion between this part and its parent by defining the number of degrees of freedom of the joint and the movement allowed for each degree of freedom. The different joint-types allowed are: *rigid, revolute, twist, universal, bend-twist, twist-bend*, or *spherical*. *Joint-limits* determine the point beyond which restoring spring forces will be exerted for each degree of freedom. A *recursive-limit* parameter determines how many times this node should generate a phenotype part when in a recursive cycle. A set of local *neurons* is also included in each node, and will be explained further in the next section. Finally, a node contains a set of *connections* to other nodes.

Each connection also contains information. The placement of a child part relative to its parent is decomposed into *position, orientation, scale*, and *reflection*, so each can be mutated independently. The position of attachment is constrained to be on the surface of the parent part. Reflections cause negative scaling, and allow similar but symmetrical sub-trees to be described. A *terminal-only* flag can cause a connection to be applied only when the recursive limit is reached, and permits tail or hand-like components to occur at the end of chains or repeating units.

Figure 1 shows some simple hand-designed graph topologies and resulting phenotype morphologies. Note that the parameters in the nodes and connections such as *recursive-limit* are not shown for the genotype even though they affect the morphology of the phenotype. The nodes are anthropomorphically labeled as "body," "leg," etc. but the genetic descriptions actually have no concept of specific categories of functional components.

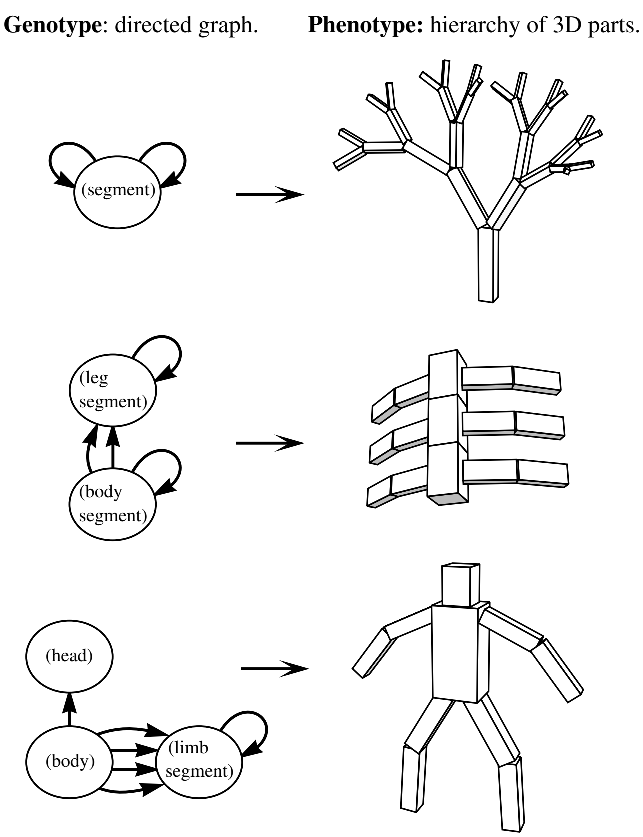

Figure 1: Designed examples of genotype graphs and corresponding creature morphologies.

3 Creature Control

A virtual "brain" determines the behavior of a creature. The brain is a dynamical system that accepts input sensor values and provides output effector values. The output values are applied as forces or torques at the degrees of freedom of the body's joints. This cycle of effects is shown in Figure 2.

Sensor, effector, and internal neuron signals are represented here by continuously variable scalars that may be positive or negative. Allowing negative values permits the implementation of single effectors that can both push and pull. Although this may not be biologically realistic, it simplifies the more natural development of muscle pairs.

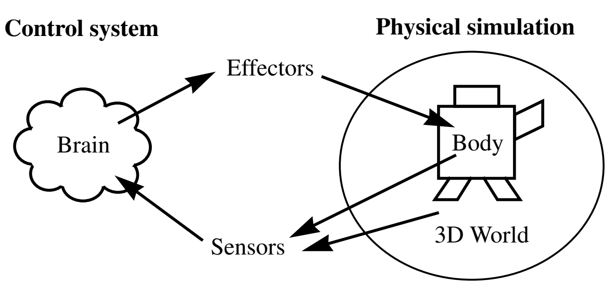

Figure 2: The cycle of effects between brain, body and world.

3.1 Sensors

Each sensor is contained within a specific part of the body, and measures either aspects of that part or aspects of the world relative to that part. Three different types of sensors were used for these experiments:

1. *Joint angle sensors* give the current value for each degree of freedom of each joint.

2. *Contact sensors* activate (1.0) if a contact is made, and negatively activate (-1.0) if not. Each contact sensor has a sensitive region within a part's shape and activates when any contacts occur in that area. In this work, contact sensors are made available for each face of each part. No distinction is made between self-contact and environmental contact.

3. *Photosensors* react to a global light source position. Three photosensor signals provide the coordinates of the normalized light source direction relative to the orientation of the part. This is the same as having pairs of opposing photosensitive surfaces in which the left side negates its response and adds it to the right side for the total response.

Other types of sensors, such as accelerometers, additional proprioceptors, or even sound or smell detectors could also be implemented, but these basic three are enough to allow interesting and adaptive behaviors to occur. The inclusion of the different types of sensors in an evolving virtual brain can be enabled or disabled as appropriate depending on the physical environment and behavior goals. For example, contact sensors are enabled for land environments, and photosensors are enabled for following behaviors.

3.2 Neurons

Internal neural nodes are used to give virtual creatures the possibility of arbitrary behavior. Ideally a creature should be able to have an internal state beyond its sensor values, or be affected by its history.

In this work, different neural nodes can perform diverse functions on their inputs to generate their output signals. Because of this, a creature's brain might resemble a dataflow computer program more than a typical neural network. This approach is probably less biologically realistic than just using sum and threshold functions, but it is hoped that it makes the evolution of interesting behaviors more likely. The set of functions that neural nodes can have is: *sum, product, divide, sum-threshold, greater-than, sign-of, min, max, abs, if, interpolate, sin, cos, atan, log, expt, sigmoid, integrate, differentiate, smooth, memory, oscillate-wave, and oscillate-saw.*

Some functions compute an output directly from their inputs, while others such as the oscillators retain some state and can give time varying outputs even when their inputs are constant. The number of inputs to a neuron depends on its function, and here is at most three. Each input contains a connection to another neuron or a sensor from which to receive a value. Alternatively, an input can simply receive a constant value. The input values are first scaled by weights before being operated on.

For each simulated time interval, every neuron computes its output value from its inputs. In this work, two brain time steps are performed for each dynamic simulation time step so signals can propagate through multiple neurons with less delay.

3.3 Effectors

Each effector simply contains a connection to a neuron or a sensor from which to receive a value. This input value is scaled by a constant weight, and then exerted as a joint force which affects the dynamic simulation and the resulting behavior of the creature. Different types of effectors, such as sound or scent emitters, might also be interesting, but only effectors that exert simulated muscle forces are used here.

Each effector controls a degree of freedom of a joint. The effectors for a given joint connecting two parts, are contained in the part further out in the hierarchy, so that each non-root part operates only a single joint connecting it to its parent. The angle sensors for that joint are also contained in this part.

Each effector is given a *maximum-strength* proportional to the maximum cross sectional area of the two parts it joins. Effector forces are scaled by these strengths and not permitted to exceed them. Since strength scales with area, but mass scales with volume, as in nature, behavior does not always scale uniformly.

3.4 Combining Morphology and Control

The genotype descriptions of virtual brains and the actual phenotype brains are both directed graphs of nodes and connections. The nodes contain the sensors, neurons, and effectors, and the connections define the flow of signals between these nodes. These graphs can also be recurrent, and as a result the final control system can have feedback loops and cycles.

However, most of these neural elements exist within a specific part of the creature. Thus the genotype for the nervous system is a nested graph: the morphological nodes each contain graphs of the neural nodes and connections. Figure 3 shows an example of an evolved nested graph.

When a creature is synthesized from its genetic description, the neural components described within each part are generated along with the morphological structure. This causes blocks of neural control circuitry to be replicated along with each instanced part, so each duplicated segment or appendage of a creature can have a similar but independent local control system.

These local control systems can be connected to enable the possibility of coordinated control. Connections are allowed between adjacent parts in the hierarchy: the neurons and effectors within a part can receive signals from sensors or neurons in their parent part or in their child parts.

Creatures are also given a set of neurons that are not associated

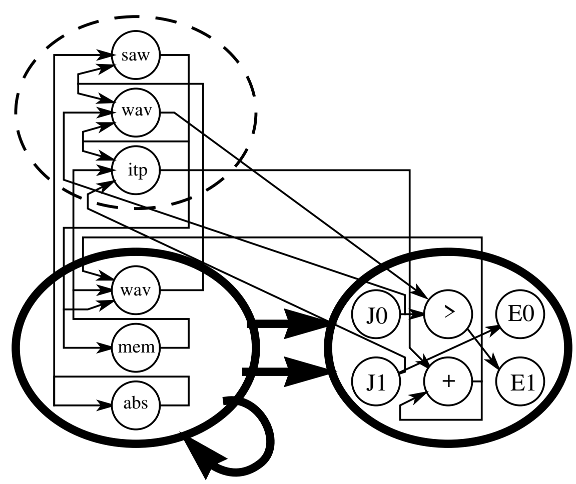

Figure 3: Example evolved nested graph genotype. The outer graph in bold describes a creature's morphology. The inner graph describes its neural circuitry. J0 and J1 are joint angle sensors, and E0 and E1 are effector outputs. The dashed node contains centralized neurons that are not associated with any part.

with a specific part, and are copied only once into the phenotype. This gives the opportunity for the development of global synchronization or centralized control. These neurons can receive signals from each other or from sensors or neurons in specific instances of any of the creature's parts, and the neurons and effectors within the parts can optionally receive signals from these unassociated-neuron outputs.

In this way the genetic language for morphology and control is merged. A local control system is described for each type of part, and these are copied and connected into the hierarchy of the creature's body to make a complete distributed nervous system. Figure 4a shows the creature morphology resulting from the genotype in figure 3. Again, parameters describing shapes, recursive-limits, and weights are not shown for the genotype even though they affect the phenotype. Figure 4b shows the corresponding brain of this creature. The brackets on the left side of figure 4b group the neural components of each part. Some groups have similar neural systems because they are copies from the same genetic description. This creature can swim by making cyclic paddling motions with four similar flippers. Note that it can be difficult to analyze exactly how a control system such as this works, and some components may not actually be used at all. Fortunately, a primary benefit of using artificial evolution is that understanding these representations is not necessary.

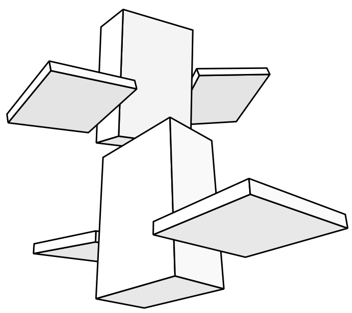

Figure 4a: The phenotype morphology generated from the evolved genotype shown in figure 3.

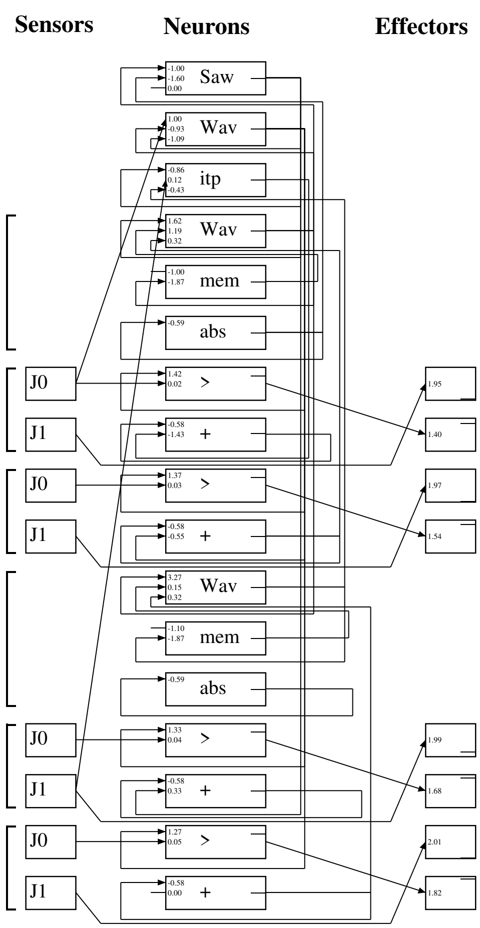

Figure 4b: The phenotype "brain" generated from the evolved genotype shown in figure 3. The effector outputs of this control system cause paddling motions in the four flippers of the morphology above.

4 Physical Simulation

Dynamics simulation is used to calculate the movement of creatures resulting from their interaction with a virtual three-dimensional world. There are several components of the physical simulation used in this work: articulated body dynamics, numerical integration, collision detection, collision response, friction, and an optional viscous fluid effect. These are only briefly summarized here, since physical simulation is not the emphasis of this paper.

Featherstone's recursive O(N) articulated body method is used to calculate the accelerations from the velocities and external forces of each hierarchy of connected rigid parts [3]. Integration determines the resulting motions from these accelerations and is performed by a Runge-Kutta-Fehlberg method which is a fourth order Runge-Kutta with an additional evaluation to estimate the error and adapt the step size. Typically between 1 and 5 integration time steps are performed for each frame of 1/30 second.

The shapes of parts are represented here by simple rectangular solids. Bounding box hierarchies are used to reduce the number of collision tests between parts from $O(N^2)$. Pairs whose world-space bounding boxes intersect are tested for penetrations, and collisions with a ground plane are also tested if one exists. If necessary, the previous time-step is reduced to keep any new penetrations below a certain tolerance. Connected parts are permitted to interpenetrate but not rotate completely through each other. This is achieved by using adjusted shapes when testing for collisions between connected parts. The shape of the smaller part is clipped halfway back from its point of attachment so it can swing freely until its remote end makes contact.

Collision response is accomplished by a hybrid model using both impulses and penalty spring forces. At high velocities, instantaneous impulse forces are used, and at low velocities springs are used, to simulate collisions and contacts with arbitrary elasticity and friction parameters.

A viscosity effect is used for the simulations in underwater environments. For each exposed moving surface, a viscous force resists the normal component of its velocity, proportional to its surface area and normal velocity magnitude. This is a simple approximation that does not include the motion of the fluid itself, but is still sufficient for simulating realistic looking swimming and paddling dynamics.

It is important that the physical simulation be reasonably accurate when optimizing for creatures that can move within it. Any bugs that allow energy leaks from non-conservation, or even round-off errors, will inevitably be discovered and exploited by the evolving creatures. Although this can be a lazy and often amusing approach for debugging a physical modeling system, it is not necessarily the most practical.

5 Behavior Selection

In this work, virtual creatures are evolved by optimizing for a specific task or behavior. A creature is grown from its genetic description as previously explained, and then it is placed in a dynamically simulated virtual world. The brain provides effector forces which move parts of the creature, the sensors report aspects of the world and the creature's body back to the brain, and the resulting physical behavior of the creature is evaluated. After a certain duration of virtual time (perhaps 10 seconds), a *fitness* value is assigned that corresponds to the success level of that behavior. If a creature has a high fitness relative to the rest of the population, it will be selected for survival and reproduction as described in the next section.

Before creatures are simulated for fitness evaluation, some simple viability checks are performed, and inappropriate creatures are removed from the population by giving them zero fitness values. Those that have more than a specified number of parts are removed. A subset of genotypes will generate creatures whose parts initially interpenetrate. A short simulation with collision detection and response attempts to repel any intersecting parts, but those creatures with persistent interpenetrations are also discarded.

Computation can be conserved for most fitness methods by discontinuing the simulations of individuals that are predicted to be unlikely to survive the next generation. The fitness is periodically estimated for each simulation as it proceeds. Those are stopped that are either not moving at all or are doing somewhat worse than the minimum fitness of the previously surviving individuals.

Many different types of fitness measures can be used to perform evolutions of virtual creatures. Four examples of fitness methods are described here.

5.1 Swimming

Physical simulation of a water environment is achieved by turning off gravity and adding the viscous water resistance effect as described. Swimming speed is used as the fitness value and is measured by the distance traveled by the creature's center of mass per unit time. Straight swimming is rewarded over circling by using the maximum distance from the initial center of mass. Continuing movement is rewarded over that from a single initial push, by giving the velocities during the final phase of the test period a stronger relative weight in the total fitness value.

5.2 Walking

The term *walking* is used loosely here to indicate any form of land locomotion. A land environment is simulated by including gravity, turning off the viscous water effect, and adding a static ground plane with friction. Additional inanimate objects can be placed in the world for more complex environments. Again, speed is used as the selection criteria, but the vertical component of velocity is ignored.

For land environments, it can be necessary to prevent creatures from generating high velocities by simply falling over. This is accomplished by first running the simulation with no friction and no effector forces until the height of the center of mass reaches a stable minimum.

5.3 Jumping

Jumping behavior can be selected for by measuring the maximum height above the ground of the lowest part of the creature. An alternative method is to use the average height of the lowest part of the creature during the duration of simulation.

5.4 Following

Another evaluation method is used to select for creatures that can adaptively follow a light source. Photosensors are enabled, so the effector output forces and resulting behavior can depend on the relative direction of a light source to the creature. Several trials are run with the light source in different locations, and the speeds at which a creature moves toward it are averaged for the fitness value. Following behaviors can be evolved for both water and land environments.

Fleeing creatures can also be generated in a similar manner, or following behavior can be transformed into fleeing behavior by simply negating a creature's photo sensor signals.

Once creatures are found that exhibit successful following behaviors, they can be led around in arbitrary paths by movement of the light sources.

6 Creature Evolution

An evolution of virtual creatures is begun by first creating an initial population of genotypes. These initial genotypes can come from several possible sources: new genotypes can be synthesized "from scratch" by random generation of sets of nodes and connections, an existing genotype from a previous evolution can be used to seed the initial population of a new evolution, or a seed genotype can be designed by hand. However, no hand-designed seed genotypes were used in the examples shown here.

A *survival-ratio* determines the percentage of the population that will survive each generation. In this work, population sizes were typically 300, and the survival ratio was 1/5. If the initially generated population has fewer individuals with positive fitness than the number that should survive, another round of seed genotypes is generated to replace those with zero fitness.

For each generation, creatures are grown from their genetic descriptions, and their fitness values are measured by a method such as those described in the previous section. The individuals whose fitnesses fall within the survival percentile are then reproduced, and their offspring fill the slots of those individuals that did not survive. The survivors are kept in the population for the next generation, and the total size of the population is maintained. The number of offspring that each surviving individual generates is proportional to its fitness – the most successful creatures make the most children.

Offspring are generated from the surviving creatures by copying and combining their directed graph genotypes. When these graphs are reproduced they are subjected to probabilistic variation or mutation, so the corresponding phenotypes are similar to their parents but have been altered or adjusted in random ways.

6.1 Mutating Directed Graphs

A directed graph is mutated by the following sequence of steps:

1. The internal parameters of each node are subjected to possible alterations. A mutation frequency for each parameter type determines the probability that a mutation will be applied to it at all. Boolean values are mutated by simply flipping their state. Scalar values are mutated by adding several random numbers to them for a Gaussian-like distribution so small adjustments are more likely than drastic ones. The scale of an adjustment is relative to the original value, so large quantities can be varied more easily and small ones can be carefully tuned. A scalar can also be negated. After a mutation occurs, values are clamped to their legal bounds. Some parameters that only have a limited number of legal values are mutated by simply picking a new value at random from the set

of possibilities.

2. A new random node is added to the graph. A new node normally has no effect on the phenotype unless a connection also mutates a pointer to it. Therefore a new node is always initially added, but then garbage collected later (in step 5) if it does not become connected. This type of mutation allows the complexity of the graph to grow as an evolution proceeds.

3. The parameters of each connection are subjected to possible mutations, in the same way the node parameters were in step 1. With some frequency the connection pointer is moved to point to a different node which is chosen at random.

4. New random connections are added and existing ones are removed. In the case of the neural graphs these operations are not performed because the number of inputs for each element is fixed, but the morphological graphs can have a variable number of connections per node. Each existing node is subject to having a new random connection added to it, and each existing connection is subject to possible removal.

5. Unconnected elements are garbage collected. Connectedness is propagated outwards through the connections of the graph, starting from the root node of the morphology, or from the effector nodes of neural graphs. Although leaving the disconnected nodes for possible reconnection might be advantageous, and is probably biologically analogous, at least the unconnected newly added ones are removed to prevent unnecessary growth in graph size.

Since mutations are performed on a per element basis, genotypes with only a few elements might not receive any mutations, where genotypes with many elements would receive enough mutations that they rarely resemble their parents. This is compensated for by temporarily scaling the mutation frequencies by an amount inversely proportional to the size of the current graph being mutated, such that on the average, at least one mutation occurs in the entire graph.

Mutation of nested directed graphs, as are used here to represent creatures, is performed by first mutating the outer graph and then mutating the inner layer of graphs. The inner graphs are mutated last because legal values for some of their parameters (inter-node neural input sources) can depend on the topology of the outer graph.

6.2 Mating Directed Graphs

Sexual reproduction allows components from more than one parent to be combined into new offspring. This permits features to evolve independently and later be merged into a single individual. Two different methods for mating directed graphs are presented.

The first is a *crossover* operation (see figure 5a). The nodes of two parents are each aligned in a row as they are stored, and the nodes of the first parent are copied to make the child, but one or more crossover points determine when the copying source should switch to the other parent. The connections of a node are copied with it and simply point to the same relative node locations as before. If the copied connections now point out of bounds because of varying node numbers they are randomly reassigned.

A second mating method *grafts* two genotypes together by connecting a node of one parent to a node of another (see figure 5b). The first parent is copied, and one of its connections is chosen at random and adjusted to point to a random node in the second parent. Newly unconnected nodes of the first parent are removed and the newly connected node of the second parent and any of its descendants are appended to the new graph.

A new directed graph can be produced by either of these two mating methods, or asexually by using only mutations. Offspring

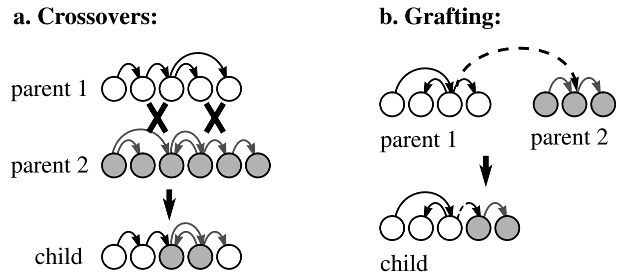

Figure 5: Two methods for mating directed graphs.

from matings are sometimes subjected to mutations afterwards, but with reduced mutation frequencies. In this work a reproduction method is chosen at random for each child to be produced by the surviving individuals using the ratios: 40% asexual, 30% crossovers, and 30% grafting. A second parent is chosen from the survivors if necessary, and a new genotype is produced from the parent or parents.

After a new generation of genotypes is created, a phenotype creature is generated from each, and again their fitness levels are evaluated. As this cycle of variation and selection continues, the population is directed towards creatures with higher and higher fitness.

6.3 Parallel Implementation

This genetic algorithm has been implemented to run in parallel on a Connection Machine® CM-5 in a master/slave message passing model. A single processing node performs the genetic algorithm. It farms out genotypes to the other nodes to be fitness tested, and gathers back the fitness values after they have been determined. The fitness tests each include a dynamics simulation and although most can execute in nearly real-time, they are still the dominant computational requirement of the system. Performing a fitness test per processor is a simple but effective way to parallelize this genetic algorithm, and the overall performance scales quite linearly with the number of processors, as long as the population size is somewhat larger than the number of processors.

Each fitness test takes a different amount of time to compute depending on the complexity of the creature and how it attempts to move. To prevent idle processors from just waiting for others to finish, new generations are started before the fitness tests have been completed for all individuals. Those slower simulations are simply skipped during that reproductive cycle, so all processors remain active. With this approach, an evolution with population size 300, run for 100 generations, might take around three hours to complete on a 32 processor CM-5.

7 Results

Evolutions were performed for each of the behavior selection methods described in section 5. A population of interbreeding creatures often converges toward homogeneity, but each separately run evolution can produce completely different locomotion strategies that satisfy the requested behavior. For this reason, many separate evolutions were performed, each for 50 to 100 generations, and the most successful creatures of each evolution were inspected. A selection of these is shown in figures 6-9. In a few cases, genotypes resulting from one evolution were used as seed genotypes for a second evolution.

The swimming fitness measure produced a large number of simple paddling and tail wagging creatures. A variety of more complex strategies also emerged from some evolutions. A few creatures pulled themselves through the water with specialized sculling appendages. Some used two symmetrical flippers or even large numbers of similar flippers to propel themselves, and several multi-segmented watersnake creatures evolved that wind through the water with sinusoidal motions.

The walking fitness measure also produced a surprising number of simple creatures that could shuffle or hobble along at fairly high speeds. Some walk with lizard-like gaits using the corners of their parts. Some simply wag an appendage in the air to rock back and forth in just the right manner to move forward. A number of more complex creatures emerged that push or pull themselves along, inchworm style. Others use one or more leg-like appendages to successfully crawl or walk. Some hopping creatures even emerged that raise and lower arm-like structures to bound along at fairly high speeds.

The jumping fitness measure did not seem to produce as many different strategies as the swimming and walking optimizations, but a number of simple jumping creatures did emerge.

The light-following fitness measure was used in both water and land environments, and produced a wide variety of creatures that can swim or walk towards a light source. Some consistently and successfully follow the light source at different locations. Others can follow it some of the time, but then at certain relative locations fail to turn towards it. In the water environment, some developed steering fins that turn them towards the light using photosensor inputs. Others adjust the angle of their paddles appropriately as they oscillate along.

Sometimes a user may want to exert more control on the results of this process instead of simply letting creatures evolve entirely automatically. Aesthetic selection is a possible way to achieve this,

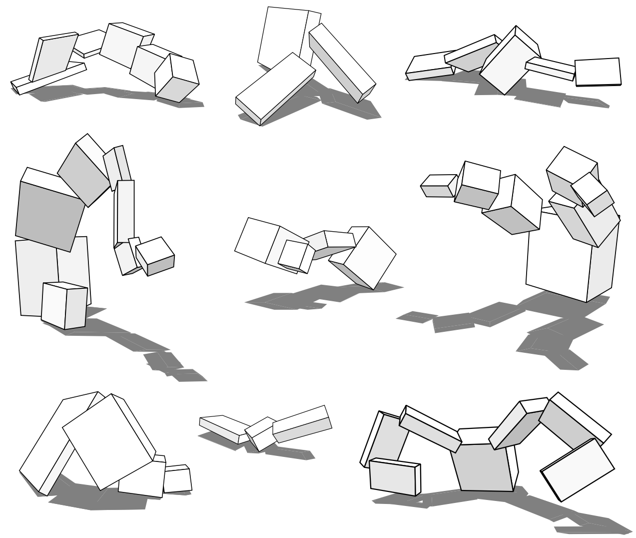

Figure 7: Creatures evolved for walking.

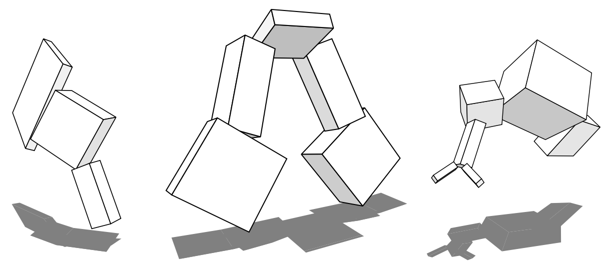

Figure 8: Creatures evolved for jumping.

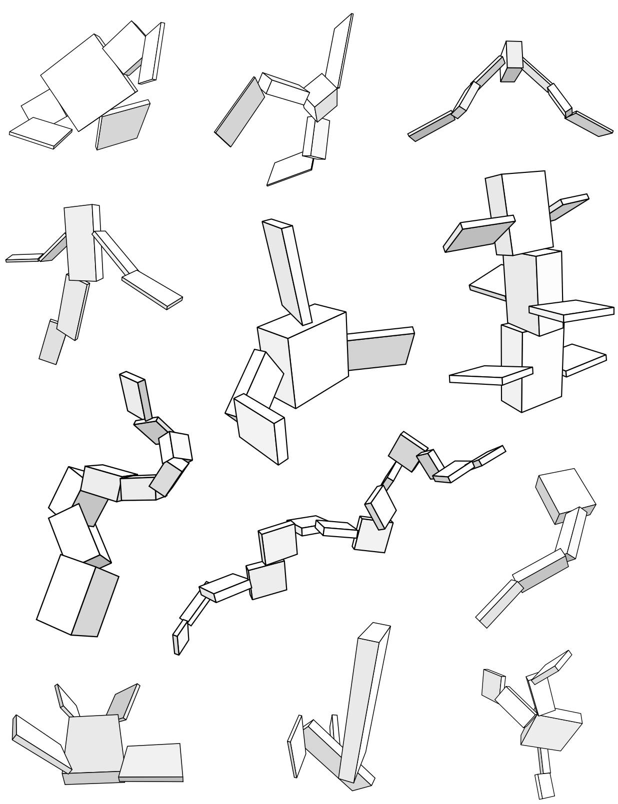

Figure 6: Creatures evolved for swimming.

but observation of the trial simulations of every creature and providing every fitness value interactively would require too much patience on the part of the user. A convenient way of mixing automatic selection with aesthetic selection, is to observe the final successful results of a number of evolutions, and then start new evolutions with those that are aesthetically preferred. Although the control may be limited, this gives the user some influence on the creatures that are developed.

Another method of evolving creatures is to interactively evolve a morphology based on looks only, or alternatively hand design the morphology, and then automatically evolve a brain for that morphology that results in a desirable behavior.

Creatures that evolved in one physical world can be placed in another and evolved further. An evolved watersnake, for example, was placed on land and then evolved to crawl instead of swim.

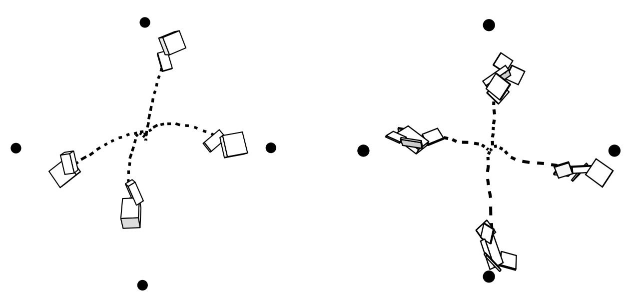

Figure 9: Following behavior. For each creature, four separate trials are shown from the same starting point toward different light source goal locations.

8 Future Work

One direction of future work would be to experiment with additional types of fitness evaluation methods. More complex behaviors might be evolved by defining fitness functions that could measure the level of success at performing more difficult tasks, or even multiple tasks. Fitness could also include the efficiency at which a behavior was achieved. For example, a fitness measure might be the distance traveled divided by the amount of energy consumed to move that distance.

Alternatively, fitness could be defined in a more biologically realistic way by allowing populations of virtual creatures to compete against each other within a physically simulated changing world. Competition has been shown to facilitate complexity, specialization, or even social interactions [17,22]. It becomes difficult to define explicit evaluations that can select for "interesting" behavior, but perhaps systems like these could help produce such results.

Another direction of future work might be to adjust the genetic language of possible creatures to describe only those that could actually be built as real robots. The virtual robots that can best perform a given task in simulation would then be assembled, and would hopefully also perform well in reality.

Much work could be done to dress up these virtual creatures to give them different shapes and improved rendered looks. Flexible skin could surround or be controlled by the rigid components. Various materials could be added such as scales, hair, fur, eyes, or tentacles, and they might flow or bounce using simple local dynamic simulations, even if they did not influence the overall dynamics. The shape details and external materials could also be included in the creatures' genetic descriptions and be determined by artificial evolution.

9 Conclusion

In summary, a system has been described that can generate autonomous three-dimensional virtual creatures without requiring cumbersome user specifications, design efforts, or knowledge of algorithmic details. A genetic language for representing virtual creatures with directed graphs of nodes and connections allows an unlimited hyperspace of possible creatures to be explored. It is believed that these methods have potential as a powerful tool for the creation of desirable complexity for use in virtual worlds and computer animation.

As computers become more powerful, the creation of virtual actors, whether animal, human, or completely unearthly, may be limited mainly by our ability to design them, rather than our ability to satisfy their computational requirements. A control system that someday actually generates "intelligent" behavior might tend to be a complex mess beyond our understanding. Artificial evolution permits the generation of complicated virtual systems without requiring design, and the use of unbounded genetic languages allows evolving systems to increase in complexity beyond our understanding. Perhaps methods such as those presented here will provide a practical pathway toward the creation of intelligent behavior.

Acknowledgments

Thanks to Gary Oberbrunner and Matt Fitzgibbon for Connection Machine and software help. Thanks to Lew Tucker and Thinking Machines Corporation for supporting this research. Thanks to Bruce Blumberg and Peter Schröder for dynamic simulation help and suggestions. And special thanks to Pattie Maes.

References

1. Cramer, N.L., "A Representation for the Adaptive Generation of Simple Sequential Programs," *Proceedings of the First International Conference on Genetic Algorithms*, ed. by J. Grefenstette, 1985, pp.183-187.
2. Dawkins, R., *The Blind Watchmaker*, Harlow Longman, 1986.
3. Featherstone, R., *Robot Dynamics Algorithms*, Kluwer Academic Publishers, Norwell, MA, 1987.
4. de Garis, H., "Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules," *Proceedings of the 7th International Conference on Machine Learning*, 1990, pp.132-139.
5. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
6. Hart, J., "The Object Instancing Paradigm for Linear Fractal Modeling," *Graphics Interface*, 1992, pp.224-231.
7. Holland, J.H., *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan Press, 1975.
8. Kitano, H., "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, Vol.4, pp.461-476, 1990.
9. Koza, J., *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
10. Lindenmayer, A., "Mathematical Models for Cellular Interactions in Development, Parts I and II," *Journal of Theoretical Biology*, Vol.18, 1968, pp.280-315.
11. McKenna, M., and Zeltzer, D., "Dynamic Simulation of Autonomous Legged Locomotion," *Computer Graphics*, Vol.24, No.4, July 1990, pp.29-38.
12. Miller, G., "The Motion Dynamics of Snakes and Worms," *Computer Graphics*, Vol.22, No.4, July 1988, pp.169-178.
13. Mjolsness, E., Sharp, D., and Alpert, B., "Scaling, Machine Learning, and Genetic Neural Nets," *Advances in Applied Mathematics*, Vol.10, pp.137-163, 1989.
14. Ngo, J.T., and Marks, J., "Spacetime Constraints Revisited," *Computer Graphics*, Annual Conference Series, 1993, pp.343-350.
15. van de Panne, M., and Fiume, E., "Sensor-Actuator Networks," *Computer Graphics*, Annual Conference Series, 1993, pp.335-342.
16. Raibert, M., and Hodgins, J.K., "Animation of Dynamic Legged Locomotion," *Computer Graphics*, Vol.25, No.4, July 1991, pp.349-358.
17. Ray, T., "An Approach to the Synthesis of Life," *Artificial Life II*, ed. by Langton, Taylor, Farmer, & Rasmussen, Addison-Wesley, 1991, pp.371-408.
18. Sims, K., "Artificial Evolution for Computer Graphics," *Computer Graphics*, Vol.25, No.4, July 1991, pp.319-328.
19. Sims, K., "Interactive Evolution of Dynamical Systems," *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, ed. by Varela, Francisco, & Bourgine, MIT Press, 1992, pp.171-178.
20. Smith, A.R., "Plants, Fractals, and Formal Languages," *Computer Graphics*, Vol.18, No.3, July 1984, pp.1-10.
21. Todd, S., and Latham, W., *Evolutionary Art and Computers*, London, Academic Press, 1992.
22. Yaeger, L., "Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context," *Artificial Life III*, ed. by C. Langton, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Vol. XVII, Addison-Wesley, 1994, pp.263-298.