Evolving 3D Morphology and Behavior by Competition

Karl Sims
Thinking Machines Corporation
245 First Street
Cambridge MA 02142

Keywords
artificial evolution, coevolution, virtual creatures, evolutionary programming, dynamic simulation, artificial life

Abstract This article describes a system for the evolution and coevolution of virtual creatures that compete in physically simulated three-dimensional worlds. Pairs of individuals enter one-on-one contests in which they contend to gain control of a common resource. The winners receive higher relative fitness scores allowing them to survive and reproduce. Realistic dynamics simulation including gravity, collisions, and friction, restricts the actions to physically plausible behaviors.

The morphology of these creatures and the neural systems for controlling their muscle forces are both genetically determined, and the morphology and behavior can adapt to each other as they evolve simultaneously. The genotypes are structured as directed graphs of nodes and connections, and they can efficiently but flexibly describe instructions for the development of creatures' bodies and control systems with repeating or recursive components. When simulated evolutions are performed with populations of competing creatures, interesting and diverse strategies and counter-strategies emerge.

1 Introduction

Interactions between evolving organisms are generally believed to have a strong influence on their resulting complexity and diversity. In natural evolutionary systems the measure of fitness is not constant: the reproducibility of an organism depends on many environmental factors including other evolving organisms, and is continuously in flux. Competition between organisms is thought to play a significant role in preventing static fitness landscapes and sustaining evolutionary change.

These effects are a distinguishing difference between natural evolution and optimization. Evolution proceeds with no explicit goal, but optimization, including the genetic algorithm, usually aims to search for individuals with the highest possible fitness values where the fitness measure has been predefined, remains constant, and depends only on the individual being tested.

The work presented here takes the former approach. The fitness of an individual is highly dependent on the specific behaviors of other individuals currently in the population. The hope is that virtual creatures with higher complexity and more interesting behavior will evolve than when applying the selection pressures of optimization alone.

Many simulations of coevolving populations have been performed that involve competing individuals [1, 2]. As examples, Lindgren has studied the evolutionary dynamics of competing game strategy rules [13], Hillis has demonstrated that coevolving parasites can enhance evolutionary optimization [8], and Reynolds evolves vehicles for

competition in the game of tag [18]. The work presented here involves similar evolutionary dynamics to help achieve interesting results when phenotypes have three-dimensional bodies and compete in physically simulated worlds.

In several cases, optimization has been used to automatically generate dynamic control systems for given two-dimensional articulated structures: de Garis has evolved weight values for neural networks [5], Ngo and Marks have applied genetic algorithms to generate stimulus-response pairs [15], and van de Panne and Fiume have optimized sensor-actuator networks [16]. 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. Previously, control systems were generated for fixed structures that were user designed, but here entire creatures are evolved: The evolution determines the creature morphologies as well as their control systems. The physical structure of a creature can adapt to its control system, and vice versa, as they evolve together. Also, here the creatures' bodies are three-dimensional and fully physically based. In addition, 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 [7, 10, 12, 14, 22]. Finally, the previous work on articulated structures relies only on optimization, and competitions between individuals were not considered.

A different version of the system described here has also been used to generate virtual creatures by optimizing for specific defined behaviors such as swimming, walking, and following [21].

Genotypes used in simulated evolutions and genetic algorithms have traditionally consisted of strings of binary digits [6, 9]. Variable length genotypes such as hierarchical Lisp expressions 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 program genetically solutions to a variety of problems [3, 11], as well as to explore procedurally generated images and dynamical systems [19, 20].

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 neural systems of virtual creatures. The level of complexity is variable for both genotype and phenotype. New features and functions can be added to creatures or existing ones removed, as they evolve.

The next section of this article describes the environment of the simulated contest and how the competitors are scored. Section 3 discusses different simplified competition patterns for approximating competitive environments. Sections 4 and 5 present the genetic language that is used to represent creatures with arbitrary structure and behavior, and section 6 summarizes the physical simulation techniques used. Section 7 discusses the evolutionary simulations including the methods used for mutating and mating directed graph genotypes, and finally sections 8 and 9 provide results, discussion, and suggestions for future work.

2 The Contest

Figure 1 shows the arena in which two virtual creatures will compete to gain control of a single cube. The cube is placed in the center of the world, and the creatures start on opposite sides of the cube. The second contestant is initially turned by 180° so the

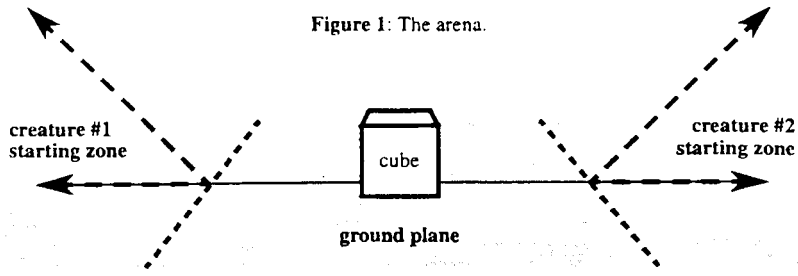

Figure 1. The arena.

relative position of the cube to the creature is consistent from contest to contest no matter which starting side it is assigned. Each creature starts on the ground and behind a diagonal plane slanting up and away from the cube. Creatures are wedged into these "starting zones" until they contact both the ground plane and the diagonal plane, so taller creatures must start further back. This helps prevent the inelegant strategy of simply falling over onto the cube. Strategies like this that utilize only potential energy are further discouraged by relaxing a creature's body before it is placed in the starting zone. The effect of gravity is simulated until the creature reaches a stable minimum state.

At the start of the contest the creatures' nervous systems are activated, and a physical simulation of the creatures' bodies, the cube, and the ground plane begins. The winner is the creature that has the most control over the cube after a certain duration of simulated time (8 seconds were given). Instead of just defining a winner and loser, the margin of victory is determined in the form of a relative fitness value, so there is selection pressure not just to win, but to win by the largest possible margin.

The creatures' final distances to the cube are used to calculate their fitness scores. The shortest distance from any point on the surface of a creatures's parts to the center of the cube is used as its distance value. A creature gets a higher score by being closer to the cube, but also gets a higher score when its opponent is further away. This encourages creatures to reach the cube, but also gives points for keeping the opponent away from it. If d_1 and d_2 are the final shortest distances of each creature to the cube, then the fitnesses for each creature, f_1 and f_2, are given by:

$$f_1 = 1.0 + \frac{d_2 - d_1}{d_1 + d_2}$$

$$f_2 = 1.0 + \frac{d_1 - d_2}{d_1 + d_2}$$

This formulation puts all fitness values in the limited range of 0.0 to 2.0. If the two distances are equal the contestants receive tie scores of 1.0 each, and in all cases the scores will average 1.0.

Credit is also given for having "control" over the cube, beyond just as measured by the minimum distance to it. If both creatures end up contacting the cube, the winner is the one that surrounds it the most. This is approximated by further decreasing the distance value, as used above, when a creature is touching the cube on the side that opposes its center of mass. Because the initial distances are measured from the center of the cube they can be adjusted in this way and still remain positive.

During the simulated contest, if neither creature shows any movement for a full second, the simulation is stopped and the scores are evaluated early to save unnecessary computation.

3 Approximating Competitive Environments

There are many trade-offs to consider when simulating an evolution in which fitness is determined by discrete competitions between individuals. In this work, pairs of individuals compete one-on-one. At every generation of a simulated evolution the individuals in the population are paired up by some pattern, and a number of competitions are performed to determine eventually a fitness value for every individual. The simulations of the competitions are by far the dominant computational requirement of the process, so the total number of competitions performed for each generation and the effectiveness of the pattern of competitions are important considerations.

In one extreme, each individual competes with all the others in the population and the average score determines the fitness (Figure 2a). However, this requires $(N^2 - N)/2$ total competitions for a single-species population of N individuals. For large populations this is often unacceptable, especially if the competition time is significant, as it is in this work.

In the other extreme, each individual competes with just a single opponent (Figure 2b). This requires only $N/2$ total competitions, but can cause inconsistency in the fitness values because each fitness is often highly dependent on the specific individual that happens to be assigned as the opponent. If the pairing is done at random, and especially if the mutation rate is high, fitness can be more dependent on the luck of receiving a poor opponent than on an individual's actual ability.

One compromise between these extremes is for each individual to compete against several opponents chosen at random for each generation. This can somewhat dilute the fitness inconsistency problem, but at the expense of more competition simulations.

A second compromise is a tournament pattern (Figure 2c), which can efficiently determine a single overall winner with $N - 1$ competitions. But this also does not necessarily give all individuals fair scores because of the random initial opponent assignments. Also, this pattern does not easily apply to multi-species evolutions where competitions are not performed between individuals within the same species.

A third compromise is for each individual to compete once per generation, but all against the same opponent. The individual with the highest fitness from the previous generation is chosen as this one-to-beat (Figure 2d). This also requires $N - 1$ competitions per generation, but effectively gives fair relative fitness values because all are playing against the same opponent, which has proven to be competent. Various interesting instabilities can still occur over generations, however, because the strategy of the "best" individual can change suddenly between generations.

The number of species in the population is another element to consider when simulating evolutions involving competition. A species may be described as an interbreeding subset of individuals in the population. In single-species evolutions individuals will compete against their relatives, but in multi-species evolutions individuals can optionally compete only against individuals from other species. Figure 2 shows graphical representations of some of the different competition patterns described above for both one and two species.

The resulting effects of using these different competition patterns is unfortunately difficult to quantify in this work, because by its nature a simple overall measure of success is absent. Evolutions were performed using several of the methods described above with both one and two species, and the results were subjectively judged. The most "interesting" results occurred when the "all vs. best" competition pattern was used. Both one- and two-species evolutions produced some intriguing strategies, but the multi-species simulations tended to produce more interesting interactions between the evolving creatures.

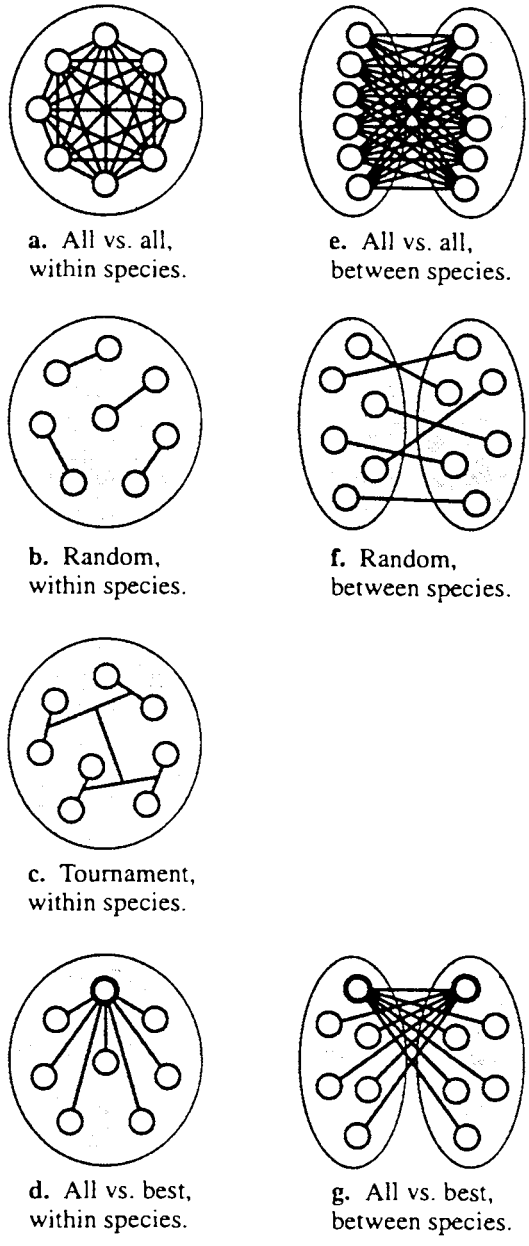

Figure 2. Different pair-wise competition patterns for one and two species. The gray areas represent species of interbreeding individuals, and lines indicate competitions performed between individuals.

4 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 section 5.2. 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 by symmetrical subtrees 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 3 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 segment, and so forth, but the genetic descriptions actually have no concept of specific categories of functional components.

5 Creature Behavior

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 4.

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.

5.1 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.

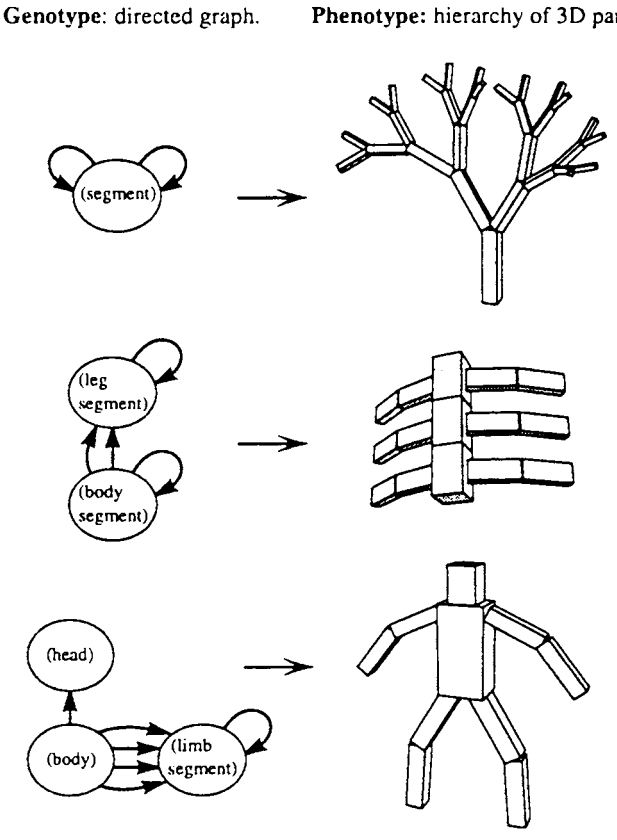

Figure 3. Designed examples of genotype graphs and corresponding creature morphologies.

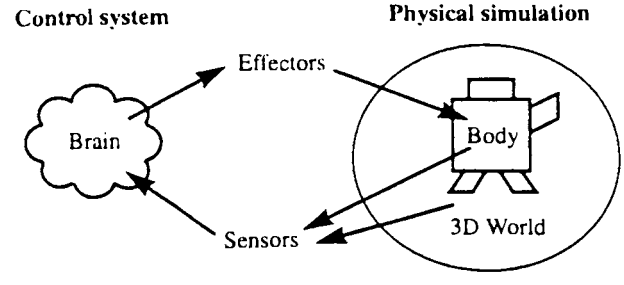

Figure 4. Cycle of effects between brain, body, and world.

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. Shadows are not simulated, so photosensors continue to sense a light source even if it is blocked. Photosensors for two independent colors are made available. The source of one color is located in the desirable cube, and the other is located at the center of mass of the opponent. This effectively allows evolving nervous systems to incorporate specific "cube sensors" and "opponent sensors."

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 some interesting and adaptive behaviors to occur.

5.2 Neurons

Internal neural nodes are used to give virtual creatures the possibility of arbitrary behavior. They allow a creature to have an internal state beyond its sensor values, and 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 artificial 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. The genetic parameters for each neural node include these weights as well as the function type and the connection information.

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.

5.3 Effectors

Each effector simply contains a connection from 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 that 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 nonroot 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. This is similar to the strength limits of natural muscles. As in nature, mass scales with volume but strength scales with area, so behavior does not always scale uniformly.

5.4 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

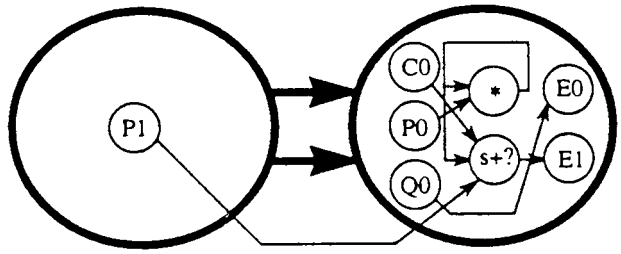

Figure 5. Example of evolved nested-graph genotype. The outer graph in bold describes a creature's morphology. The inner graph describes its neural circuitry. C0, P0, P1, and Q0 are contact and photosensors, E0 and E1 are effector outputs, and those labeled "*" and "s+?" are neural nodes that perform *product* and *sum-threshold* functions.

each contain graphs of the neural nodes and connections. Figure 5 shows an example of an evolved nested graph that describes a simple three-part creature as shown in Figure 6.

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 allowed a set of neurons that are not associated 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 creatures'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 6a shows the creature morphology resulting from the genotype in Figure 5. Again, parameters describing shapes and weight values are not shown for the genotype even though they affect the phenotype. Figure 6b shows the corresponding brain of this creature. The brackets on the left side of Figure 6b group the neural components of each part. Two groups have similar neural systems because they were synthesized from the same genetic description. This creature can roll over the ground by making cyclic tumbling motions with its two arm-like appendages. 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.

6 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

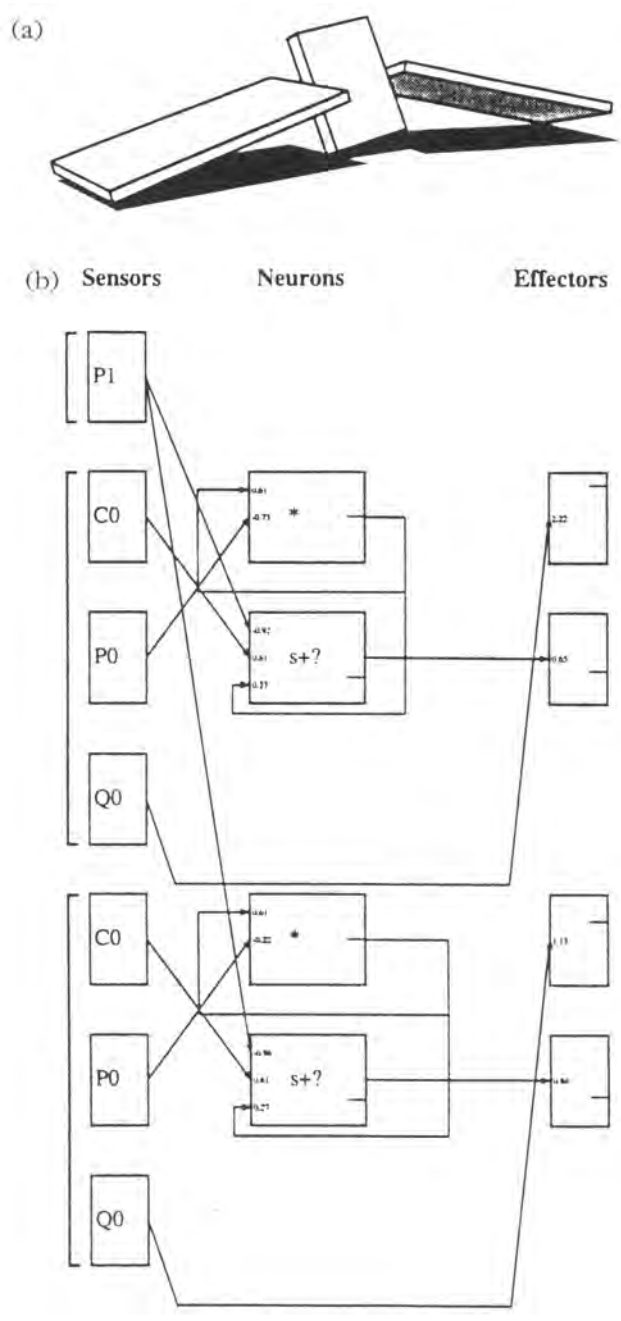

Figure 6. (a) The phenotype morphology generated from the evolved genotype shown in Figure 5. (b) The phenotype "brain" generated from the evolved genotype shown in Figure 5. The effector outputs of this control system cause the morphology in part (a) to roll forward in tumbling motions.

integration, collision detection, and collision response with friction. These are only briefly summarized here, because physical simulation is not the emphasis of this article.

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 [4]. 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 necessary, the previous time step is reduced to keep any new penetration depths 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 responds 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.

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.

7 Creature Evolution

An evolution of virtual creatures is begun by first creating an initial population of genotypes. Seed genotypes are synthesized "from scratch" by random generation of sets of nodes and connections. Alternatively, an existing genotype from a previous evolution can be used to seed an initial population.

Before creatures are paired off for competitions and 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.

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 genotypes, and their fitness values are measured by simulating one or more competititons with other individuals as described. 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 number of offspring that each surviving individual generates is proportional to its fitness. The survivors are kept in the population for the next generation, and the total size of the population is maintained. In multi-species evolutions, each subpopulation is independently treated in this way so the number of individuals in each species remains constant and species do not die out.

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.

7.1 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 is 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 that is chosen at random.
4. New random connections may be added and existing ones may be 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, and from the effector nodes of the 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.

Because mutations are performed on a per element basis, genotyes with only a few elements might not receive any mutations, where genotypes with many elements would receive enough mutations that they would rarely resemble their parents. This is compensated for by 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 (internode neural input sources) can depend on the topology of the outer graph.

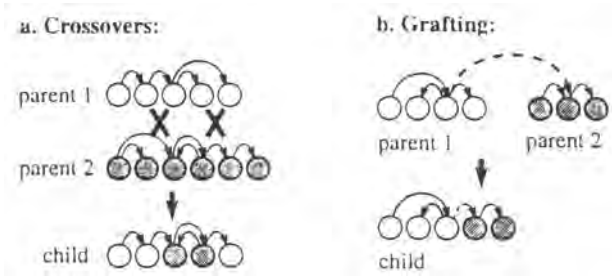

Figure 7. Two methods for mating directed graphs.

7.2 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 used in this work.

The first is a *crossover* operation (Figure 7a). 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 (Figure 7b). 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 from matings are sometimes subjected to mutations afterward, 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 values are evaluated. As this cycle of variation and selection continues, the population is directed toward creatures with higher fitness.

7.3 Parallel Implementation

This process has been implemented to run in parallel on a Connection Machine® CM-5 in a master/slave message passing model. A single processing mode contains the population and performs all the selection and reproduction operations. It farms out pairs of 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 for the competition and although many 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 process, 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 creatures and how they attempt to move. To prevent idle processors from just waiting for others to finish, the slowest few simulations at the end of a generation are suspended and those individuals are removed from the population by giving them zero fitness. With this approach, an evolution with population size 300, run for 100 generations, might take about four hours to complete on a 32 processor CM-5.

8 Results and Discussion

Many independent evolutions were performed using the all vs. best competition pattern as described in section 3. Some single-species evolutions were performed in which all individuals both compete and breed with each other, but most included two species where individuals only compete with members of the opponent species.

Some examples of resulting two-species evolutionary dynamics are shown in Figure 8. The relative fitness of the best individuals of each species are plotted over 100 generations. The rate of evolutionary progress varied widely in different runs. Some species took many generations before they could even reach the cube at all, while others discovered a fairly successful strategy in the first 10 or 20 generations. Figure 8c shows an example where one species was successful fairly quickly and the other species never evolved an effective strategy to challenge it. The other three graphs in Figure 8 show evolutions where more interactions occurred between the evolving species.

A variety of methods for reaching the cube were discovered. Some extended arms out onto the cube, and some reached out while falling forward to land on top of it. Others could crawl inchworm style or roll toward the cube, and a few even developed leg-like appendages that they used to walk toward it.

The most interesting results often occurred when both species discovered methods for reaching the cube and then further evolved strategies to counter the opponent's behavior. Some creatures pushed their opponent away from the cube, some moved the cube away from its initial location and then followed it, and others simply covered up the cube to block the opponent's access. Some counter-strategies took advantage of a specific weakness in the original strategy and could be easily foiled in a few generations by a minor adaptation to the original strategy. Others permanently defeated the original strategy and required the first species to evolve another level of counter-counter-strategy to regain the lead. In some evolutions the winners alternated between species many times with new strategies and counter-strategies. In other runs one species kept a consistent lead with the other species only providing temporary challenges.

After the results from many simulations were observed, the best were collected and then played against each other in additional competitions. The different strategies were compared, and the behavior and adaptibility of creatures were observed as they faced new types of opponents that were not encountered during their evolutions. A few evolutions were also performed starting with an existing creature as a seed genotype for each species so they could further evolve to compete against a new type of opponent.

Figure 9 shows some examples of evolved competing creatures and demonstrates the diversity of the different strategies that emerged. Some of the behaviors and interactions of these specific creatures are described briefly here. The larger creature in Figure 9b nudges the cube aside and then pins down his smaller opponent. The crab-like creature in 9c can successfully walk forward, but then continues blindly past the cube and over the opponent. Figure 9d shows a creature that has just pushed its opponent away from the cube, and the arm-like creature in 9e also jabs at its opponent before curling around the cube.

Most creatures perform similar behavior independently of the opponent's actions,

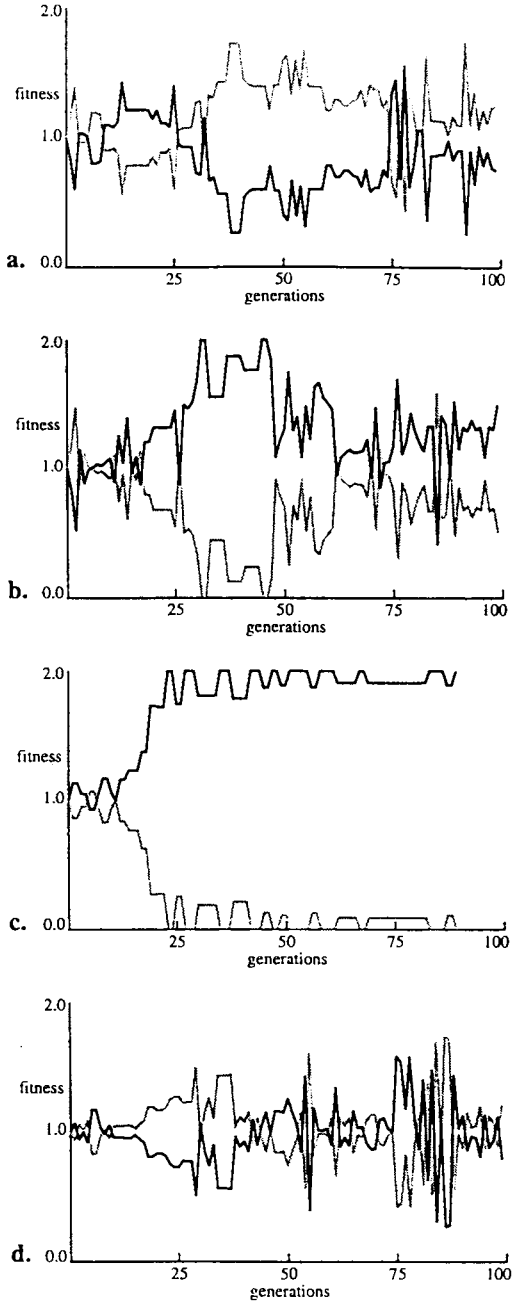

Figure 8. Relative fitness between two coevolving and competing species, from four independent simulations.

but a few are adaptive in that they can reach toward the cube wherever it moves. For example the arm-like creature in Figure 9f pushes the cube aside and then uses photosensors to follow it adaptively. If its opponent moves the cube in a different direction it will successfully grope toward the new location.

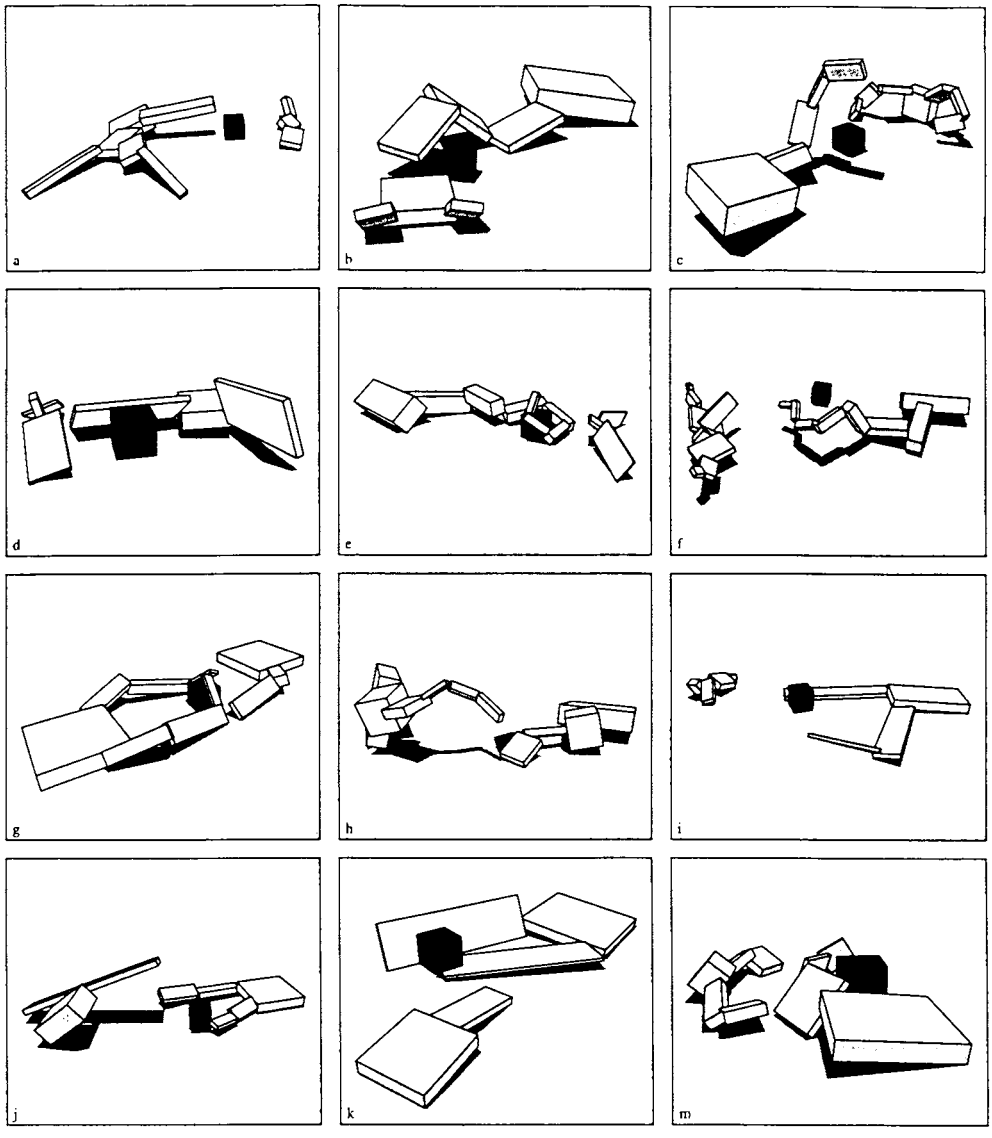

Figure 9. Evolved competing creatures.

The two-armed creature in Figure 9g blocks access to the cube by covering it up. Several other two-armed creatures in 9i, 9j, and 9k use the strategy of batting the cube to the side with one arm and catching it with the other arm. This seemed to be the most successful strategy of the creatures in this group, and the one in 9k was actually the overall winner because it could whisk the cube aside very quickly. However, it was a near tie between this and the photosensitive arm in 9f. The larger creature in 9m wins by a large margin against some opponents because it can literally walk away with the cube, but it does not initially reach the cube very quickly and tends to lose against faster opponents.

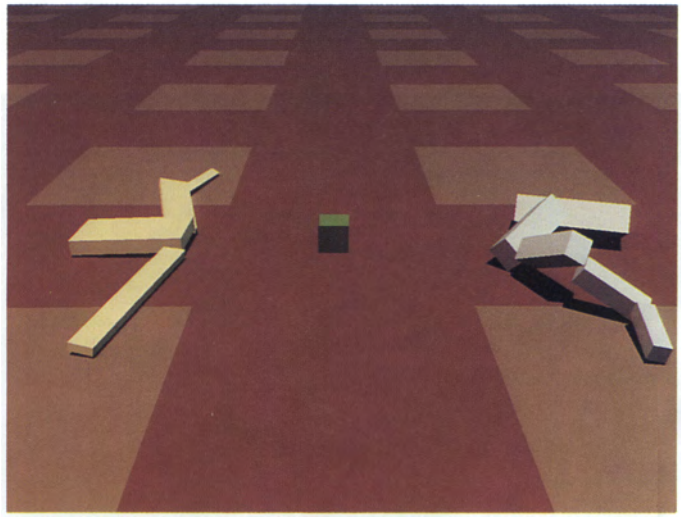

Plate 1. Start.

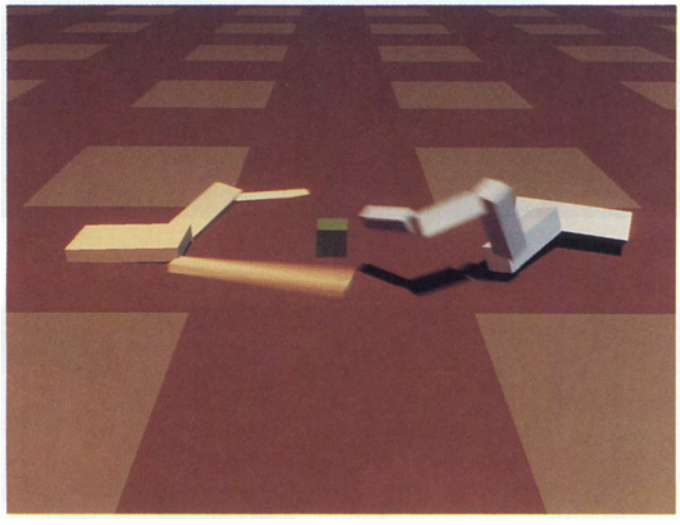

Plate 2. Action.

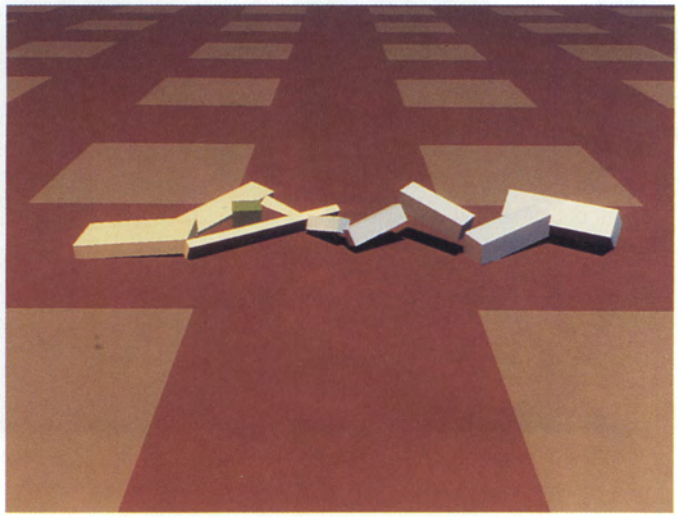

Plate 3. Finish.

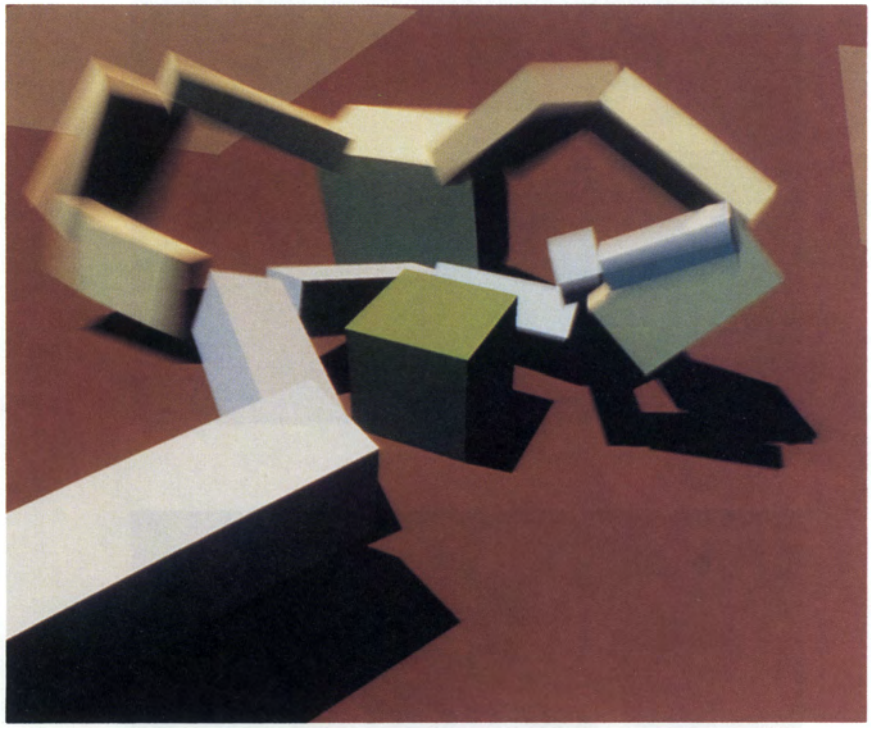

Plate 4.

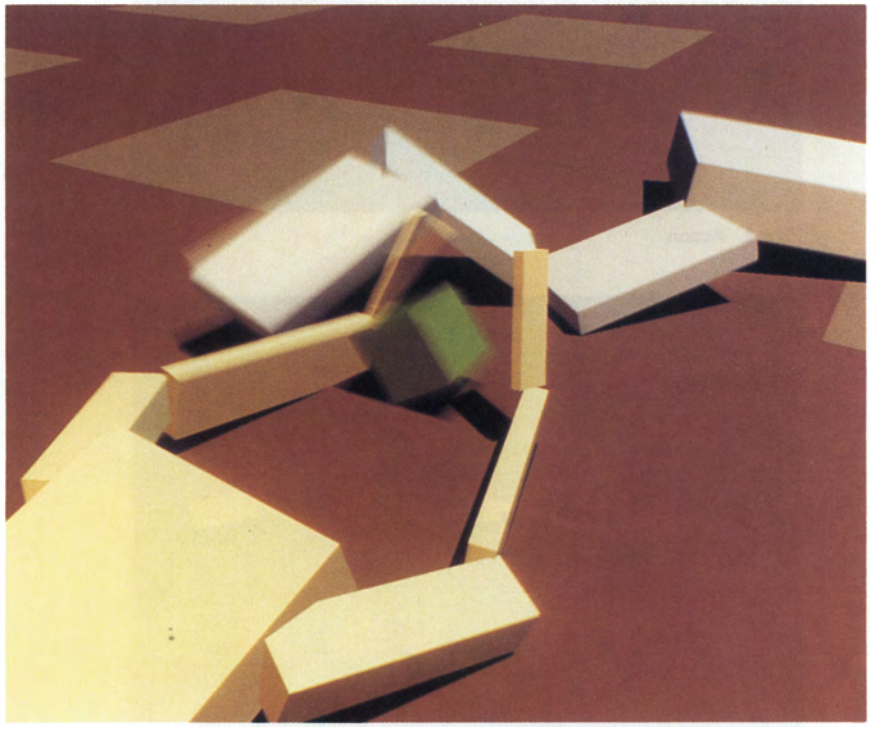

Plate 5.

It is possible that adaptation on an evolutionary scale occurred more easily than the evolution of individuals that were themselves adaptive. Perhaps individuals with adaptive behavior would be significantly more rewarded if evolutions were performed with many species instead of just one or two. To be successful, a single individual would then need to defeat a larger number of different opposing strategies.

9 Future Work

Several variations on this system could be worth further experimentation. Other types of contests could be defined in which creatures compete in different environments and different rules determine the winners. Creatures might also be rewarded for cooperative behavior as well as competitive, and teams of interacting creatures could be simulated.

Evolutions containing larger numbers of species should certainly be performed, with the hope of increasing the chances for emergence of more adaptive individuals as hypothesized above.

An additional extension to this work would be to simulate a more complex but more realistic environment in which many creatures simultaneously compete and/or cooperate with each other, instead of pairing off in one-on-one contests. Speciation, mating patterns, competing patterns, and even offspring production could all be determined by one long ecological simulation. Experiments like this have been performed with simpler organisms and have produced interesting results including specialization and various social interactions [17, 23].

Perhaps the techniques presented here should be considered as an approach toward creating artificial intelligence. When a genetic language allows virtual entities to evolve with increasing complexity, it is common for the resulting system to be difficult to understand in detail. In many cases it would also be difficult to design a similar system using traditional methods. Techniques such as these have the potential of surpassing those limits that are often imposed when human understanding and design is required. The examples presented here suggest that it might be easier to evolve virtual entities exhibiting intelligent behavior than it would be for humans to design and build them.

10 Conclusion

In summary, a system has been described that can automatically generate autonomous three-dimensional virtual creatures that exhibit diverse competitive strategies in physically simulated worlds. A genetic language that uses directed graphs to describe both morphology and behavior defines an unlimited hyperspace of possible results, and a variety of interesting virtual creatures have been shown to emerge when this hyperspace is explored by populations of evolving and competing individuals.

Acknowledgments

Thanks to Gary Oberbrunner and Matt Fitzgibbon for Connection Machine and software support. Thanks to Thinking Machines Corporation and Lew Tucker 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. Angeline, P. J., Pollack, J. B. (1993). Competitive environments evolve better solutions for complex tasks, In S. Forrest, (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms.* (pp. 264–270), Morgan Kaufmann.

2. Axelrod, R. (1989). Evolution of strategies in the iterated prisoner's dilemma, In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann.

3. Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms* (pp. 183–187).

4. Featherstone, R. (1987). *Robot dynamics algorithms.* Norwell, MA: Kluwer Academic Publishers.

5. de Garis, H. (1990). Genetic programming: Building artificial nervous systems using genetically programmed neural network modules. *Proceedings of the 7th International Conference on Machine Learning* (pp. 132–139).

6. Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley.

7. Hart, J. (1992). The object instancing paradigm for linear fractal modeling, In *Graphics Interface* (pp. 224–231).

8. Hillis, W. D. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, (Eds.), *Artificial Life II,* (pp. 313–384). Addison-Wesley.

9. Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

10. Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems, 4,* 461–476.

11. Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection,* Cambridge, MA: MIT Press.

12. Lindenmayer, A. (1968). Mathematical models for cellular interactions in development: Parts I and II. *Journal of Theoretical Biology, 18,* 280–315.

13. Lindgren, K. (1991). Evolutionary phenomena in simple dynamics, In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen, (Eds.), *Artificial Life II,* (pp. 295–312). Addison-Wesley.

14. Mjolsness, E., Sharp, D., & Alpert, B. (1989). Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics, 10,* 137–163.

15. Ngo, J. T., & Marks, J. (1993). Spacetime constraints revisited. *Computer Graphics,* Annual Conference Series, (pp. 343–350). New York: ACM/SIGGRAPH.

16. van de Panne, M., & Fiume, E. (1993). Sensor-actuator networks. *Computer Graphics,* Annual Conference Series, (pp. 335–342). New York: ACM/SIGGRAPH.

17. Ray, T. (1991). An approach to the synthesis of life, In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), *Artificial Life II,* (pp. 371–408). Addison-Wesley.

18. Reynolds, C. (1994). Competition, coevolution and the game of tag, In R. Brooks & P. Maes, (Eds.), *Artificial Life IV Proceedings,* (pp. 59–69). Cambridge, MA: MIT Press.

19. Sims, K. (1991). Artificial evolution for computer graphics. *Computer Graphics, 25,* 319–328.

20. Sims, K. (1992). Interactive evolution of dynamical systems, In Varela, Francisco, & Bourgine, (Eds.), *Toward a practice of autonomous systems: Proceedings of the first European conference on artificial life.* (pp. 171–178). Cambridge, MA: MIT Press.

21. Sims, K. (1994). Evolving virtual creatures. *Computer Graphics,* Annual Conference Series, (pp. 15–22). New York: ACM/SIGGRAPH.

22. Smith, A. R. (1984). Plants, fractals, and formal languages. *Computer Graphics, 18,* 1–10.

23. Yaeger, L. (1994). Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or PolyWorld: Life in a new context. In C. Langton, (Ed.), *Artificial Life III,* Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Vol. XVII, (pp. 263–298). Addison-Wesley.