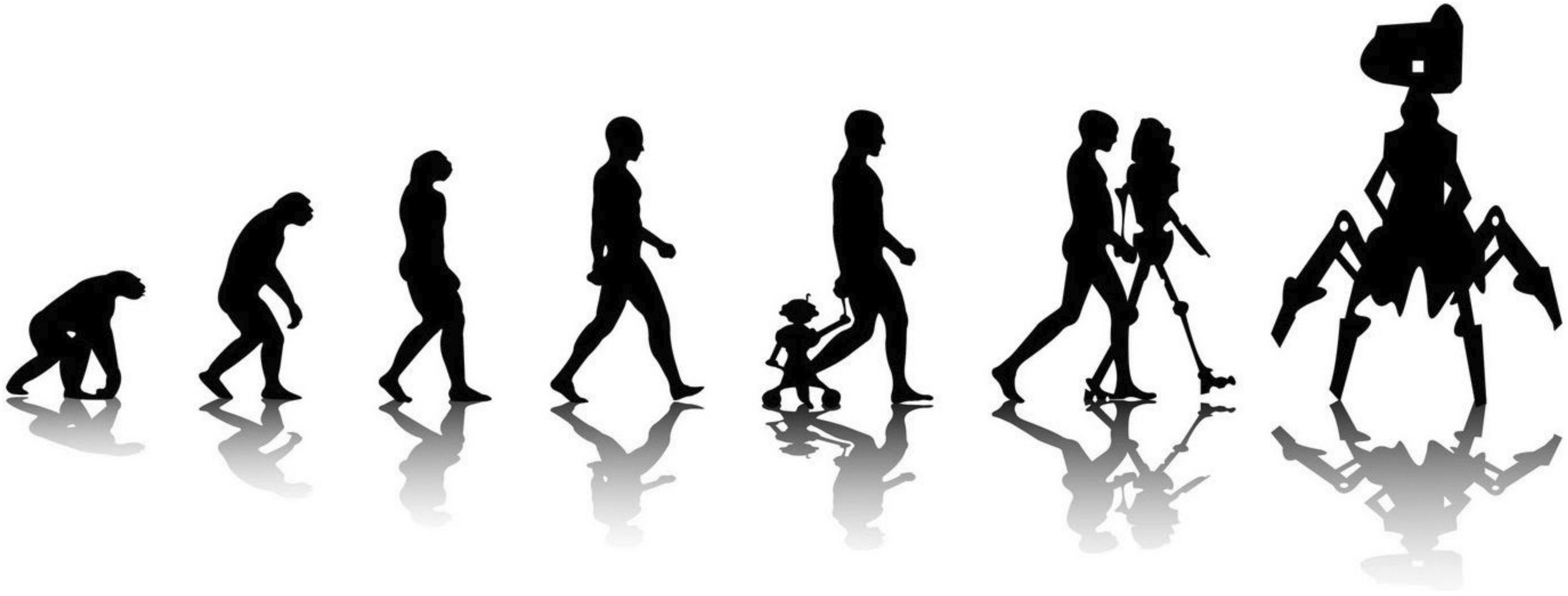


Modern Robotics: Evolutionary Robotics

COSC 4560 / COSC 5560

Professor Cheney

2/12/18

No class on Feb 26 or March 30

Evolving Virtual Creatures

Karl Sims

Thinking Machines Corporation

Evolved Virtual Creatures

Karl Sims

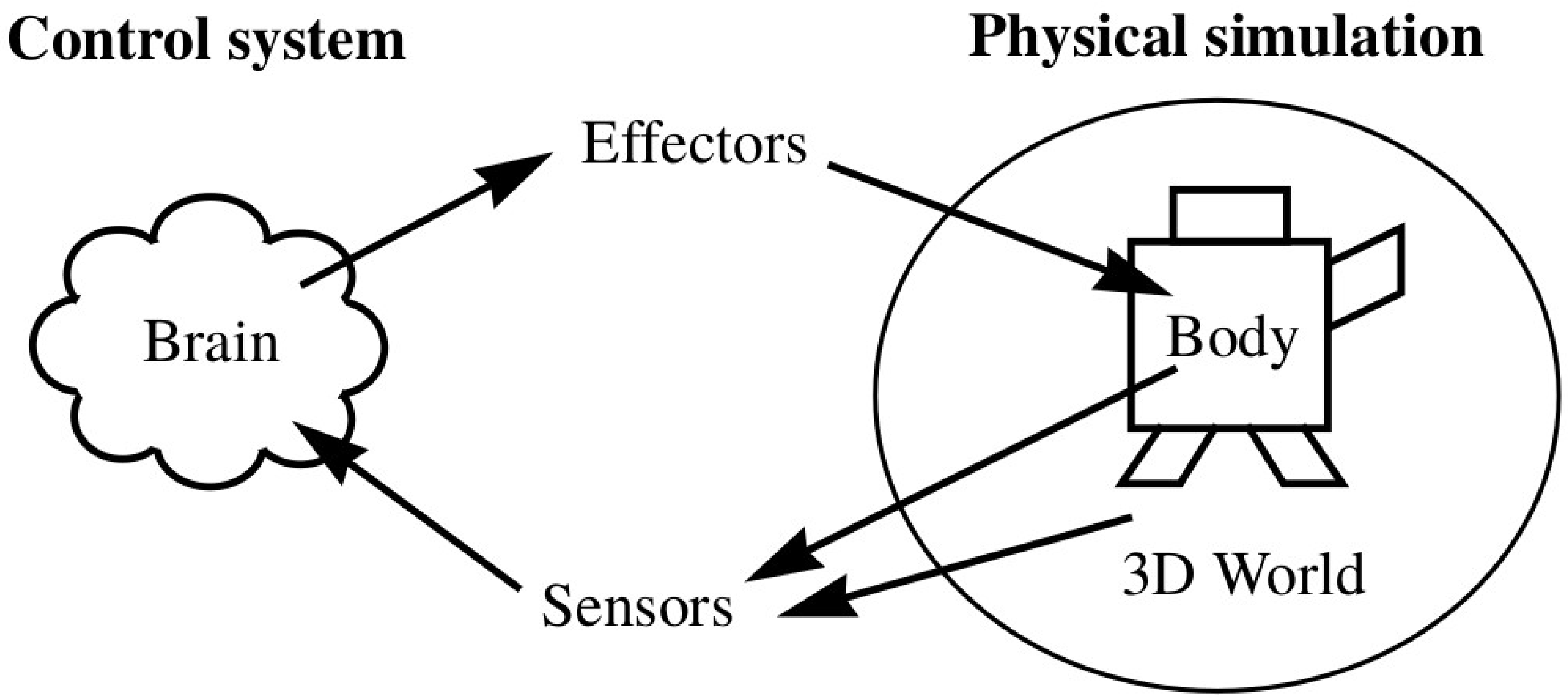


Figure 2: The cycle of effects between brain, body and world.

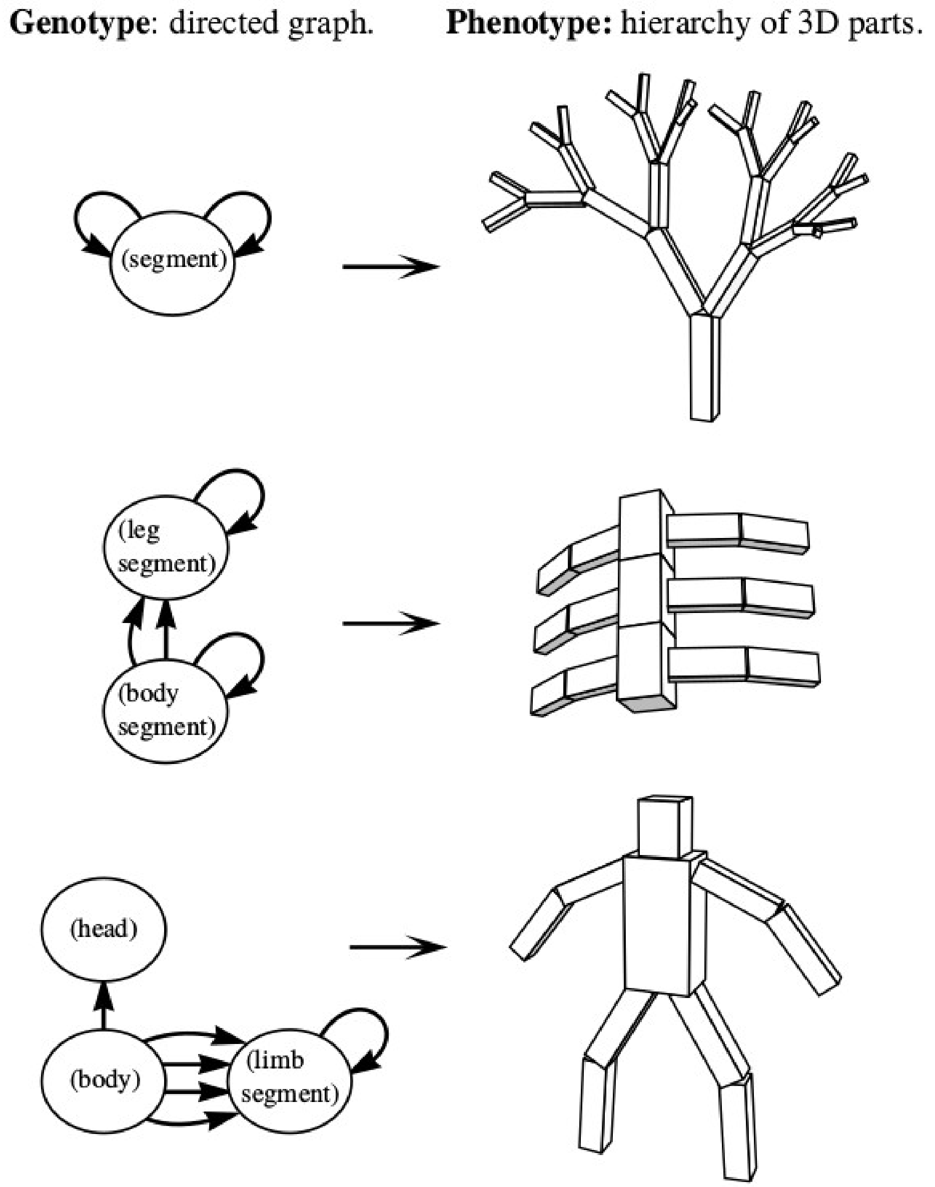
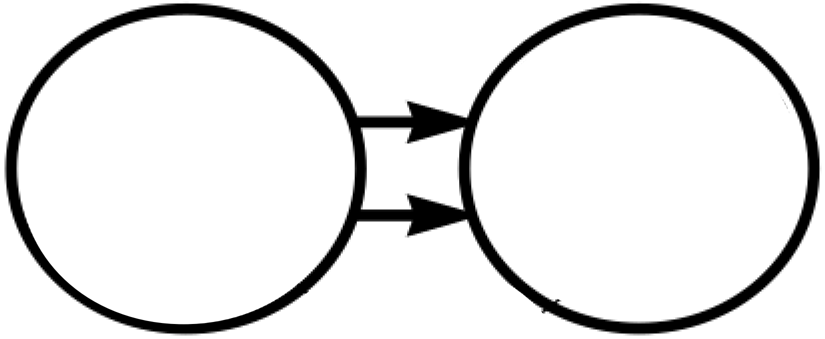


Figure 1: Designed examples of genotype graphs and corresponding creature morphologies.



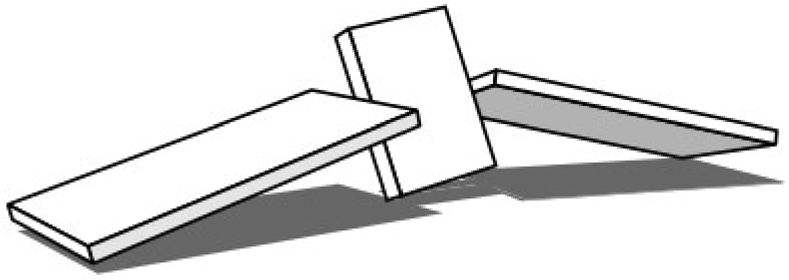


Figure 6a: The phenotype morphology generated from the evolved genotype shown in figure 5.

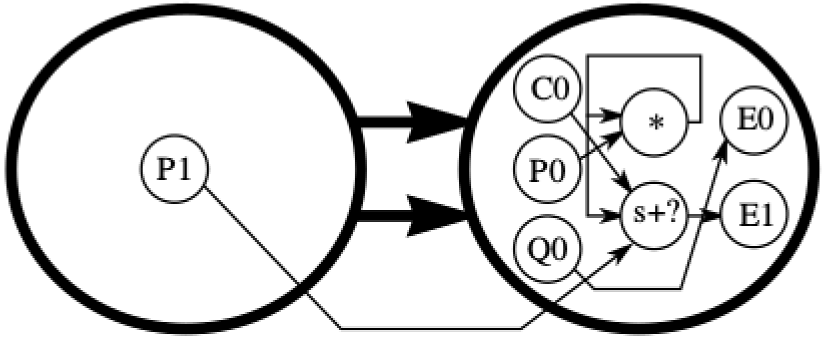


Figure 5: Example evolved nested graph genotype. The outer graph in bold describes a creature's morphology. The inner graph describes its neural circuitry. C0, P0, P1, and Q0 are contact and photosensors, E0 and E1 are effector outputs, and those labeled "*" and "s+?" are neural nodes that perform *product*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

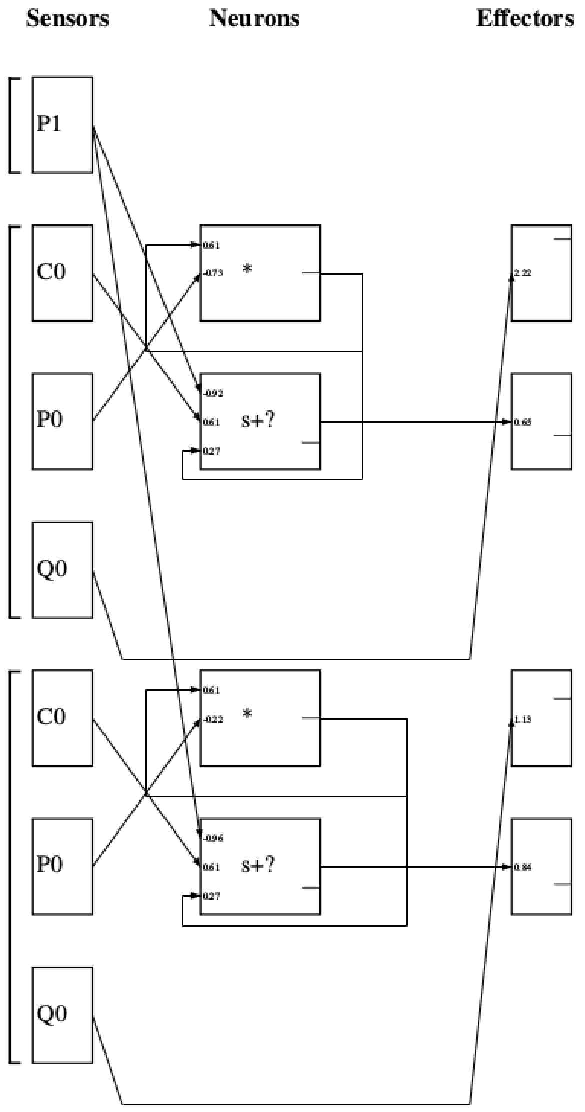


Figure 6b: The phenotype "brain" generated from the evolved genotype shown in figure 5. The effector outputs of this control system cause the morphology above to roll forward in tumbling motions.

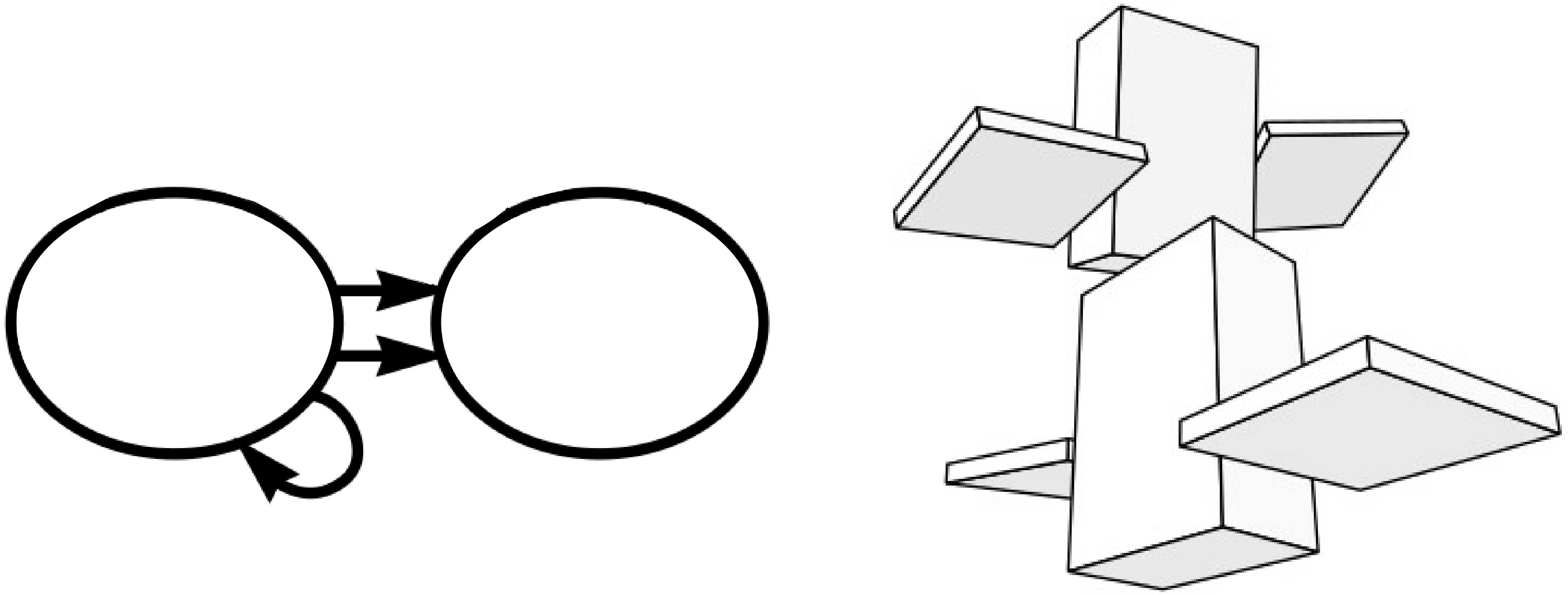


Figure 4a: The phenotype morphology generated from the evolved genotype shown in figure 3.

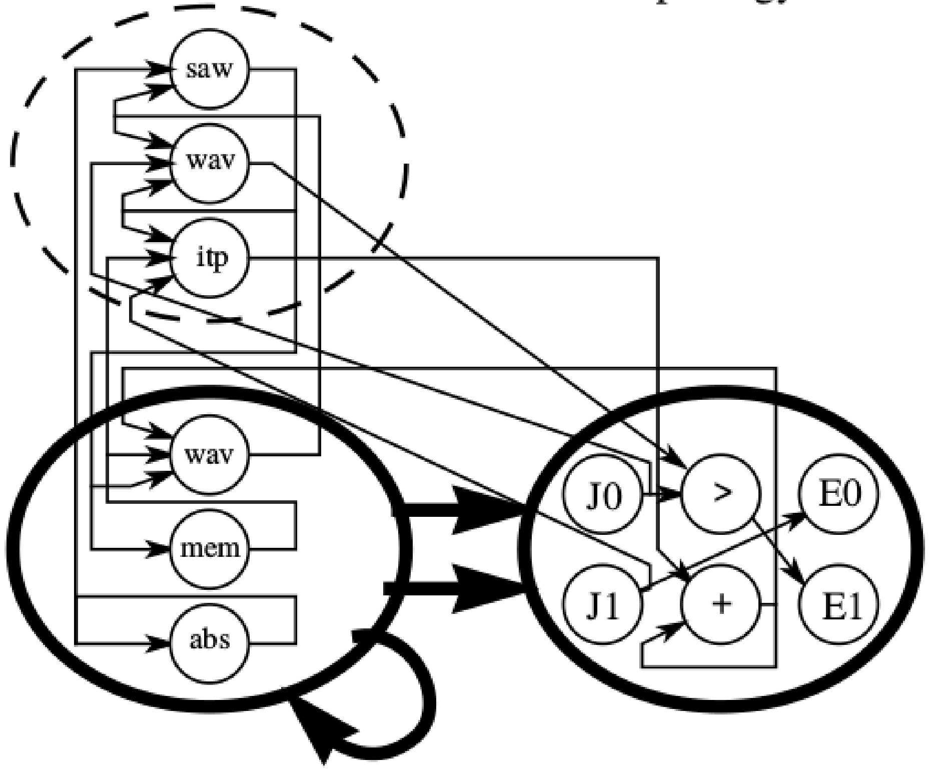


Figure 3: Example evolved nested graph genotype. The outer graph in bold describes a creature's morphology. The inner graph describes its neural circuitry. J0 and J1 are joint angle sensors, and E0 and E1 are effector outputs. The dashed node contains centralized neurons that are not associated with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

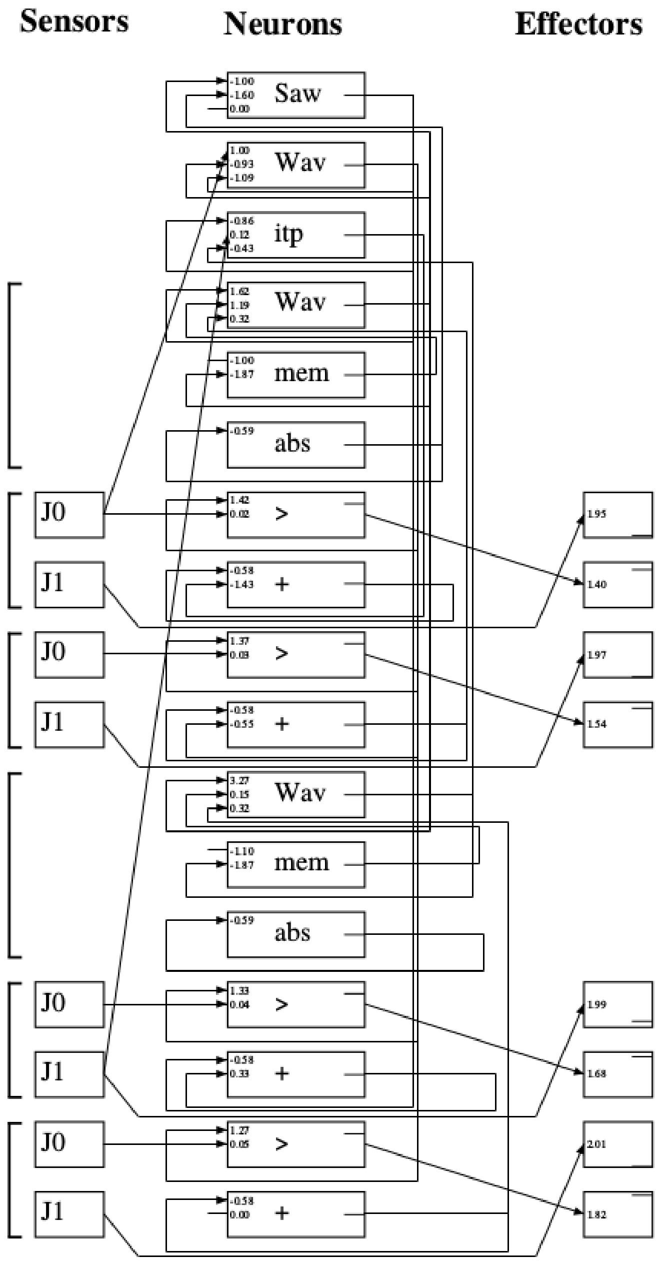


Figure 4b: The phenotype "brain" generated from the evolved genotype shown in figure 3. The effector outputs of this control system cause paddling motions in the four flippers of the morphology above.

Mating Directed Graphs

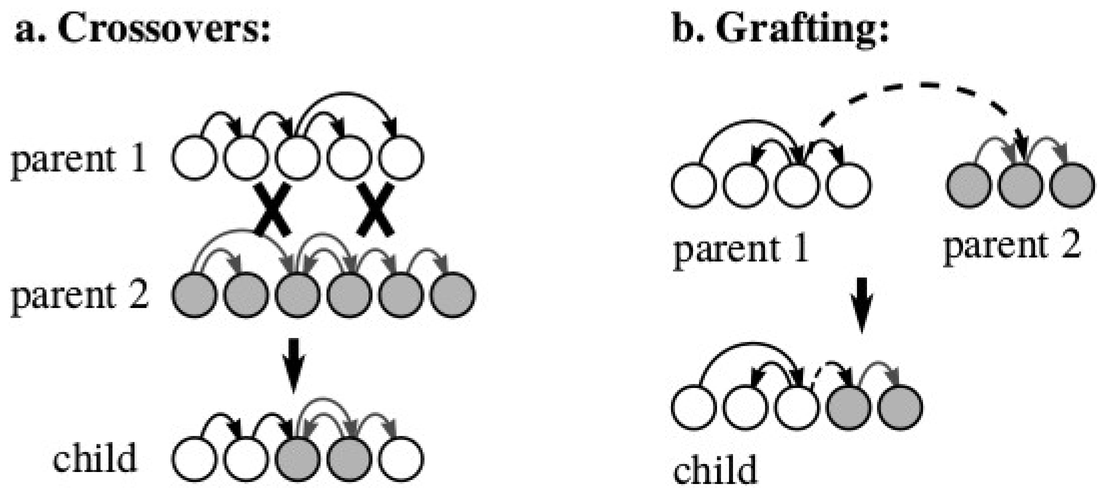
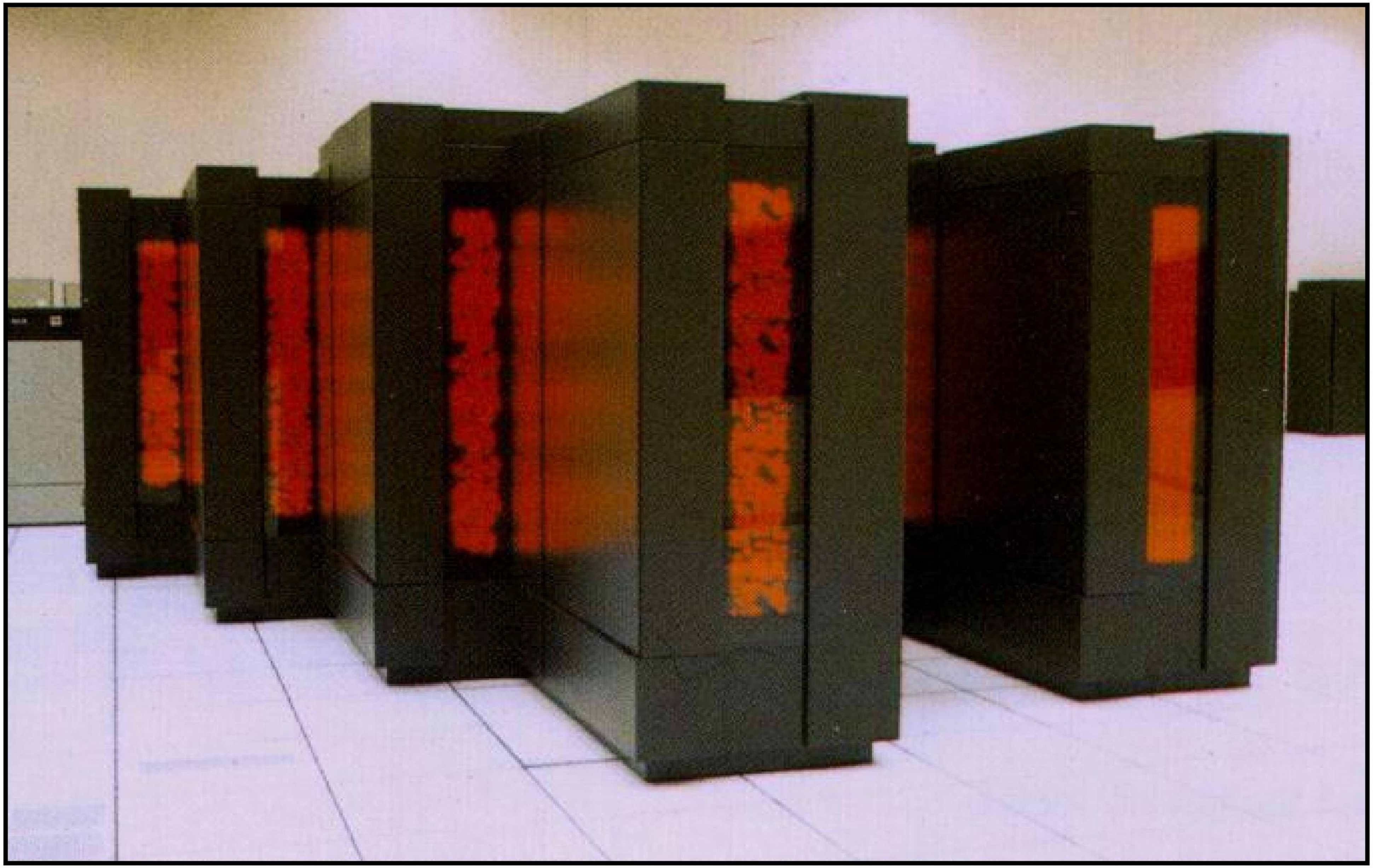


Figure 5: Two methods for mating directed graphs.

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.



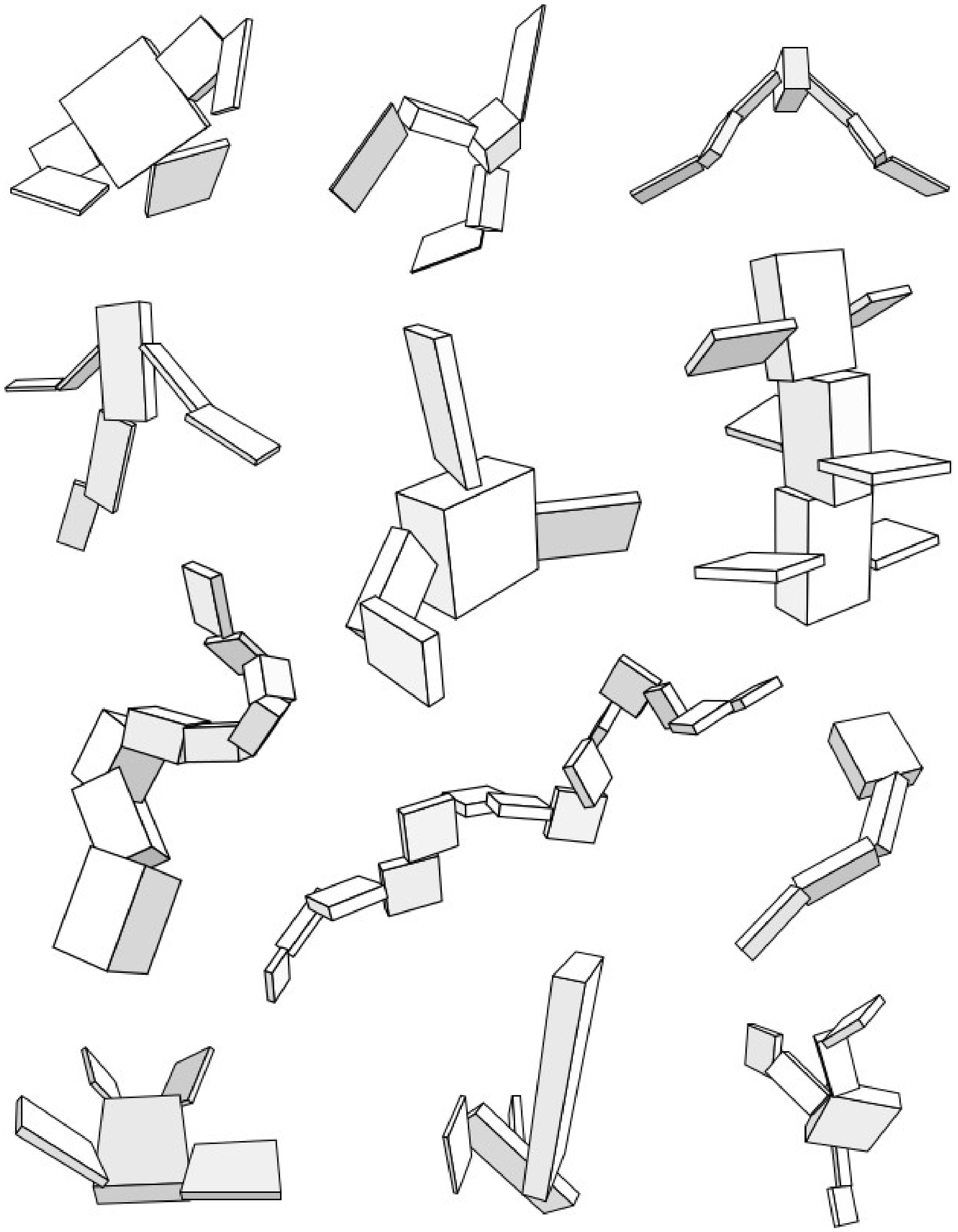


Figure 6: Creatures evolved for swimming.

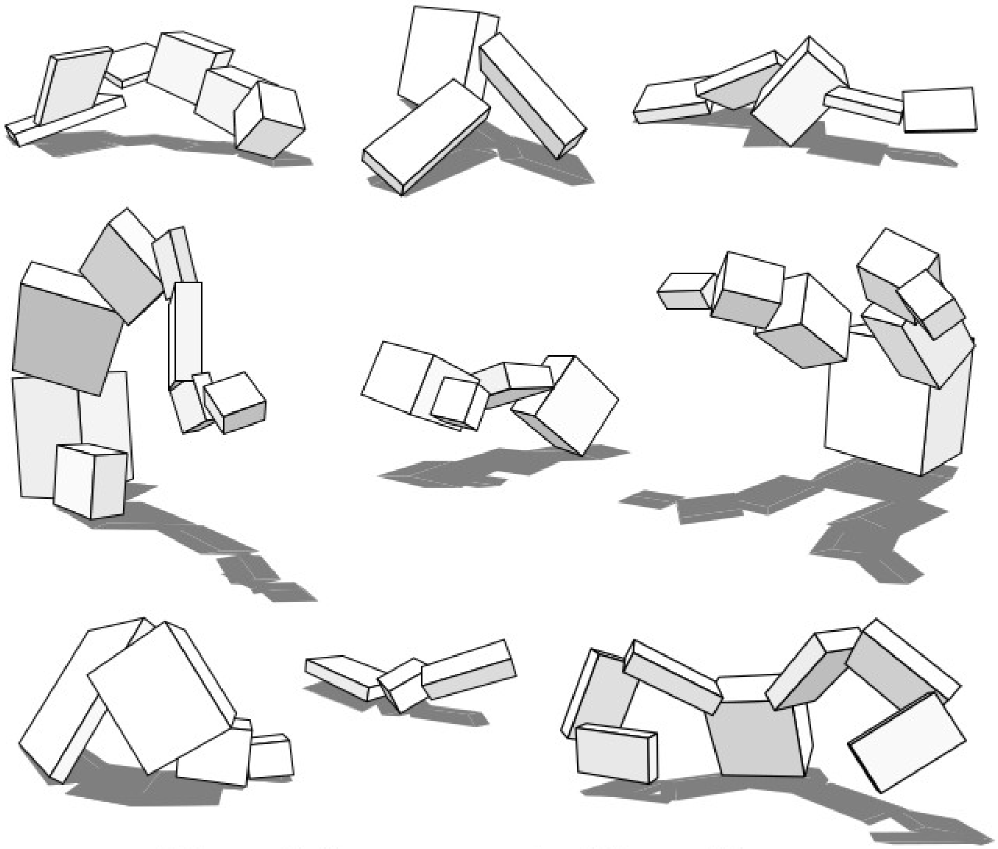


Figure 7: Creatures evolved for walking.

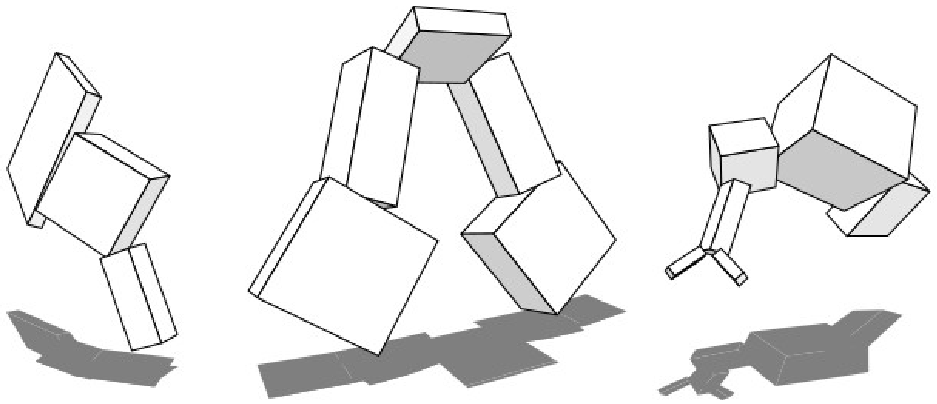


Figure 8: Creatures evolved for jumping.

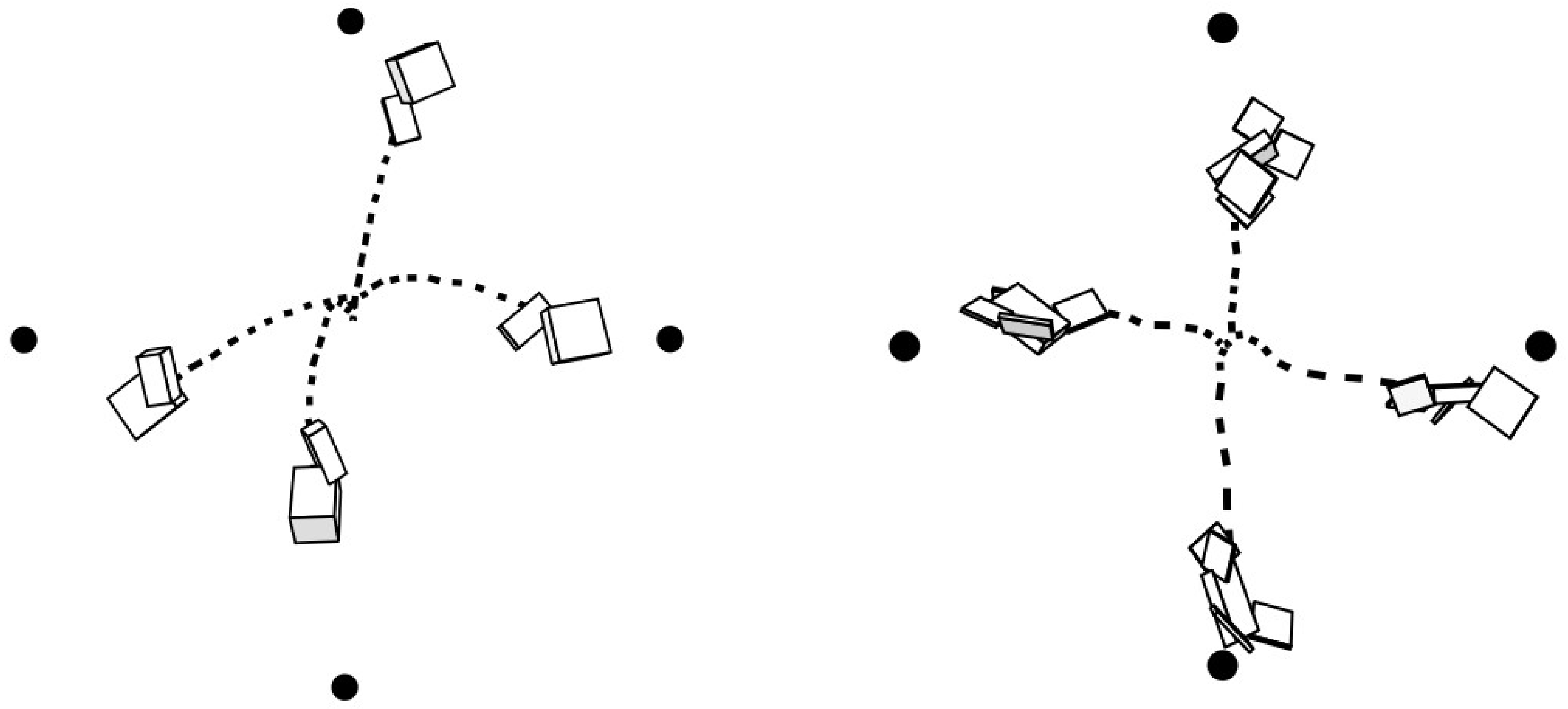


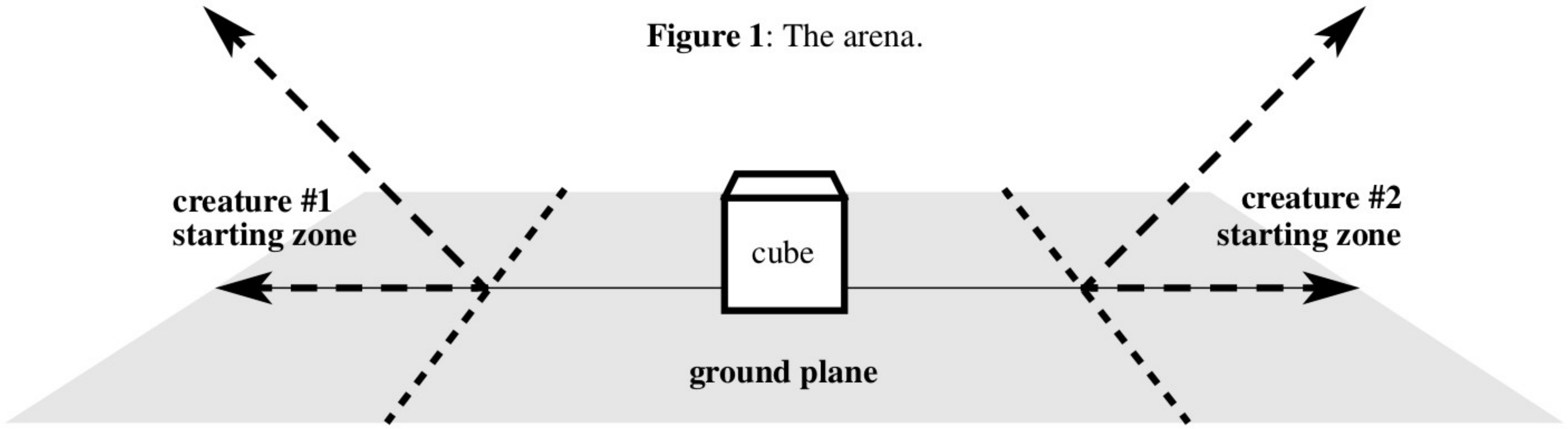
Figure 9: Following behavior. For each creature, four separate trials are shown from the same starting point toward different light source goal locations.

Evolving 3D Morphology and Behavior by Competition

Karl Sims

Thinking Machines Corporation

Figure 1: The arena.



The Contest

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.

Approximating Competitive Environments

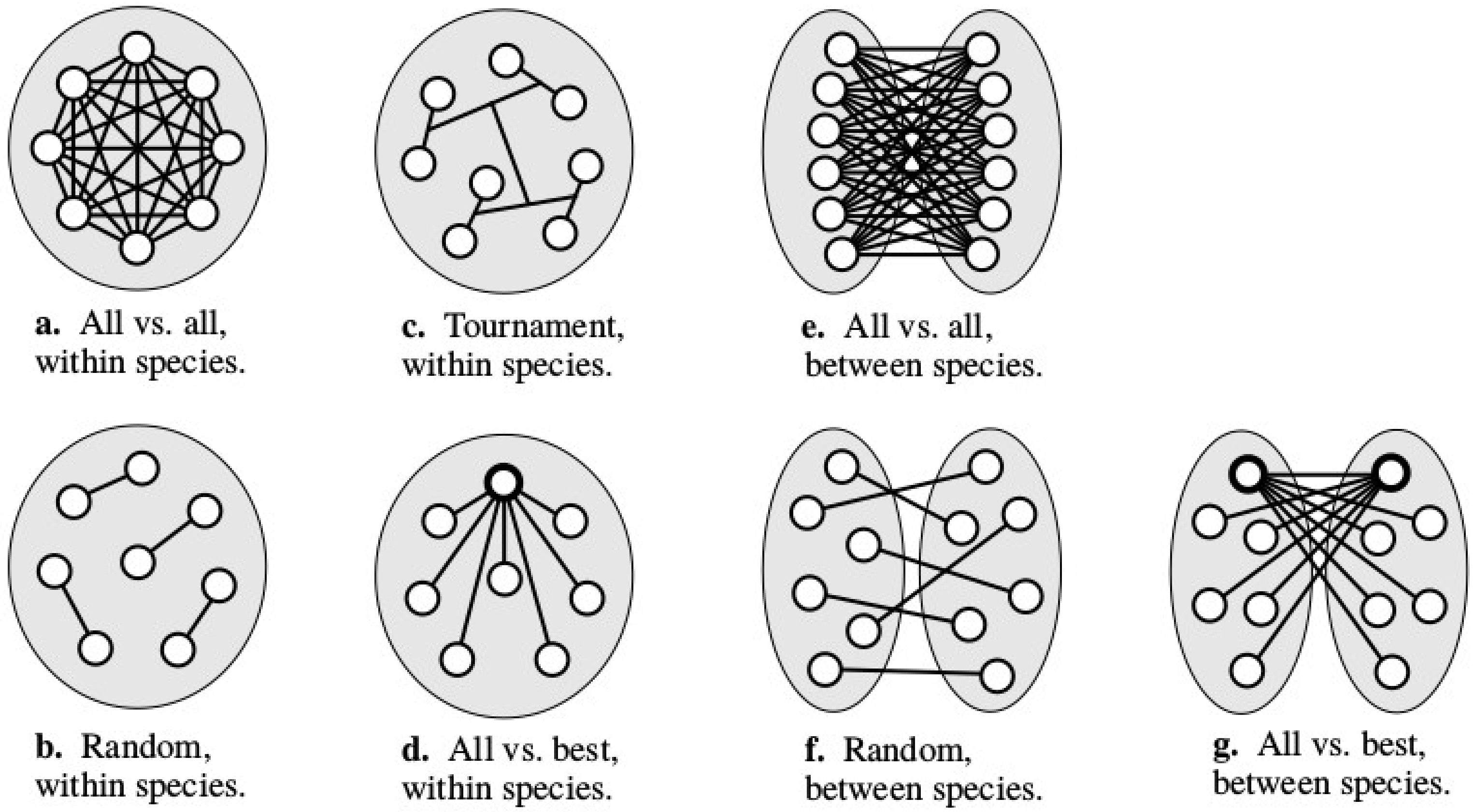
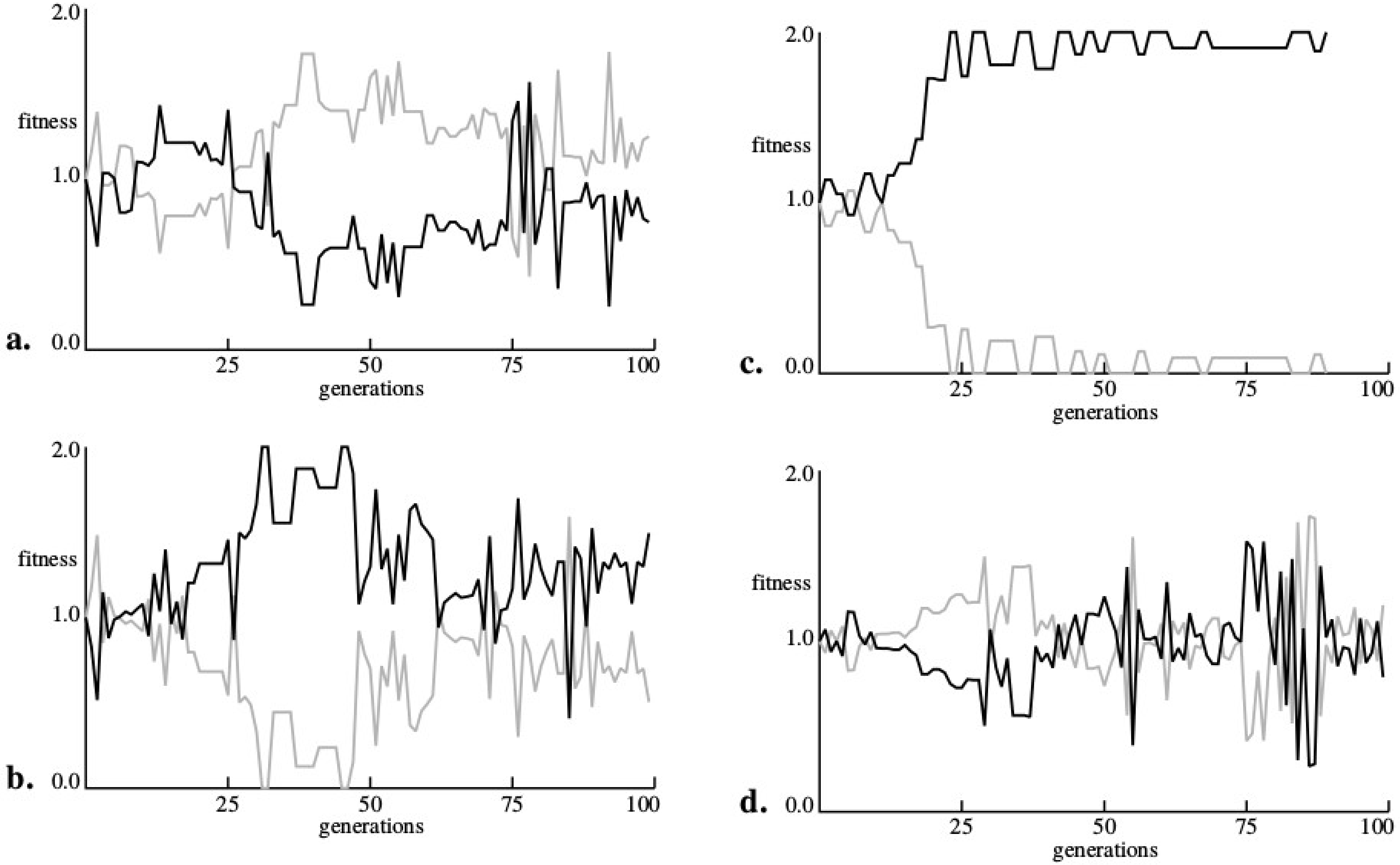

a. All vs. all, within species.
c. Tournament, within species.
e. All vs. all, between species.
b. Random, within species.
d. All vs. best, within species.
f. Random, between species.
g. All vs. best, between species.

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.

Figure 8: Relative fitness between two co-evolving and competing species, from four independent simulations.



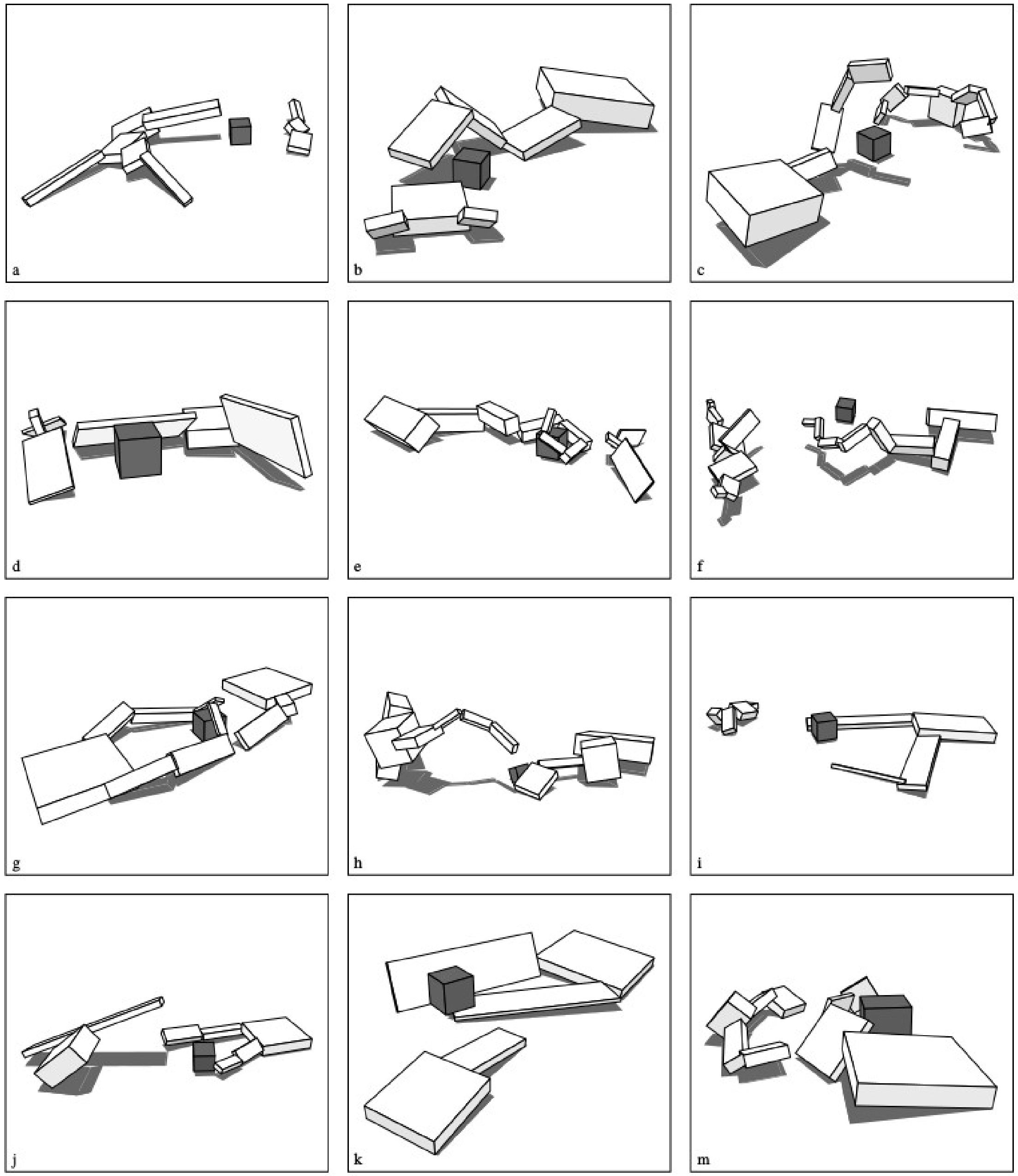
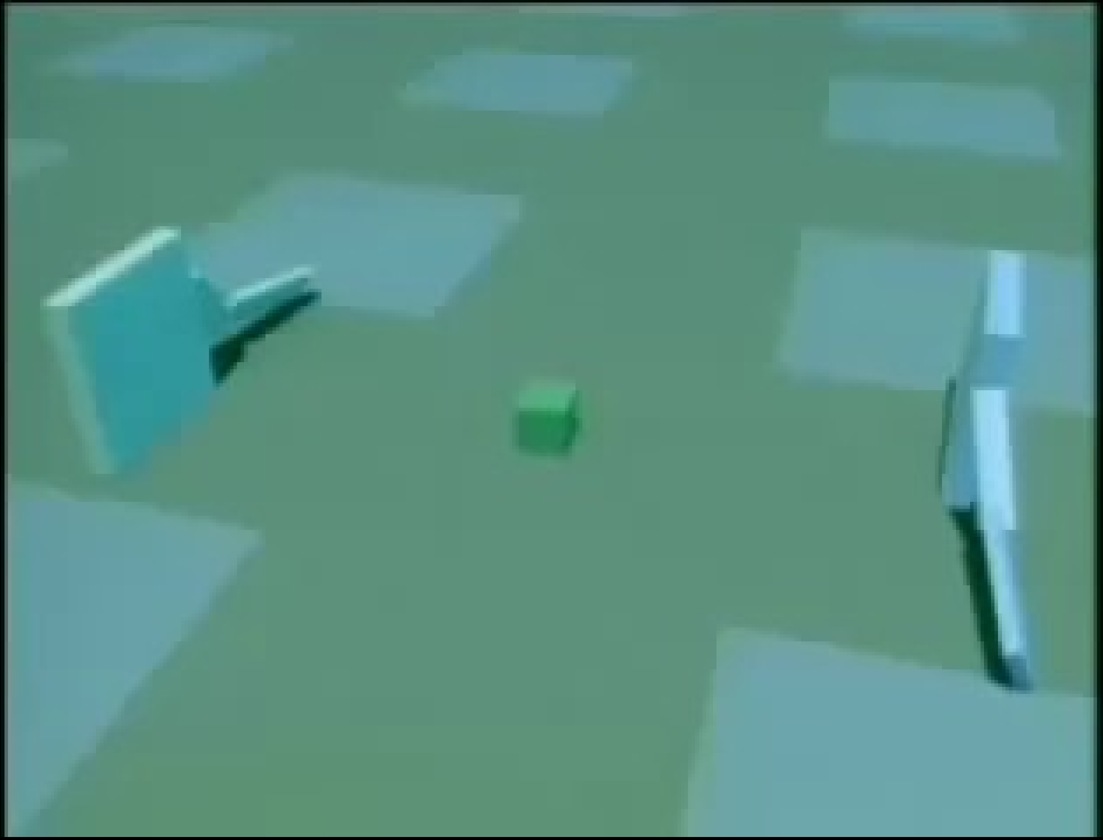


Figure 9: Evolved competing creatures.



GENARTS

Artificial Evolution for Computer Graphics

Karl Sims

Thinking Machines Corporation

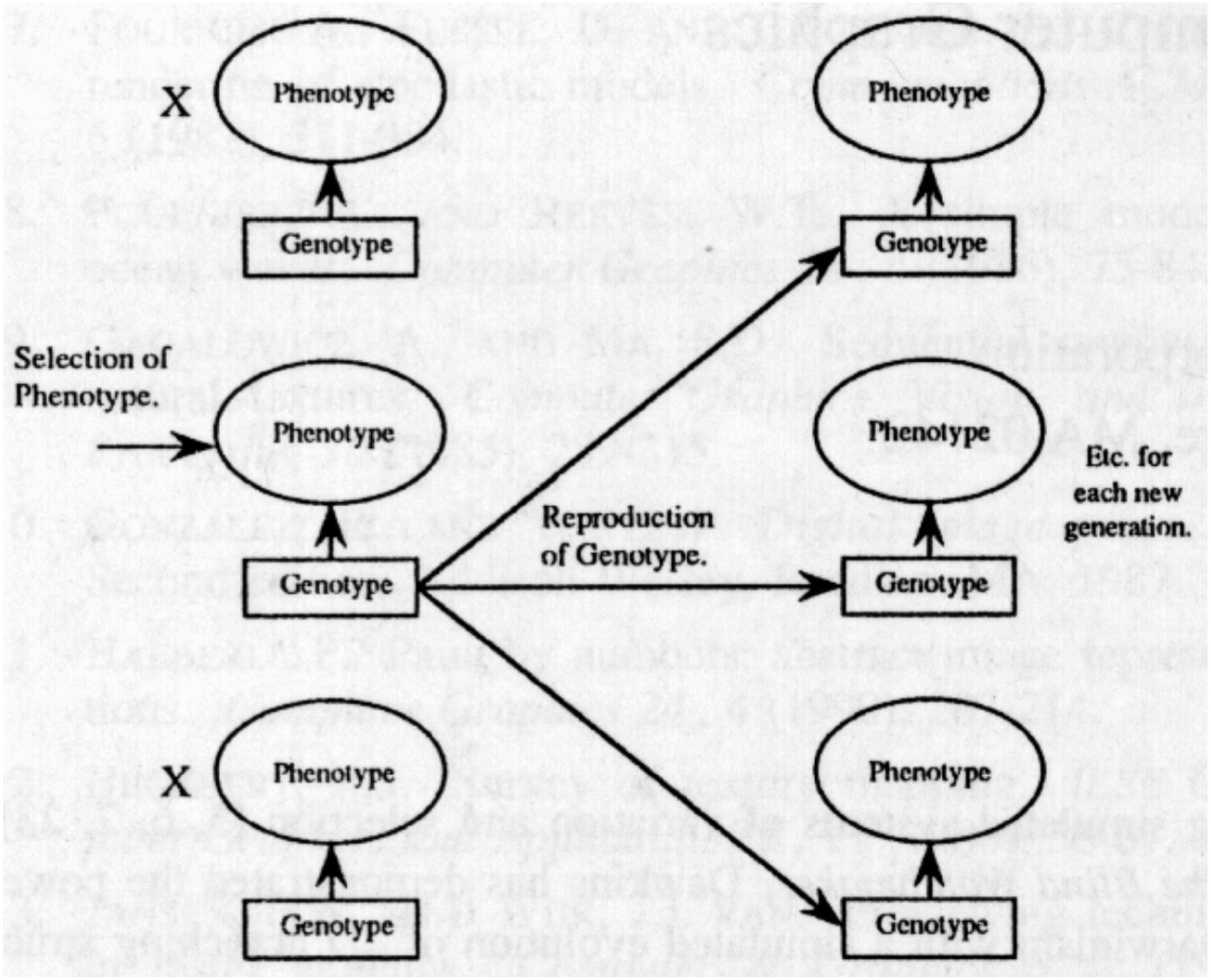


Figure 1: Phenotype selection, genotype reproduction.

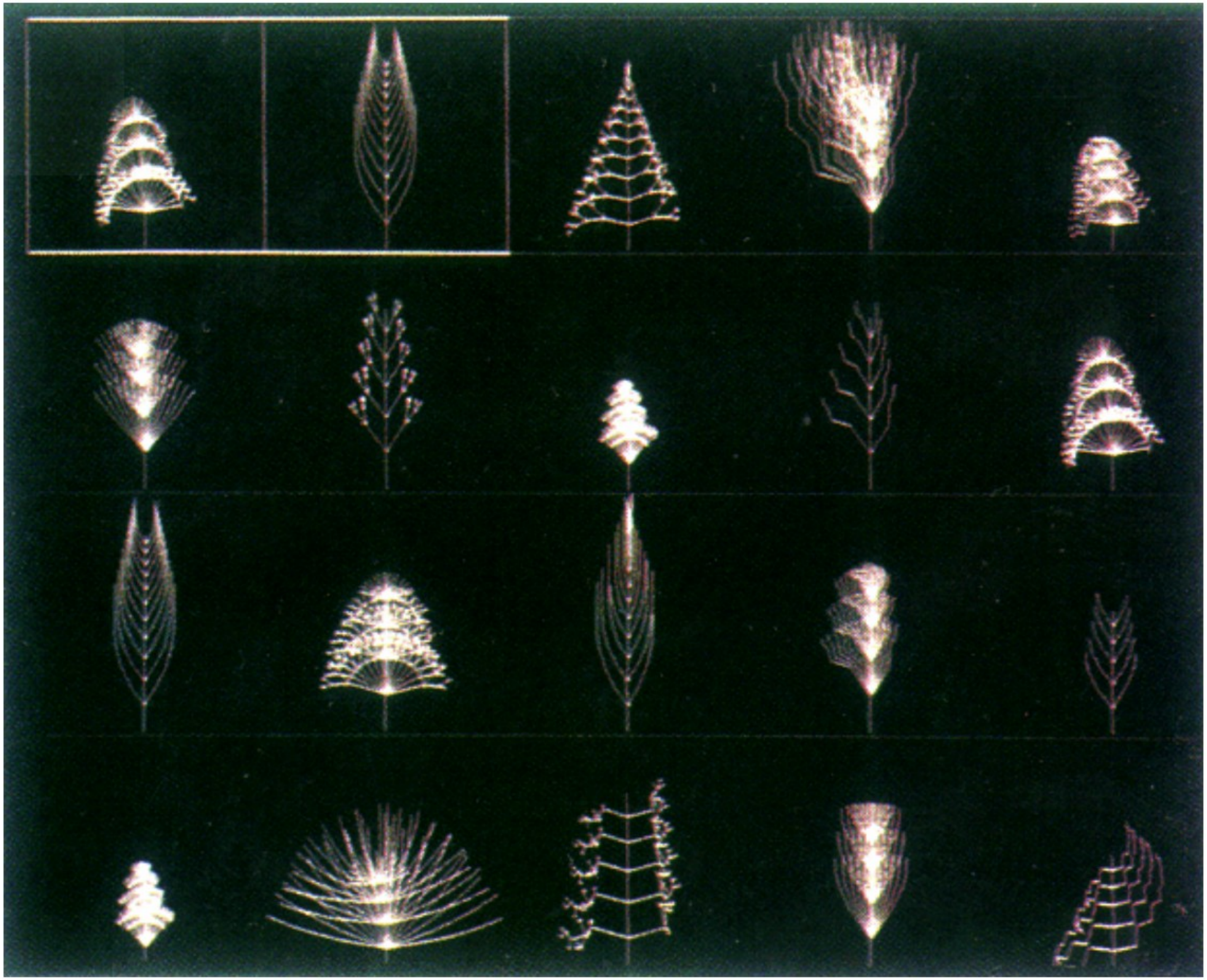


Figure 2: Mating plant structures.

Evolving Images

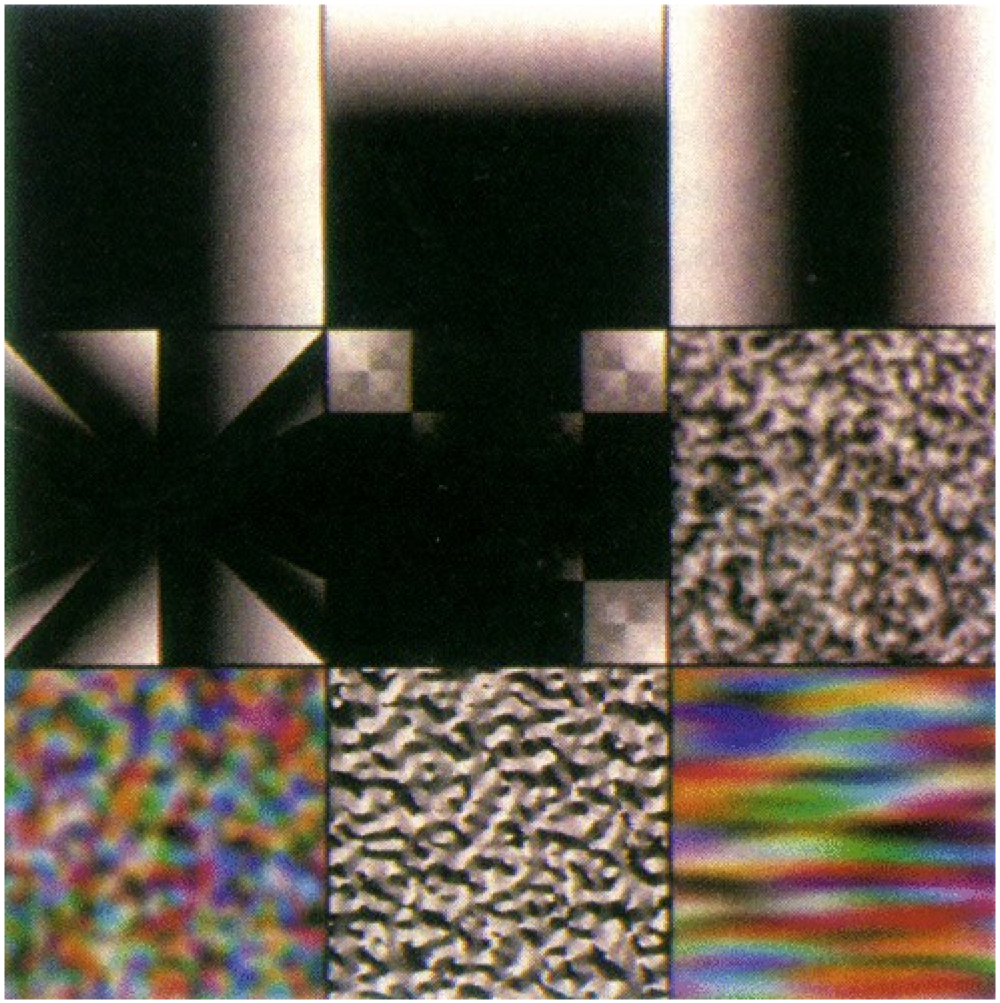
The second example of artificial evolution involves the generation of textures by mutating symbolic expressions. Equations that calculate a color for each pixel coordinate *(x,y)* are evolved using a *function set* containing some standard common lisp functions [26], vector transformations, procedural noise generators, and image processing operations:

```
+, -, *, /, mod, round, min, max, abs, expt, log, and,
or, xor, sin, cos, atan, if, dissolve, hsv-to-rgb, vector,
transform-vector, bw-noise, color-noise, warped-bw-noise,
warped-color-noise, blur, band-pass, grad-mag, grad-dir,
bump, ifs, warped-ifs, warp-abs, warp-rel, warp-by-grad.
```

Figure 4: Simple expression examples.

(reading left to right, top to bottom)

a. *X*
b. *Y*
c. *(abs X)*
d. *(mod X (abs Y))*
e. *(and X Y)*
f. *(bw-noise .2 2)*
g. *(color-noise .1 2)*
h. *(grad-direction (bw-noise .15 2) .0 .0)*
i. *(warped-color-noise (* X .2) Y .1 2)*



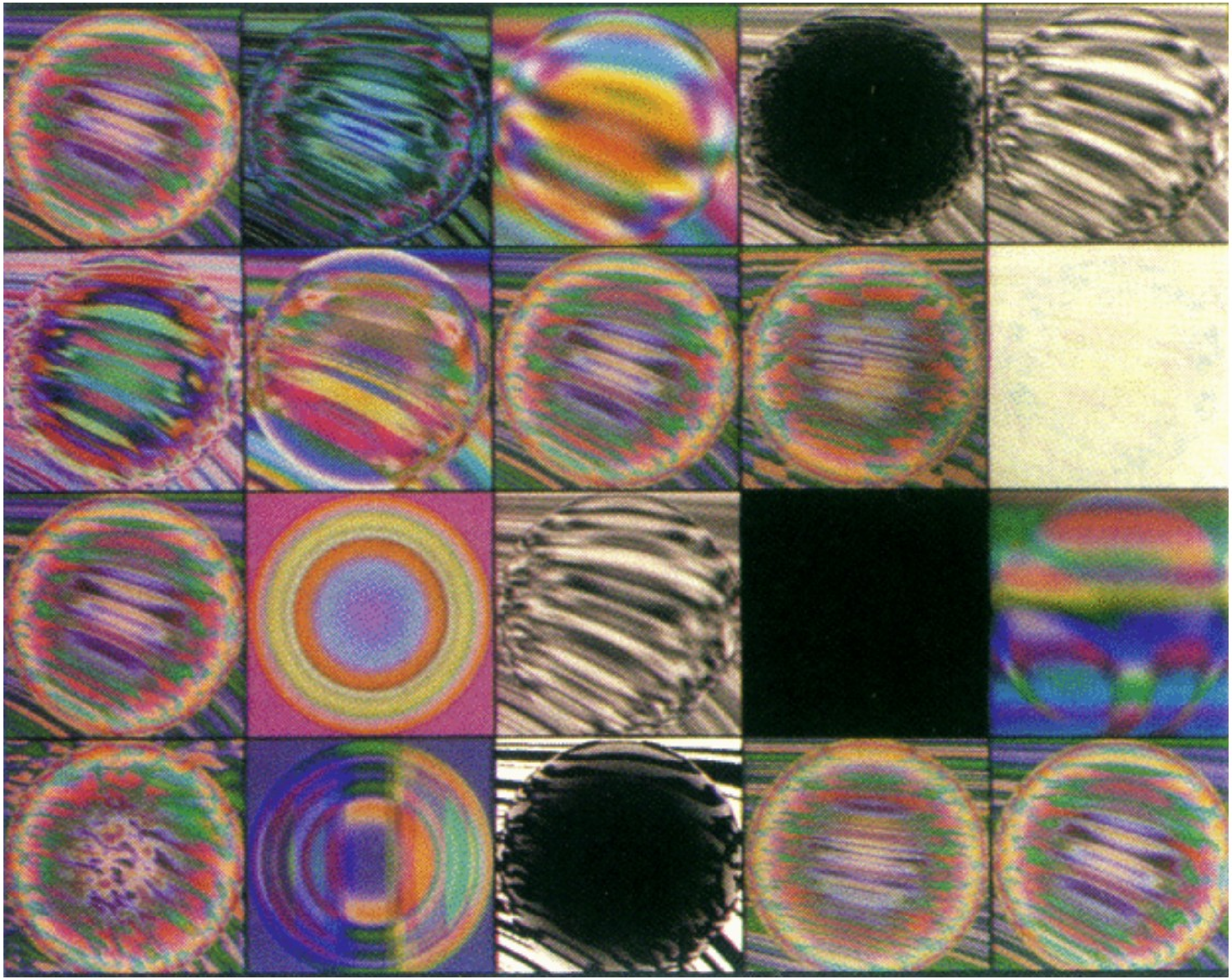
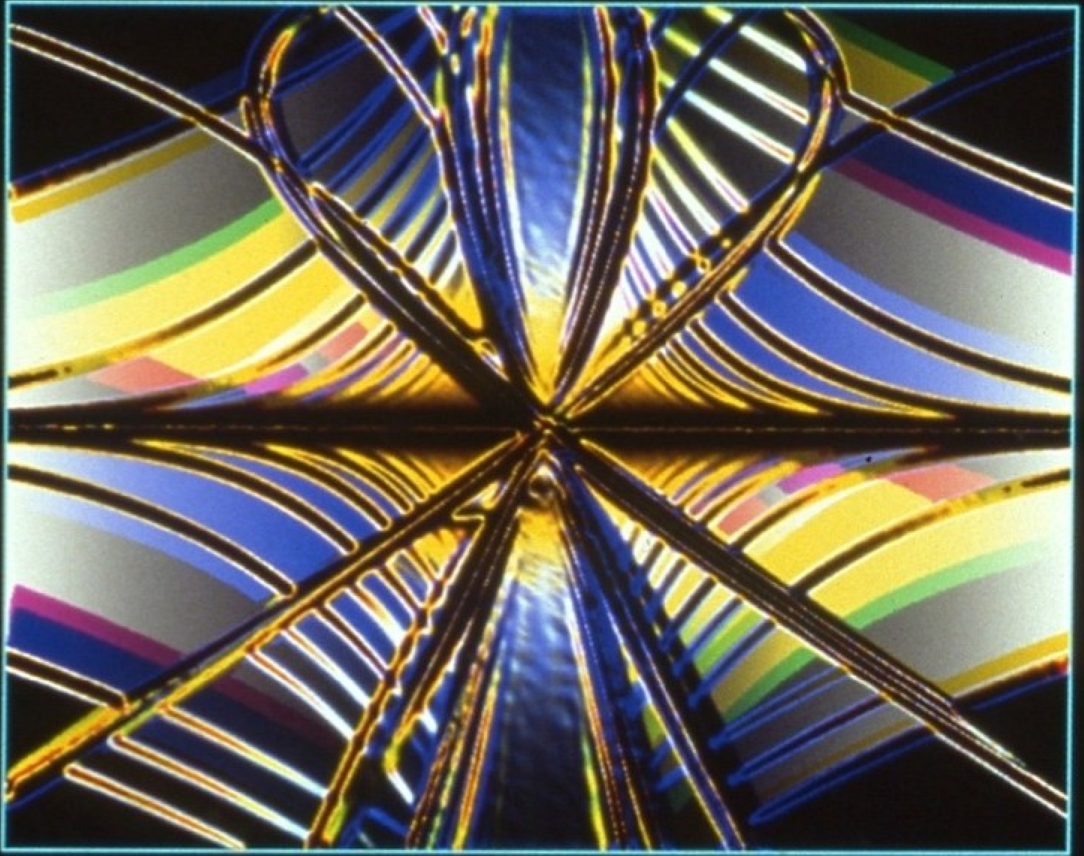


Figure 5: Parent with 19 random mutations.

Images are generated procedurally by symbolic Lisp expressions:

Color <= F(x,y)

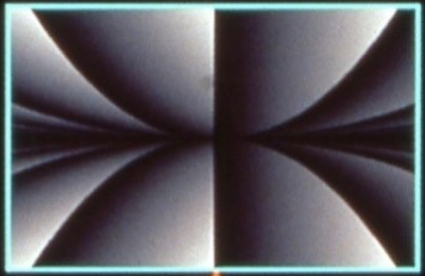
Phenotype:
(Image)



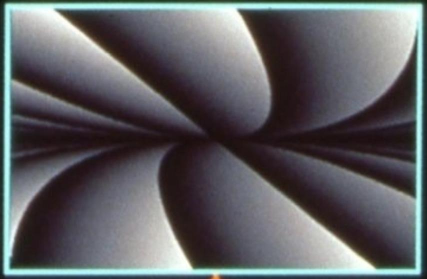
Genotype:
(Lisp code)

```
(round (log (+ y (color-grad (round (+ (abs (round (log
(+ y (color-grad (round (+ y (log (invert y) 15.5)) x) 3.1
1.86 #(0.95 0.7 0.59) 1.35)) 0.19) x)) (log (invert y) 15.5))
x) 3.1 1.9 #(0.95 0.7 0.35) 1.35)) 0.19) x)
```

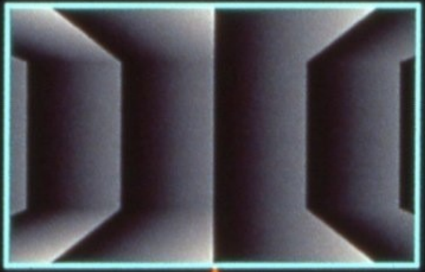
Computer reproduces Lisp expressions with random mutations:



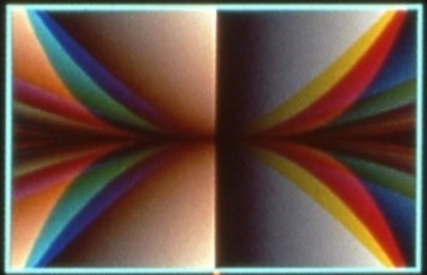
(mod x (expt (abs y) .46)



(mod (+ y x) (expt (abs y) .46))

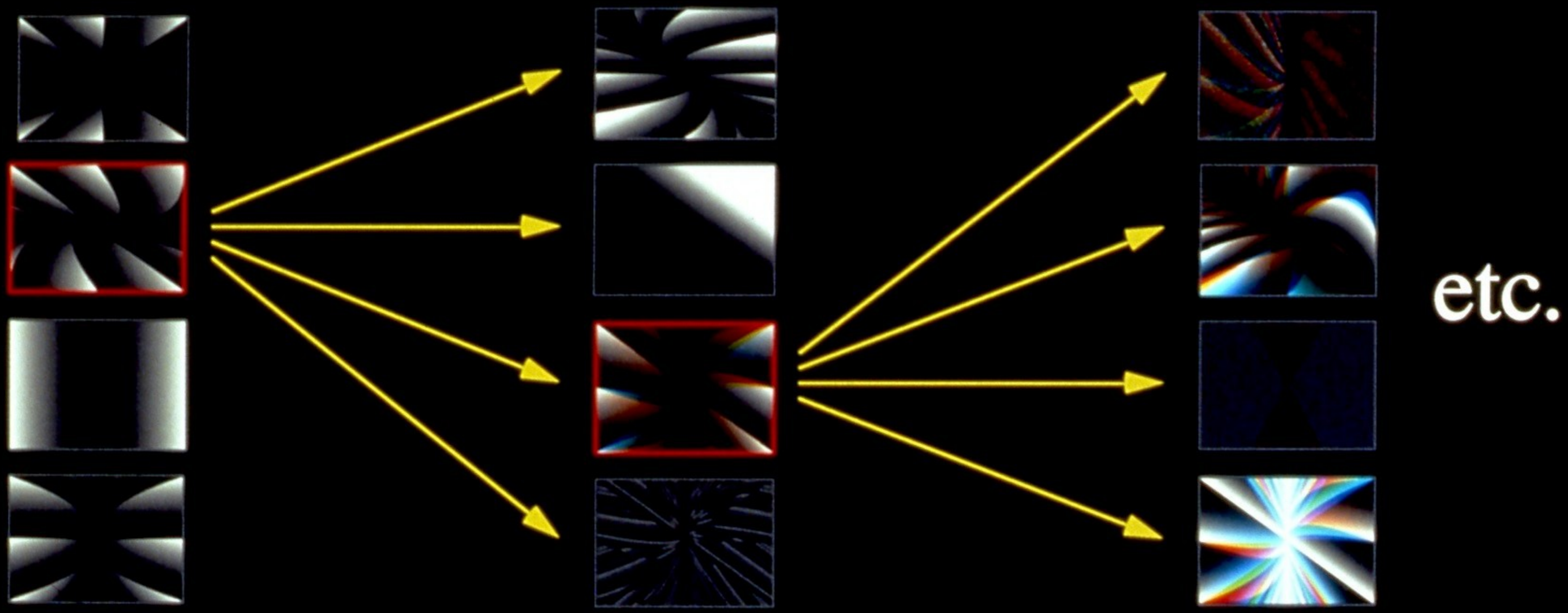


(mod x (max (abs y) .46))

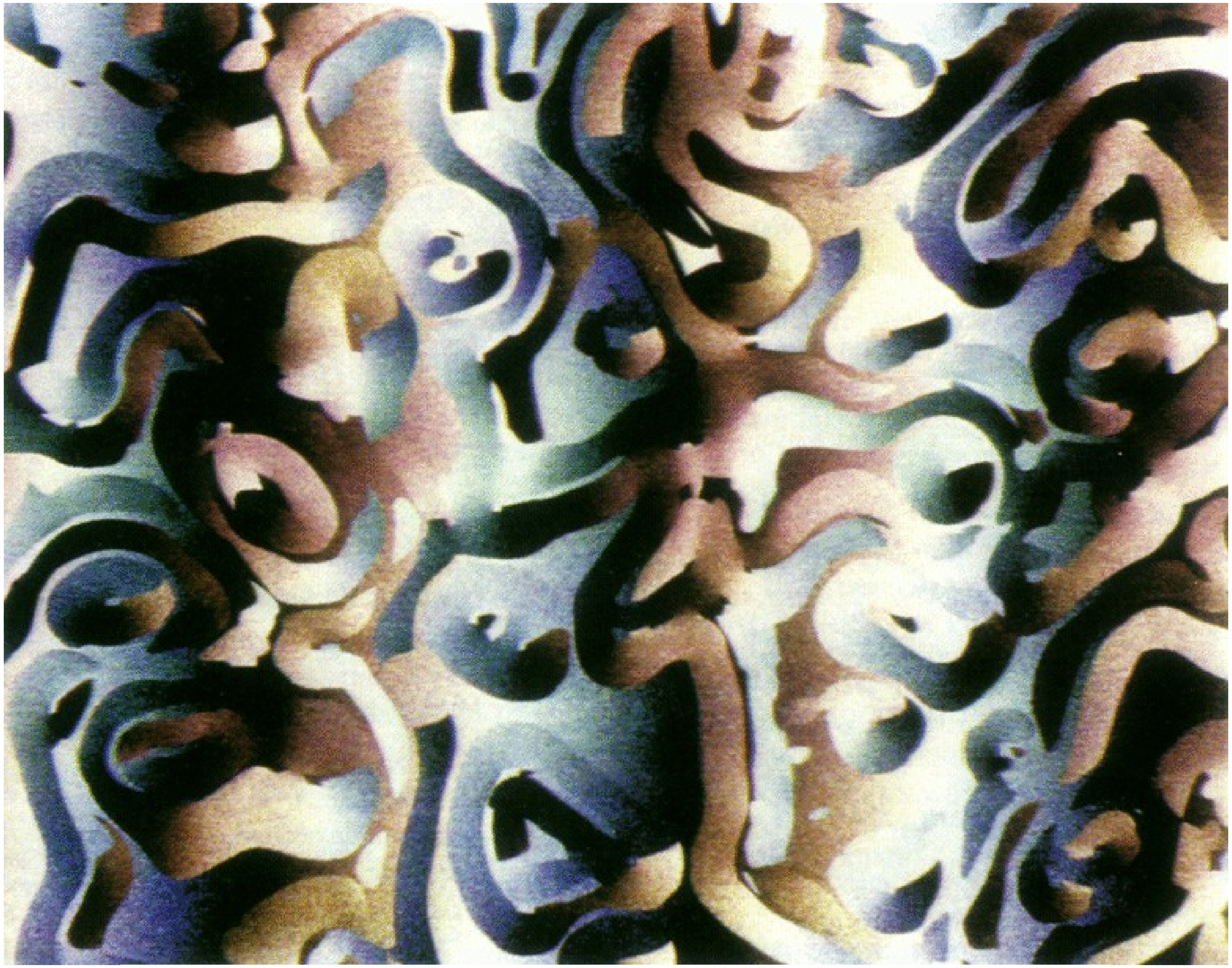


(mod x (expt (abs y) #(.26 .49 .73))

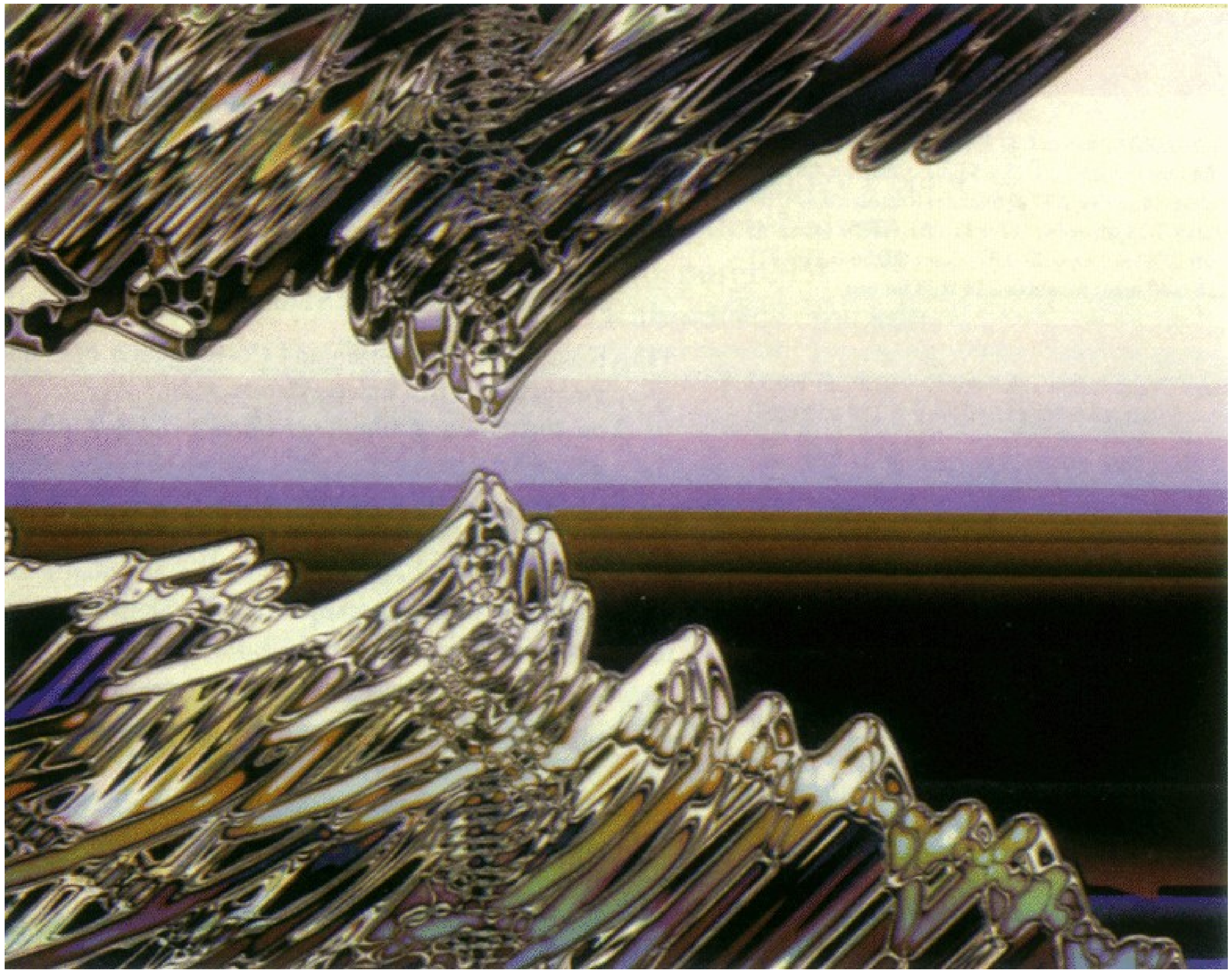
User selects the more aesthetically interesting images for survival:



"Interactive Evolution"

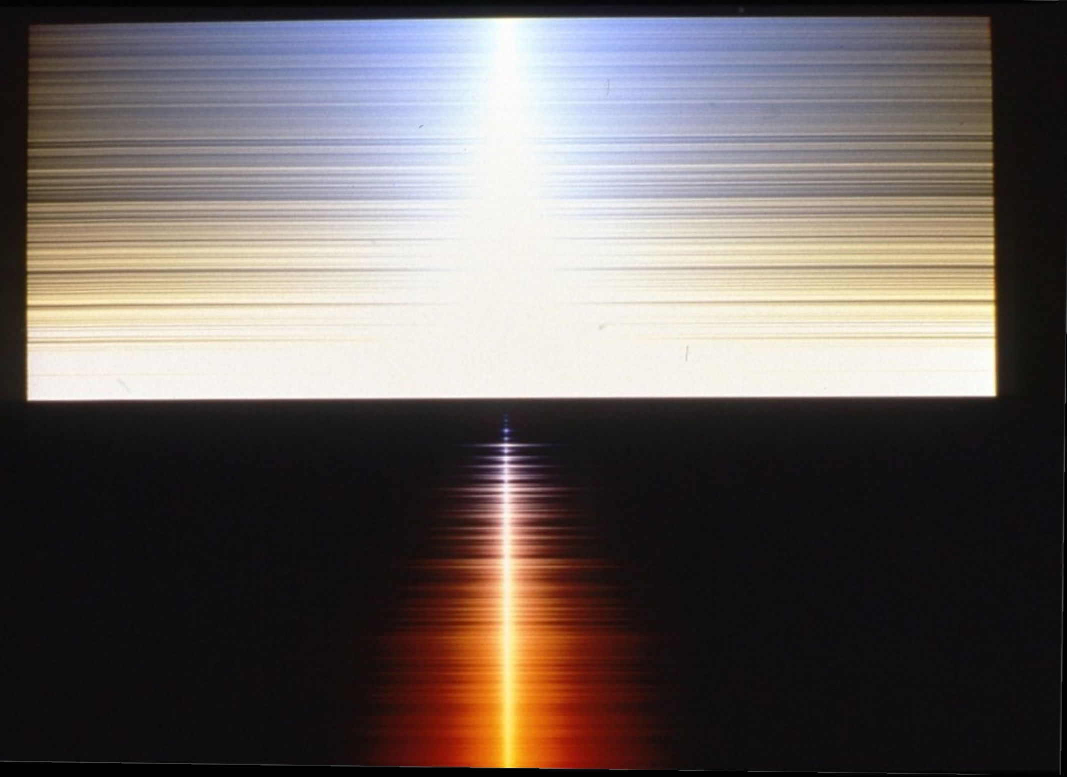
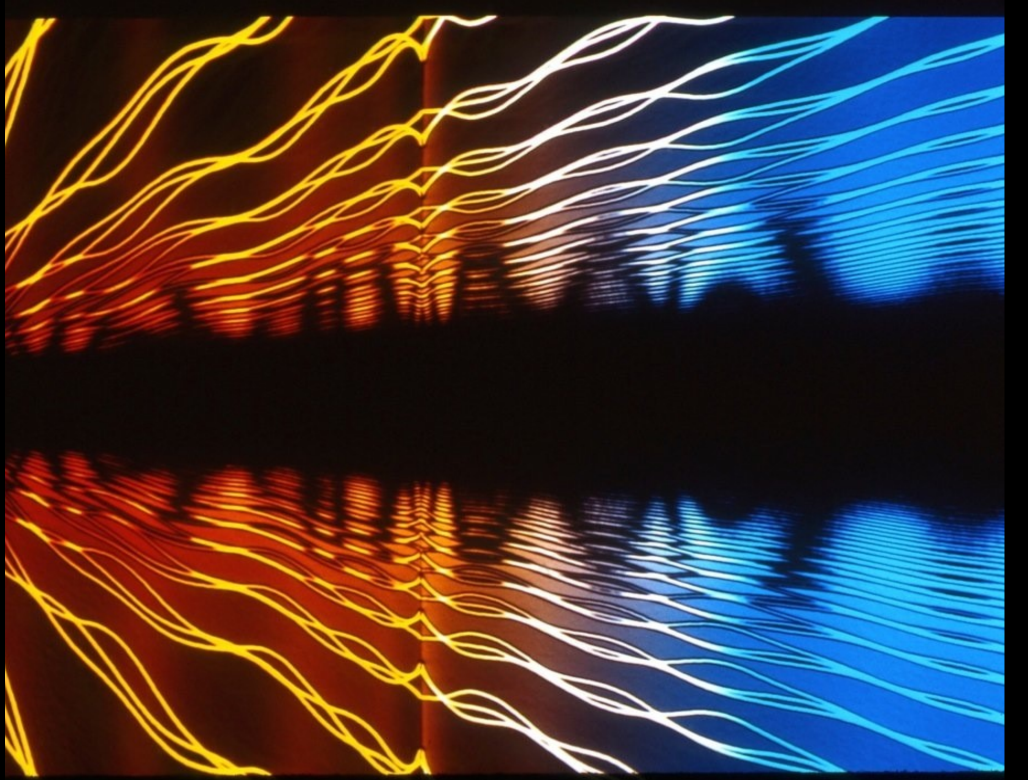


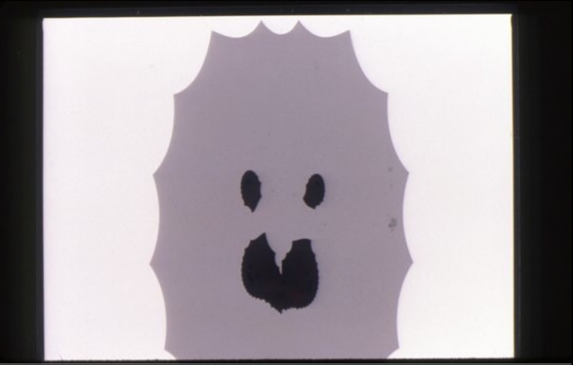
```
(sin (+ (- (grad-direction (blur (if (hsv-to-rgb (warped-
color-noise #(0.57 0.73 0.92) (/ 1.85 (warped-color-
noise x y 0.02 3.08)) 0.11 2.4)) #(0.54 0.73 0.59) #(1.06
0.82 0.06)) 3.1) 1.46 5.9) (hsv-to-rgb (warped-color-
noise y (/ 4.5 (warped-color-noise y (/ x y) 2.4 2.4))
0.02 2.4))) x))
```

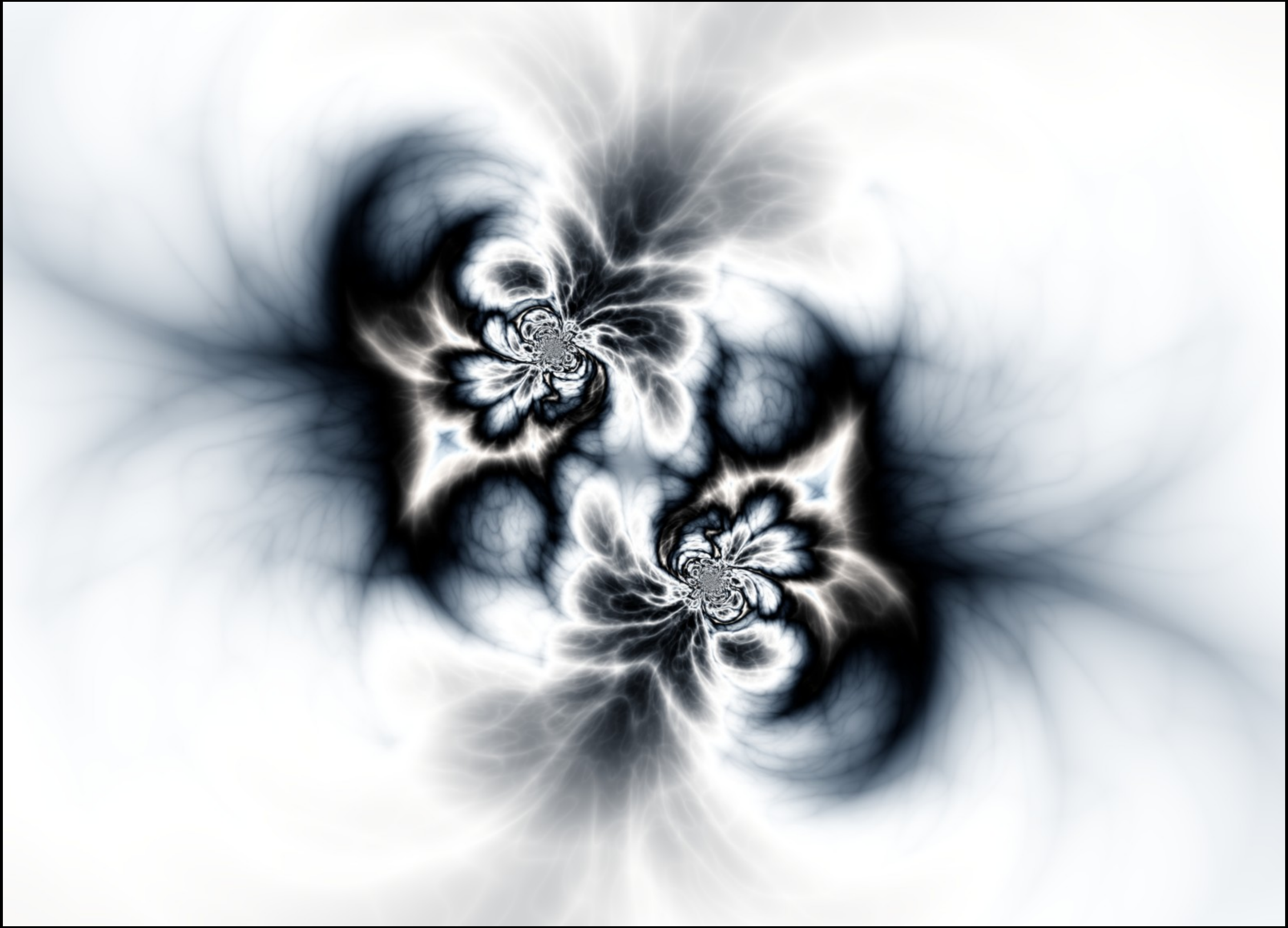


```
(cos (round (atan (log (invert y) (+ (bump (+ (round x y) y) #(0.46 0.82 0.65) 0.02
#(0.1 0.06 0.1) #(0.99 0.06 0.41) 1.47 8.7 3.7) (color-grad (round (+ y y) (log (invert x)
(+ (invert y) (round (+ y x) (bump (warped-ifs (round y y) y 0.08 0.06 7.4 1.65 6.1 0.54 3.1 0.26
0.73 15.8 5.7 8.9 0.49 7.2 15.6 0.98) #(0.46 0.82 0.65) 0.02 #(0.1 0.06 0.1)
#(0.99 0.06 0.41) 0.83 8.7 2.6))))) 3.1 6.8 #(0.95 0.7 0.59) 0.57))) #(0.17 0.08 0.75) 0.37)
(vector y 0.09 (cos (round y y)))))
```

How do we rank the “prettiness” of art?





```
wave(noise(chroma_scale( cinverse(xy - [-3.564, 4.51,1])
 - cinverse(xy + [-3,564, 4.51,1]), 0.465)
+ 27.36, 2.7, 1, 8, 2, 0),   [2.02, 2, 1.98], 0, 1, 0)
```