NEAT Particles: Design, Representation, and Animation of Particle System Effects

Erin Hastings, Ratan Guha, and Kenneth O. Stanley
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando, FL 32816
{hastings, guha, kstanley}@cs.ucf.edu
In *Proceedings of the IEEE 2007 Symposium on Computational Intelligence and Games* (CIG'07). Piscataway, NJ: IEEE

***Abstract*— Particle systems are a representation, computation, and rendering method for special effects such as fire, smoke, explosions, electricity, water, magic, and many other phenomena. This paper presents NEAT Particles, a new design, representation, and animation method for particle systems tailored to real-time effects in video games and simulations. In NEAT Particles, the NeuroEvolution of Augmenting Topologies (NEAT) method evolves artificial neural networks (ANN) that control the appearance and motion of particles. NEAT Particles affords three primary advantages over traditional particle effect development methods. First, it decouples the creation of new particle effects from mathematics and programming, enabling users with little knowledge of either to produce complex effects. Second, it allows content designers to evolve a broader range of effects than typical development tools through a form of Interactive Evolutionary Computation (IEC). And finally, it acts as a concept generator, allowing users to interactively explore the space of possible effects. In the future such a system may allow content to be evolved in the game itself, as it is played.**

Keywords: particle systems, NeuroEvolution of Augmenting Topologies, NEAT, interactive evolutionary computation, IEC

I. Introduction

Particle systems are ubiquitous in computer graphics, producing animated effects including fire, smoke, clouds, gunfire, water, cloth, explosions, magic, lighting, electricity, atoms, flocking, and many others [1] [2]. Particle Systems are defined by (1) a set of points in space, and (2) a set of *rules* guiding their behavior and appearance, e.g. velocity, color, size, shape, transparency, rotation, etc. Since particle systems follow complex rules, creating new or unique particle effects requires considerable mathematical and technical skill. For example, consider designing a particle effect for a magical spell that *originates from the wizard's hand, swirls in a spiral toward a target, and changes color and size*. In current practice the precise mechanics for this scenario must be hand coded by a programmer as a new particle effect

This paper presents NEAT Particles, a new design approach for particle systems based on the NeuroEvolution of Augmenting Topologies (NEAT) method

NEAT Particles aims to (1) enable users with little programming or artistic skill to evolve unique particle system effects through Interactive Evolutionary Computation (IEC), (2) broaden the range of possible effects, and (3) provide a way for developers to explore the range of possible effects. This paper describes how NEAT Particles creates effects for real-time games and simulations; in the future it will be extended other applications as well. NEAT Particles can evolve behavior such as the wizard's spell without the need for knowledge of physics or programming. NEAT Particles is a step toward the long-term goal of automated graphics content generation for games, simulations, and movies.

II. Background

This section reviews particle systems, IEC, and NEAT, which are all components of the NEAT Particles system presented in Section III.

A. Particle System Background

Particle systems often implement special effects in movies [3] and games [1] [2]. Particle systems can also model more tangible objects such as unique trees in a forest [4], folded cloth and fabric [5] [6], and simulate fluid motion [7] [8]. Realistic particle movement is often achieved by simulating real-world physics [9]. At a more abstract level, particle systems have simulated animal and insect flocking and swarming behavior as well [10]. The diversity of particle system applications demonstrates their importance to modern interactive media and games.

B. Interactive Evolutionary Computation (IEC) Background

IEC is an approach to evolutionary computation (EC) in which human evaluation partially or wholly replaces the fitness function [11]. IEC has enabled a broad range of graphical content generation. An early implementation of IEC was Biomorphs, which aimed to illustrate theories about natural evolution [12]. Biomorphs are simple patterns encoded as Lindenmayer Systems (L-systems) [13], which are grammars used to specify the order in which a set of replacement rules are carried out. Figures that resemble animals or plants can be interactively evolved in this way. IEC digital art systems have also utilized representations such as linear or non-linear functions, fractals, and automata. Notable examples include Mutator [14], a cartoon and facial animation system, and SBART [15], a two-dimensional art system.

Figure 1 illustrates IEC's capabilities with Mattias Fagerlund's Delphi-NEAT Genetic Art application [16]. The figure shows four champions in the evolution of a spaceship [17]. In this example, the user starts by selecting a simple image that somewhat resembles what they wish to create and continues

to evolve more complex images through selection until satisfied with the output. The series of images demonstrates the potential of IEC art tools.

C. NeuroEvolution of Augmenting Topologies (NEAT) Background

The NEAT method was originally developed to solve difficult control and sequential decision tasks. The ANNs evolved with NEAT control agents that select actions based on their sensory inputs. While previous methods that evolved ANNs, i.e. *neuroevolution* methods, evolved either fixed topology networks [18], [19], [20], or arbitrary random-topology networks [21], [22], [23], [24], [25], [26], [27], NEAT is the first to begin evolution with a population of small, simple networks and *complexify* the network topology over generations, leading to increasingly sophisticated behavior. Compared to traditional reinforcement learning techniques, which predict the long-term reward for taking actions in different states [28], the recurrent networks that evolve in NEAT are robust in continuous domains and in domains that require memory, making many applications possible. This section briefly reviews the NEAT method; Stanley and Miikkulainen [29], [30] provide complete introductions.

NEAT is based on three key principles. First, in order to allow ANN structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover without the need for expensive topological analysis. That way, genomes of different organizations and sizes stay compatible throughout evolution, solving the previously open problem of matching different topologies [31] in an evolving population.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before competing with other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong.

Third, unlike other systems that evolve network topologies and weights [32], [24], [27], [25], [24], [27] NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem. This process of complexification has important implications for search. While it may not be practical to find a solution in a high-dimensional space by searching in that space directly, it may be possible to find it by first searching in lower dimensional spaces and complexifying the best solutions into the high-dimensional space.

Since its inception, NEAT has been applied to a broad array of research areas, most notably NERO, a real-time war game with ANN-controlled soldiers [33]. Because NEAT is a strong method for evolving controllers for dynamic physical systems, it can naturally be extended to evolve the motion of particles in particle effects as well. The next section explains how NEAT is combined with IEC to produce NEAT Particles.

III. Approach: NEAT Particles

NEAT Particles consists of five major components: 1) particle systems, 2) ANNs, 3) physics, 4) rendering, and 5) evolution.

A. Particle System Representation

A *particle system* is specified by a *system position* in space and a set of particles. Each individual *particle* is specified by a position in space and its velocity, color, transparency, and size. Particle lifespan proceeds in three phases. At birth particles are introduced into the scene based the system's position and its *generation shape*, which defines the volume within which particles spawn. During a particle's lifetime it changes and moves according to the system's *update function*. Finally, a particle is removed from the system when *time to live* has expired.

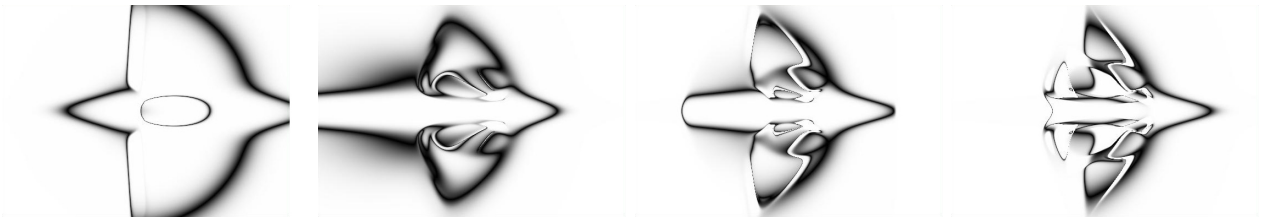
Four classes of particle system are implemented in NEAT Particles (figure 2), each designed to model type of effect common in games:

1) The *generic particle system* (figure 2a) models effects such as fire, smoke, and explosions. Each particle has a position, velocity, color, and size.
2) The *beam system* (figure 2b) models beam or laser-like effects using Bezier curves. Each particle in the beam system is a control point for the Bezier curve and has position, velocity, and color attributes.
3) The *plane system* (figure 2c) warps individual particles into different shapes. A single particle in the plane system is represented by four points, each of which has position, velocity, and color.
4) Finally, in the *trail system* (figure 2d), each particle drops a trail of particles behind it. It behaves similarly to a generic particle system but additionally has an array of trail particles.

By evolving within classes, game designers can create a wide variety of effects for different situations.

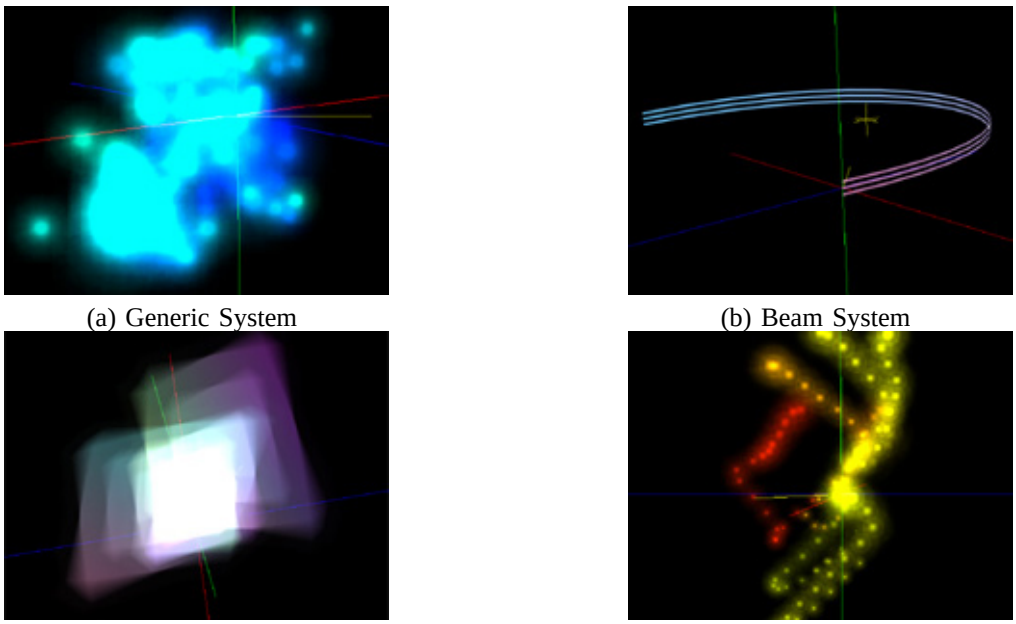
B. ANN Implementation

The ANN for each particle effect dictates the characteristics and behavior of the system. Therefore each particle effect class has its own ANN input and output configuration. In NEAT Particles, the ANN replaces the math and physics rules that must be programmed in traditional particle systems. Every particle in a single system is guided by the same ANN.



(a) (b) (c) (d)

Fig. 1. **IEC Example**. Using Delphi-NEAT Genetic Art (DGNA) [16], a spaceship is evolved [17]. The initial spaceship-like image (a) is evolved from an initial population of random images. An intermediate stage of evolution (b) suggests a tail section, wing section, and nose section. (c) Evolution proceeds and the components become more defined as interesting details become apparent. By the final stage (d), a spaceship model evolves with elegant lines, a nose section, and tail stabilizers.



(a) Generic System (b) Beam System

(c) Plane System (d) Trail System

Fig. 2. **Particle System Classes**. Each particle system class models a different kind of effect. (a) The generic particle system models effects such as fire, smoke, and explosions. (b) The beam system simulates beam and laser-like effects. (c) In the plane system individual particles are warped or stretched. (d) Finally, in the trail system, each particle drops a trail of smaller particles.

However, the ANN is activated separately for each particle. During every frame of animation in NEAT Particles an *update function* is executed during which inputs are loaded into each particle's ANN and each ANN is activated. The outputs of the ANN determine particle behavior for the next frame of animation. An appropriate set of inputs and outputs is assigned uniquely to each effect class.

Figure 3 depicts the ANNs for several NEAT Particles classes. The generic particle system ANN (figure 3a) takes the current position of the particle (P_X, P_Y, P_Z) and distance from the center of the system (D_C) as inputs. Distance from center introduces additional variety into the behavior of particles by allowing them to move in relation to the system center. The outputs are the velocity (V_X, V_Y, V_Z) and color (R, G, B) of the particle for the next frame of animation. The generic particle system produces behaviors suitable for explosion, fire, and smoke effects.

The beam system ANN (figure 3b) controls directed beam effects. To produce twisting beams a Bezier curve is implemented with mobile control points directed by the ANN. The inputs are the position of each Bezier control point (P_X, P_Y, P_Z) and distance of the control point from some target (D_T). In NEAT Particles the target is set at a fixed position away from the system position. In a game however, the target could be the position of an enemy player, or the point at which a weapon is pointed. The outputs are the velocity (V_X, V_Y, V_Z) and color (R, G, B) of the control point for the next frame of animation. Beam systems produce curving, multi-colored beams typically found in futuristic weapons, magic spells, lightning, and energy effects aimed at specific targets.

Each particle in a plane system consists of four points that form a plane that may be warped into different shapes. Since the corners must be coplanar for rendering purposes, the y component of velocity for each corner is fixed. Thus, the inputs to the plane system ANN (figure 3c) are the position of each corner (P_X, P_Z) and the distance from the center of the quad (D_C). The warped quads of plane systems are commonly

used in explosions, engine thrust, and magical weapon glow effects.

Because the differences between the trail system and generic particle system are cosmetic, the trail system ANN (figure 3d) uses the same inputs and outputs as the generic particle system.

While ANN topology largely dictates particle behavior, activation functions play a significant role as well. For simplicity, in this NEAT Particles implementation, all hidden nodes of each ANN have the same activation function; however, that activation function is selected from a set of possibilities. Activation functions with a smooth curve (e.g. sigmoid or sine) generally produce fluid movement patterns and smooth color transitions. The sine and cosine functions produce patterns with cyclic motion, while the tangent function tends to produce patterns with disjoint motion such as teleporting. Functions with linear sections of output range, such as ramp or step, generally contribute to angular, mechanical motion. To illustrate, suppose an ANN generates a particle system with an upward velocity and a color change from red to blue. If the ANN activation function is hyperbolic, the upward motion would likely be fluid and the color transition gradual. In contrast, a cosine activation might produce wavy, rising pattern. Ramp or step functions might produce upward motion and color change that is abrupt or disjoint. In general, different activation functions provide (1) greater variety and (2) patterns within patterns. The effects of different activation functions on particle motion and color are summarized in Table I.

C. Physics

This initial implementation of NEAT Particles is tailored to explosions, beams, and magical effects not generally subject to the effects gravity or collision. Therefore, a linear motion model calculates the position of a particle at time t based on *time elapsed* T since the last frame of animation:

$$P_t = P_{t-1} + SVT, \tag{1}$$

where P_t is the particle's new position, P_{t-1} is the particle's position in the previous animation frame, V is the particle's velocity, and S is a scaling value to adjust the speed of animation.

D. Rendering

NEAT Particles uses the *billboarding* technique [34] to render particles to the screen in generic systems, trail systems, and rotator systems. In billboarding, a texture is mapped onto a simple plane with four corners (i.e. a quad) that faces perpendicular to the camera. The corners of the quad are based upon the center of the particle. Thus, only the position of the particle needs to be stored rather than all four corners of the quad. The textured quads in beam systems and plane systems are not billboarded since they need not face the camera.

There are many ways to optimize particle system rendering including point sprites, level of detail (LOD), batch rendering, and GPU acceleration. NEAT Particles is compatible with all such methods, however they are not explored in this initial implementation.

E. Evolution

Evolution in NEAT Particles proceeds similarly to other IEC applications. The user is initially presented with nine simple, randomized particle systems (figure 4a). Each individual system and its ANN may be inspected by *zooming in* (figure 4b). If the initial population of nine systems is unsatisfactory, a new batch can be generated with the *reset function*. When a suitable starting system is found, the user may begin evolution by spawning a new generation based on the selected system. In the *new generation function*, a population of nine new systems (offspring) is generated from the ANN of the selected system (parent). Offspring ANNs are based on the parent ANN, but with modified connection weights and possibly new nodes and connections, that is, they *complexify* following the NEAT method. Evolution proceeds with repeated rounds of selection and offspring production until the user is satisfied with the results.

IV. Experimental Results

This section shows how NEAT Particles works in practice to produce an appropriate effect with several systems. All particle systems reported were evolved in approximately ten minutes in between 20 and 30 generations..

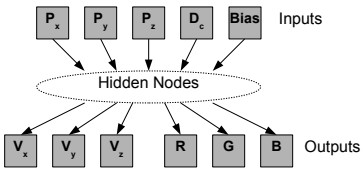
Figure 5 illustrates NEAT Particles interactively evolving an effect for a hypothetical video game. Suppose a particle effect is needed for the wizard spell *Color Spray* that should (1) emit multiple beams in all directions from the wizard's hands that (2) change color as they move and (3) spiral in an orbit pattern as they move away from the wizard. To generate this effect in NEAT Particles, random populations were generated until a suitable starting particle system was found. The initial simple system (figure 5a) was chosen because its color scheme is similar to the desired output and its movement along the axis in both directions is suggestive of potential spiral motion. After some generations of complexification a rough orbital pattern emerged (figure 5b); however, there was not yet sufficient color variation. Several generations later an almost perfect spiral pattern was evolved along with significant color transitions (figure 5c). Finally, a particle system with a wider spiral pattern and brighter colors was achieved (figure 5d), producing a remarkably vibrant rendition of the desired effect.

Similar results were achieved with other systems described in Section III. Figure 6 shows animation frames from additional evolved effects. The top row, 6a though 6d, depicts an evolved *Psychic Scream* effect, in which an imperfect ring of concentric waves radiate from the player. The bottom row, 6e though 6h, illustrates a *Warding Whip* effect, in which a beam-like energy whip lashes out from the player.

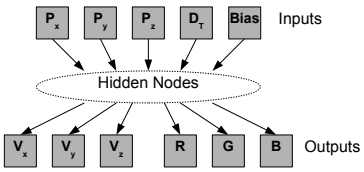
These results demonstrate how NEAT Particles can be used to evolve pleasing effects without user knowledge of programming or mathematics.

V. Performance

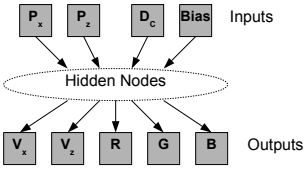
NEAT Particles' computational requirements scale at $O(n)$, where n is the number of particles. The position of each particle is input to the ANN once per frame. Similarly, in



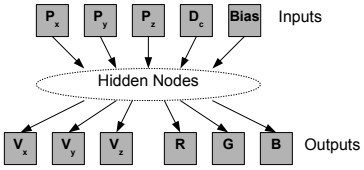
(a) Generic System ANN



(b) Beam System ANN



(c) Plane System ANN



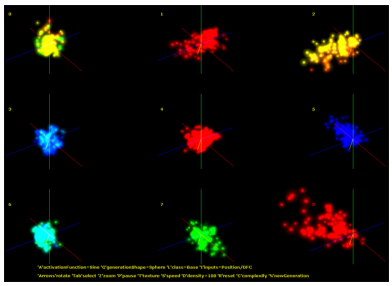
(d) Trail System ANN

Fig. 3. **Particle System ANNs**. To produce a specific range of effects, each particle class ANN uses different inputs and outputs. (a) The generic particle system ANN inputs are the current position of the particle, the distance from the system center, and a bias. The outputs are the particle velocity and color. (b) The beam system ANN inputs are the current position of a Bezier control point, the distance of the control point from the target, and a bias. The outputs are the control point velocity and color. (c) Since the four corners of a plane system particle must remain coplanar, the y component of each corner's velocity is fixed. Therefore the plane system ANN inputs include the position of each corner, the distance of each corner from the center of the plane, and a bias. Outputs are the velocity and color of each corner. (d) The trail system ANN has inputs and outputs similar to the generic system.

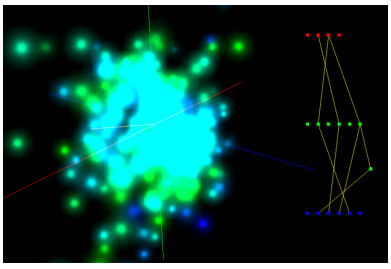
TABLE I

EFFECTS OF ACTIVATION FUNCTIONS ON PARTICLE SYSTEM BEHAVIOR

Function	**Movement Patterns**	**Color Transitions**	**Applications**
bipolar sigmoid, hyperbolic	fluid, organic motion	smooth color transitions	explosions, magic spells
sine, cosine	fluid, cyclic motion	smooth, cyclic color transitions	fire, smoke, water
tangent	disjoint motion teleportation	flashing colors, no transitions	sparks, fireworks
ramp, step	angular, linear motion	flashing colors, no transitions	robotic movement, seeking missiles



(a) Main Interface



(b) Zoom Mode

Fig. 4. **NEAT Particles Interface**. In the main interface (a), the user is presented with 9 particle systems. Variables such as activation function, generation shape, and inputs are displayed on the bottom. In zoom mode (b), a single particle system and its ANN may be inspected. The top row of nodes are inputs and the bottom row are outputs.

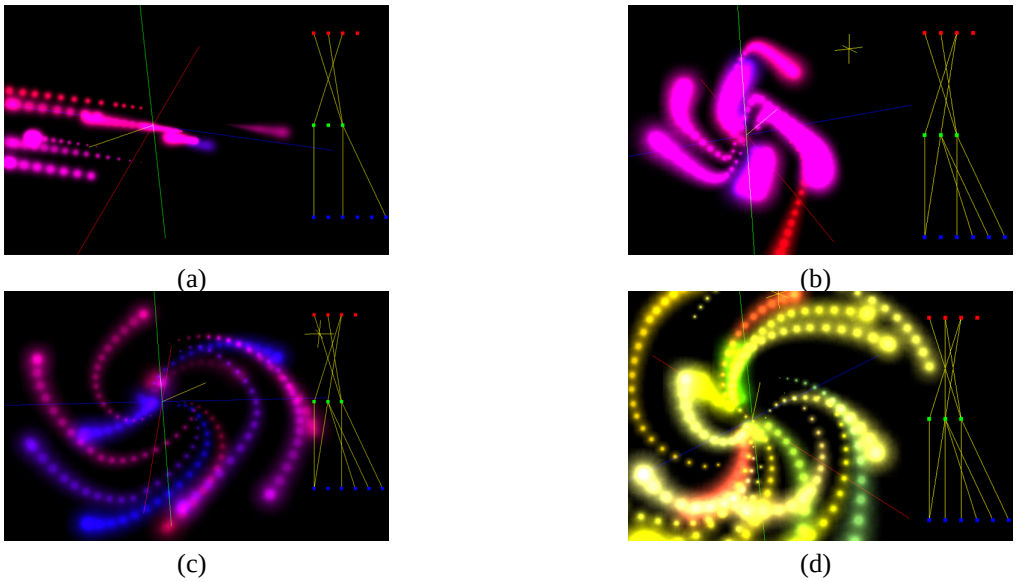


Fig. 5. **Evolution Example**. This series of images shows the evolution of a *Color Spray* effect using a trail system. (a) An initial particle system is selected to start evolution. (b) After some generations a spiral pattern emerges. (c) Soon a full spiral pattern develops along with prominent color transitions. (d) A wider spiral pattern and brighter color scheme is selected as the final spell effect.

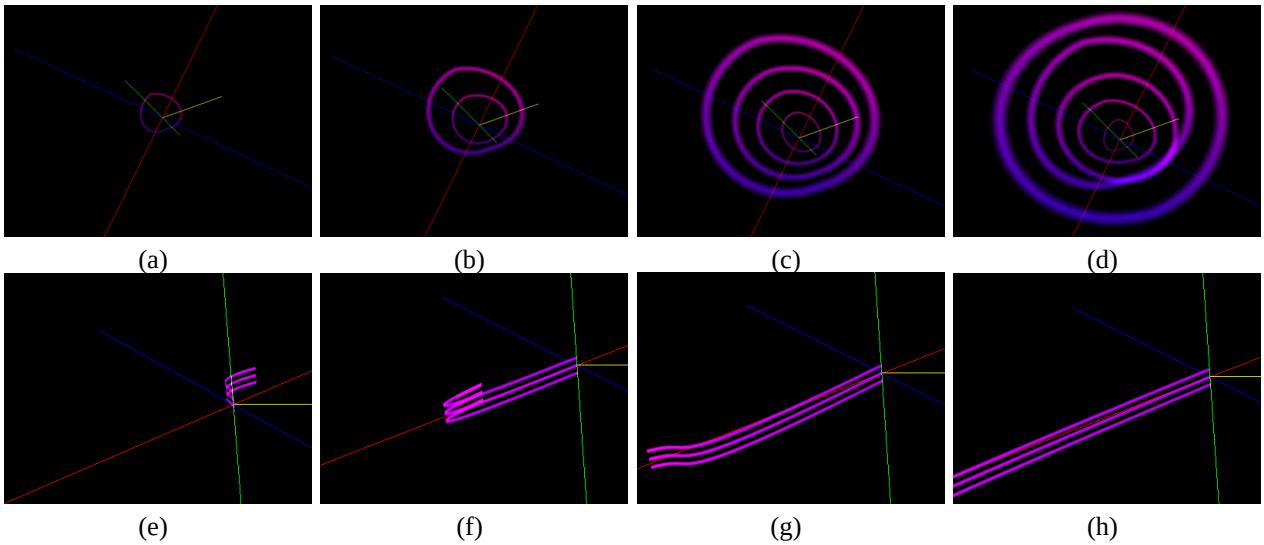


Fig. 6. **Example Evolved Systems**. Each series shows consecutive frames in animations from two evolved effects. Images (a) through (d) show the expanding rings from a *Psychic Scream* effect evolved with the plane system. Images (e) though (h) are frames from a *Warding Whip* effect, evolved with the beam system.

traditional particle systems each particle passes through an *update function* once per frame. While the complexity of the ANN increases with the complexity of the effect, the same is likely true for traditional formulations. Thus NEAT Particles can be expected to perform comparably to traditional particle systems.

VI. Discussion and Future Work

The purpose of many IEC systems is simply to interactively explore a search space. In contrast, the objective of NEAT Particles is to generate useful content. Therefore, NEAT Particles constrains the search space for the user. The search space should be large enough to explore many interesting and useful results, yet not so large that producing useful output is too time-consuming. The class system implemented in NEAT Particles provides such constraint.

Besides intentionally evolving specific particle systems that the user has in mind, the IEC approach of NEAT Particles acts also as a concept generation tool. While evolving a specific effect, the user often generates novel, useful effects that were not initially planned. Thus an additional advantage of NEAT Particles over traditional particle system implementations is that it may act as an idea or concept generator.

Future research will focus on the continued exploration of inputs, outputs, activation functions, and other variables to evolve new types of particle systems. Fire, smoke, water,

electricity, and other realistic effects will require specialized sets of ANN input and outputs.

In addition to acting as a design and rendering system for effects, NEAT Particles potentially applies to other game-related applications. For example, novel content is coveted in games. With an in-game NEAT Particles system, each spell or character could be associated with a unique ANN. A player might indirectly affect the attributes of his or her ANN in various ways. For example, by gaining levels or researching new spells the player might gain or encounter new effects. In this manner players could potentially acquire signature spell effects, like wizards in popular fantasy novels, and thereby implicitly and collectively search the space of effects. Spells could also be combined through crossover, generating new effects. Content can potentially be evolved in real time in this way with the real-time NEAT (rtNEAT) method [33], which has proven effective in video games in the past. Automatic content generation is a Holy Grail in the game industry, and evolutionary content generation is an intriguing and untested solution.

VII. Conclusions

NEAT Particles is a design, representation, and animation method for particle systems that allows particle effects to be evolved with NEAT and IEC rather than hand coded. The advantages provided by NEAT Particles over traditional particle system implementations are that (1) it allows users to produce complex effects without mathematical or programming knowledge, (2) a wider range of effects can be produced, and (3) it can generate novel concepts. NEAT Particles is designed to produce effects appropriate for real-time games and simulations; however its most significant implication may be the generation of novel content during game play itself.

References

[1] J. Lander, "The ocean spray in your face," *Game Developer Magazine*, July 1997.

[2] J. V. der Berg, "Building an advanced particle system," *Game Developer Magazine*, March 2000.

[3] W. Reeves, "Particle systems: A technique for modeling a class of fuzzy objects," *ACM Transactions on Computer Graphics*, vol. 17, no. 3, 1983.

[4] ——, "Approximate and probabilistic algorithms for shading and rendering structured particle systems," *ACM Transactions on Computer Graphics*, vol. 19, no. 3, 1985.

[5] D. Breen, "A particle based model for simulating draping behavior of woven cloth," *Textile Research Journal*, vol. 64, no. 11, pp. 663–685, 1994.

[6] B. Eberhardt, A. Weber, and W. Strasser, "A fast, flexible, particle-system model for cloth draping," *IEEE Transactions on Computer Graphics and Applications*, vol. 16, no. 5, 1996.

[7] D. Obrien, S. Fisher, and M. Lin, "Automatic simplification of particle system dynamics," 2001.

[8] M. Muller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[9] C. Reynolds, "Steering behaviors of autonomous characters," in *Proceedings of the Game Developers Conference*, 1999.

[10] ——, "Flocks, herds, and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987.

[11] H. Takagi, "Interactive evolutionary computation: Fusion of the capacities of EC optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.

[12] R. Dawkins, *The Blind Watchmaker.* Essex, U.K.: Longman, 1986.

[13] A. Lindenmayer, "Mathematical models for cellular interaction in development parts I and II," *Journal of Theoretical Biology*, vol. 18, pp. 280–299 and 300–315, 1968.

[14] S. Todd and W. Latham, *Evolutionary Design by Computers*. Morgan Kaufman, 1999.

[15] T. Unemi, "Genetic algorithms and computer graphic arts," *Journal of Japan Society for Artificial Intelligence*, vol. 9, no. 4, 1994.

[16] M. Fagerlund, "NEAT based genetic art," 2006. [Online]. Available: http://www.hypeskeptic.com/mattias/GeneticArt/

[17] K. O. Stanley, "Exploiting regularity without development," in *Proceedings of the AAAI Fall Symposium on Developmental Systems*. Menlo Park, CA: AAAI Press, 2006.

[18] F. Gomez and R. Miikkulainen, "Solving non-Markovian control tasks with neuroevolution," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. San Francisco: Kaufmann, 1999, pp. 1356–1361.

[19] N. Saravanan and D. B. Fogel, "Evolving neural control systems," *IEEE Expert*, pp. 23–27, June 1995.

[20] A. Wieland, "Evolving neural network controllers for unstable systems," in *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA). Piscataway, NJ: IEEE, 1991, pp. 667–673.

[21] H. Braun and J. Weisbrod, "Evolving feedforward neural networks," in *Proceedings of ANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms*. Berlin: Springer, 1993.

[22] J. C. F. Pujol and R. Poli, "Evolution of the topology and the weights of neural networks using genetic programming with a dual representation," School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK, Tech. Rep. CSRP-97-7, 1997.

[23] J. C. Bongard and R. Pfeifer, "Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny," in *Proceedings of the Genetic and Evolutionary Computation Conference*, L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds. San Francisco: Kaufmann, 2001, pp. 829–836.

[24] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, 1996, pp. 81–89.

[25] B.-T. Zhang and H. Muhlenbein, "Evolving optimal neural networks using genetic algorithms with Occam's razor," *Complex Systems*, vol. 7, pp. 199–220, 1993.

[26] D. W. Opitz and J. W. Shavlik, "Connectionist theory refinement: Genetically searching the space of network topologies," *Journal of Artificial Intelligence Research*, vol. 6, pp. 177–209, 1997.

[27] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[29] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[30] ——, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, pp. 63–100, 2004.

[31] N. J. Radcliffe, "Genetic set recombination and its application to neural network topology optimization," *Neural computing and applications*, vol. 1, no. 1, pp. 67–90, 1993.

[32] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 54–65, 1993.

[33] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, vol. 9, no. 6, pp. 653–668, 2005.

[34] A. Fernandes, "Lighthouse 3d billboarding tutorial," 2006. [Online]. Available: http://www.lighthouse3d.com/opengl/billboarding/