Interactive Evolution of Particle Systems for Computer Graphics and Animation

Erin J. Hastings, Ratan K. 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
To appear in: *IEEE Transactions on Evolutionary Computation*, New York: IEEE Press, 2009.

***Abstract*—Interactive Evolutionary Computation (IEC) creates the intriguing possibility that a large variety of useful content can be produced quickly and easily for practical computer graphics and gaming applications. To show that IEC can produce such content, this paper applies IEC to particle system effects, which are the de facto method in computer graphics for generating fire, smoke, explosions, electricity, water, and many other special effects. While particle systems are capable of producing a broad array of effects, they require substantial mathematical and programming knowledge to produce. Therefore, efficient particle system generation tools are required for content developers to produce special effects in a timely manner. This paper details the design, representation, and animation of particle systems via two IEC tools called NEAT Particles and NEAT Projectiles. Both tools evolve artificial neural networks (ANN) with the NeuroEvolution of Augmenting Topologies (NEAT) method to control the behavior of particles. NEAT Particles evolves general-purpose particle effects, whereas NEAT Projectiles specializes in evolving particle weapon effects for video games. The primary advantage of this NEAT-based IEC approach is to decouple the creation of new effects from mathematics and programming, enabling content developers without programming knowledge to produce complex effects. Furthermore, it allows content designers to produce a broader range of effects than typical development tools. Finally, it acts as a concept generator, allowing content creators to interactively and efficiently explore the space of possible effects. Both NEAT Particles and NEAT Projectiles demonstrate how IEC can evolve useful content for graphical media and games, and are together a step toward the larger goal of automated content generation.**

***Index Terms*—Interactive Evolutionary Computation, IEC, NeuroEvolution of Augmenting Topologies, NEAT, particle systems.**

I. Introduction

Content generation means creating models, levels, textures, animations, lighting, etc. for computer graphics in games, movies, and television. For media developers, content generation consumes significant time and money to produce today's complex graphics and game content [1], [2]. In part to address this problem, in the video game industry, it is becoming increasingly popular to provide extensive character customization tools within games and to distribute tools that allow users to create their own content outside of the game as well [3], [4], [5]. Furthermore, there is a new trend towards *content generation tools as games themselves*, that is, *sandbox games* such as The Sims[1], Second Life[2], and Spore[3]. These games feature creating houses, vehicles, clothing, and creatures as primary game play features [6]. Thus, there is a growing need for powerful and user-friendly content generation tools both to reduce the content bottleneck and further empower users.

An emerging approach to this problem is *automated content generation* through *Interactive Evolutionary Computation* (IEC), that is, automating content creation though user interaction. This paper presents such an automated content generation method for *particle systems*, demonstrating the promise of IEC for practical content generation.

Particle systems are ubiquitous in computer graphics for producing animated effects such as fire, smoke, clouds, gunfire, water, cloth, explosions, magic, lighting, electricity, flocking, and many others [7], [8]. They 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 such rule sets are often complex, creating each new effect requires considerable mathematics and programming knowledge. For example, consider designing a *a spherical flame shield of pulsing colors* effect for a futuristic video game or movie. Alternatively, consider designing a *particle weapon effect that fires multiple curving arcs toward the target*. In current practice, the precise mechanics for either scenario must be hand coded by a programmer. To simplify design, particle effect packages typically provide developers with a set of particle system classes, each suitable for a certain type of effect. Content developers manipulate the parameters of each particle system class by hand to produce the desired effect. The problem is that there is no way to efficiently explore the range of effects within each class.

To address this problem, this paper presents a new design, representation, and animation approach for particle systems in which (1) artificial neural networks (ANNs) control particle system behavior, (2) the *NeuroEvolution of Augmenting Topologies* (NEAT) method [9], [10] produces sophisticated particle system behaviors by evolving increasingly complex ANNs, and (3) evolution is guided by user preference through an IEC interface.

Two prototype systems are discussed, NEAT Particles, a general-purpose particle effect generator, and NEAT Projectiles, which is specialized to evolve particle weapon effects

[1]Copyright 2007 Electronic Arts, http://thesims.ea.com/
[2]Copyright 2003 Linden Research Inc., http://secondlife.com/
[3]Copyright 2007 Electronic Arts, http://www.spore.com/

for video games. Both systems interactively evolve ANNs with NEAT to control the motion and appearance of particles. An IEC interface provides a user-friendly method to evolve unique content.

In this way, NEAT Particles shows how IEC can enable practical content generation that provides an easy alternative to current, potentially cumbersome practice. In particular, NEAT Particles and NEAT Projectiles (1) enable users without programming or artistic skill to evolve unique particle system effects through a simple interface, (2) allow developers to evolve a broad range of effects within each particle class, and (3) serve as concept generators, enabling novel effect types to be easily discovered. By allowing users to evolve particle behavior without knowledge of physics or programming, NEAT Particles and NEAT Projectiles are a step toward the larger goal of automated content generation for games, simulations, and movies.

II. Background

This section reviews particle systems, IEC, and NEAT, which are components of NEAT Particles and NEAT Projectiles.

A. Particle Systems

The first computer-generated particle system in commercial computer graphics, called the *Genesis Effect*, appeared in Star Trek II: The Wrath of Khan[1] [11] (figure 1a). Soon after, particle systems effects became widespread on television as well (figure 1b). Nearly all modern video games include a particle system engine [7], [8]; special effects in games such as magical spells (figure 1c) and futuristic weapons (figure 1d) are usually implemented with particle systems.

In addition to diffuse phenomena such as fire, smoke, and explosions, particle systems can also model concrete objects such as dense trees in a forest [12], folded cloth and fabric [13], [14], and simulated fluid motion [15], [16]. Realistic particle movement is often achieved by simulating real-world physics [17]. At a more abstract level, particle systems can simulate animal and insect flocking as well as swarming behavior [18]. The prevalence and diversity of particle system applications demonstrates their importance to computer graphics in modern media and games.

B. Interactive Evolutionary Computation (IEC)

IEC is an approach to evolutionary computation (EC) in which human evaluation replaces the fitness function [19]. A typical IEC application presents to the user the current generation of content. The user then interactively determines which members of the population will reproduce and the IEC application automatically generates the next generation of content based on the user's input. Through repeated rounds of content generation and fitness assignment, IEC enables unique content to evolve that suits the user's preferences. In some cases such content cannot be discovered or created in any other way.

IEC aids especially in evolving content for which fitness functions would be difficult or impossible to formalize (e.g. for aesthetic appeal). Thus, graphical content generation is a common application of IEC [20], [21], [22], [23], [24], [25].

IEC was first introduced in Biomorphs (figure 2a), which aims to illustrate theories about natural evolution [20]. Biomorphs are patterns encoded as *Lindenmayer Systems* (L-systems) [26], i.e. grammars that specify the order in which a set of replacement rules are carried out. Abstract figures that resemble animals or plants are interactively evolved in this manner.

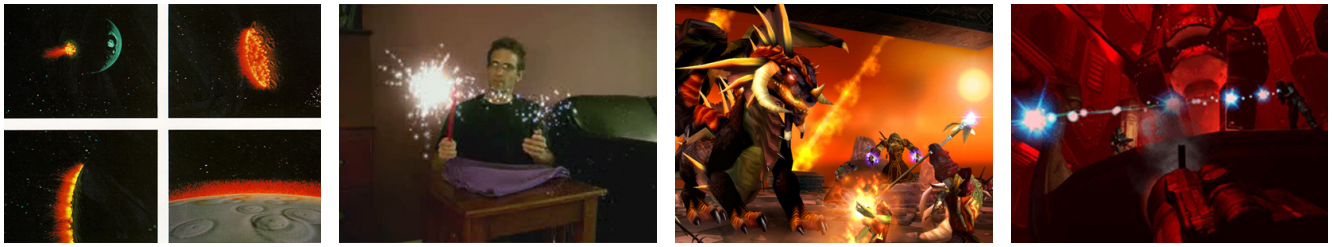
Representations in *genetic art* (i.e. IEC applied to art) often vary, including linear or non-linear functions, fractals, and automata. Some notable examples include (1) Mutator [21], a cartoon and facial animation system, (2) SBART (figure 2b) [22], a two-dimensional art exploration tool, (3) a tool that evolves implicit surface models such as fruits and pots (figure 2c) [23], [27], [28], and (4) a system for evolving quadric models used as machine components (figure 2d) [24].

Figure 3 illustrates IEC's content generation capabilities. The figure shows a progression of four user-selected parents in the evolution of a spaceship with a genetic art tool [25], [29], [30]. In the example, the user starts by selecting a simple image that vaguely resembles what they wish to create and continues to evolve more complex images through selection until satisfied with the result. The sequence of images demonstrates the potential of IEC as an engine for content generation. These images, from Delphi NEAT Genetic Art (DNGA), are produced by ANNs evolved by NEAT, which is discussed in the next section.

C. NeuroEvolution of Augmenting Topologies

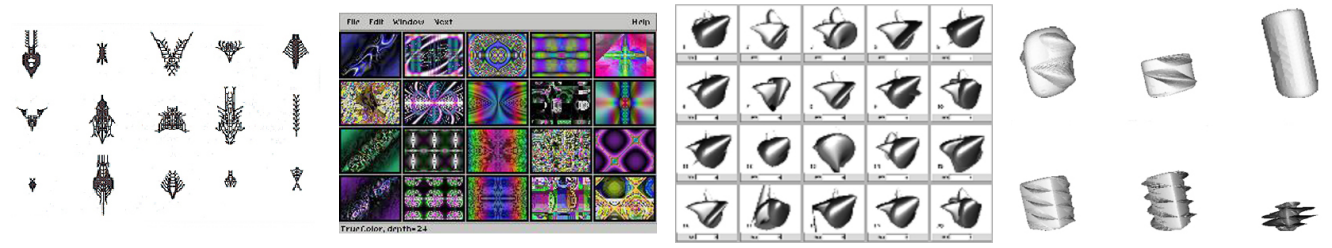
The NEAT method was originally developed to solve control and sequential decision tasks. The ANNs evolved with NEAT can control agents that select actions based on their sensory inputs. While previous methods that evolved ANNs (i.e. neuroevolution methods) evolved either fixed topology networks [31], [32], [33], or arbitrary random-topology networks [34], [35], [36], [37], [38], [39], [40], NEAT is the first to begin evolution with a population of small, simple networks and *complexify* the network topology into diverse species 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 [41], the recurrent networks that evolve in NEAT are robust in continuous domains and in domains that require memory, making many applications possible. In this paper, particle systems are controlled by ANNs evolved by NEAT. NEAT is well-suited to this task because (1) it is a proven method for evolving ANNs, and (2) it was employed successfully in prior genetic art applications [25]. This section briefly reviews the NEAT method; Stanley and Miikkulainen [9], [10] provide complete descriptions.

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



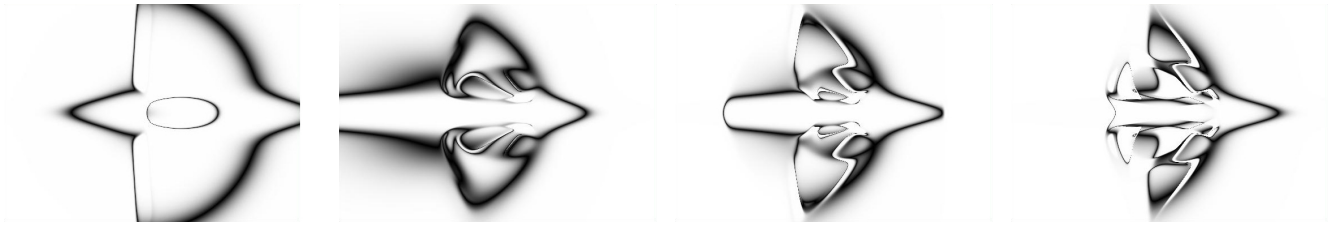
(a) (b) (c) (d)

Fig. 1. **Particle System Examples**. Particle systems are ubiquitous in computer graphics for both movies and games. (a) *The Genesis Effect* from Star Trek 2: The Wrath of Khan[1] [11] is one of the earliest applications of particle systems in commercial computer graphics. (b) Particle systems appear extensively in television, such as the effects in this live footage produced with the ParticleIllusion[2] software. (c) Magical spells, glowing weapons, and dragon breath from the World of Warcraft[3] video game are implemented with particle systems. (d) Finally, futuristic weapons are usually implemented with particle systems, such as in this shot of the Doom 3[4] video game.



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

Fig. 2. **IEC Art Examples**. This figure depicts examples several IEC program interfaces. (a) Biomorphs are simple plant or animal-like figures evolved with L-systems; they are one of the earliest examples of IEC [20]. (b) SBART evolves complex 2D images [22]. (c) A creative design system [23], [27], [28] evolves simple three-dimensional objects such as green peppers. (d) Finally, an IEC interface [24] generates three-dimensional machine components. These examples illustrate typical IEC interfaces and demonstrate the range of content that can be evolved.



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

Fig. 3. **IEC Evolution Example**. In this example a spaceship is interactively evolved with DelphiNEAT-based Genetic Art (DNGA) [25], [29], [30]. 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) As evolution proceeds the components become more defined and interesting details become apparent. By the final stage (d), a spaceship model evolves with elegant lines, a nose section, and tail stabilizers. This sequence illustrates how complex digital art can be evolved by user preference.

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 [42] in an evolving population.

Second, traditionally 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. However, in this work, because a human performs selection rather than an automated process, the usual speciation procedure in NEAT is unecessary.

Third, unlike other systems that evolve network topologies and weights [43], [37], [40], [38], 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

[1]Copyright 1982 Paramount Pictures, http://www.paramount.com/
[2]Copyright 2006 Wondertouch Software, http://www.wondertouch.com/
[3]Copyright 2004 Blizzard Entertainment, http://www.blizzard.com/
[4]Copyright 2005 Id Software, http://www.idsoftware.com/

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. For IEC, complexification means that content can become more elaborate and intricate over generations.

Since its inception, NEAT has been applied to a broad array of research areas [44], [45], [25], [46]. Most notable for the approach in this paper is NERO [45], an interactive, real-time war game in which ANN-controlled soldiers are evolved. 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 a broad array of effects with NEAT Particles.

III. Approach - NEAT Particles

NEAT Particles combines IEC and NEAT to enable users to evolve complex particle systems. ANNs control particle system behavior, NEAT evolves the ANNs, and an IEC interface gives the user control over evolution. 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 an absolute *system position* in three-dimensional space and a set of particles. Each individual particle is defined by its position, velocity, color, and size. Particle lifespan unfolds in three phases.

1) At birth particles are introduced into space relative to system position and according to a *generation shape* (figure 4) that defines the volume within which new particles may spawn.
2) During its lifetime, each particle changes and moves according to a set of rules, i.e. an *update function.*
3) Each particle dies, and is removed from the system, when its *time to live* has expired.

NEAT Particles effects are divided into *classes* for two primary reasons: (1) user convenience and (2) performance. First, to evolve effects in a reasonable time frame, it is helpful to divide the search space for the user. Second, effects may be highly dependent upon certain variables, and unaffected by other variables. For performance reasons, it is not feasible to evolve all possible particle variables simultaneously. A better approach is implemented in NEAT Particles, in which only key variables are evolved in each particle effect class. Five particle system classes are implemented in NEAT Particles to facilitate evolving a variety of common types of effects.

- The *generic system* (figure 5a) models effects such as fire, smoke, and explosions. Each particle has a position, velocity, color, and size.
- The *plane system* (figure 5b) warps individual particles into different shapes for bright flashes, lens flares, and engine exhaust effects. A single particle in the plane system is represented by four points, each of which has position, velocity, and color.
- The *beam system* (figure 5c) models beam, laser, or electricity effects using Bezier curves. Each particle in the beam system is a control point for the Bezier curve, including its position, velocity, and color attributes.
- The *rotator system* (figure 5d) models effects whose primary behavior is orbital rotation, common in many applications. Each particle in a rotator system has rotation, position, and color attributes.
- The *trail system* (figure 5e) behaves similarly to the generic system, but additionally drops a trail of static particles behind each moving particle.

By providing an array of particle system classes, NEAT Particles allows designers to evolve a substantial variety of effects while conveniently constraining the search space during any particular run.

B. Artificial Neural Network Implementation

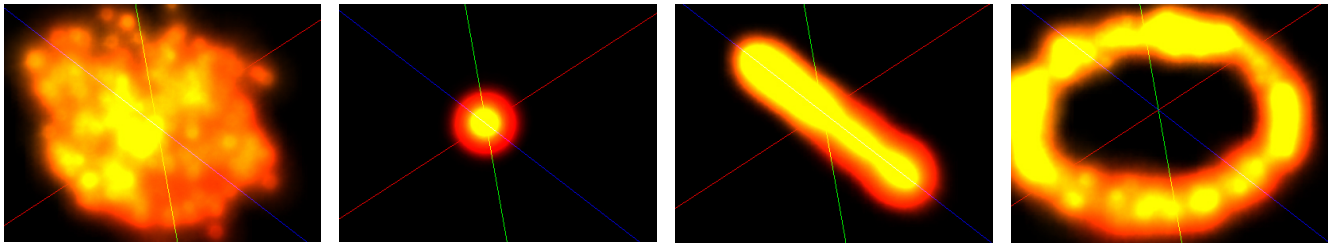
ANNs control particle behavior in NEAT Particles for two primary reasons. First, ANNs are a proven method for autonomous control. Second, NEAT is a powerful method for evolving ANNs for control and sequential decision tasks.

An important question is why evolving ANNs is preferable to directly evolving the variables of a traditional particle system implementation. While feasible, such an approach still ultimately relies on hand-coded rules (which constitute such systems), which thus depend on programmers to make the search possible. For example, in a traditional particle system implementation, when a new effect class is needed it requires programmers to define the effect parameters (e.g. color change, motion pattern physics, etc.). In contrast, in NEAT Particles the effects of any class are represented by the same structure: ANNs.

The ANN for each particle effect dictates the characteristics and behavior of the system. Therefore, each particle effect class includes 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. Because special effects in most movie and game graphics need to be visually appealing yet not necessarily physically plausible, ANNs do not need to equate to physically realistic models. However, evolved ANN-controlled particle behaviors (e.g. *spin in a spiral while changing color from green to orange*) are still compatible with rules in physically accurate particle simulations such as gravity, friction, or collision.

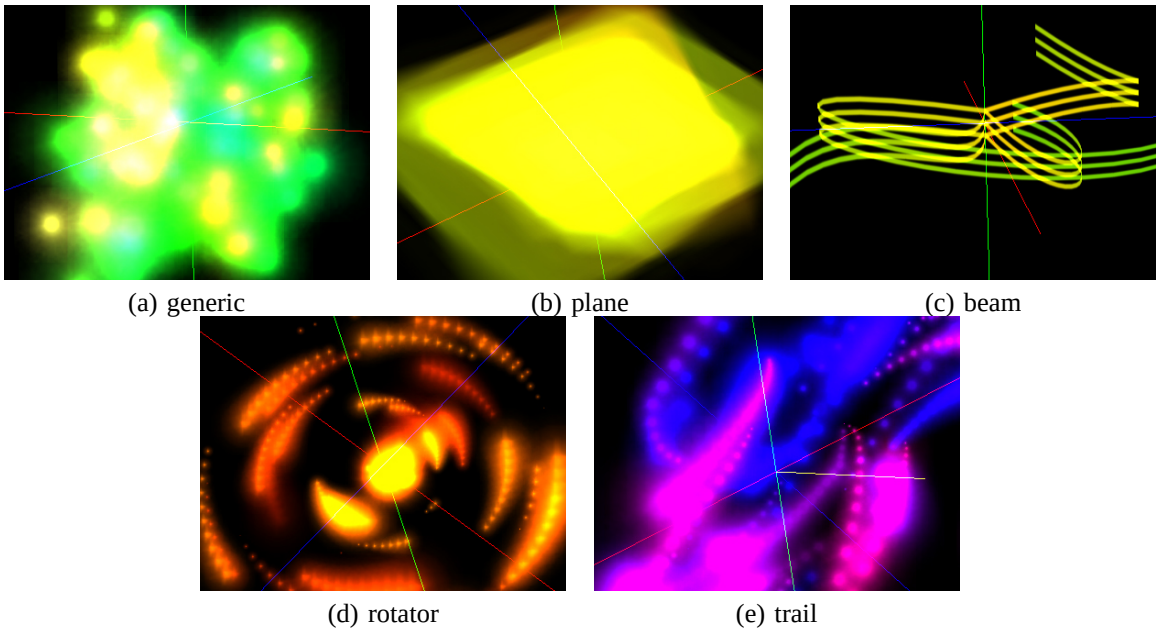
Every particle in a system is guided by the same ANN. However, the ANN is activated separately for each particle. During every frame of animation in NEAT Particles an update function (figure 6) is executed that (1) loads inputs, (2) activates the ANN, and (3) reads outputs. The ANN outputs determine particle behavior for the current frame of animation. An appropriate set of inputs and outputs is associated with each effect class as follows.

The primary inputs in NEAT Particles are position and distance from center of the system. The main outputs are velocity and color. These are good inputs and outputs because they can encode significant variety over the long term. However, because animation happens in real-time, the change in position



(a) sphere (b) point (c) line (d) circle

Fig. 4. **Generation Shapes**. A particle system's generation shape defines the region in which new particles spawn. (a) Spherical generation produces area effects such as smoke and explosions. (b) Point generation facilitates effects that are attached to specific points on objects, such as vehicle thrust and muzzle flash. (c) Line generation commonly produces effects attached to characters or melee weapons, such as glowing swords. (d) Circular generation enables effects that surround objects, such as energy fields.



(a) generic (b) plane (c) beam

(d) rotator (e) trail

Fig. 5. **Particle System Classes**. Predefined classes constrain the search space for designers. (a) The generic particle system models effects such as fire, smoke, and explosions. (b) The plane system warps and stretches individual particles for flashes, lens flares, and other effects. (c) The beam system simulates beam, laser, or electricity effects. (d) The rotator system models effects based on orbital rotation common in explosions, energy, and magic. (e) The trail system is similar to the generic system; however each particle drops a trail of smaller particles. Trail systems commonly implement magic, energy, weapon, and exhaust effects.

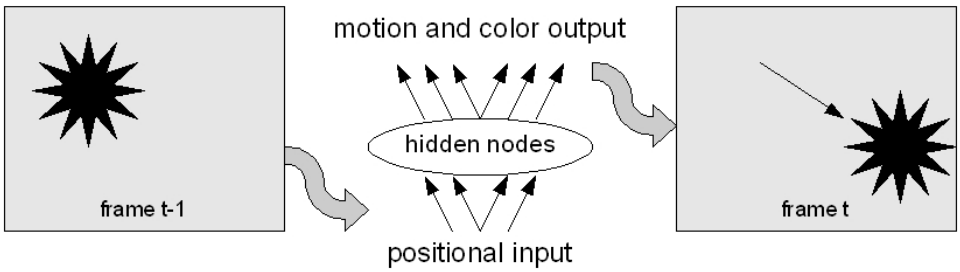


Fig. 6. **Update Function**. Every frame of animation, each particle passes through an update function to compute velocity and color for that frame. Suppose animation for a particle is being computed at frame t (on the right). The particle's position and distance from center in the previous frame $t-1$ (on the left) are input into the particle system ANN. After the ANN is activated, its outputs are interpreted as velocity and color at frame t. The high frame rate of real-time animation produces small position changes; thus animation and color change is fluid. Over the long term; however, position changes are large, producing a variety of patterns and behaviors.

and distance from center are small from one frame to the next, producing incremental changes that look smooth.

The generic particle system ANN (figure 7a) 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 the potential for symmetry by allowing particles 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 explosions, fire, and smoke effects.

Each particle in the plane system consists four co-planar points that may be warped into different shapes. Because 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 7b) are the position of each corner (p_x, p_z) and the distance from the center of the plane (d_c). The warped quads of plane systems are commonly found in explosions, engine thrust, and glow effects.

The beam system ANN (figure 7c) 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 a the center of the system (d_c). 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.

The rotator system (figure 7d) enables evolving rotation-based effects. The inputs to the ANN are particle position (p_x, p_y, p_z) and distance from the center of the system (d_c). The outputs are rotation around the x, y, and z axes (r_x, r_y, r_z) and color (R, G, B). Rotation-based particle systems are common in explosions, halos, and energy effects.

The trail system behaves similarly to the generic system yet provides a more complex visual effect by periodically dropping stationary particles that shrink and fade out. Therefore, the trail system ANN takes the same inputs and emits the same outputs as the generic ANN. Trail systems are convenient because they provide a computationally inexpensive form of motion blur or visual trail behind moving objects.

ANNs control particle behavior and ANN input/outputs divide effects into classes, which shrinks the search space for users. While ANN topology and weights significantly contribute to particle behavior, activation functions within each node play an important role as well; they are detailed in the next section.

C. Activation Functions

Unlike traditional ANNs, NEAT Particles ANN hidden nodes and output nodes contain an activation function selected from a set of eight possibilities (figure 8). Theoretically, ANNs with a single activation function can evolve any behavior [47]; however, multiple activation functions are preferable in NEAT Particles because the user can obtain variety more quickly and thereby evolve toward the intended effect sooner.

D. Physics

Each frame of animation, after the ANN is activated, the velocity for each particle is determined by the outputs. To animate a particle each frame (i.e. move the particle through space) a linear motion model calculates the position of the particle at time t based on *time elapsed* τ since the last frame of animation:

$$P_t = P_{t-1} + V\tau s, \tag{1}$$

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

E. Rendering

NEAT Particles renders particles to the screen with *billboarding* [48], a technique in which two-dimensional bitmap textures are mapped onto a plane (i.e. a *quad*) that faces perpendicular to the camera. The corners of the quad are offsets from the particle position. By facing the quad toward the camera the billboarding method convincingly conveys the illusion of translucent three-dimensional particles in space.

The billboarding technique is implemented in NEAT Particles because it is the most common and versatile method to render particles. An alternative particle rendering method is point sprites [49]; however, they do not allow arbitrary warping of particle shape required for the beam and plane systems.

There are several ways to optimize particle system rendering including level of detail (LOD) [15], batch rendering [49], and GPU acceleration [50]. NEAT Particles is compatible with all such methods; however they are not explored in this implementation.

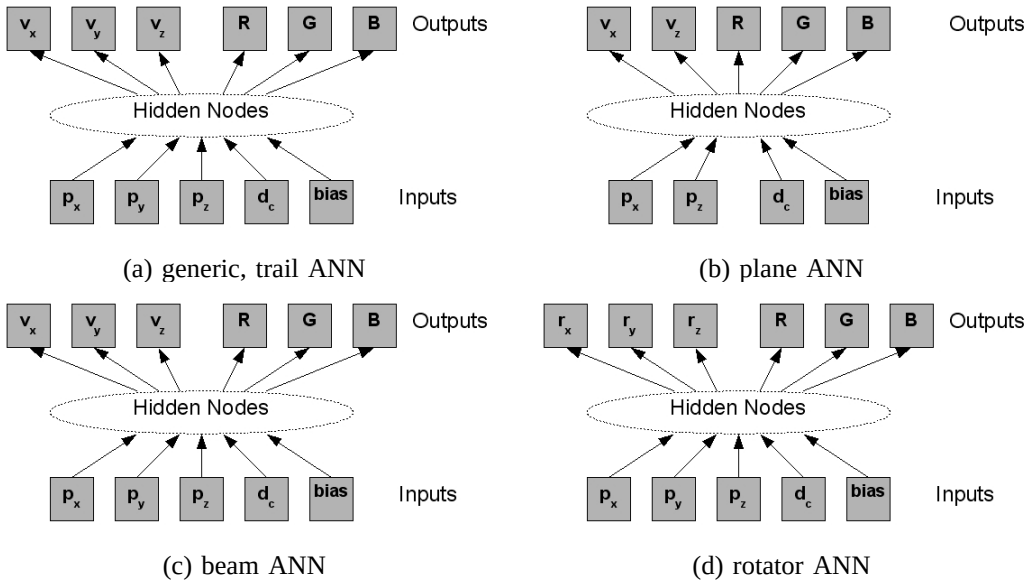
The next section explains how particle classes, ANNs, physics, and rendering combine to enable particle effect evolution.

F. Evolution

Evolution in NEAT Particles follows a similar procedure to other IEC applications (Section II-B). The user is initially presented a population of nine randomized particle systems represented by simple ANNs (figure 9a). Each individual system and its ANN can be inspected by *zooming in* on the system (figure 9b). If the initial population of nine systems is unsatisfactory, a new random batch of effects can be generated by restarting evolution.

The user begins evolution by selecting a single system from the population to spawn a new generation. A population of eight new systems (i.e. offspring) is then generated from the ANN of the selected system (i.e. parent) by mutating its connection weights and possibly adding new nodes and connections. That is, offspring complexify following the NEAT method. Evolution proceeds with repeated rounds of selection and offspring production until the user is satisfied with the results. If the user is unsatisfied with an entire new generation, an *undo* function recalls the previous generation.

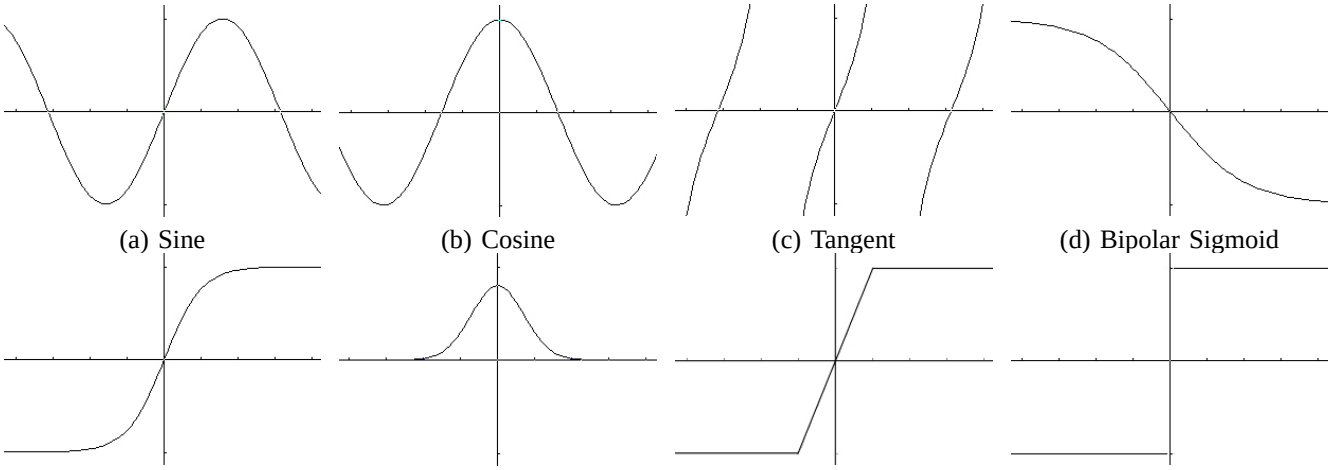
Specifically, each new generation preserves the parent exactly and the other eight members of the population are



(a) generic, trail ANN (b) plane ANN

(c) beam ANN (d) rotator ANN

Fig. 7. **Particle System ANNs**. To produce a specific range of effects, each particle class ANN uses different inputs (i.e. position and distance from center) and outputs (i.e. velocity, color, and rotation), which are shown for (a) the generic and trail particle system, (b) the beam particle system, (c) the plane particle system, and (d) the rotator particle system. The beam system ANN appears similar to the generic and trail system ANNs; however a generic system ANN controls individual particles, whereas a beam system ANN controls Bezier curve control points. The plane system ANN controls four corners of a warped quad, and the rotator system ANN controls individual particle rotation. Each ANN is evolved in NEAT Particles to connect the inputs to the outputs of each class.



(a) Sine (b) Cosine (c) Tangent (d) Bipolar Sigmoid

(e) Hyperbolic (f) Gaussian (g) Ramp (h) Step

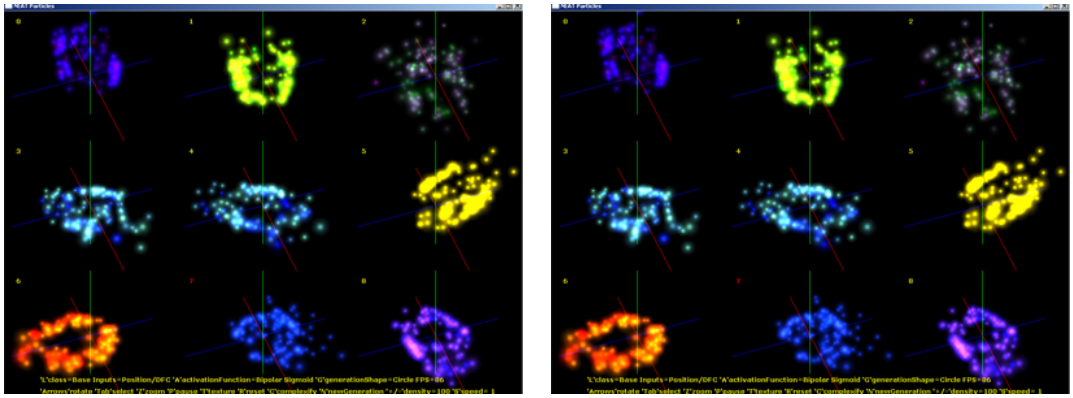
Fig. 8. **ANN Activation Functions**. In NEAT Particles all ANN hidden nodes and output nodes are randomly assigned one of eight activation functions: (a) sine, (b) cosine, (c) tangent, (d) bipolar sigmoid ($\frac{[1-exp(-x)]}{[1+exp(-x)]}$), (e) hyperbolic ($\frac{[e^{(x)}-e^{(-x)}]}{[e^{(x)}+e^{(-x)}]}$), (f) Gaussian ($[\frac{1}{\sqrt{(0.5*PI)}}]*e^{(-x^2)}$), (g) ramp ($x=\begin{vmatrix}-1 & if(x<-1)\\ 1 & if(x>1)\end{vmatrix}$), or (h) step ($x=\begin{vmatrix}-1 & if(x<0)\\ 1 & if(x\geq 0)\end{vmatrix}$).

mutated from the parent. For each offspring, a uniformly random number of connections (between one and the number of connections in the network) are mutated by a uniformly random value between -0.5 and 0.5. Adding new nodes and connections is controlled by separate mutation rates. The probability of adding a new connection is 0.3 and the probability of adding a new node is 0.2. New nodes are assigned a random activation function and connected into the existing ANN [9]. These parameters were found to be effective for IEC in preliminary experimentation.

Through complexification, particle system effects become increasingly sophisticated as evolution progresses. Thus, complex and unique effects are discovered that follow user preferences. The next section explains evolving particle system content for a more specialized purpose, weapons effects for video games.

IV. NEAT Projectiles

NEAT Projectiles is an extension of NEAT Particles designed to evolve particle weapon effects for video games. The aim is to exhibit a concrete, practical application of NEAT Particles that can potentially enhance content generation in existing real-world products. NEAT Projectiles uses similar rendering, physics, and activation functions as NEAT Particles.



(a) Main Interface (b) Zoom Mode

Fig. 9. **NEAT Particles Interface**. In the main interface (a), the user is presented with nine particle systems. System parameters such as generation shape and inputs are displayed on the bottom of the screen. In zoom mode (b), a single particle system and its ANN can be inspected.

Furthermore, the same IEC interface (figure 10) drives evolution. The major differences are (1) the projectile classes, (2) the projectile constraints, and (3) the ANN inputs and outputs.

A. Projectile Classes

Three classes of weapon-like systems are implemented in NEAT Projectiles to mirror common weapon models in video games: (1) *dumb weapons*, (2) *directed weapons*, and (3) *smart weapons*. Dumb weapons fire simple, non-target aware projectiles and exhibit a fixed behavior in flight. Directed weapons fire projectiles that may be steered by the user during flight. Smart weapons see the target; like a heat-seeking missile, the in-flight behavior of smart projectiles is influenced by target motion.

B. Projectile Constraint

Particle weapons provide two new significant constraints on particle motion beyond generic particle effects. First, to avoid weapons firing backward, projectile velocity is limited to overall forward motion. Second, evolved projectile weapons fire in the same pattern regardless of what direction the weapon is facing. It would not make sense for projectiles emitted from a weapon to behave differently when a user points the weapon in different directions. Therefore, projectile coordinates are defined relative to the heading of the gun when it is fired.

The new projectile classes and constraint mechanisms also influence the interpretation of NEAT Projectiles ANNs, as explained next.

C. Projectile ANNs

Because there is more than one way to make particles act as projectiles, two approaches are implemented and tested in NEAT Particles: (1) the *offset-constrained model* and (2) the *force constrained model*.

In the offset-constrained model (figure 11a), a 90° *offset cone* in front of each particle is computed in each frame. The outputs from each particle's ANN represent a vector within the offset cone, which becomes the particle's new velocity. Offset angles are computed differently for each weapon type. A particle fired from the *dumb weapon* has a fixed offset in the direction the gun was facing on discharge (figure 12a). The *directed weapon* allows the user to influence projectiles while in flight; therefore particle offset is constrained to a 90° cone around the vector the weapon is currently facing (figure 12b). Particles fired from the *smart weapon* seek their target. Therefore, the smart particle's offset is constrained to the 90° cone around a vector from the projectile to the target (figure 12c).

In the force-constrained model (figure 11b), the ANN is similar to that used in the generic system of NEAT Particles; however a push force is applied to constrain particle movement to a general direction. The direction of the push force depends on the weapon type. The dumb weapon projectile is pushed in the direction of the gun when it discharges. The directed projectile pushes in the direction the gun is currently facing. The smart weapon pushes projectiles in the direction of the target.

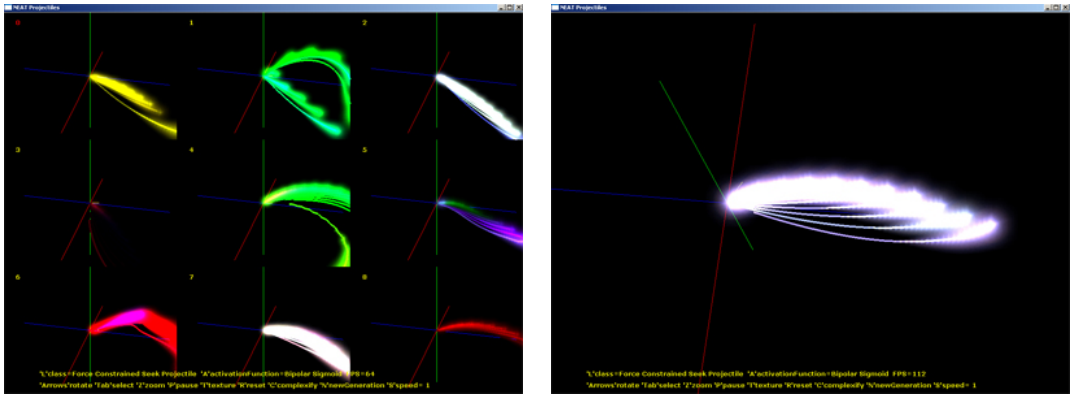
The combination of constraint model, classes, and correct ANN design minimizes defective offspring while allowing a sufficiently large variety of unique weapons to evolve, which is integral to efficiently producing useful content though IEC.

V. Experimental Results

This section shows how NEAT Particles and NEAT Projectiles work in practice to produce useful particle system content. All particle systems reported were evolved in between five and ten minutes and between 20 and 30 user-guided generations. The NEAT particles executable, source code, and examples effects in this paper can be downloaded at `http://eplex.cs.ucf.edu`.

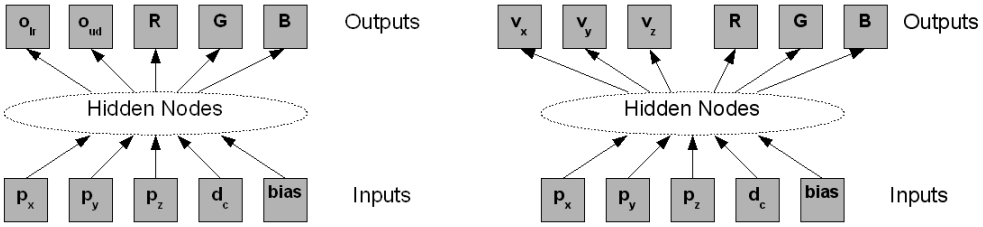
Figure 13 illustrates evolving a *flame shield* effect with NEAT Particles. The goal of the effect is a halo of flaming red particles around the user. Evolution begins with a red ring (figure 13a). As evolution proceeds, particles begin breaking away from the ring (figure 13b). A full orbital sphere of particles develops eventually (figure 13c) and the final effect (figure 13d) exhibits brighter colors.

Figure 14 depicts the evolution of an *arc gun* effect with NEAT Projectiles. The aim is to create a multi-beam effect that tracks a target. The starting point is a single curving



(a) Main Interface (b) Zoom Mode

Fig. 10. **NEAT Projectiles Interface**. In the main interface (a), the user chooses among nine projectile systems of any weapon class. The weapons can be rotated (as in this figure) and the refire rate adjusted to display their full behavior. In zoom mode (b), a single projectile system and its ANN can be inspected.



(a) offset constrained (b) force constrained

Fig. 11. **Projectile System ANNs**. Projectile models are designed to minimize undesirable behaviors (e.g. firing backward). (a) In the offset constrained model, projectile movement is constrained by an offset cone in front of the projectile. The offset cone is computed by adding two vectors *up-down* (o_{ud}) and *left-right* (o_{lr}). ANN inputs include the current position of the particle, distance from center of the weapon, and a bias. The outputs are left-right offset, up-down offset, and color. (b) In the force constrained model, projectile motion is perturbed by an additional push force applied to the projectile after ANN processing. 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. Both models constrain projectile behavior while allowing sufficient evolutionary variety.

beam to the target, which is marked with a cross (figure 14a). During evolution the beam splits (figures 14b, 14c). Finally, the desired effect is achieved with two stylized, parallel arcs that track the target (figure 14d).

Preliminary testing of both NEAT Projectiles constraint models suggests that, compared to the force-constrained model, the offset-constrained model over-constrains evolution. It generates less variety in evolved weapon effects. However, unlike the force-constrained model, it also produces no offspring that fire back at the user. Thus, both models have their pros and cons.

Keyframe animations for evolved particle and projectile systems are depicted in figures 15 and 16. Additional evolved systems are presented in figures 17 and 18.

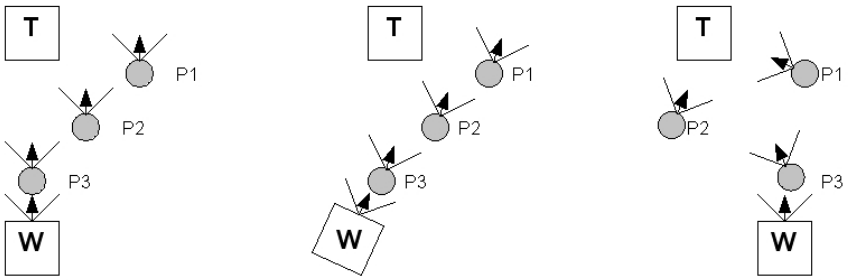
A. Comparisons

To compare the quality of IEC particle effects to those generated by traditional methods, two hand-coded particle emitters were implemented with the same rendering method as NEAT Particles (figure 19). The resulting effects exhibit similar visual quality; however, they are limited to simple behaviors because the behavioral complexity of hand-coded particle systems is dependent upon mathematics, physics, and programming, which become increasingly difficult to coordinate through hand-coded policies as more is added.

Another interesting comparison can be drawn with the IEC fireworks application by Tsuneto [51] (figure 20), which produces a specialized class of particle effects. In this system, fireworks are defined by real-world attributes such as powder type, explosive payload, number of stages, stage configuration, etc. A rule-based physics system defines the behavior of fireworks based on these attributes. Through repeated selection in an IEC interface, users can evolve fireworks to suit their preferences. Thus, unlike NEAT Particles, this system demonstrates evolving the variables of a rule system. In contrast, NEAT Particles evolves the behavior rules themselves. Both approaches offer unique advantages. The special rule set of the fireworks application allows it to focus on a specific class of effects. NEAT Particles in contrast can evolve effects in a large variety of classes because of its generality and lack of domain-specific parameters.

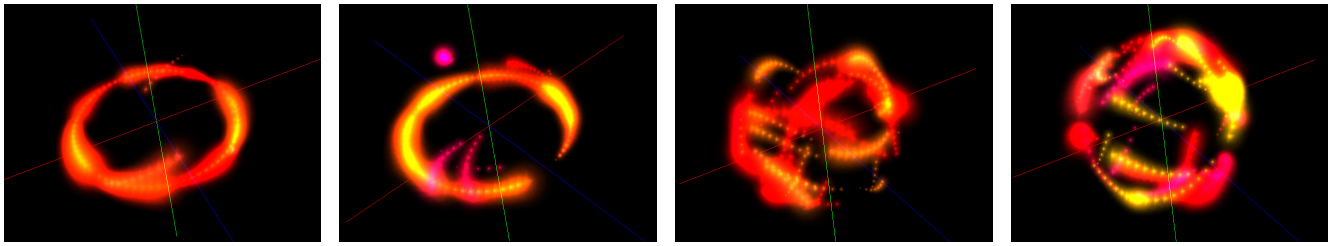
B. User Study

An informal user study was conducted to test the viability of the NEAT Particles method. In this study, eight users were introduced to the system and encouraged to explore the search space to evolve effects that please them. Samples of the user-evolved effects are presented in figure 21. In general, users found the generic and trail particle system classes to produce the most variety, while the rotator, plane, and beam systems



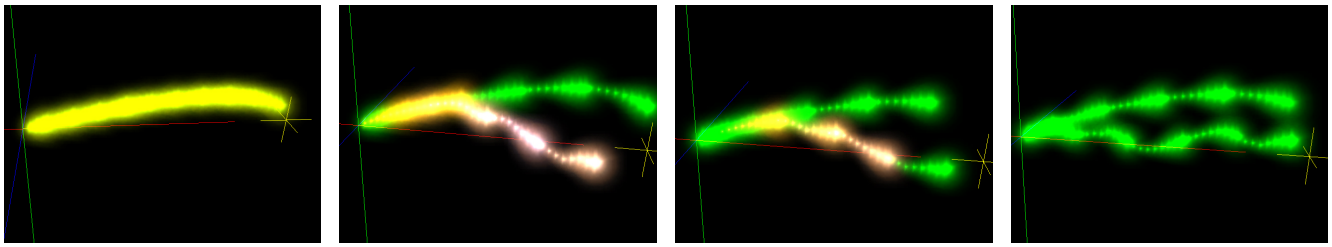
(a) dumb (b) directed (c) seek

Fig. 12. **Projectile Constraint Mechanics**. To ensure that weapons behave as projectiles, particle velocities are constrained to the 90° cones shown above. In the figure, W is the weapon and T is the target. (a) Dumb weapon particles are not target-aware so they are constrained in velocity to a fixed 90° cone in front of the weapon at the moment of discharge. (b) Directed weapon particles are not target-aware but may be influenced while in flight by the weapon. Thus directed particle velocity is constrained to the 90° cone in which the weapon is currently facing. (c) Smart particles are target-aware; therefore they are constrained to a 90° cone around a vector from the particle to the target.



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

Fig. 13. **Particle System Evolution**. This figure illustrates evolution of a *flame shield* effect with NEAT Particles, in which red and yellow particles are evolved to orbit a player at the center. (a) Evolution begins with a ring shaped rotator system with an appropriate red and yellow color scheme. (b) After a few generations particles begin to detach from the ring. (c) Several generations later a prominent orbital behavior becomes apparent. (d) Evolution concludes with a full orbital pattern and a brighter color scheme, producing a convincing flame shield effect suitable for use in a video game.



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

Fig. 14. **NEAT Projectiles Evolution**. A double-arc beam weapon that seeks a target is evolved with NEAT Projectiles. The weapon emits particles from the left side of each frame and the target is marked with a cross on the right side. (a) Evolution begins with a single arc that connects to the target. (b) After several generations, the beam begins to split in the middle. (c) Continuing evolution, the double arc becomes more pronounced. (d) Eventually the arcs become fully disjoint and the intended projectile behavior is achieved.

were acknowledged to be more constrained. The examples presented in figure 21, all evolved within 15 generations, demonstrate that the IEC approach enables users to quickly and easily generate complex and unique particle effects.

In summary, the totality of results demonstrate the ability of NEAT Particles and NEAT Projectiles to evolve particle systems of similar complexity to those in mainstream games.

VI. 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 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 is expected to perform comparably to traditional particle systems.

VII. Discussion and Future Work

The complexity and cost of producing content for modern graphics and games creates the need for tools that quickly and efficiently generate novel content. IEC can alleviate this problem by enabling automated content generation guided by user preferences. NEAT Particles demonstrates the promise of this approach by constraining the search space for the user, thereby defining a content space large enough to evolve many interesting and useful results, yet not so large that producing useful output is too time-consuming. The separate classes implemented in NEAT Particles provide this constraint.

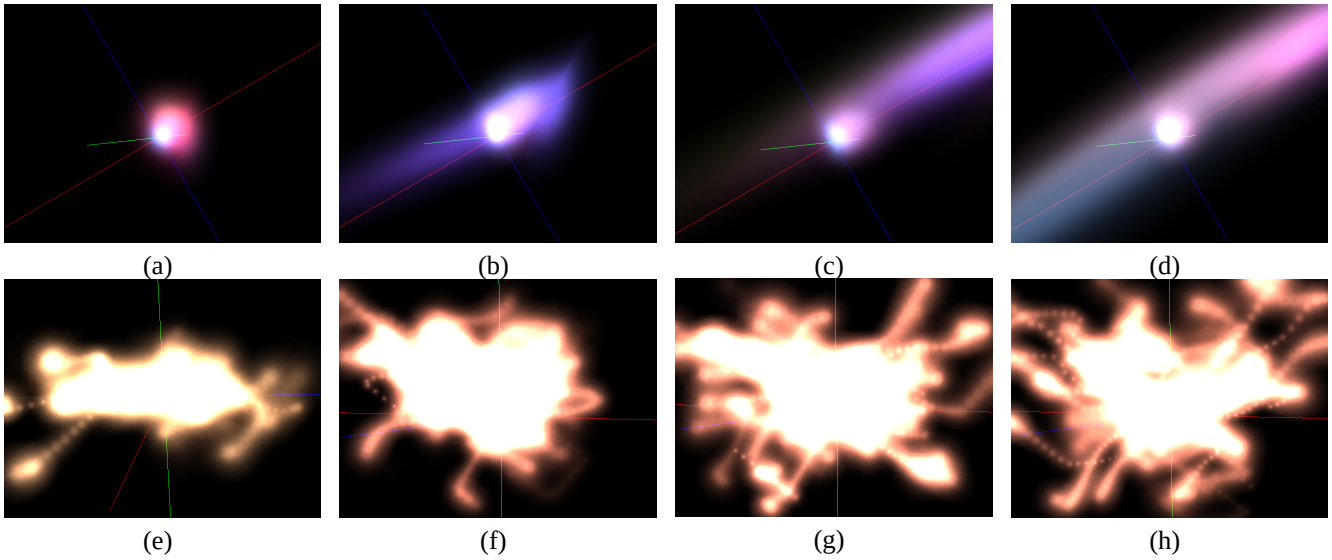


Fig. 15. **Evolved Particle Animations**. Key frames from animations of two evolved particle systems are presented in this example. Figures (a) through (d) depict an expanding vortex or explosion-like effect of an evolved plane system. Figures (e) through (h) depict a realistic billowing smoke cloud or explosion effect produced by an evolved trail system. Such effects illustrate that NEAT Particles can evolve effects appropriate for graphics and games.

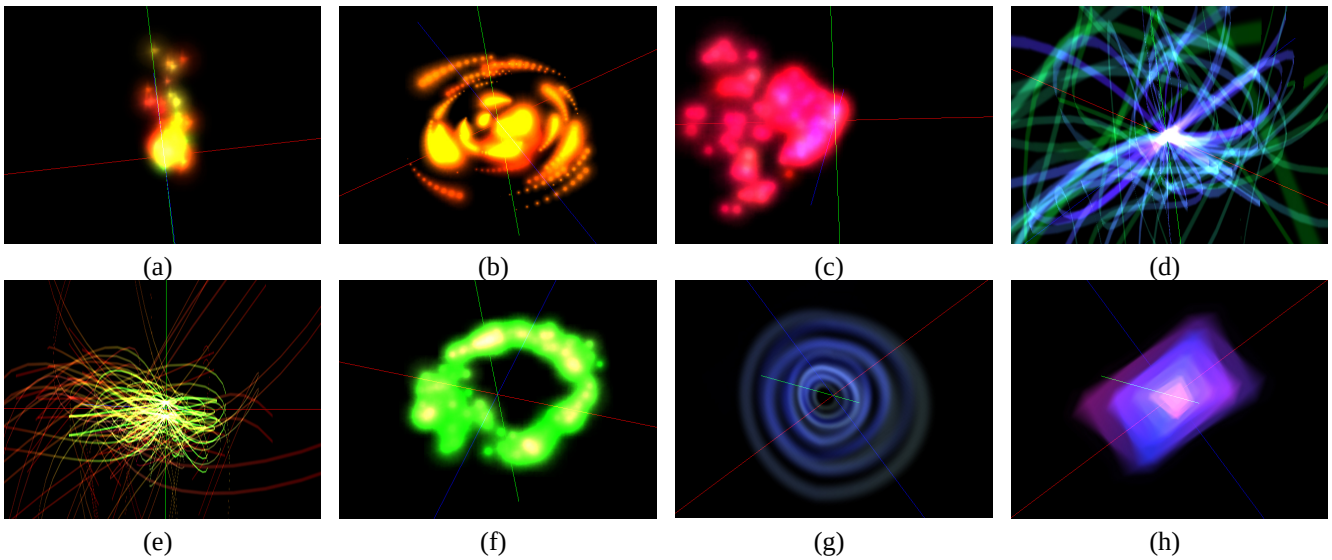


Fig. 16. **Evolved Projectile Animations**. Key frames of evolved single projectiles are displayed above. The projectiles are fired from the left side of the screen towards the right side. The trailing lines mark motion over time. Figures (a) through (d) display a solid yellow projectile with a smooth curving behavior. Figures (e) through (h) depict a spiraling projectile that changes colors. Multiple projectiles evolved within the constraints combine to form complex weapon effects in NEAT Projectiles.

Besides intentionally evolving specific particle systems that they have in mind, users can also employ the IEC approach of NEAT Particles as a concept generation tool. While evolving a specific effect, the user often generates novel, compelling 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 (1) user-designation of inputs and outputs, decoupling particle system creation from programming even further, and (2) evolution during a game, while it is played, which is a most significant implication of automated content generation. For example, consider a game in which content such as weapons, items, spells, etc. are automatically generated based on characteristics users prefer. Such a system is especially suited to multi-player virtual world games (e.g. *Massively Multiplayer Online Games*; MMOGs), in which unique content is coveted and potentially thousands of players can contribute to evolutionary content within the game.

Another promising area for future research is applying similar techniques to evolving other graphical content, such as three dimensional models and programmable shader effects.

VIII. Conclusion

There is an increasing need for powerful graphics and game content generation tools. Content developers require such tools to augment and assist the slow and expensive content pipeline.

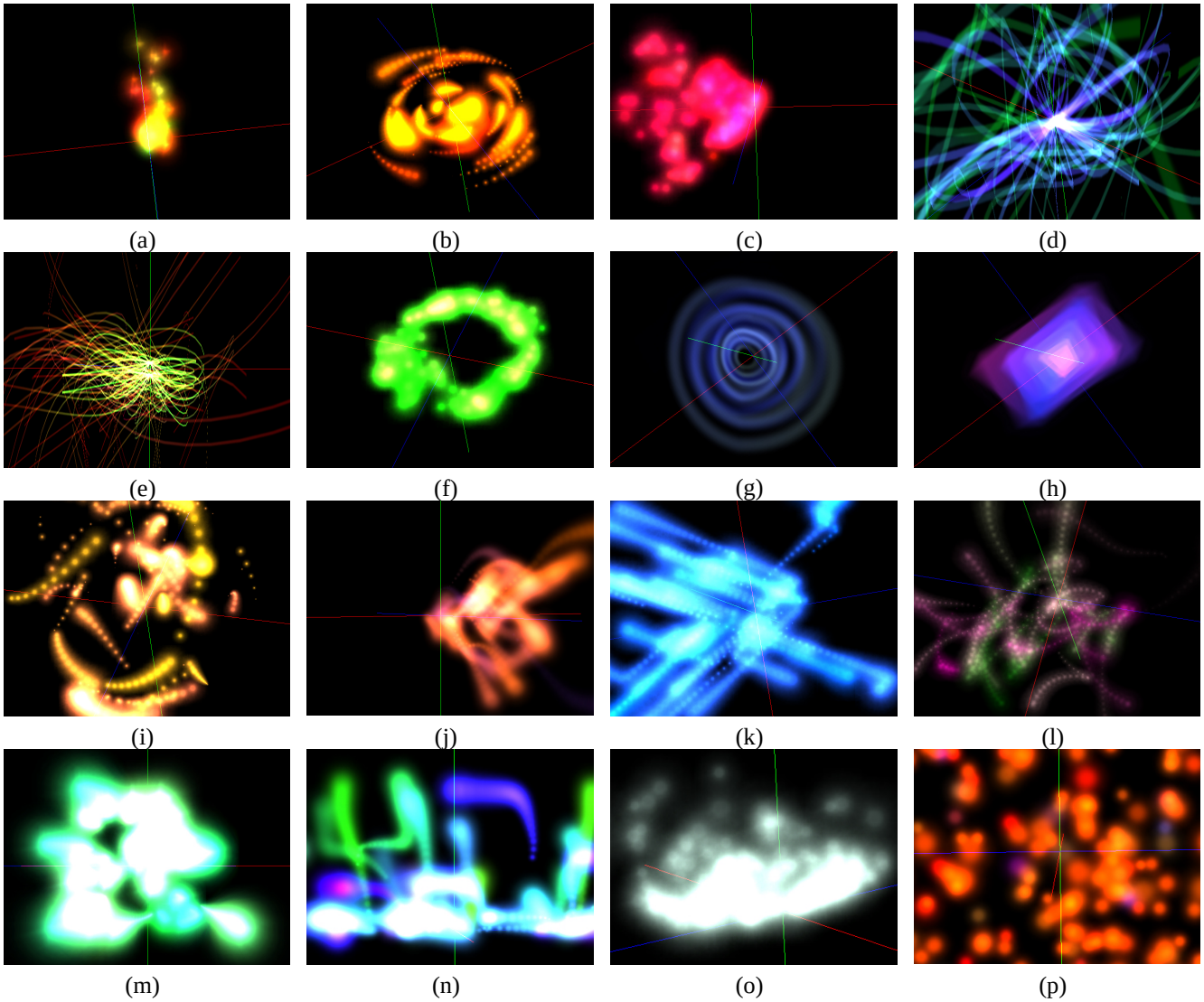


Fig. 17. **Sample Evolved Particle Systems**. The images in this figure are single animation frames from effects evolved with NEAT Particles. These images demonstrate the variety of effects evolved through IEC.

End-users benefit from such tools because today's games are distributed with content generation tools for users to customize or build their own content. Additionally, there is the emerging trend of content generation as a major part of game play itself. IEC can potentially solve this problem by providing an intuitive way to easily generate complex and unique content by user preference.

NEAT Particles and NEAT Projectiles demonstrate how particle system effects for graphics and video games can be interactively evolved through user preference. In this approach, particle systems are represented by ANNs, the ANNs are evolved by NEAT, and an IEC interface enables the user to guide evolution. By replacing the complex, hand-coded rules of traditional particle systems with ANNs, the dependence on programmers to create new effects is reduced. The IEC interface provides easy, interactive exploration of the search space and a way to discover novel effects useful to both developers of graphics and gaming media, and to end users of the content generation tools provided with such software.

While the focus of this work is on evolving particle system effects, the techniques are applicable to generating other types of graphical and gaming content. Thus, automated content generation is a promising research direction in which evolutionary computation can significantly contribute to popular media and games.

References

[1] P. Stiff, "Special effects cost studios big bucks," *Digital Spy*, 2006. [Online]. Available: http://www.digitalspy.co.uk/movies/a33092/special-effects-cost-studios-big-bucks.html/

[2] B. Lomborg, "These holywood special effects may cost the world 15 trillion," *Telegraph.co.uk*, 2004. [Online]. Available: http://www.telegraph.co.uk/opinion/main.jhtml?xml=/opinion/2004/05/09/do0903.xml&sSheet=/portal/2004/05/09/ixportal.html/

[3] V. Software, "Source engine sdk," 2007. [Online]. Available: http://developer.valvesoftware.com/wiki/Main_Page/

[4] E. Games, "Unreal engine sdk," 2007. [Online]. Available: http://www.unrealtechnology.com/

[5] I. Software, "Quake wars sdk," 2007. [Online]. Available: http://www.idsoftware.com/

[6] G. Entis, "Recent accomplishments and upcoming challenges for interactive graphics in videogames," 2007. [Online]. Available: http://www.zcorp.com/

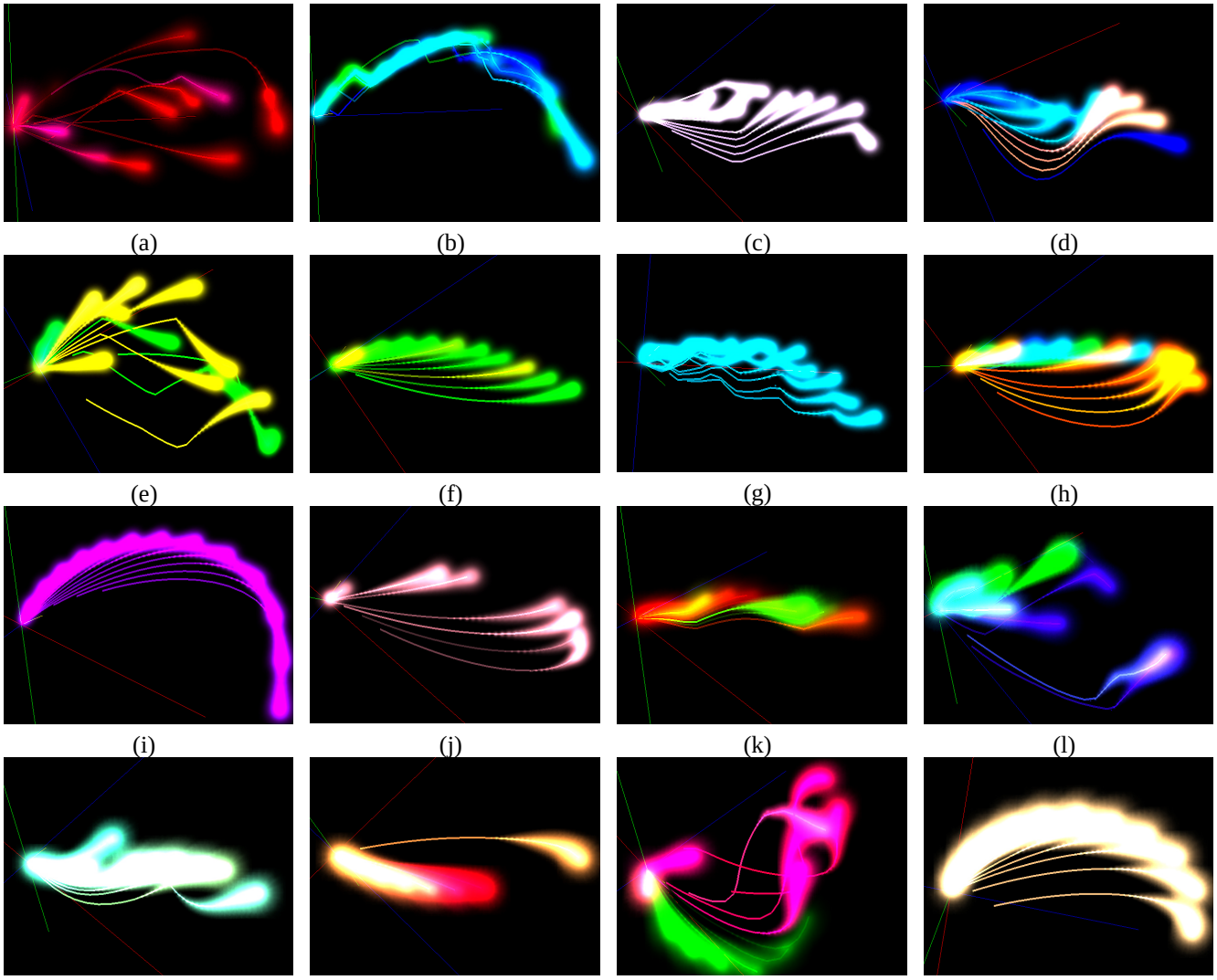
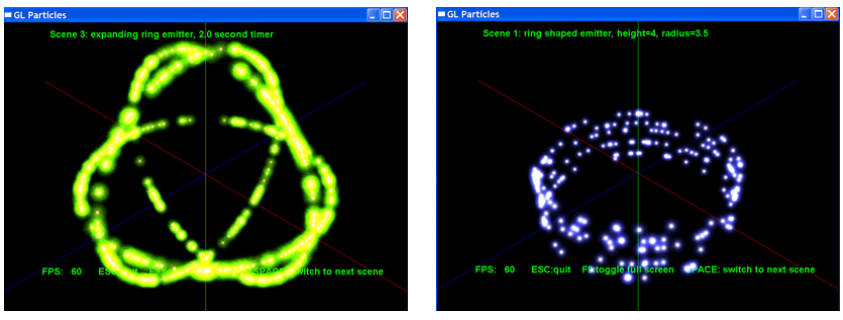
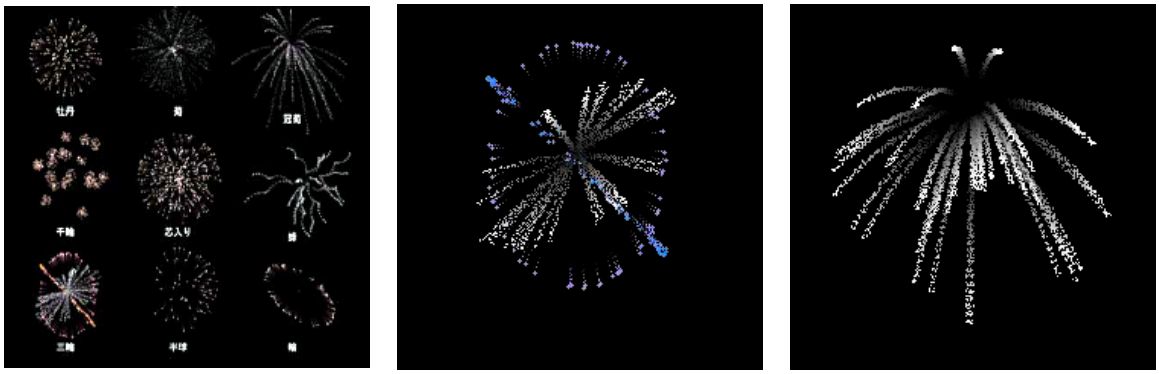


Fig. 18. **Sample Evolved Projectile Systems**. Single animation frames are shown of evolved NEAT Projectile systems fired from the left side of the screen toward targets on the right. The trailing lines plot motion over time. These images demonstrate the variety of weapon behaviors evolved by NEAT Projectiles.



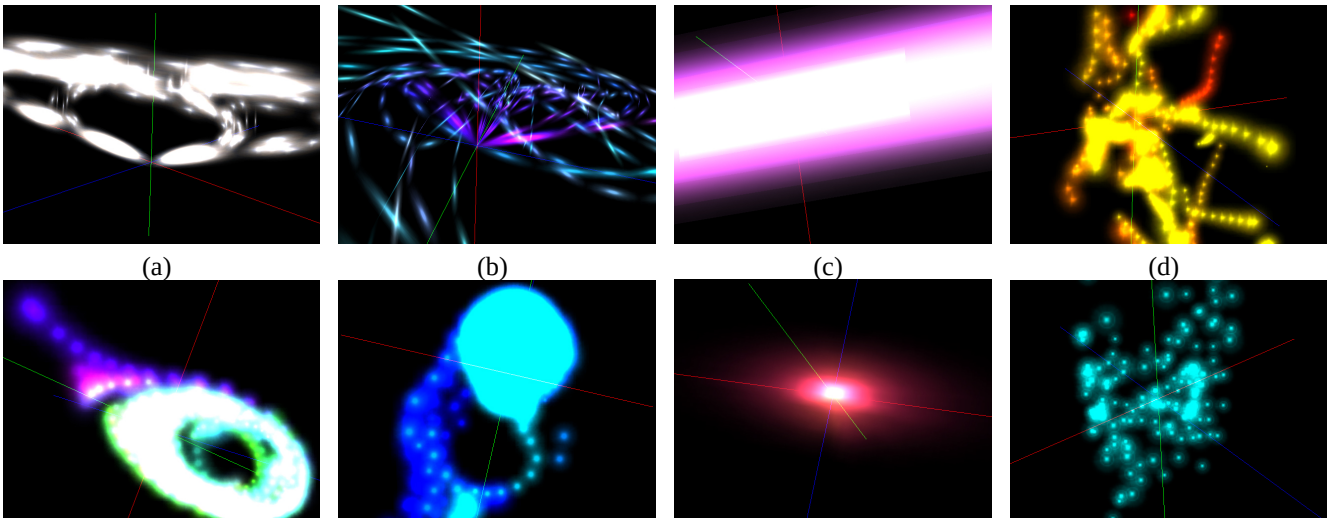
(a) (b)

Fig. 19. **Traditional Particle System Comparison**. To compare IEC-generated particle systems to traditional ones, two hand-coded particle emitters were implemented within the same renderer as NEAT Particles. The expanding ring emitter (a) supports explosion effects and the simple ring emitter (b) can convey a variety of magical and force effects. Both systems display similar visual quality to NEAT Particles. However, they are capable of comparatively much less behavioral complexity. This example demonstrates the dependence on math and programming of traditional particle system implementations.



(a) (b) (c)

Fig. 20. **Tsuneto's IEC Fireworks** [51]). In this IEC fireworks application, users evolve combinations of real-world fireworks properties such as size, powder type, explosive payload, etc., through repeated selection. This exemplifies evolving the variables of a rule system, whereas in NEAT Particles the rules themselves are evolved.



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

(e) (f) (g) (h)

Fig. 21. **User-Evolved Particle Effects**. The images in this figure depict effects evolved with NEAT Particles by participants in the user study. The variety and complexity of these examples demonstrate that the IEC approach of NEAT Particles enables users to quickly evolve compelling particle system effects.

[7] J. Lander, "The ocean spray in your face," *Game Developer Magazine*, pp. 13–20, July 1997.

[8] J. V. der Berg, "Building an advanced particle system," *Game Developer Magazine*, pp. 44–50, March 2000.

[9] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/keyword?stanley:ec02

[10] ——, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, pp. 63–100, 2004. [Online]. Available: http://nn.cs.utexas.edu/keyword?stanley:jair04

[11] W. Reeves, "Particle systems: A technique for modeling a class of fuzzy objects," *ACM Transactions on Computer Graphics*, vol. 17, no. 3, pp. 91 – 108, 1983.

[12] ——, "Approximate and probabilistic algorithms for shading and rendering structured particle systems," *ACM Transactions on Computer Graphics*, vol. 19, no. 3, pp. 313 – 322, 1985.

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

[14] 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.

[15] D. Obrien, S. Fisher, and M. Lin, "Automatic simplification of particle system dynamics," in *Proceedings of the 14th Conference on Computer Animation*, 2001, pp. 210–257.

[16] 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, pp. 154–159.

[17] C. Reynolds, "Steering behaviors of autonomous characters," in *Proceedings of the Game Developers Conference*, 1999, pp. 763–782.

[18] ——, "Flocks, herds, and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987, pp. 25 – 34.

[19] 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. [Online]. Available: http://ieeexplore.ieee.org/iel5/5/20546/00949485.pdf?tp=&arnumber=949485&isnumber=20546

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

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

[22] T. Unemi, "Genetic algorithms and computer graphic arts," *Journal of Japan Society for Artificial Intelligence*, vol. 9, no. 4, pp. 518–523, 1994.

[23] H. Nishino, H. Takagi, S. Cho, and K. Utsumiya, "A 3d modeling system for creative design," in *Proceedings of the 15th International Conference on Information Networking*, 2001, pp. 479–487.

[24] P. Husbands, G. Jenny, M. McIlhagga, and R. Ives, "Two applications of genetic algorithms to component design," *Lecture Notes in Computer Science*, vol. 1143, pp. 50–62, 1996.

[25] M. Fagerlund, "DelphiNEAT-based genetic art homepage," http://www.cambrianlabs.com/mattias/GeneticArt/, 2005.

[26] A. Lindenmayer, "Mathematical models for cellular interaction in de-

velopment parts I and II," *Journal of Theoretical Biology*, vol. 18, pp. 280–299 and 300–315, 1968.
[27] H. Nishino, H. Takagi, S. Cho, and K. Utsumiya, "A 3d modeling system for creative design," vol. 100, no. 461, pp. 1–8, 2000.
[28] ——, "A digital prototyping sytem for designing novel 3d geometries," pp. 473–482, October 2000.
[29] K. O. Stanley, "Exploiting regularity without development," in *Proceedings of the AAAI Fall Symposium on Developmental Systems*. Menlo Park, CA: AAAI Press, 2006.
[30] ——, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, pp. 131 – 162, 2007.
[31] 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. [Online]. Available: http://nn.cs.utexas.edu/keyword?gomez:ijcai99
[32] N. Saravanan and D. B. Fogel, "Evolving neural control systems," *IEEE Expert*, pp. 23–27, June 1995.
[33] 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.
[34] 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.
[35] 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.
[36] 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. [Online]. Available: http://www-illigal.ge.uiuc.edu:8080/gecco-2001/
[37] 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.
[38] 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.
[39] 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.
[40] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
[41] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[42] 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.
[43] 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.
[44] M. E. Taylor, S. Whiteson, and P. Stone, "Comparing evolutionary and temporal difference methods in a reinforcement learning domain," in *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, July 2006, pp. 1321–1328.
[45] 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.
[46] H. Ferstl, "SharpNEAT-based genetic art homepage," http://www.cs.ucf.edu/~kstanley/GenArt.zip, 2006.
[47] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, 1989.
[48] A. Fernandes, "Lighthouse 3d billboarding tutorial," 2006. [Online]. Available: http://www.lighthouse3d.com/opengl/billboarding/
[49] F. D. Luna, *3D Game Programming with Direct X 9.0*. Wordware, 2003.
[50] L. Latta, "Building a million particle system," in *Proceedings of Game Developers Conference*, 2004.
[51] C. Tsuneto, "A fireworks animation support system using interactive evolutionary computation," Master's thesis, Kyushu Institute of Design, 2002.

Erin J. Hastings is currently pursuing a Ph.D. at the University of Central Florida. He earned a B.S. in Computer Science from University of Florida in 2001 and an M.S. in Computer Science from University of Central Florida in 2004.

His research interests include evolutionary computation, interactive evolution, neural networks, graphical content generation, particle systems, collision detection, and range monitoring. He has recently published papers in the IEEE Symposium on Computational Intelligence and Games, the ICST International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities, and the SCS Summer Computer Simulation Conference.

Ratan K. Guha received the B.S. degree with honors in Mathematics and the M.S. degree in Applied Mathematics from the University of Calcutta. He received the Ph.D. degree in Computer Science from the University of Texas at Austin in 1970.

He is a Professor of Computer Science at the University of Central Florida, Orlando. His research interests include distributed systems, networks, security protocols, modeling, simulation, and graphics. His research has been supported by grants from ARO, NSF, STRICOM, PM-TRADE, NASA, and the State of Florida.

Dr. Guha is a member of ACM, IEEE, SCS, and served on the Board of Directors of SCS. He is currently serving on the editorial board for the International Journal of Internet Technology and Secured Transactions and the editorial board for Modelling and Simulation in Engineering.

Kenneth O. Stanley received the B.S.E. degree *magna cum laude* in computer science engineering and a minor in cognitive science from the University of Pennsylvania, Philadelphia, in 1997, and the M.S. degree in computer science and the Ph.D. degree from the University of Texas at Austin (UT-Austin), in 1999 and 2004, respectively.

He is an Assistant Professor in the School of Electrical Engineering and Computer Science, University of Central Florida. He has published papers in JAIR, Evolutionary Computation, and Artificial Life journals. His research focuses on evolutionary algorithms and complexity.

Dr. Stanley is a member of AAAI. He has won Best Paper Awards at the 2002 Genetic and Evolutionary Computation Conference, for his work on NEAT, and at the IEEE 2005 Symposium on Computational Intelligence and Games, for his work on NERO.