1

Evolution and Collective Intelligence of the Electric Sheep

Scott Draves

Spotworks, San Francisco CA, USA

Summary. *Electric Sheep* is a collective intelligence composed of 40,000 computers and people mediated by a genetic algorithm. It is made with an open-source screen-saver that harnesses idle computers into a render farm with the purpose of animating and evolving artificial life-forms known as *sheep.* The votes of the users form the basis for a fitness function for exploring a space of abstract animations. Users also may design sheep by hand for inclusion in the gene pool.

The name *Electric Sheep* is an homage to Philip K. Dick's novel *Do Androids Dream of Electric Sheep*, the basis for the film *Blade Runner.* The metaphor compares the screen-saver to the computer's dream.

After the introduction, we dig into the system starting with Section 1.2 on its architecture and implementation. Section 1.3 covers the genetic code, including its basis in the equations of classic Iterated Function Systems. The equation is then generalized into the Fractal Flame algorithm, which translates the genetic code into an image. The next two sections treat color and motion.

Section 1.4 shows how the genetic algorithm decides which sheep die, which ones reproduce, and how. Section 1.5 defines the primary dataset and its limitations, and reports some of its statistics. Section 1.5.1 uses the dataset to determine that the genetic algorithm functions more as an amplifier of its human collaborators' creativity rather than as a traditional genetic algorithm that optimizes a fitness function.

The goal of Electric Sheep is to create a self-supporting, network-resident life-form. Section 1.6 speculates on how to make the flock support more of a self-sustaining reaction rather than functioning as an amplifier. At last, Section 1.6.1, explains how *Dreams in High Fidelity* addresses the support issue.

images elided

Fig. 1.1. Above is Sheep 191.21054 (sheep generation 191, number 21054) and below is 198.19616, born in August and December 2005, respectively.

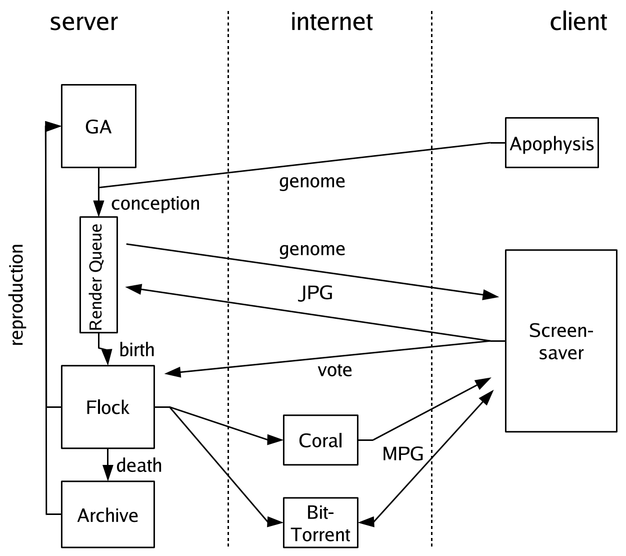


Fig. 1.2. System block diagram. The dotted lines divide the diagram into 3 parts: on the left are the components that run on the server, on the right those on the client, and in the middle is the internet. A sheep is conceived by the genetic algorithm (GA, described in Section 1.4), sits in the render queue until all its frames have been received, then is born into the flock. It can then be downloaded into the client and voted upon until its death. Apophysis is the sheep design GUI (See Section 1.4.1). Coral and BitTorrent are download accelerators (See Section 1.2.1).

1.1 Introduction

The Electric Sheep project began in 1999, and is ongoing [6]. To the public, it appears as a screen-saver client that can be downloaded and installed on almost any computer. When one of these computers is idle and goes to sleep, the sheep animations appear, and in parallel the computer goes to work rendering new sheep and sharing its results with all other users. It was inspired by the SETI@Home distributed screen-saver [1].

Figure 1.1 shows still images of two example sheep. After installation, no interaction is required for a user to enjoy the imagery and for their computer to contribute to its creation.

Each sheep's shape, motion, and color are specified by a genetic code, a string of hundreds of floating point numbers. When a user sees a sheep they like, they may press the up arrow key to vote for it, and increase its rating. Sheep with higher ratings live longer and are more likely to reproduce. This fitness function captures the desire of the audience, hence the sheep are a product of aesthetic evolution, a concept first realized by Karl Sims [11].

Users can also download GUI software called *Apophysis* [13] to design sheep genomes and post them to the server. If they prove popular they may interbreed with the artificially evolved population. Hence a human design team collaborates and competes with the artificial intelligence.

1.2 Architecture and Implementation

Electric Sheep has a client/server architecture as illustrated in Figure 1.2. The client initiates all communication between them, and if no client were running, the server would not run at all.

The screen-saver client has three main threads. One thread downloads sheep animations from the server to a local disk cache. It downloads those with the most votes first. The default size of the cache is 1GB (enough for 216 animations) but the user may change it. Another thread reads the sheep from the cache and displays them in a continuous sequence on the screen. The third thread contacts the server, receives a genome specifying a frame to render, renders the frame, then uploads the resulting JPEG file.

The server maintains several collections of sheep. Sheep are numbered as they are created and are identified by this sequence number. Freshly conceived genomes start out in the render queue. Each frame is sent out to a different computer. When all the frames of a sheep have been received, they are compressed into MPEG and deleted, and the sheep is made available for download and voting. Sheep average 4.6MB each. Eventually the sheep dies (Section 1.4.2 explains when) and the MPEG file is deleted.

All these sheep are referred to collectively as a generation. Each time the server is reset the database is wiped, all sheep are deleted from the server and from all client caches, the generation number is incremented, and evolution starts fresh. The generation current in November 2006 is 202, and the sheep that are analyzed below are members of generation 165, from 2004. The major generations last for many months, contain thousands of sheep, and are preserved on the server. Most generations last only a few moments during debugging and are discarded.

1.2.1 Bandwidth

The primary server is a commodity Linux x86 server running Apache. It runs the evolutionary algorithm, collects frames and votes, compresses frames, and sends genomes to clients for rendering. This server received 220Kb/s from the clients and transmits 260Kb/s to them (measured average of July to October 2004). On average since its inception in 1999, the server traffic has doubled every 9 months. This machine does not have nearly enough bandwidth available to distribute the MPEG files to all the clients. With the current audience size, this would require 20TB/day (1.8Gb/s)!

Over the years the mechanism and source for the bandwidth has changed. Right now, the primary server copies them to a high volume server with 15Mb/s allocated to sheep. This machine feeds the Coral web-cache, an NSF-funded network of hundreds of servers located world-wide. Coral limits Electric Sheep to 250GB/day of traffic.

Because each user only gets a fraction of the flock, their experience is less than ideal. A freshly installed client may take a *long* time to download its first sheep. When it does run, the playback will probably be repetitive and discontinuous because the client has only a subset of the flock. Right now this is the factor limiting user growth. Bandwidth is the bottleneck.

To address this, the sheep are adopting BitTorrent, a peer-to-peer file sharing protocol [4]. The idea is to share the bandwidth load among the clients the same way the computational load already is. A client with BitTorrent built-in has been released and is delivering about 2TB/day of sheep. Unfortunately BitTorrent depends on each user configuring their firewall to allow their computer to act as a server. Only about 15% of them do so successfully, and this subset cannot support the whole network. Because the sheep are batched into torrents of 100MB each (about 22 sheep), with this protocol the download is not prioritized by rating.

The system is open-source and the code is licensed under the GPL (General Public License version 2) [10]. The fractal flame utilities are written in C and, alas, the server is written in Perl. The clients are written in C, C++, and Objective-C. The genome format is XML.

1.3 The Genetic Code

Fractal flames [5] were developed in 1992 as a generalization and refinement of the Iterated Function System (IFS) category of fractals [3]. The genetic code used by Electric Sheep is the parameter set for these fractals. It consists of up to several hundred floating-point numbers. The parameters control the scattering of billions of particles from which an image emerges.

The genetic code is a visual language and the core of the system. The language is intended to be abstract, expressive, and robust. *Abstract* means that the codes are small relative to the images. *Expressive* means that a variety of images can be drawn. And *robust* means that useful codes are easy to find. These are conflicting goals.

1.3.1 Iterated Function Systems

A classic IFS consists of a recursive set-equation on the plane:

$$S = \bigcup_{i=0}^{n-1} T_i(S)$$

The solution S is a subset of the plane (and hence a two-tone image). The T_i are a small collection of n affine transforms of the plane, and S is their fixed-point. Affine transformations consist of combinations of scale, rotate, translate, and skew. Hence S is composed of several distorted copies of itself. Normally each copy is smaller than the whole. This is illustrated in Figure 1.3. Overlap is allowed.

The process is closely related to both video feedback (with each camera corresponding to one transformation), and iterated photocopying with zoom and cut&paste. Its implementation however is more like a particle system: the transformations are iterated to generate a stream of colored particles, each of which contributes luminance to a pixel. Unlike a regular particle system, the particles' positions are not saved, and with each frame they are all generated from scratch using just the genome. This allows the renderer to run in parallel across many computers.

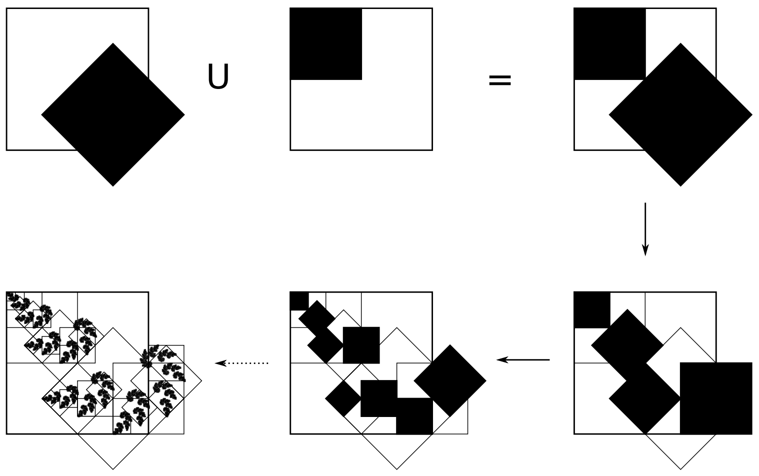


Fig. 1.3. Construction of S from two transformations T_0 and T_1. The first two diagrams in the top row represent T_0 and T_1 by showing how they map the biunit square (outlined) into a smaller square (shaded). Their union is the top-right diagram. The bottom row represents further applications of the transformations, right to left. The solution S appears in the lower-left along with the first four levels of construction squares.

1.3.2 Fractal Flames

A fractal flame is based on the same recursive equation, but the transforms may be non-linear and the solution algorithm produces a full-color image. The transforms are linear blends of a set of 31 basis functions known as *variations*. The variations are composed with an affine matrix, like a classic IFS. So each transform T_i is:

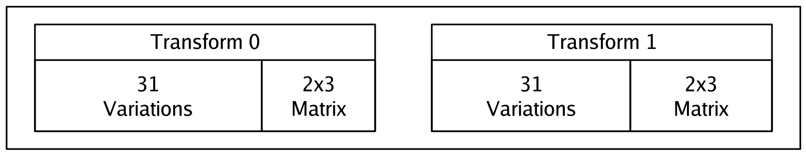
$$T_i(x, y) = \sum_j v_{ij} V_j(a_i x + b_i y + c_i, d_i x + e_i y + f_i)$$

where v_{ij} are the blending coefficients for T_i, and a_i through f_i are 6 affine matrix coefficients. The V_j are the variations, for example:

$$\begin{array}{ll}
V_0(x, y) = (x, y) & V_3(x, y) = (r\cos(\theta + r), r\sin(\theta + r)) \\
V_1(x, y) = (\sin x, \sin y) & V_4(x, y) = (r\cos(2\theta), r\sin(2\theta)) \\
V_2(x, y) = (x/r^2, y/r^2) & V_5(x, y) = (\theta/\pi, r - 1)
\end{array}$$

where r and θ are the polar coordinates for the point (x, y) in rectangular coordinates. V_0 is the identity function so this space of non-linear functions is a super-set of the space of linear functions. See [5] for the complete list.

There are 3 additional parameters for density, color, and symmetry. Density effects the relative brightness; color effects which part of the palette is used, and symmetry the motion. They are also explained in [5]. Together these 40 (31 for v_{ij} plus 6 for a_i to f_i plus 3 is 40 total) parameters make up one transform, and are roughly equivalent to a gene in biological genetics. The order of the transforms in the genome does not effect the solution image. Many transforms have visually identifiable effects on the solution, for example particular shapes, structures, textures, angles, or locations. The genome and its relation to the recursive set equation is depicted in Figure 1.4.



$$S = (\sum_j v_{0j} V_j(a_0 x + b_0 y + c_0, d_0 x + e_0 y + f_0))(S) \cup (\sum_j v_{1j} V_j(a_1 x + b_1 y + c_1, d_1 x + e_1 y + f_1))(S)$$

$$S = T_0(S) \cup T_1(S)$$

Fig. 1.4. A genome with two transforms, its formula, and its formula again written more abstractly.

On average there are five transforms in the function system, making for 200 (5×40) floating-point numbers in the genome. Note however that most sheep have most variational coefficients set to zero, which reduces the effective dimensionality of the space. However some sheep have many more than five transforms. When the XML representation of a large collection of genomes were compressed with gzip, they averaged 1.7KB each.

At the time of generation 165, there were only up to six transforms in the function system, and there were only 18 variations, resulting in 162 dimensions. The next version has 40 variations plus 24 additional parameters per transform.

1.3.3 Color and Palettes

The colors of the image are determined by a palette, which is a map from [0, 1] to color. During generation 165, the palette was determined by a single number, which selected one of several hundred built-in palettes. This integer color parameter has since been replaced with an arbitrary palette (768 bytes) in the genome. This is like a classic color-map of an 8-bit frame-buffer, but it is used just to determine the color of each particle, which is then drawn into a full-color image.

The original palettes were algorithmically derived from photographs of landscapes and famous paintings. The algorithm extracts colors from an input image (the trivial part) and then orders them to reduce the difference between adjacent colors in the palette (the hard part—finding an optimal solution is equivalent to the Traveling Salesman Problem). Today users who submit genomes can use their own source images to create palettes with this algorithm, or import arbitrary palettes.

This contrasts with the more common genetic approaches to colors, which are to either evolve an algebraic expression for the palette, or to evolve separate expressions for each color component (red, green, and blue, or hue, saturation, and value). The problem with these methods is that the color-space used (RGB or HSV) predominates, resulting in either mostly grays and unsaturated colors (from RGB) or garish rainbows (from HSV).

1.3.4 Animation and Transitions

The previous sections described how a single image is defined by the genome. To create animations, Electric Sheep rotates over time the 2×2 matrix part (a_i, b_i, d_i, and e_i) of each of the transforms. After a full circle, the solution image returns to the first frame, so sheep animations loop smoothly. Sheep are 128 frames long, and by default are played back at 23 frames per second making them 5.5 seconds long.

The client does not just cut from one looping animation to another. It displays a continuously morphing sequence. To do this the system renders transitions between sheep in addition to the sheep themselves. The transitions are genetic crossfades based on pair-wise linear interpolation, but using a spline to maintain C^1 continuity with the endpoints. This means that the derivative of the motion is also continuous, hence the motion is free of jerks.

Transitions are also 128 frames long. For each sheep created, three transitions are also created: one from another random flock member to the new sheep, one from the new sheep to a random flock member, and another one between two other random members. Most of the rendering effort is spent on transitions.

1.4 The Genetic Algorithm

There are three parts of the genetic algorithm: the rating system that collects the votes and computes the fitness of individual sheep, the genetic operators used to create new genomes, and the main loop that controls which live and die.

As already mentioned, users can vote for a sheep they like by pressing the up arrow key. If the sheep is alive its rating is incremented. Pressing the down arrow key decrements the rating. Votes for dead sheep are discarded. Votes during transitions are discarded. Users may also vote for or against a sheep by pressing buttons on its web page.

The ratings decay over time. Each day the ratings are divided by four with integer arithmetic rounding down.

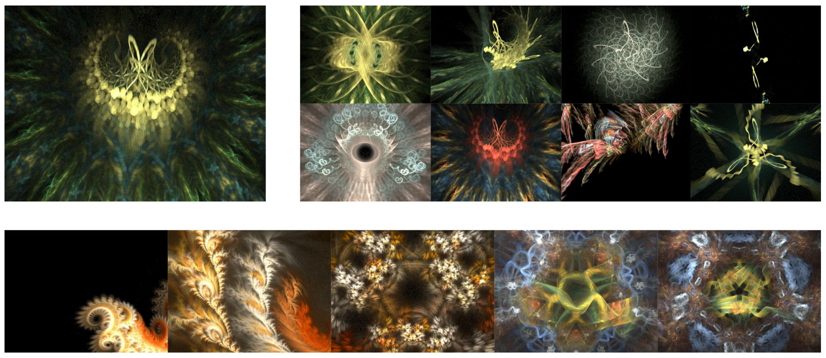


Fig. 1.5. Sheep 165.15875 (generation 165, number 15875), on the top-left, was born on August 16 and died 24 hours later after receiving one vote. It was one of 42 siblings. It was reincarnated on October 28 as sheep 165.29140, received a peak rating of 29, lived 7 days, and had 26 children, 8 of which appear to its right. Below are five descendants of a sheep in order parent to child, starting on the left. Their numbers are 165.01751, 165.01903, 165.02313, 165.02772, and 165.02975. The last is a result of mutation, the previous three of crossover, the first was posted by Liz Tomchek.

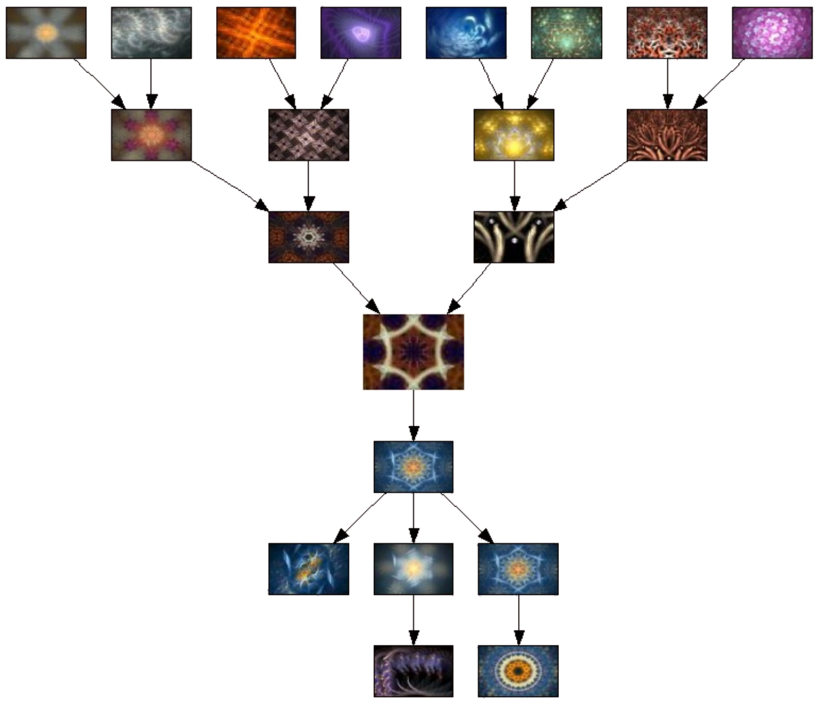


Fig. 1.6. The closest ancestors and descendants of Sheep 202.43868, the brown snowflake-symmetric sheep drawn slightly larger in the center. This sheep and these ancestors were generated by crossover, so each has two parents. The in-laws of the descendants generated by crossover are not shown, so whether they are mutants or children is not depicted.

1.4.1 Genetic Operators

There are four sources of genomes for new sheep: randomness, mutation, crossover, and posts from Apophysis. The parents for mutation and crossover operators are randomly picked from the current population weighted by rating. The probability of being selected is proportional to $\log_2(1+r)$, where r is the rating . Sheep that have received no votes have rating zero and so cannot be selected. When the study below was conducted, the probability was linear in the rating. The results appeared to follow a winner-take-all pattern though, with a few popular sheep dominating reproduction. Sample families appear in Figures 1.5 and 1.6.

randomness The affine matrix coefficients are chosen with uniform distribution from [-1, 1]. The variational coefficients are set to zero except for one variation chosen at random that is set to one.

crossover The crossover operation has three methods. The main method creates a genome by taking each transform (gene) from one parent or the other at random. Another method does pair-wise linear interpolation between the two parent genomes where the blend factor is chosen uniformly from [0, 1]. The last method takes the union of the two genomes. 1/10 crossovers use union, 2/10 use interpolation.

mutation The mutation operator has several different methods: randomizing just the variational coefficients, randomizing just the matrix coefficients of one transform, adding noise (-10 decibels, or numbers from [-0.1, 0.1]) to all the matrix coefficients, changing just the colors, and adding symmetry.

When applying these three automatic operators, the server renders a low-resolution frame and tests if the image is too dark or too bright. The operator is iterated until the resulting genome passes. For random genomes, 43% are rejected (in a test run 177 tries were required to get 100 passing genomes). This is a simple viability test.

post Human designers may post genomes to the server with Apophysis. Apophysis represents the matrix coefficients as triangles that the designer can drag, scale, and rotate. The variations are represented with type-in boxes, and the rest of the parameters are editable in one of several dialog windows. Apophysis is scriptable, and scripts are also often shared. Scripts are essentially user-defined, high-level genetic operators. Discovery plays a large roll in working with Apophysis.

In generation 202, all genomes are required to be under the CreativeCommons Attribution license [9] to allow derived works such as by the genetic algorithm. As a result, anyone can download any sheep, learn from its genome, improve it, and re-post it.

The server has a queue of sheep and transitions that are currently being rendered. When the queue is left with fewer than about 60 sheep, it is filled with genomes derived with one of the three automatic operators, or posted genomes if any are available.

In addition to picking parents for mutation and crossover from the current population, there are two additional sources of genomes. One is an archive of dead sheep from the current generation with peak rating of 2 or more, the other is a gene-bank of the sheep from previous generations with peak ratings of 4 or more. A sheep's peak rating is the highest rating obtained during its lifetime. The gene-bank is not represented in Figure 1.2.

In contrast with the older genetic algorithm of used during generation 165, the GA now substitutes previous good sheep for most random sheep. It also favors crossover over mutation. See Section 1.6 below for an explanation of *broods*, a further improvement of the genetic algorithm.

1.4.2 The Main Loop

The server maintains a single flock of sheep and continuously updates their ratings, creates new sheep, and kills off old ones. During generation 165, the server had 510MB of disk space for storing sheep animations, enough for 28 sheep and 83 transitions. Now it has 2.5GB. Each time a sheep is born, when it finishes rendering, the sheep with the lowest rating is killed to make room. If several sheep are tied for worst, then the oldest is taken (usually several sheep have received no votes and are tied with a rating of zero). Sheep also have a maximum lifespan of 7 days.

Killing a sheep removes the animation file from the server, but not from clients who may have allocated more disk space to their caches. The other records, including the peak rating, parentage, genome, a filmstrip of 16 thumbnails, and the first frame are kept. This archive may be browsed on the server either sorted by peak rating, as extended family trees, or by designer.

This on-line or steady-state approach contrasts with the more traditional genetic algorithm's off-line main loop that divides the population into generations and alternates between rating all the individuals in a generation and then deriving the next generation from the ratings. Note that Electric Sheep does have 'generations', but it means something else as explained in Section 1.2.

1.5 Empirical Results

The primary dataset of 4,100 genomes was collected from the server's database starting July 13th 2004 until October 13th. Previous versions of the server did not keep a record of the sheep: when they died they were completely deleted from the server. The data we have collected are a starting point to understanding the system and its behavior. However, they are somewhat confounded:

- The client uses the ratings to prioritize downloading. Since the server is busy enough that most clients cannot download all the sheep, this causes a snowball effect where a high rating itself causes more votes.
- The audience is fickle: sheep with identical genomes regularly receive completely different ratings (See Figure 1.5). Possibly the audience becomes fatigued by repeated exposure to variations of a successful genome, and stops voting for them. Even once popular sheep reintroduced much later do not necessarily fare well.
- Designers enlist others to vote for their sheep, post many similar sheep, or re-post the results of automatic evolution. There are three administrators who occasionally kill sheep, explicitly direct mating, mutation, reincarnation, and vote without limit.

In April 2006 there were 30,000 users of the screen-saver almost everyday. About 900 of them vote by pushing the arrow keys on the keyboard, and 20 vote while browsing the database on the web. On average each day, 14 genomes are submitted by 8 different designers. Over the six months of generation 198, 44 people submitted 5 or more genomes. By September 2006 the number of daily users has increased to 40,000.

Previously, user counts had to be estimated from unique IP addresses [6], but starting May 2005 with v2.6 the client generates a unique identifier (like a web cookie). By comparison over a one-day period, the IP address estimate was 10% over, but if counted over weeks, it was about double.

The collective intelligence has other more traditional channels as well: In the past month the general client user forum has averaged 13 messages per day. The genetic design discussion list has gotten 1.5 messages per day, and the Apophysis email list 10 messages per day. Wikis are used to supplement traditional documentation.

1.5.1 Amplification of Creativity

In a system with human-computer collaboration, we propose defining the *creative amplification* as the ratio of total content divided by the human-created content. If we compare the posted genomes with their evolved descendants we can measure how much creative amplification Electric Sheep provides.

In the primary dataset there were 21% hand-designed, posted sheep and 79% evolved sheep. If the sum is weighted by rating, then we get 48% to 52%, for an amplification factor of 2.08 (1+52/48). One could say the genetic algorithm is doubling the output of the human posters.

Of the 79% evolved sheep, 42% of them result from the totally random genetic operator. Their fraction of total ratings is only 3.8%.

There are some caveats to this metric. For example, if the genetic algorithm just copied the posted genomes, it might receive some votes for its 'creativity'. Or if it ignored the posted genomes and evolved on its own, it would receive some votes but they would not represent 'amplification'.

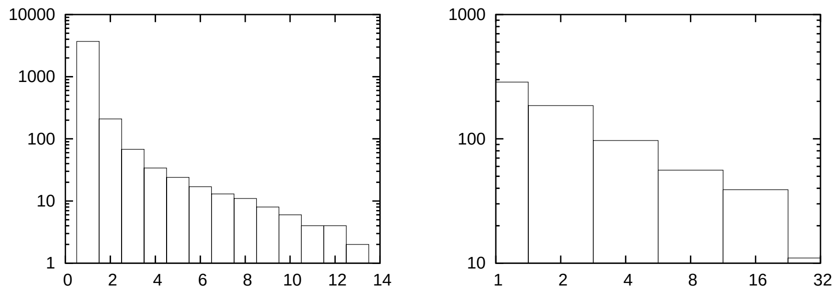


Fig. 1.7. On the left is a histogram of lengths of lineages. The length is on the horizontal axis, and the number of sheep is on the vertical. On the right is a histogram of the ratings of the sheep. Again, the rating is on the horizontal and the number of sheep is on the vertical.

Figure 1.7 shows the distribution of lengths of lineages of the sheep. The lineage length of a sheep is the maximum number of parent-to-child relationships that issue from it. Sheep with no children are assigned one, and sheep with children are assigned one plus the maximum of the lineage lengths of those children. Instead of fitness increasing along lineages, we find it dying out: the rating of the average parent is 6.7 but the average maximum rating of direct siblings is only 3.8.

The decay in ratings may result from the audience losing interest in a lineage because it fails to change fast enough, rather than a decay of absolute quality of those sheep. The viewpoint of watching the screen-saver and seeing sheep sequentially is different from the viewpoint of browsing the archive and comparing all the sheep. Neither can be called definitive.

Genetic algorithms normally run for many tens to hundreds or thousands of generations. In contrast, the lineages of the sheep are very short: the longest is 13.

1.6 Motivation and Direction

Electric Sheep illustrates the process by which the longer and closer one studies something, the more detail and structure appears. It investigates the role of experiencers in creating the experience. If nobody ran the client, there would be nothing to see.

Because the collective guides the evolution, no one has the burden of voting. Instead people act on inspiration. The network is used to assemble these bits of judgment efficiently. This avoids the common pitfall of aesthetic evolution, which is a dearth of human input.

The sheep are parasites of human attention. The goal of Electric Sheep is to create a self-supporting, network-resident life-form. Right now the sheep depend on a central server, requiring disk, bandwidth, and administration. Hopefully these inputs can be eliminated and the network can be made symmetric by using Distributed Hash Tables [2] and BitTorrent (see Section 1.2).

Furthermore the genetic algorithm depends on input from human designers. Our goal here is not to remove the input, but to improve what the genetic algorithm does with it: to increase the creative amplification factor. The ultimate goal then is divergence of the factor. In *Can a machine do anything new?* [14], Alan Turing wrote:

> One could say that a man can "inject" an idea into the machine, and that it will respond to a certain extent and then drop into quiescence, like a piano string struck by a hammer. Another simile would be an atomic pile of less than critical size: an injected idea is to correspond to a neutron entering the pile from without. Each such neutron will cause a certain disturbance which eventually dies away. If, however, the size of the pile is sufficiently increased, their disturbance caused by such an incoming neutron will very likely go on and on increasing until the whole pile is destroyed. Is there a corresponding phenomenon for minds, and is there one for machines? ... Adhering to this analogy we ask, "Can a machine be made to be supercritical?"

A big problem with the Electric Sheep's genetic algorithm is that the population size is too small. Good mutations are rare. So just eliminating the constraints of the central server and growing the population might help. If sheep have more children their chance of having one whose rating exceeds their own increases. The question is, will the audience become bored first?

A more direct way to improve the genetic algorithm would be to develop a model of the historic sheep ratings, and then use this model to screen the output of the genetic algorithm. This would be a more sophisticated viability test than the one described in Section 1.4.1. One useful input to this model could be the fractal dimension of the sheep, as it correlates well with aesthetics [12]. A histogram of total rating by dimension has a characteristic sharp peak between 1.5 and 1.7 [7]. This way risks homogenizing the sheep, however.

In the meantime we are experimenting with putting a human filter on the genetic algorithm, a technique we call the *brood.* The server now daily generates 256 potential children, but only renders one frame of each (this is about as expensive as rendering two ordinary sheep). The shepherd picks the best 40 or so of the brood. Future invocations of the genetic algorithm then use these picks, if available. Early indications are that the lengths of lineages have increased: so far the maximum in generation 202 is 30, compared to 13 for generation 165 (see Section 1.5.1).

A more fundamental problem with achieving open-ended evolution is the finiteness of the genetic code. One way to address that may be to replace the current fixed set of variations and coefficients with algebraic expressions, as in Genetic Programming [8]. Another would be to embed the current genetic code into a per-pixel, image-arithmetic language, like Sims' [11].

1.6.1 Dreams in High Fidelity

One way that the Electric Sheep are *not* self-supporting is financially: the developers are volunteers. There are many ways of supporting open-source software: for the Electric Sheep, donations, DVD sales, and advertisements on the website have proved to be inadequate. The new plan is to turn the bandwidth bottleneck (see Section 1.2.1) to our advantage.

Dreams in High Fidelity consists of a small computer driving a large 1280×720 liquid crystal display: a painting that evolves. It plays animations rendered by the Electric Sheep, but at triple the resolution and six times more frames per sheep. The image quality is striking on a large display. It requires 20 times the computation to make a high fidelity sheep.

The artist selects favorite sheep from the archives and public flock, and sends them back to be re-rendered at higher resolution: heaven for an electric sheep. So far two such flocks have been completed. The first is 55GB, totaling eight hours if played end-to-end, and requiring over one million CPU hours to render. The second is 100GB.

The *Dreams* have a symbiotic relationship to the screen-saver. The free version provides the design laboratory and gene pool from which the best sheep are extracted. It also provides the distributed supercomputer needed to realize the high fidelity content. Ideally the hifi version will fetch the income required to keep the whole project in operation, and develop it further.

1.7 Conclusion

The Electric Sheep demonstrate the feasibility of large-scale distributed interactive evolution. The network serves both as an artwork itself, and a platform for further research. In particular, this framework can be applied to other genetic codes besides fractal flames.

I believe the free flow of code is an increasingly important social and artistic force. The proliferation of powerful computers with high-bandwidth network connections forms the substrate of an expanding universe. The Electric Sheep and we their shepherds are colonizing this new frontier.

I look forward to many more generations of sheep at ever higher resolutions, with more expressive genetic codes, in three dimensions, responding to music, performing feats not yet imagined.

References

1. David Anderson et al. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45:56–61, 2002.
2. Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.
3. Michael Barnsley. *Fractals Everywhere.* Academic Press, 1988.
4. Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.
5. Scott Draves. The fractal flame algorithm. `http://flam3.com/flame.pdf`, 2004.
6. Scott Draves. The electric sheep screen-saver: A case study in aesthetic evolution. In *Applications of Evolutionary Computing, LNCS 3449.* Springer Verlag, 2005.
7. Scott Draves, Fred Abraham, Ralph Abraham, J. C. Sprott, and Pablo Viotti. Aesthetics and the fractal dimension of electric sheep. Presented at the annual meeting of the *Society for Chaos Theory in Psychology and the Life Sciences*, 2005.
8. John Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems).* The MIT Press, 1992.
9. Lawrence Lessig. *The Future of Ideas.* Vintage, 2002.
10. Bruce Parens. The open source definition. In *Open Sources: Voices from the Open Source Revolution.* O'Reilly, 1999.
11. Karl Sims. Artificial evolution for computer graphics. In *Proceedings of SIGGRAPH.* ACM, 1991.
12. J. C. Sprott. Automatic generation of iterated function systems. *Computers and Graphics*, 18:417–425, 1994.

13. Mark Townsend. Apophysis. `http://apophysis.org`, 2004.
14. Alan Turing. Can a machine do anything new? *Computing Machinery and Intelligence*, 59:433–460, 1950.

Acknowledgments

Many thanks to Dean Gaudet, Paul Graham, Scott Hassan, Tristan Horn, Nick Long, David McGrath, Erik Reckase, Matt Reda, Jeremy Richardson, Mark Townsend, and Chris Ursitti.