SBArt4 – Breeding Abstract Animations in Realtime

Tatsuo Unemi

***Abstract*— SBART was developed in early 1990's as one of the derivatives from Artificial Evolution by Karl Sims. It has a functionality to create a movie from a bred image through post-processing. The innovation of graphics processing unit (GPU) in these years improved the calculation performance to be fast enough to realize breeding animations in realtime on the personal computer. SBArt4 utilizes the advantage of GPU by compiling each expression in genotype into a type of OpenGL shading language. Even when it renders each frame of the animation in realtime, it achieves enough speed for users to evaluate the product of an abstract animation immediately. Though there were a number of problems because of architectural difference between CPU and GPU, almost compatible functionalities with the previous version have been implemented including reference to an external image and integer-based bitwise operation. Through experimental executions on some different hardware configurations, it was certified that it runs fast enough on recent consumer machines though some older machines are not powerful enough.**

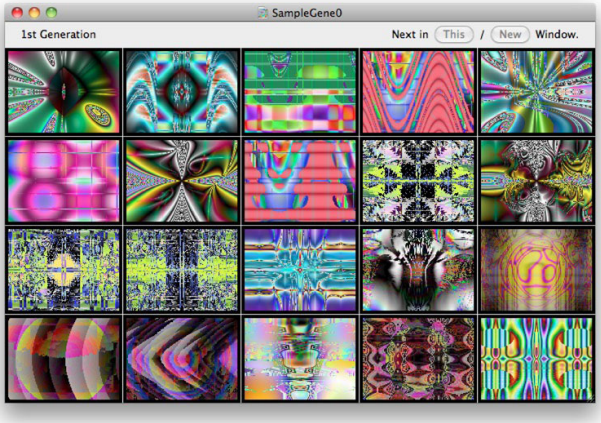

Fig. 1. An example field window of SBArt4.

I. Introduction

One of the active application fields of Interactive Evolutionary Computing (IEC) is Evolutionary Art mainly on abstract drawings. SBART [1] by the author is one of the derivatives from Artificial Evolution by Karl Sims [2], a tool to breed two dimensional abstract image utilizing a functional style of genotype to calculate a color value from xy coordinates for each pixel.

The previous version SBART3 [3] also has a functionality to create a movie from the result of breeding in a post-processing. To produce a series of frame images for an animation, the image for each frame is drawn using not only the spatial coordinate but also the time variable. The processing speed of personal computers was not fast enough for realtime rendering in this purpose at that time. It took some seconds for each frame for screen resolution of standard TV quality.

The performance of CPU for personal computers has improved as ten times faster than the age when the first version of SBART on UNIX workstation was released. However, it is still not enough for realtime rendering because it requires faster than 0.07 seconds per frame even when the expression includes more than 100 symbols and the image's resolution is of Full High-Definition. Another innovation in recent years is on Graphics Processing Unit (GPU). It was originally developed for acceleration of 3D rendering in polygon-based computer graphics, but it is attracting attention from various engineering fields in which hi-performance computing is required, such as fluid dynamics, structural analysis, molecular structure calculation, and so on, because it potentially has a huge power for parallel processing of number crunching. The rendering process in SBART is not on 3D polygons but on 2D pixels. In this type of purpose, OpenGL shading language (GLSL) [4] is a suitable tool for software development. It is a high level programming language that has a similar syntax with C language for description of a process applied to each pixel. A program fragment is compiled into a set of machine codes, and then is sent from CPU to GPU with data to compute large number of pixels in parallel.

The new version of software named SBArt4 utilizes the advantage of GPU by compiling each expression in genotype into Core Image Kernel Language [5], that is a dialect of GLSL supported by Apple Corp. in Mac OS X 10.4 or later. It is relatively easy to embed these functionalities in an application software by using Core Image framework available in the standard free programming environment of Xcode for Mac OS X.

The following sections describe system overview, compilation into GLSL, performance, and typical examples. Then this paper is closed with concluding remarks and future extension.

II. System Overview

SBART is a typical IEC application based on Simulated Breeding that allows the user to select individuals as parents in order to reproduce a population in the next generation. Differently from the framework of Interactive Genetic Algorithm, it uses no scalar grade for evaluation, that is, selected individuals become parents and the others are discarded. A typical graphical user interface named "field window" is shown in Figure 1 in which whole of the population containing 20 individuals are displayed in a window.

Tatsuo Unemi is with the Department of Information Systems Science, Soka University, 1-236 Tangi-machi, Hachioji, Tokyo, 192-8577 Japan (phone: +81 42 691 9429; email: unemi@iss.soka.ac.jp).

TABLE I
SYMBOLS FOR EXPRESSION IN GENOTYPE

Binary Operator	Unary Operator	Variable
$+$	$-$	(x, y, t)
$-$	abs	(x, t, y)
$\times$	sin	(y, x, t)
$\div$	cos	(y, t, x)
pow	log	(t, x, y)
hypot	exp	(t, y, x)
max	sqrt	
min	sign	
and	image	
mdist		

A. Genotype

Each individual has a genotype that includes a tree structure for a mathematical expression constructed from operators as non-terminal symbols and variables and constants as terminal symbols. A genetic operation of mutation or crossover is applied to the genotypes of the selected individuals in a style of Genetic Programming [6], that is, mutation by replacing a node by a symbol randomly chosen, and crossover by exchanging subtrees between parents.

Table I shows a set of symbols for operators and variables implemented in SBArt 4.1. Some of the operations have constraints for the domain of argument values such as square root and logarithmic function. To avoid an error during the calculation, some types of modification are applied to these cases as described later. For the trigonometric functions, sin and cos, each argument is multiplied by π to adjust it to the domain of the argument value given by another function.

Each variable is a vector constructed from three scalar variables, x, y, and t. x and y express spatial location of the target pixel, and t is a time variable for animation. A constant is represented in one scalar value, but it is expanded to a vector of three elements of the same values.

In addition to an expression, the range of time variable t is also included in a genotype because it is an important parameter to create an animation. This feature was not implemented in the previous version because the animation was not a target for breeding but for post-processing. The range is represented by a pair of floating point numbers, the start value t_0 and the span value s, that is, the range is $[t_0, t_0 + s]$ if $s > 0$. The value starts from t_0 and increases until it reaches $t_0 + s$ when s is positive, decreases when s is negative.

One of the genetic operations applied to the range of time variable is a type of mutation commonly used in a domain of continuous numerical values by adding a random number within a restricted range and distribution. In the current implementation, simple uniform distribution is employed of which range is depending on the value of span s in the genotype. The crossover is done simply by random choice of a value from one of the parents for each value of start or span.

B. Phenotype

The expression described above is to calculate the color value from xy coordinate and time t for each pixel in the image as the phenotype of individual. Each of the arguments and the results of any operation is a vector of three elements where the final result is interpreted as a color value in a space of hue, saturation, and brightness (HSB). Therefore, the time complexity in sequential computation to generate one frame image of a phenotype from a genotype is proportional to the multiplication between the number of pixels in the image and the number of operations in the expression. The number of steps to execute operations can be reduced to the depth of the tree structure by a parallel processing, but this point is not so much effective because the reduced steps depends on how many binary operators the expression includes. The parallel processing is more effective on the concurrent calculation among pixels, because it is theoretically possible for all pixels to be calculated in parallel. The rendering time is depending on the number of processing cores in GPU because it is usually less than the number of pixels in a single image.

The coordinate system in an image is arranged so that the center is the origin $(0, 0)$, the y coordinate of the top and bottom edges are 1 and -1 respectively, and the x coordinate is arranged in the same scale of y axis. If the aspect ratio of the image is $4 : 3$ for example, the range of x coordinate is $[-1.3\dot{3}, 1.3\dot{3}]$.

The calculation for each pixel is executed under the substitutions on the position of pixel for x and y, and time value for t.

C. Time variables

In addition to the time variable t in variable symbols, constant values are also the target for varying along the time axis by multiplying a factor $u + 1$ where u is another time variable in the same range of t. Two types of progression on the time variables were implemented. One is to simply iterate the value from t_0 to $t_0 + s$ for both t and u where these two variables always share the same value, that is,

$$t = u = t_0 + s\,T \tag{1}$$

where T is a variable repeatedly moving from 0 to 1.

The other one is to create a loop movie of which frame at the end smoothly connectes to the frame at the beginning. In this case, the values of two time variables are calculated based on equations

$$t = t_0 + s \sin 2\pi T, \tag{2}$$

$$u = t_0 + s \cdot \frac{1 - \cos 2\pi T}{2}. \tag{3}$$

These were designed so that both values start with the same one as t_0 but their phases in every moment are different.

D. Example

Figure 2 shows an example of genotype illustrated in a tree structure and the image as its phenotype. The expression of

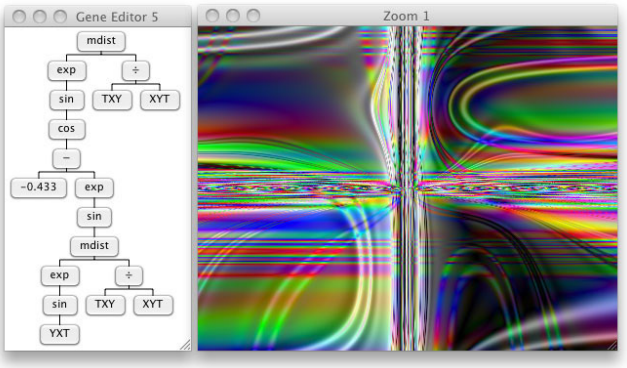

Fig. 2. An example of genotype and phenotype of an individual.

```
 1.// Core Image Kernel produced with SBArt4 (4.1 beta 1)
 2.// by Tatsuo Unemi, 2010-01-25 18:16:35 +0900
 3.// Time variable t = [-0.312925, 1.068027] (Linear)
 4.vec3 divide(vec3 a, vec3 b){
 5.    return a/compare(b,b,compare(-b,b,vec3(1.)));}
 6.vec3 cnst(float u,float c){return vec3(c+c*u);}
 7.kernel vec4 individual(float t, float u){
 8.    vec2 p=destCoord();
 9.    vec3 v=((exp(sin((cos(((cnst(u,-0.433000))-(exp(
10.        sin((((exp(sin((vec3(p.y,p.x,t))*3.141592)))+
11.        (divide(vec3(t,p.x,p.y),vec3(p.x,p.y,t))))/2.)*
12.        3.141592))))*3.141592))*3.141592)))+
13.        (divide(vec3(t,p.x,p.y),vec3(p.x,p.y,t))))/2.;
14.    float th=v.z*6.28394,
15.        x=v.y*cos(th)/3.,y=v.y*sin(th)/sqrt(3.);
16.    v=mod(abs(vec3(v.x+x+y,v.x-x-x,v.x+x-y))/2.,2.);
17.    v=compare(v-1.,v,2.-v);
18.    return vec4(compare(-v,pow(v,vec3(1.75)),vec3(0.)),
19.        1.);}
```

Fig. 3. A sample code in Core Image Kernel Language.

genotype in a mathematical formula is as follows.

$$\frac{\exp\sin\cos\left(-0.433-\exp\sin\frac{\exp\sin yxt+\frac{txy}{xyt}}{2}\right)+\frac{txy}{xyt}}{2} \quad (4)$$

The symbol named "mdist" is a function that calculates the average value between two arguments. yxt and the other sequences of three italic letters are variables listed in Table I constructed from three scalar variables. For example, yxt stands for (y, x, t). The coefficient π to be multiplied to the arguments of trigonometrical functions are omitted in the equation for simplicity.

III. Compilation into GLSL

GLSL has a syntax similar to C language but some special data types and built-in functions are available for graphic rendering and pixel handling. For example, it has data types to represent vectors, such as `vec2`, `vec3`, and `vec4` for two, three, and four scalar elements of `float`. Core Image Kernel Language is a dialect of GLSL that has more additional restrictions in the control statements including a condition, such as `if`, `for`, and `while`. This type of statement is allowed only when the condition can be determined before the execution. The three arguments' operator *X* `?` *Y* `:` *Z* and a built-in function `compare` described below are available for conditional selection of expression instead of combination of `if` and `else` statements.

A. Example

Figure 3 shows the code generated from the expression shown in Figure 2 and Equation 4. The main part representing the expression is in the five lines from ninth line that starts with `vec3 v=`.

The first three lines are comments including the range of time value. The next two lines are the definition of subfunction named `divide` for division symbolized by ÷ in Table I and Figure 2. To avoid an error because of zero divisor, division must not complied simply into normal divisor denoted by `/`. Instead, the definition of safe version of division function is added when the expression in genotype includes one or more operators of division. The built-in function `compare` takes three arguments of vectors to pick up elements from second or third argument depending on the value of corresponding element in first argument. The element in second argument is chosen if the element in first argument is negative, otherwise the element in third argument is chosen. The functional expression `vec3(1.)` means a vector of three 1.s. The definition specifies division is applied only when the divisor is not zero.

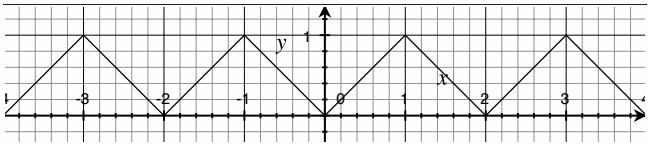

Fig. 4. The saw-shaped function for mapping the final scalar value within $[0, 1]$.

The sixth line is the definition of subfunction named `cnst` for a constant value. Because it varies with time variable u as described above, this subfunction is necessary when the expression includes a constant.

The seventh line is a declaration of main function. It takes two scalar arguments `t` and `u` corresponding to time variables t and u, and returns a vector of four elements that represents a color value for a pixel. The eighth line refers to a built-in function `destCoord` to assign spatial coordinate of the target pixel to the vector variable `p`. The elements `p.x` and `p.y` are used in a variable. For example, `vec3(p.y,p.x,t)` in the tenth line expresses a variable vector (y, x, t).

The final part from the fourteenth line is to convert the result of expression in genotype to the final value of RGBA color space. It employs a simplified version of translation from HSB to RGB with a side effect that makes the produced image more interesting. The sixteenth line that includes `mod` and `abs` and the seventeenth line with `compare` are to confine arbitrary scalar values of each element within $[0, 1]$ by mapping them using a saw-shaped function,

$$f(x) = \begin{cases} x \bmod 2 & \text{if } x \bmod 2 < 1 \\ 2 - x \bmod 2 & \text{otherwise} \end{cases} \quad (5)$$

illustrated in Figure 4. The final line includes a power func-

tion for gamma correction to darken the color appropriately. The alpha component in the final value in RGBA space is always 1. to indicate it is opaque.

B. Modified functions

As described above, some of the operators must be modified to avoid an undesirable runtime error due to a violation on the domain of the arguments. This special consideration must be applied to three of the operators listed in Table I, $\div$, log, and sqrt. The modified definitions are as follows.

$$x \div y = \begin{cases} x & \text{if } y = 0 \\ \frac{x}{y} & \text{otherwise} \end{cases} \quad (6)$$

$$\log x = \begin{cases} 0 & \text{if } x = 0 \\ \ln |x| & \text{otherwise} \end{cases} \quad (7)$$

$$\text{sqrt } x = \begin{cases} \sqrt{x} & \text{if } x \geq 0 \\ -\sqrt{-x} & \text{otherwise} \end{cases} \quad (8)$$

Similar type of care must be taken for the power function because the compiler producing the machine code breaks the built-in function `pow` down into a combination of exponential and logarithmic functions. This means it causes an error when the first argument is zero unless any special handling is made.

C. Selective functions

Almost all of the operators calculate a scalar value for each element independently, but the selective functions "min" and "max" depend on the combination of elements. Both functions take two arguments and pick up one of them according to the value of first element that expresses brightness. "min" returns the darker argument, and "max" returns the brighter argument. The code of definition of these functions are as follows.

```
vec3 myMax(vec3 a,vec3 b)
    {return (a.x>b.x)?a:b;}
vec3 myMin(vec3 a,vec3 b)
    {return (a.x<b.x)?a:b;}
```

D. Function to import external image data

The unary operator "image" is to embed data stored in an external image typically of a photograph. It takes one argument and interprets the first element as the horizontal coordinate and the second as the vertical coordinate. The result image is just identical to the external one if the argument is a variable (x, y, t). The coordinate system on the external image to indicate which pixels it refer to is adjusted so as to fit the vertical span of the external image to the height of the result image. The horizontal axis is adjusted so as to keep the aspect ratio.

Figure 5 shows the code to define the subfunction. It takes three arguments. The first is the argument value of the expression given in the genotype, the second is a two dimensional vector that expresses the size of external image in pixels, and the third is an image sampler that provides pixel values of external image. `sampler` is a built-in data type for this purpose. The values of latter two arguments are given from outside of the program fragment when rendering process starts. To receive the information, the declaration of kernel function definition in the seventh line in Figure 3 is extended with two more arguments same with these ones as follows.

```
kernel vec4 individual
    (float t, float u, vec2 s, sampler img)
```

Because a value given for coordinate has arbitrary value as the result of bred subexpression, the external image is expanded as if it has infinite size without boundary by filling the area with tiles of same or flipped images so that tiles are adjacent smoothly each other. The second line in Figure 5 picks up an RGB color value by a built-in function `sample` with the appropriate coordinate in the second argument, where `a.xy` denotes two dimensional vector constructed from first two elements in a three dimensional vector `a`.

The fourth line is for gamma correction to adjust it for further calculation, and the rest part is a simplified translation from RGB to HSB that works just as the inverse of the function from HSB to RGB in the bottom part of Figure 3.

```
vec3 image(vec3 a, vec2 s, sampler img){
    vec3 c = sample(img,s-abs(s-mod((a.xy+s.xy/vec2(s.y))
        *s.y/2.,s*2.))).xyz;
    c = compare(-c,pow(c,vec3(1./1.75)),vec3(0.));
    float x = c.x-2.*c.y+c.z, y = c.x-c.z;
    float r = sqrt(x*x+3.*y*y);
    float th = (r<=0.)?0.:(y>=0.)?
        acos(x/r):6.283184-acos(x/r);
    return vec3((c.x+c.y+c.z)/1.5,r,th/6.283184);}
```

Fig. 5. The definition of "image" subfunction.

```
vec3 myAnd(vec3 a,vec3 b){
    vec3 c=mod(a*10000.,16777216.),
        d=mod(b*10000.,16777216.),e,f=0;
    int i; for (i = 16777216; i > 0; i /= 2) {
        e = vec3(i);
        f += compare(c-e,vec3(0),compare(d-e,vec3(0),e));
        c -= compare(c-e,vec3(0),e);
        d -= compare(d-e,vec3(0),e); }
    return f/10000.; }
```

Fig. 6. The definition of bitwise "and" subfunction.

E. Integer-based bitwise calculation

Because the operators for bitwise logical calculation are not supported in Core Image Kernel Language, a kind of trick is necessary for an integer-based bitwise operator named "and" that was introduced in the previous version of SBART. Figure 6 shows its definition, utilizing a `for` loop by dividing the control variable `i` by two. This operator produces a type of discrete fractal pattern that is useful to enrich the variations of result images.

IV. Performance

The current version was tested with Mac OS X 10.6 on some different machines. It does not run fast enough on a relatively older machine, such as iMac 1.83GHz with ATI Radeon X1600, but the animation is smooth enough on recent ones, such as MacBook Pro equipped with 3.06GHz Intel

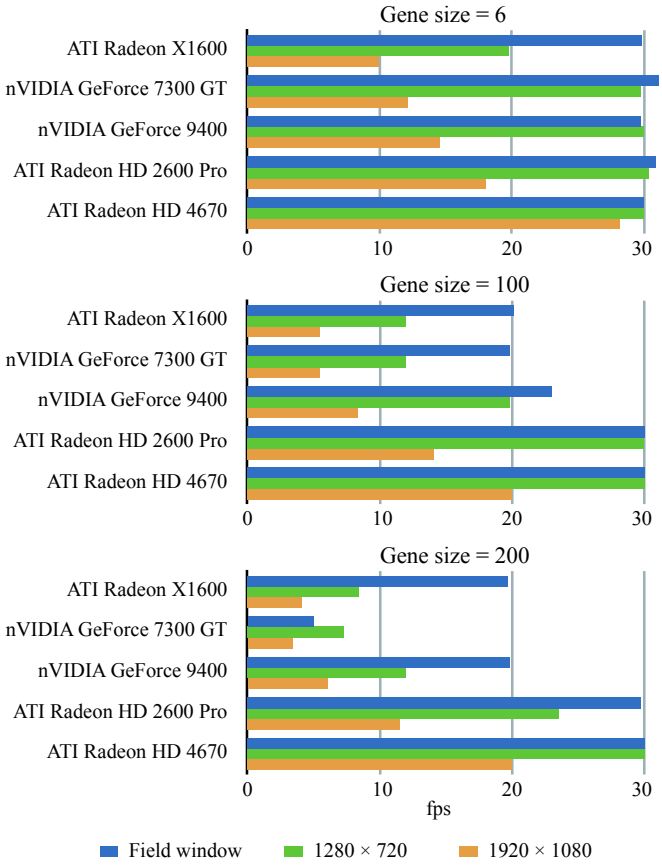

Fig. 7. Rendering Performance on Different Machines

Core 2 Duo Penryn CPU and NVIDIA GeForce 9400M GPU. The animation of single individual on a zoomed window works smoothly in the standard resolution of analog TV in all of the cases. It got slower but still acceptable for the full HD size of 1,920 by 1,080 pixels. The performance depends on a number of factors, such as frame buffer size, kernel memory size, the number of processing units, bus speed between CPU and GPU, and so on.

One of the troubles is on the simultaneous live animation for all twenty individuals in a field window. Though the image size for each individual is small, it does not work in enough smoothness on some machines. The main reason is that the program fragments for individuals are separated and they are loaded into GPU in turn for each frame. It might be able to be solved by unifying all of codes into a single program fragment, but it has not been examined yet since it is somewhat complicated to build up such a unified program. Instead, a caching mechanism was examined. It memorizes each frame image in the main memory to display it when the same frame image should be displayed again in later cycles. A capacity of memory is consumed, but it is negligible when the number of frames is less than 100 and the speed got faster enough after it completes caching the images for all of the necessary frames. It also requires a time to copy each image from GPU to CPU, but it is also negligible because the image size is small. The live animation is organized in 15 frames per second (fps) and 4 seconds in duration of one cycle for the default settings. The total number of frames in a single field window is 15 fps $\times$ 4 seconds $\times$ 20 individuals = 1,200. Therefore it occupies 120 pixels for width $\times$ 90 pixels for height $\times$ 4 RGBA components $\times$ 1,200 frames $\approx$ 5 M bytes without compression in a typical field window. This size is not so large waste of memory capacity for the recent machines even if the user opens tens of field windows.

Figure 7 summarizes the throughputs in fps on some machines of different hardware configurations for two different sizes of single images and one field window. The case of field window does not include the time to store each image in the main memory for cache. In all of the cases, three different sizes of expressions in genotypes were examined. One is a short expression that includes only six symbols, and the others contain 100 and 200 symbols that produces relatively complex pattern of phenotype image. The genotypes examined here were generated randomly excluding the unary operator "image" and the binary operator "and" because they require several steps for calculation differently from the others. Some of the measured values are near from 5, 10, 15, 20, and 30 due to the drawing mechanism in MacOS X. The rendering process for the screen is triggered every 1/60 seconds even if the data for drawing can be prepared in shorter time than the interval of refresh rate. The reason why an older machine is very slow to render a field window of large genotypes is probably the time required to load and execute the kernel program fragments.

SBArt4 also has a functionality of multi-field user interface [7] that allows the user to open arbitrary number of field windows in addition to arbitrary number of zoomed window. We restricted the animation running only on the front most window, because it would be useless for the user and overload for the machine if too many animations were running in all of windows at same time. It is clear that the animation on a hidden window is meaningless, but also the motions in windows in the outside of user's attention is just disturbance against user's observation on the focused individual.

V. Example Production

Figure 8 illustrates the frame sequences of example movies of the genotype in Figure 2. The upper one is of linear transition as explained in Equation 1 where the time variable t and u move from -0.312925 to 1.068027. The lower one is of cyclic transition in Equation 2 and 3 where t and u oscillate in different phases so that the final frame image smoothly continues to the first one.

For the production of such a movie by the previous version of SBART, we could neither breed the time parameters nor evaluate the motion in a field window. We needed to imagine during the breeding process how each image in the population would move, because it did not provide any method to take advantage of interactive evolutionary computing to breed the motion. It is not sure that an interesting still image is always connected with an interesting movie. An image of less interest might be a suitable seed to produce an amazing movie. Thus, it is important to implement the fast algorithm

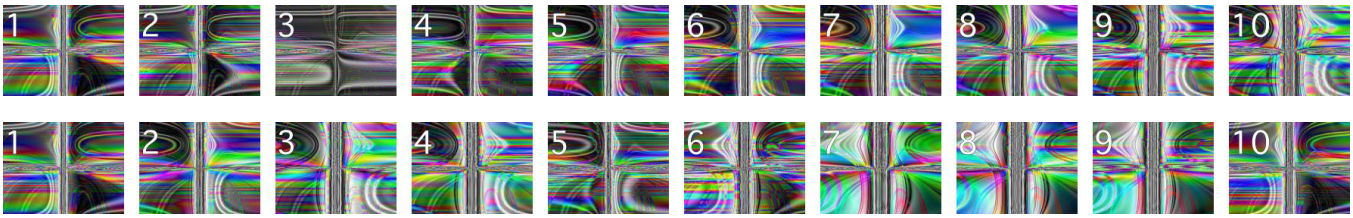

Fig. 8. Example movies. The upper one is of linear transition of time variable t from t_0 to $t_0 + s$. The lower one is of cyclic transition where the final frame smoothly continues to the first frame.

of realtime rendering to obtain a wide variation of satisfiable results.

VI. Concluding Remarks and Future Extension

A new version of the software tool named SBArt4 was introduced, that realized breeding of abstract animation in a style of interactive evolutionary computing. It requires much of computation to produce a distribution of colors in images for each frame in realtime. It would be still difficult if the calculation relied only on CPU, but recent innovation of GPU enabled it. By a method to compile a genotype to a program in GLSL, it became possible to utilize the functionality of this advanced technology effectively. There were some new type of difficulties because of difference between the architectures of CPU and GPU, but some of these problems could be solved.

The main part of functionalities in the previous version of SBART has almost completed to be exported to the new version, but still some points are remaining. Embedding an external movie instead of the still external image seems not so difficult because it is enough by feeding each frame image extracted from the given external movie in the similar manner with an external image even though it needs additional processing for extraction of appropriate frame and reconstruction of the sampler for GPU. A more difficult challenge is development of proper method to embed plural external images for making collage [8] and to import pixel data of 3D space from movie [3]. Because it is assumed that the final result of graphics is always a set of pixels in 2D space, Core Image framework has no feature to handle such 3D "boxel" data as a primitive. It does not seem impossible but looks difficult to realize within a reasonable costs for both time and space.

There are some types of functionalities that should be added or revised because of the newly introduced feature, that is, animation. The first one is on sound effects. The previous version has a facility to attach a sound track to the movie during the post-processing, but it should be embedded in the breeding process in the new version. In the current implementation, the parameters for sound wave generation is outside of the target of breeding, but an graphical user interface to manipulate them directly by sliders is provided. It is easy for these parameters to be included in a genotype, but at the same time it is required to introduce a mechanism of partial breeding [9] to allow the user to control which feature should be the target of breeding. Another consideration to be made is the design of user interface for breeding including sounds. There are also many researches on it for sounds and music, but we need to solve more complicated problem because of combination with visuals as the target of breeding. A type of user interface that allows the user to select one individual to listen to might be necessary similarly to the author's another approach for music [10].

The second one is on the user's experience. The system sometimes produces an animation that includes rapid changes of clear colors, a similar pattern that used to cause the trouble on TV animation program in Japan [11]. It is too strong stimuli for the nerves system of visual perception of the viewer. It might be necessary to introduce a method to measure and restrain such a undesirable side effect to avoid a negative effect to the users' health.

Visualization of phenotypes is one of the very important features for successful application of interactive evolutionary computing. Similarly with the system presented in this paper, developments in the other application fields that look for the optimal motion will be accelerated through improvement of the hardware for graphics processing.

References

[1] T. Unemi, "Simulated Breeding: A Framework of Breeding Artifacts on the Computer," in *Artificial Life Models in Software – Second Edition*, (Chapter 12), Edited by M. Komosinski and A. A. Adamatzky, London, UK: Springer-Verlag, 2009.

[2] K. Sims, "Artificial Evolution for Computer Graphics," *Computer Graphics*, vol. 25, pp. 319–328, 1991.

[3] T. Unemi, "Embedding Movie into SBART – Breeding Deformed Movies," in *Proc. of the IEEE Conference on Systems, Man and Cybernetics*, The Hague, Netherlands, 2004.

[4] J. Kessenich, "The OpenGL Shading Language – Language Version: 1.50," The Khronos Group Inc., http://www.opengl.org/ documentation/ glsl/, 2009.

[5] Apple Corp., "Core Image Kernel Language Reference," in *Mac OS X Reference Library*, http://developer.apple.com/mac/library/ , 2008.

[6] J. R. Koza, *Genetic Programming: On The Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[7] T. Unemi, "A Design of Multi-Field User Interface for Simulated Breeding," in *Proc. of the third Asian Fuzzy Systems Symposium*, Masan, Korea, pp. 489–494, 1998.

[8] T. Unemi, "SBART 2.4: an IEC Tool for Creating 2D Images, Movies, and Collage," *Leonardo*, vol. 35, no. 2, pp. 171, 189–191, MIT Press, 2002.

[9] T. Unemi, "Partial Breeding – a method of IEC for well-structured large scale target domains," in *Proc. of the IEEE Conference on Systems, Man and Cybernetics*, TP1D4, Yasmine Hammamet, Tunisia, 2002.

[10] T. Unemi, "A Tool for Multi-part Music Composition by Simulated Breeding," in *Proc. of the Eighth International Conference on Artificial Life*, Sydney, Australia, pp. 410–413, 2002.

[11] T. Takahashi, Y. Tsukahara, M. Nomura, and H. Matsuoka, "Pokemon Seizures," *Neurological Journal of South East Asia*, no. 4, pp. 1–11, 1999.