1 introduction – what is generative art?

The world’s favorite open source encyclopedia defines generative art as “art that has been generated, composed, or constructed in an algorithmic manner through the use of systems defined by computer software algorithms, or similar mathematical or mechanical or randomised autonomous processes.”¹. There are of course different methods for art creation. Traditionally, the artist is the genius responsible for creating a work of art. The skills and motivation of the artist are decisive for the resulting art. However, some artists have experimented with this decisive role of the artist. The composer John Cage for example used star-charts from the book *Atlas Borealis* to compose his *Etudes Boreales*². In these etudes he positioned the notes based on the positions of the stars on the map. This resulted in notes that seem to be random (but of course are not). By using semi-randomness, Cage diminished his personal role in the creation of the work of art. The music genre that is based on chance and randomness is called *Aleatoric Music*³. Of course music is not the only art form where chance and randomness can be used. Visual art can also use (semi-)randomness. For example the action painting of Jackson Pollock has the effect of basing the location where paint particles land on the canvas to semi-randomness (the different directional forces working on the individual particle and the laws of gravity). This contrasts the traditional painter who carefully places his brush on a specific point on the canvas. As a result, chance art is often more abstract and less realistic.

With the rise of the computer, the number possibilities to use chance and randomness exploded, and with the computer working more and more autonomous and without being controlled by a human artist, the term generative art was adopted. Of course it is (at the time of writing) still the human artist who supplies the computer with the algorithms that are used to generate art. Generative art can not only be music or static visual art (paintings), but also movies, poetry, 3d worlds and games, or complete (virtual) experiences. Of course these art forms are not limited to the virtual domain, although the virtual world is much easier to create these days.

---

²http://www.johncage.info/workscage/etudesboreales.html
³http://en.wikipedia.org/wiki/Aleatoric_music
2 examples – visual generative art artists

Before we start creating our own generative visual art application, it is useful to look at examples of existing generative art. A few of the greatest and best generative art artists are:

- Joshua Davis – http://www.joshuadavis.com/
- Erik Natzke – http://jot.eriknatzke.com/

Another great website about generative art is http://www.generatorx.no/

3 visual art – the painter metaphor

For this tutorial, we limit ourselves to visual generative art. Looking at the way visual art is normally created will help us recreate this method in code later on. Of course we don't necessarily have to construct our generative art program in analogy to the traditional painting process, but for simplicity we will do so in this tutorial. Usually, the painter uses paint of possibly different colors and distribute it on a canvas using a brush. The painting process is a process over time in which parts of the canvas get colored with paint one by one. The skill and motivation of the painter determine the color of the paint and the placement of paint on the canvas. The painter moves the brush to create shapes on the canvas, and the shape of the brush has influence on the shape of the paint on the canvas.

4 programming – the painter metaphor in ActionScript 3

How can we recreate this painting process in ActionScript3 code? In ActionScript3 graphics can be created by creating a BitmapData object and manipulate its pixels. Then we create a new Bitmap object and have it show the BitmapData on the screen. We can either choose to manipulate individual pixels of the BitmapData or use the ActionScript drawing api to create shapes with fill(s) and stroke(s). In analogy to the traditional painting process, we have to create a BitmapData object (canvas), move some virtual brush to a certain location on the BitmapData, and change the pixels at that location. This is a process over time. To create this process in time we will use the enterFrame event. An enterFrame event is created by the Flash Virtual Machine in which Flash applications run each time a new frame of the application is to be drawn. If a Flash movie runs at 15 frames per second, this means we will draw on the BitmapData 15 times per second.

5 in practice – building a generative art application

A visual art application

For this project we will use Flash Builder 4 and its internal Flex 4 compiler. After creating a new Flex4 web project named GenerativeArtTutorial, Flash Builder generates the default main application:

```
<?xml version="1.0" encoding="utf-8"?>
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo"
  minWidth="1024" minHeight="768">
</s:Application>
```
For this main application we set a fixed size of 800x600 pixels, and add a creationComplete handler, as well as a few MXML components. The Script component refers to an external ActionScript file, *GenerativeArtTutorial.as*, that will contain the ActionScript code for our application. The Panel is the main Visual element of our application, provides a title to the application and has a width and height of 100% so that it will occupy our complete application. Because a Bitmap object, which we will paint on, cannot be added to the application directly, we add a UIComponent component to the Panel. We will later add the Bitmap object to the UIComponent. By default it is only possible to manipulate individual pixels of a bitmap. If we want to use the flash drawing API later on to create shapes, we have to prerender them on a SpriteVisualElement component, and then copy them to the bitmap. Therefore we add an invisible (visible=false) SpriteVisualElement of the same size to the Panel.

Extended Main Application, *GenerativeArtTutorial.mxml*

```xml
<?xml version="1.0" encoding="utf-8"?>
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/halo"
    width="800" height="600" creationComplete="init(event)">
    <fx:Script source="GenerativeArtTutorial.as"/>
    <s:Panel id="panel" title="Generative_Art_Tutorial"
        width="100%" height="100%">
        <mx:UIComponent id="container" width="100%" height="100%"/>
        <s:SpriteVisualElement id="paintCanvas" width="100%"
            height="100%" visible="false"/>
    </s:Panel>
</s:Application>
```

Next we create the ActionScript file *GenerativeArtTutorial.as*, and add an empty init() function:

```ActionScript
//GenerativeArtTutorial.as
private function init(evt:Event):void {
}
```

We can now test the application using the *run* button in Flash Builder to verify that our Panel is created:

![Figure 2: Our application with the Panel and title.](image-url)
The next step is creating our own Brush object class. For this we create a new ActionScript file `Brush.as`. Our brush will just be a circle with a certain radius and a certain color:

```actionscript
package {
    import flash.display.Sprite;

    public class Brush extends Sprite {
        public var radius:Number;
        private var color:uint;

        public function Brush(radius:Number=12, color:uint=0x000000):void {
            this.radius = radius;
            this.color = color;
            draw();
        }

        public function draw():void {
            graphics.beginFill(color);
            graphics.drawCircle(0, 0, radius);
            graphics.endFill();
        }
    }
}
```

Now that we have created our Brush object class, it is time to improve our GenerativeArtTutorial.as file and add the Bitmap, BitmapData, Brush and all the event handlers. First we add global variables for the `Brush`, `BitmapData` and `Bitmap`, as well as a Boolean `mouseDown` to store the status of the left mousebutton. Then, within the `init()` function, we create a new BitmapData object `bmd` (of the same size as the container component), and pass it to the constructor of a new Bitmap object `bd`. Then we add the Bitmap object to the UIComponent object `container`, in order for the bitmap to show up in the container, and thus in the Panel.

```actionscript
private var mouseDown:Boolean;
private var brush:Brush;
private var bmd:BitmapData;
private var bm:Bitmap;
private function init(evt:Event):void {
    //setup Bitmap and BitmapData
    bmd = new BitmapData(container.width,container.height);
    bm = new Bitmap(bmd);
    container.addChild(bm);

    //setup Brush
    brush = new Brush(2,0xFF0000);
    container.addChild(brush);

    //setup MouseEvents
    this.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMove);
    this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
    this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);

    //start the painting process
    this.addEventListener(Event.ENTER_FRAME, paint);
}
```
We also instantiate the brush with radius 2 and a color 0xFF0000 (red), and add the brush to the container so it too will show in our application. Finally we add evenListeners to our application, so that the appropriate functions will be called whenever the mouse is moved, the left mouse button is pressed or released, and whenever a new frame is to be rendered by the application. We also add the corresponding handler functions to our GenerativeArtTutorial.as file:

Improved GenerativeArtTutorial.as (continued)

```flash
private function paint (evt : Event) : void {
    // paint function
}

private function onMouseMove (evt : MouseEvent) : void {
    brush.x = container.mouseX;
    brush.y = container mouseY;
}

private function onMouseDown (evt : MouseEvent) : void {
    mouseDown = true;
}

private function onMouseUp (evt : MouseEvent) : void {
    mouseDown = false;
}
```

The `onMouseMove` function will make sure that whenever the mouse is moved, the brush moves along. The `onMouseDown` and `onMouseUp` functions will allow us to keep track of whether the left mouse button is down or up. We can now test our application again (using the run button in Flash Builder) to verify that the Brush moves along with our mousepointer:

![Generative Art Tutorial](image)

Figure 3: The Brush can now be moved using the mouse.

Finally we get to the most important function of our application, the `paint()` function. Let’s start by putting black dots at the position of the brush whenever the mouse button is down. We have to draw the shapes first on the SpriteVisualElement, and then copy them to the BitmapData object using the `bmd.draw()` function. First we get the Graphics class of the SpriteVisualElement (`paintCanvas`). Then we start a fill with color 0x000000 (black). Next we check if the mouse is down, and if it is, we draw a circle with a random
size between 0 and 3 at the position of the brush. Finally we end the fill and copy the
circle shape we created on the SpriteVisualElement to the BitmapData, and clear the
SpriteVisualElement so that we start with an empty SpriteVisualElement in the next
iteration.

paint() function in GenerativeArtTutorial.as

``` ActionScript
private function paint(evt:Event):void {
    var g:Graphics = paintCanvas.graphics;
    g.beginFill(0x000000, 1);
    if (mouseDown) {
        g.drawCircle(brush.x, brush.y, Math.random() * 3);
    }
    g.endFill();
    bmd.draw(paintCanvas);
    g.clear();
}
```

If we test our application we see that dots of random size are created at the location of
our brush whenever we hold the mouse button down. We have created an art application
using Flex4 and ActionScript3. But is it a generative art application? Not entirely, as
the user’s mouse input is still an important factor in the placement of the 'paint'. If we
recapitulate the definition of Generative Art from the beginning, we can see that we have
indeed generated/composed/constructed art in an algorithmic manner through the use of
a computer system and defined by a computer software algorithm, but that it is not an
autonomous process yet.

Figure 4: A dot is placed at the location of the Brush every iteration.

More autonomy

Autonomy is always autonomy to a certain degree. Of course our art has been composed
more autonomous than it is usually created in traditional painting, because the (semi-)
random size of the dots. We could say that the more (semi-)random the art is, the less
control we as human authors have over the end result, and the more autonomous the
system is.

So let’s make the algorithm more autonomous by making it more (semi-)random. Consider this new paint() function:
More random paint() function in GenerativeArtTutorial.as

```javascript
private function paint(evt:Event):void {
    var g:Graphics = paintCanvas.graphics;
    g.beginFill(0x000000,1);
    for (var i:int = 0; i<50 ; i++) {
        if (mouseDown) {
            g.drawCircle(Math.random()*container.width,
                          Math.random()*container.height,
                          Math.random()*3);
        }
    }
    g.endFill();
    bmd.draw(paintCanvas);
    g.clear();
}
```

This paint() function is much like the previous one, but instead of drawing one dot, we are now drawing 50 dots per iteration. Also, we do not take the mouse cursor's position into account when placing the dots on the Bitmap, but just draw them at random anywhere on the paintCanvas (SpriteVisualElement).

Figure 5: Using the new paint() function, the dots are placed at random on the Bitmap.

The balance

Our application now delivers a completely different result. However, it may now be too random. In the end, art is often liked or disliked based on its visual appearance, beauty. When creating generative art, an artwork should be random (and autonomous) to a certain degree. It should not be completely random. The key point is to find the right balance. Great generative art artists like Erik Natzke and Joshua Davis are the best because they have found the best balance between randomness and structure on all or most aspects of a visual generative artwork: color palette(s), composition, density, movement.
What we have created is a very basic example of a generative art application. We have seen the basic structure of a generative art application that uses the painting metaphor. Finding the balance between randomness and organization requires a lot of experimenting with different painting strategies. Creative solutions often lead to unique results. Advanced techniques that can be used are for example: bezier curves to generate smooth lines, perlin noise to generate semirandom transitions and color palettes, double and or triple buffering to increase render quality, using input images (for example from flickr or google picasa) to find realistic color palettes and composition patterns, additional input devices such as touchscreens, camera’s and game console devices (wiimote, ps3 controller, xbox360 controller), Flex’s built-in visual effects and filters, and many more. To save resulting images the Adobe PNGEncoder or Adobe AIR file system functions can be used.

Project Files The Flash Builder 4 project files for this tutorial can be found at:
http://www.timenoithof.nl/projects/tutorials/GenerativeArtTutorial/