

JWildfire tutorial on: unleash your creativity by creating your own variations

by thargor6 (<http://thargor6.deviantart.com/>)

Version 0.1

The following tutorial describes how to create our own variations on the fly. You will not need to access the JWildfire codebase and need not any tools.

All you will need is a recent copy of JWildfire (Version 0.42 or higher is recommended) and little knowledge of the Java programming language. But if you know any other high level programming language (as Pascal oder C# or C++) you will probably also will have fun :-)

Table of Contents

1 Introduction.....	3
2 Introduction and simple example: a harmonic wave	3
2.1 The theory	3
2.2 The simplest of all variations: Linear.....	3
2.2.1 The variation preview in the editor.....	5
2.3 Creating custom variations using the custom_wf variation family.....	7
2.4 Creating our "workspace"	7
2.5 Editing code	8
2.5.1 Our first own variation	10
2.5.2 Create the wave variation	11
2.5.3 Using the variation	11
3 Advanced example: a checkerboard	12
3.1 Basic idea	12
3.2 The code	13
3.3 Using the checkerboard	15
4 Appendix.....	17
4.1 Mathematical functions and symbols	17
4.1.1 Static functions and symbols.....	17
4.1.2 Methods of the FlameTransformationsContext.....	17
4.2 Flame (with inlined sourcecode) of the first example.....	18
4.3 Flame (with inlined sourcecode) of the second example	19

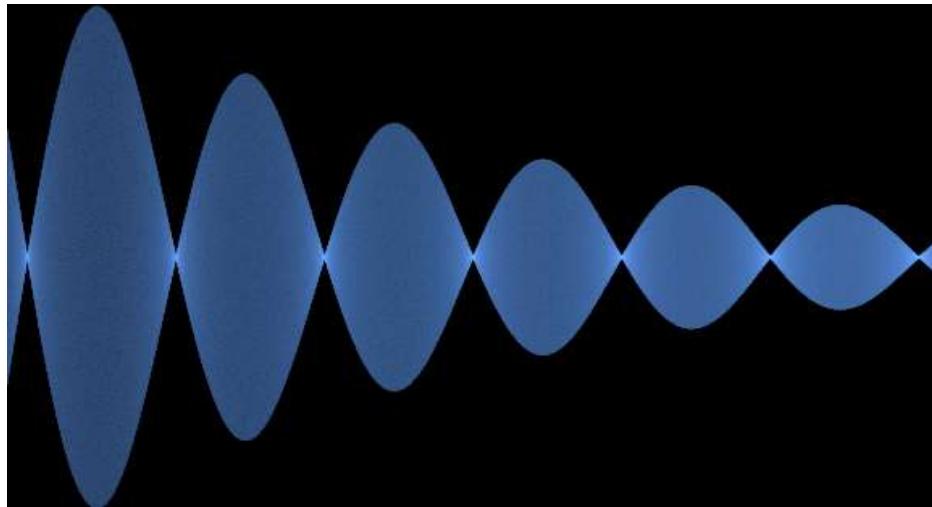
1 Introduction

JWildfire contains an integrated Java-compiler which allows to generate scripts and even variations during runtime. Those generated variations can be tested just on the fly without starting any external tool or performing any complicated step.

You just write the code and that's it :-) The code will be directly embedded as resource directly into your flame parameters.

And if you find a certain variation you created and tested this way is very useful you could integrate it in the JWildfire code base of course :-) (If you don't know how to do this just ask)

2 Introduction and simple example: a harmonic wave



2.1 The theory

A simple harmonic wave could be described by the formula:

$$y = \sin(x * \text{wavelength}) * \text{damping},$$

To use this formula in a custom variation we must go a little bit in detail of the rendering process. We remember that we create points which are transformed by

- an affine transformation: the "triangle" you see in the editor (rotate, shear, translate)
- the variation part: superimposition ("sum") of all variations you added to your transform
- one more optional affine part: the post transformation

We now are only interested in the second part. Both the affine transformation and the affine post transformation are done for us.

2.2 The simplest of all variations: Linear

Lets have a look at the original code of the simplest of all variations: the Linear variation:

```
public void transform(FlameTransformationContext pContext, XForm pXForm,  
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {
```

```
pVarTP.x += pAmount * pAffineTP.x;
pVarTP.y += pAmount * pAffineTP.y;
}
```

You see the following parameters:

- `FlameTransformationContext pContext`: the transformation context, gives access to some useful things - for example the random number generator
- `XForm pXForm`: the transform which is currently processed (not needed in the most cases)
- `XYZPoint pAffineTP`: the result from the affine transformation as input of the variation, we will **not** modify this here
- `XYZPoint pVarTP`: the point which holds the superimposition of all variations, we will modify this
- `double pAmount`: the amount of the variation as specified by the user in the editor

The transform itself does the following:

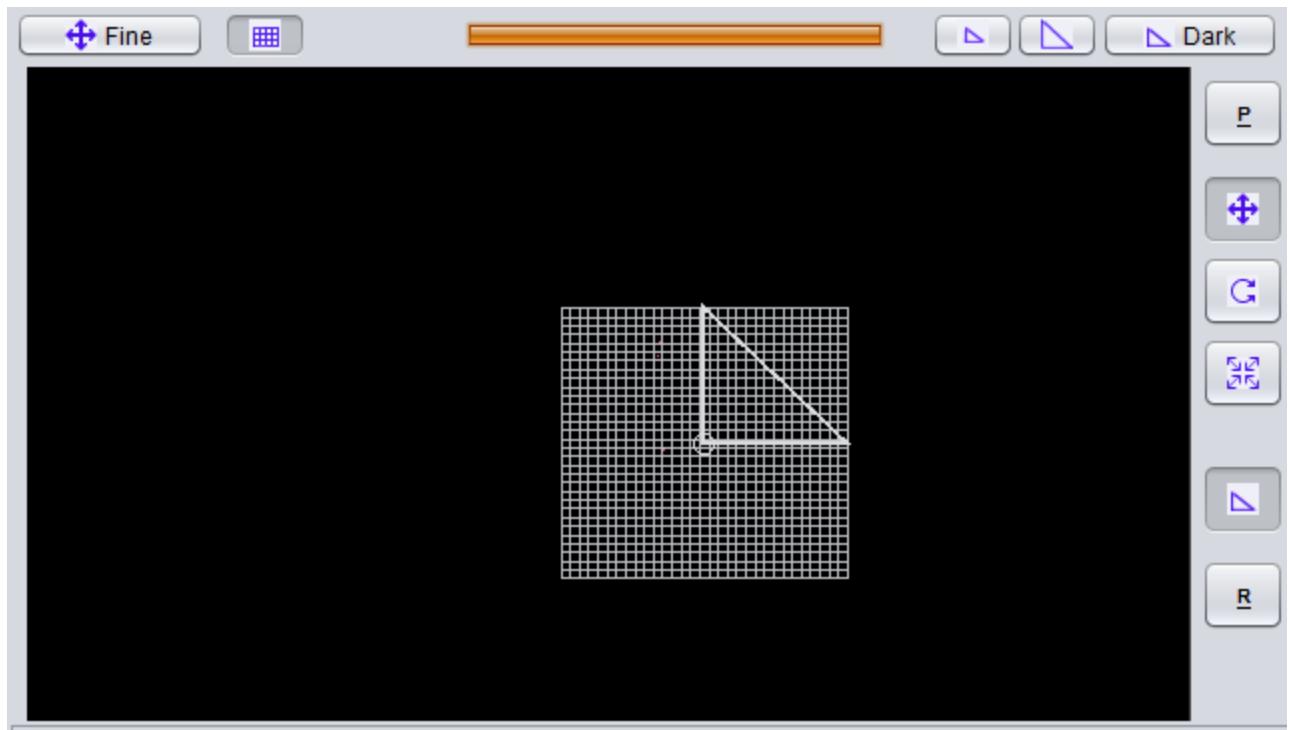
```
pVarTP.x += pAmount * pAffineTP.x;
pVarTP.y += pAmount * pAffineTP.y;
```

It simply adds the result of the affine transform (`pAffineTP`) multiplied by the amount `pAmount` the user specified to the transformation result (`pVarTP`). And this is done for the `x` and `y` coordinate respectively. (If you create a 3D-variation a third coordinate `z` comes into account, but we will not need it here.)

In the default case (where the amount is 1) the variation result is identical to result by the affine transformation. There is only a difference if you change the amount, then you are scaling your shape up or down.

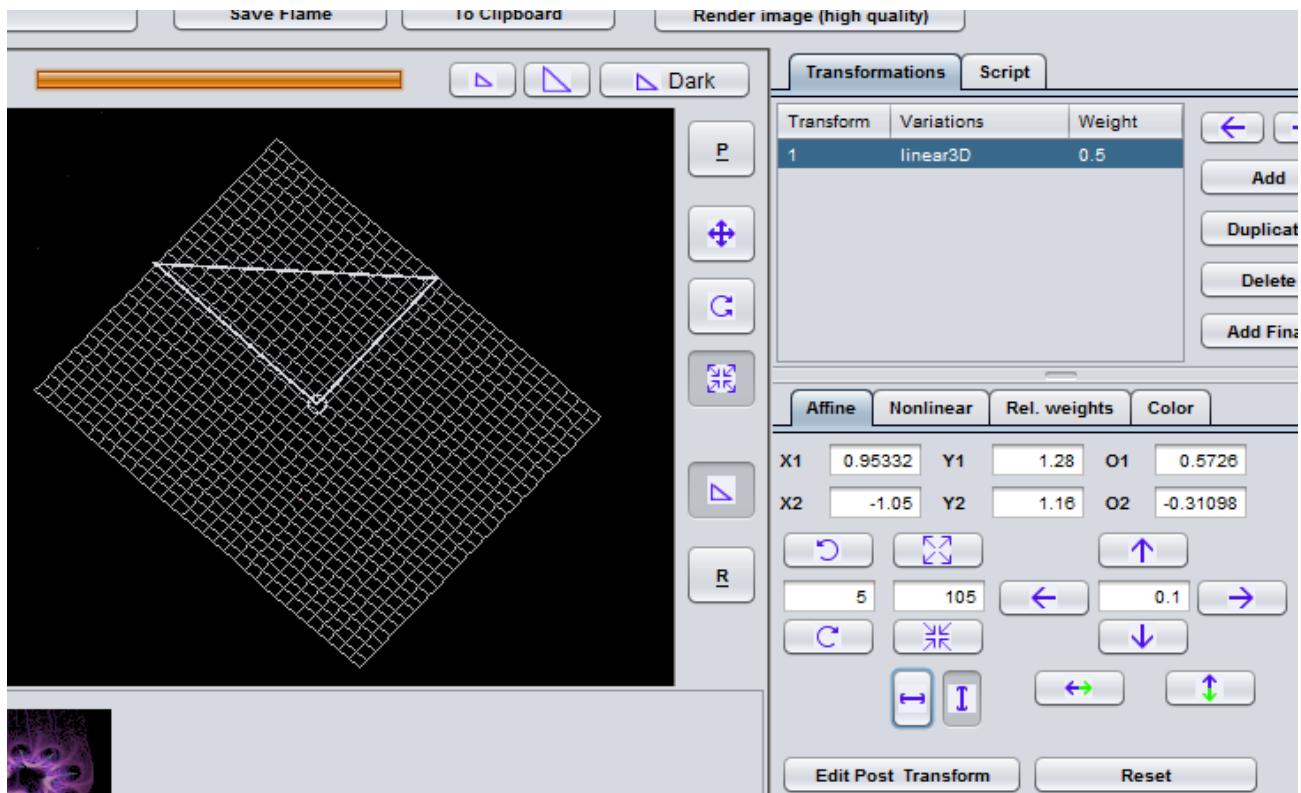
2.2.1 The variation preview in the editor

Now let's verify this in the editor: Create a blank flame by pressing "New from scratch" button and add a transform by pressing the "Add" button at the "Transformations" tab. In this case a Linear3D variation with the amount of 1 is automatically created. Activate the "Display variation effect" button in the preview area (left of the progress bar) so that you see a raster of points.



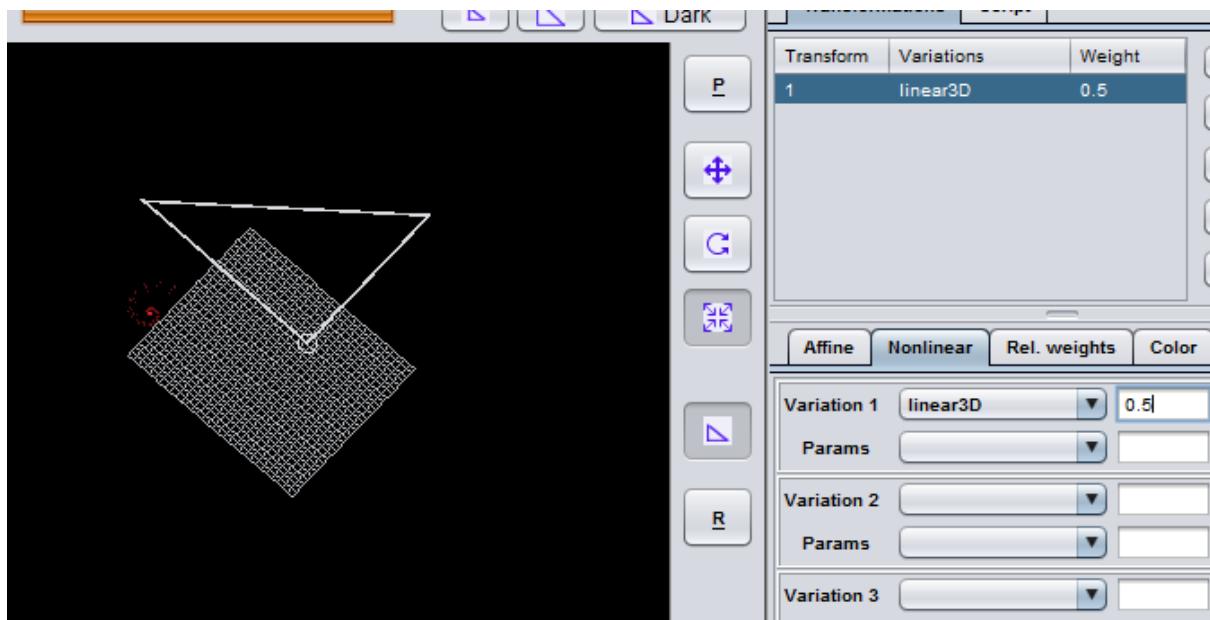
This raster shows the result of the transformation of a rectangular raster of points. In the default case the result of the transformation is the same as the original raster. I.E. the variation does not contribute anything.

But now we will change this. Just play little bit around with triangle, move it, rotate it, scale it, maybe to get an image like this:



All we changed so far was the affine part of the transformation.

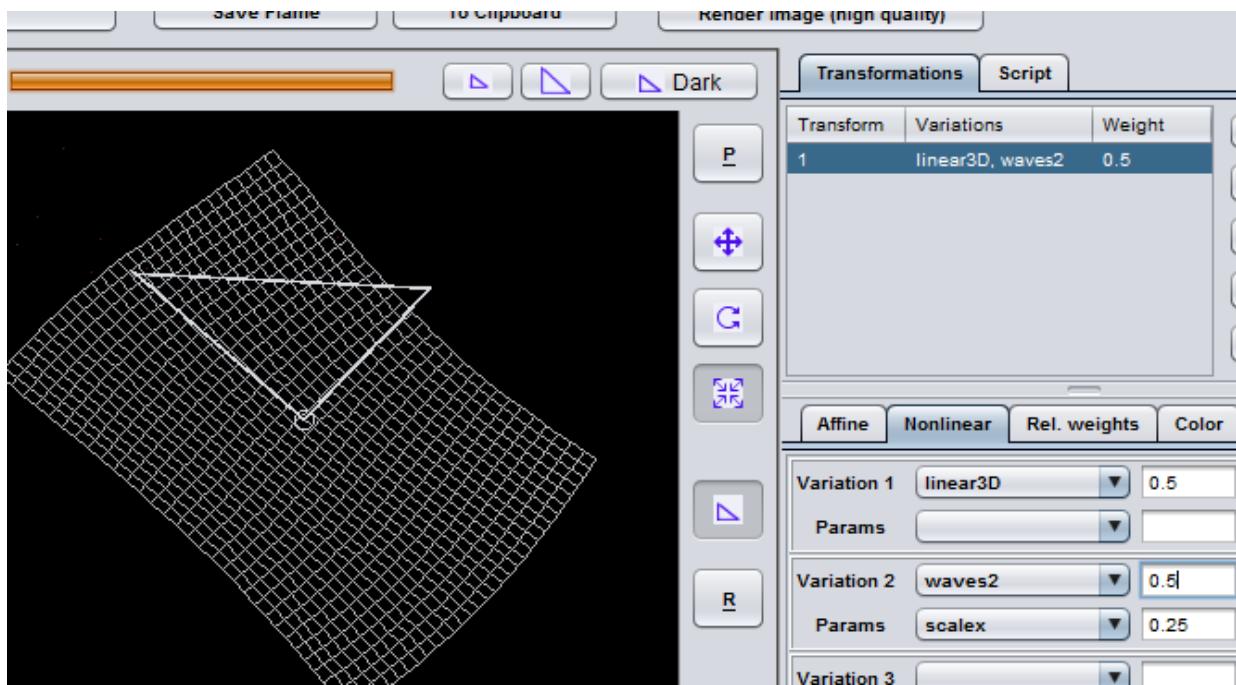
Now let's modify the variation part. On the "Transformations/Nonlinear" tab set the amount of the Linear3D variation to 0.5 and press <Return>. You see that the grid of points shrinks by the factor of 0.5:



Now let's add one more variation so see what superimposition means.

Set "Variation 2" on the "Transformations/Nonlinear" tab to "waves2". Nothing seem to happen so far because the amount of the new variation is zero. This means we are internally

calculation a waves2 contribution but multiplying this contribution by zero - which means no change. Now set the amount to 0.5 and press <Return> and you will get an slightly deformed grid of points:



You may want to play around with more variations now :-) You will discover some cases where only a "random mess" appears. This is all right then, in this case the certain variation just has a random component (for example: the blur variation).

2.3 Creating custom variations using the custom_wf variation family

To create a custom variation you use one of the three built-in variations

- pre_custom_wf: pre-variations (receiving **and modifying** the affine part)
- custom_wf: regular variations (receiving the affine part but modifying the variation part)
- post_custom_wf: post-variations (**receiving and** modifying the variation part)

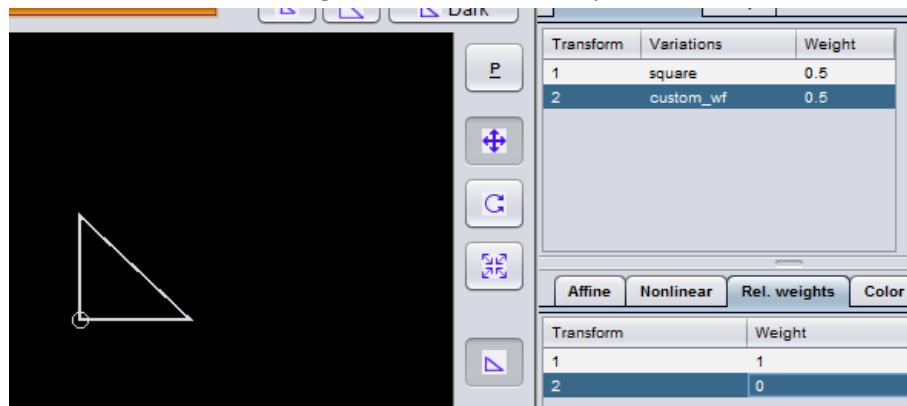
All of those variations serve you as the framework which you have to fill with the certain transformation - your Java code.

2.4 Creating our "workspace"

Now let's begin with the wave variation. At first we have to create a "workspace" - flame where we can easily apply different variations -and see a clear difference.

- Create a new flame by clicking the "New from scratch" button in the "Main" tab.
- Set the value "Pixels per unit" in the "Camera tab" to 140 and press <Return>
- Add a new transform by clicking the "Add" button in the "Transformations" tab
- Change the "Variation 1" to "square"
- Hide this transform by setting the "Draw mode" on the "Transformations/Color" tab to "HIDDEN"
- Add a new transform by clicking the "Add" button in the "Transformations" tab

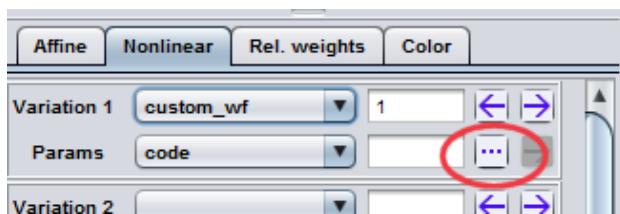
- Change the "Variation 1" to "custom_wf"
- Modify the relative weights, so that the transform 2 is linked to transforms 1:
 - Select the first transform and set the weight for transform 1 on the "Transformations/Rel. weights" tab to 0 and press <Return>
 - Select the second transform and set the weight for transform 2 on the "Transformations/Rel. weights" tab also to 0 and press <Return>.



Your flame now should only show a single point in the centre - this is because we have hidden the square and our custom variation is empty.

2.5 Editing code

To edit the code of one of the custom_ff variations you have to open the code editor of this variations. You do this by pressing the "..." after choosing the parameter "code":



Per default you will then see some example variations which are all deactivated because they are marked as comments. As in V0.42 the contents are as follows:

```
public void transform(FlameTransformationContext pContext, XForm pxForm,
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {

    // Some examples:

    // "hemisphere" variation

    //     double r = pAmount / sqrt(pAffineTP.x * pAffineTP.x + pAffineTP.y
* pAffineTP.y + 1);

    //     pVarTP.x += pAffineTP.x * r;
    //     pVarTP.y += pAffineTP.y * r;
    //     pVarTP.z += r;

    // -----
}
```

```

// change the color dynamically

//    if(pAffineTP.x<0) {

//        pVarTP.color = 0;

//    }

//    else {

//        pVarTP.color = 0.75;

//    }

// -----

// "bubble" variation

//    double r = ((pAffineTP.x * pAffineTP.x + pAffineTP.y * pAffineTP.y) / 4.0 + 1.0);

//    double t = pAmount / r;

//    pVarTP.x += t * pAffineTP.x;

//    pVarTP.y += t * pAffineTP.y;

//    pVarTP.z += pAmount * (2.0 / r - 1.0);

// -----

// "rose_wf" variation

//    final double amp=0.5;

//    final double waves=4;

//    double a0 = pAffineTP.getPrecalcAtan(pContext);

//    double r0 = pAffineTP.getPrecalcSqrt(pContext);

//    //

//    double r = amp * cos(waves * a0);

//    //

//    double nx = sin(a0) * r;

//    double ny = cos(a0) * r;

//    pVarTP.x += pAmount * nx;

//    pVarTP.y += pAmount * ny;

// -----

// ...

}

```

This all are valid variations and you can try out (and modify) them by removing the comment markers "///". You should only activate one of this variation at one time. But for now please remove all of them so that only the following code remains:

```

import org.jwildfire.create.tina.base.XForm;
import org.jwildfire.create.tina.base.XYZPoint;
import org.jwildfire.create.tina.variation.FlameTransformationContext;
import static org.jwildfire.base.MathLib.*;
}

public void init(FlameTransformationContext pContext, XForm pXForm) {

}

public void transform(FlameTransformationContext pContext, XForm pXForm,
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {
}

```

2.5.1 Our first own variation

Let's start with some very simple example. We want to transform the square to a rectangle having a width which is 3 times as large as its height.

So let's fill the function body with life now :-) Enter the two lines marked a bold:

```

public void transform(FlameTransformationContext pContext, XForm pXForm,
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {

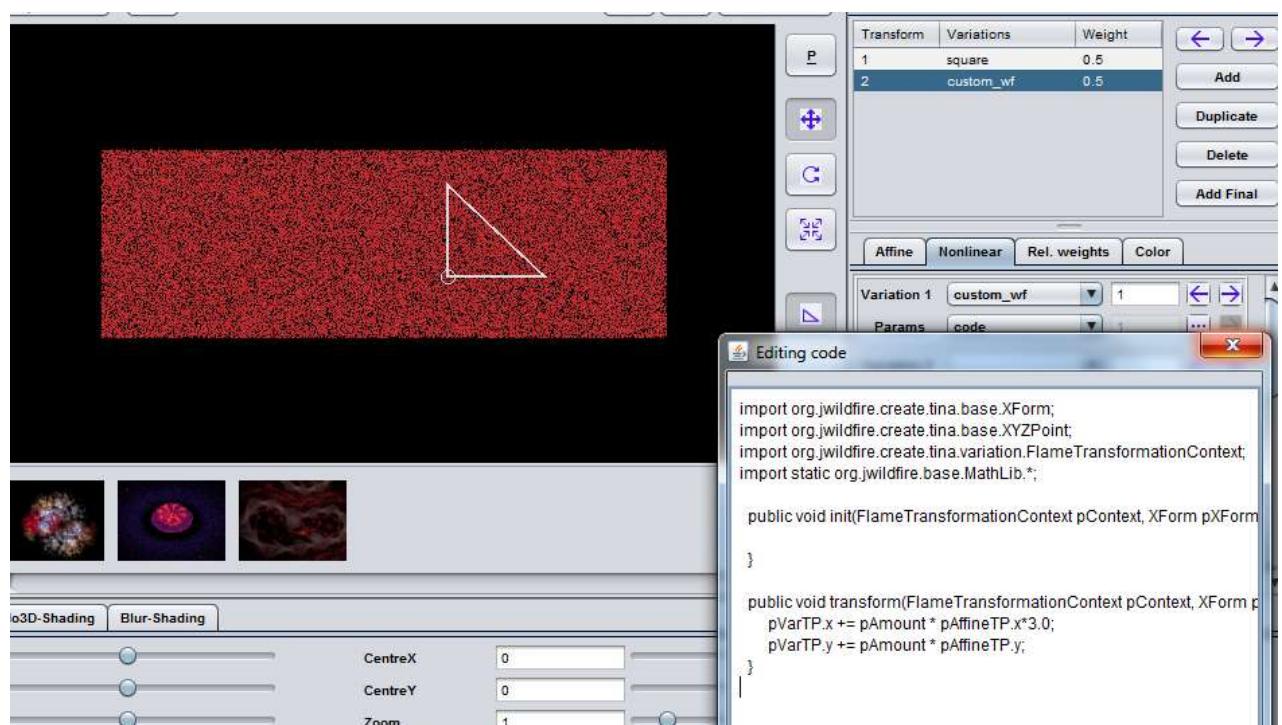
pVarTP.x += pAmount * pAffineTP.x*3.0;

pVarTP.y += pAmount * pAffineTP.y;

}

```

and press the "Ok"-Button of the window. Your code then gets compiled and integrated into JWildfire showing the change immediately. You should now see a rectangle:



2.5.2 Create the wave variation

From now on most things should be easy! You can now experiment with formulas and unleash your own creativity. Because you now have a full fledged programming language there are many ways to do things. The possibilities are endless.

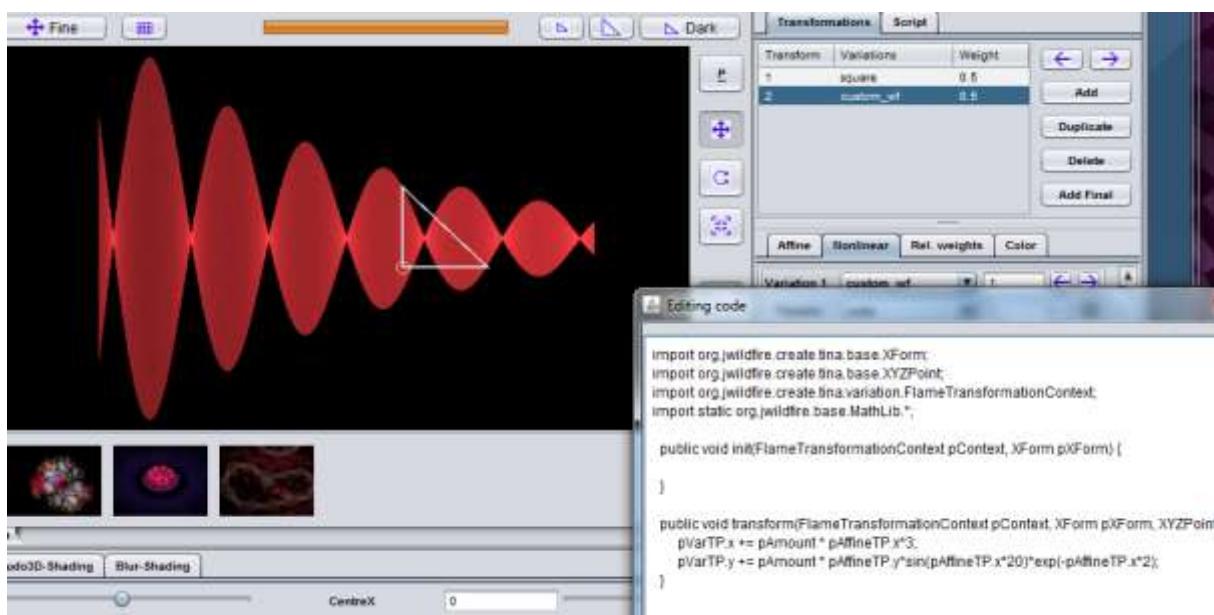
One approach to create a wave with damping could be:

```
public void transform(FlameTransformationContext pContext, XForm pxForm,
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {

    pVarTP.x += pAmount * pAffineTP.x*3;

    pVarTP.y += pAmount * pAffineTP.y*sin(pAffineTP.x*20)*exp(-
pAffineTP.x*2);

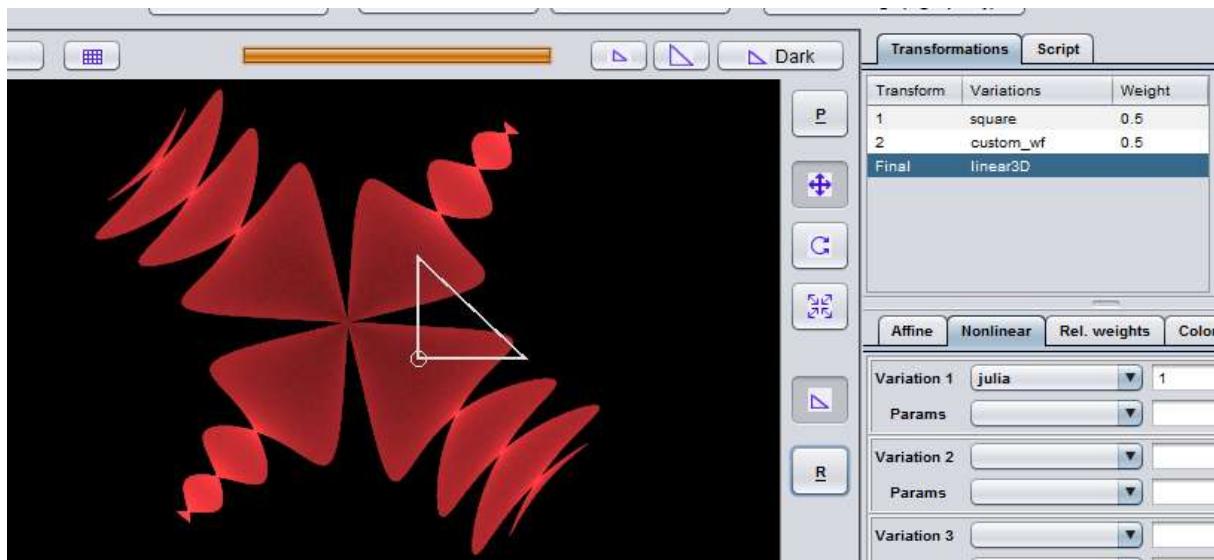
}
```



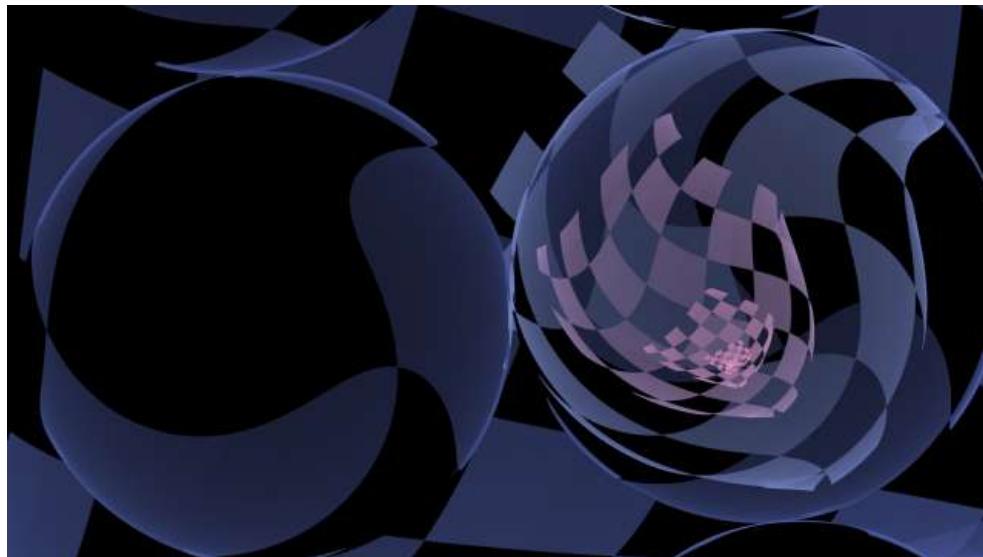
2.5.3 Using the variation

You can now use your variation as any other variation. Just let's add a final transformation to create a funny looking fractal!

Add a final transformation by pressing the "Add final" button at the "Transformations tab". Change the "Variation 1" to "julia" and you have created some unique shape :-)



3 Advanced example: a checkerboard



Let's create a checkerboard where we can specify the number and size of the fields and use this checkerboard as texture for our fractals for a refreshing different look!

3.1 Basic idea

In this example we want to create just a monochrome checkerboard (which then later can be affected by other transforms and can receive color this way).

The simplest way to create such a shape is just to fill it with random points. So we can ignore the affine transformed input vector and just create the shape.

Let's start with a simple square:

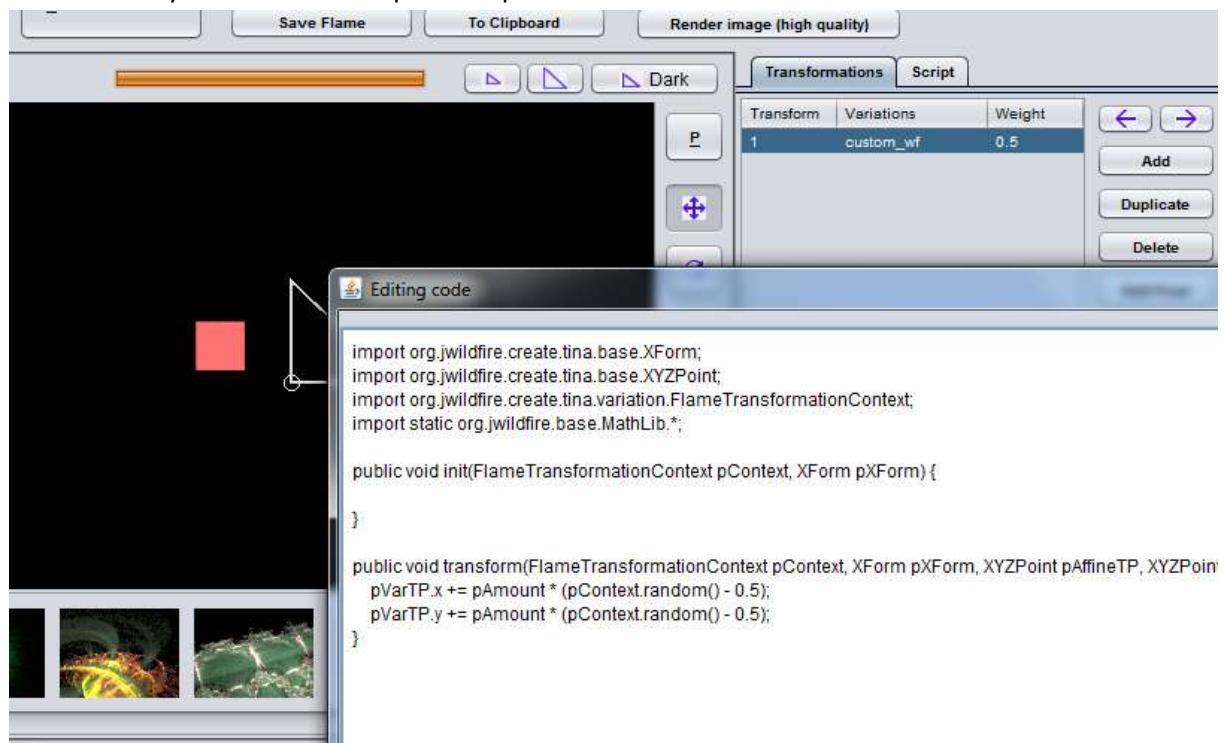
- Create a new flame by clicking the "New from scratch" button on the Main tab.
- Add a new transform by clicking the "Add" button on the "Transformations" tab
- Change the "Variation 1" to "custom_wf" and fill the transform()-method with the following code:

```

public void transform(FlameTransformationContext pContext, XForm
pXForm, XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {
    pVarTP.x += pAmount * (pContext.random() - 0.5);
    pVarTP.y += pAmount * (pContext.random() - 0.5);
}

```

- Press OK and you should see a square shape:



The basic idea is now to place more of those squares to build a checkerboard from squares. You can achieve this by adding a local random coordinate (the square shape) to a global random coordinate (the position of the square on the board).

3.2 The code

There are again many ways to do this. In my example you can specify freely the number of rows and columns:

```

public void transform(FlameTransformationContext pContext, XForm pXForm,
XYZPoint pAffineTP, XYZPoint pVarTP, double pAmount) {

    double fieldWidth=1.0;

    double fieldHeight=1.0;

    int rows=7;

    int cols=8;

    int size=rows*cols;

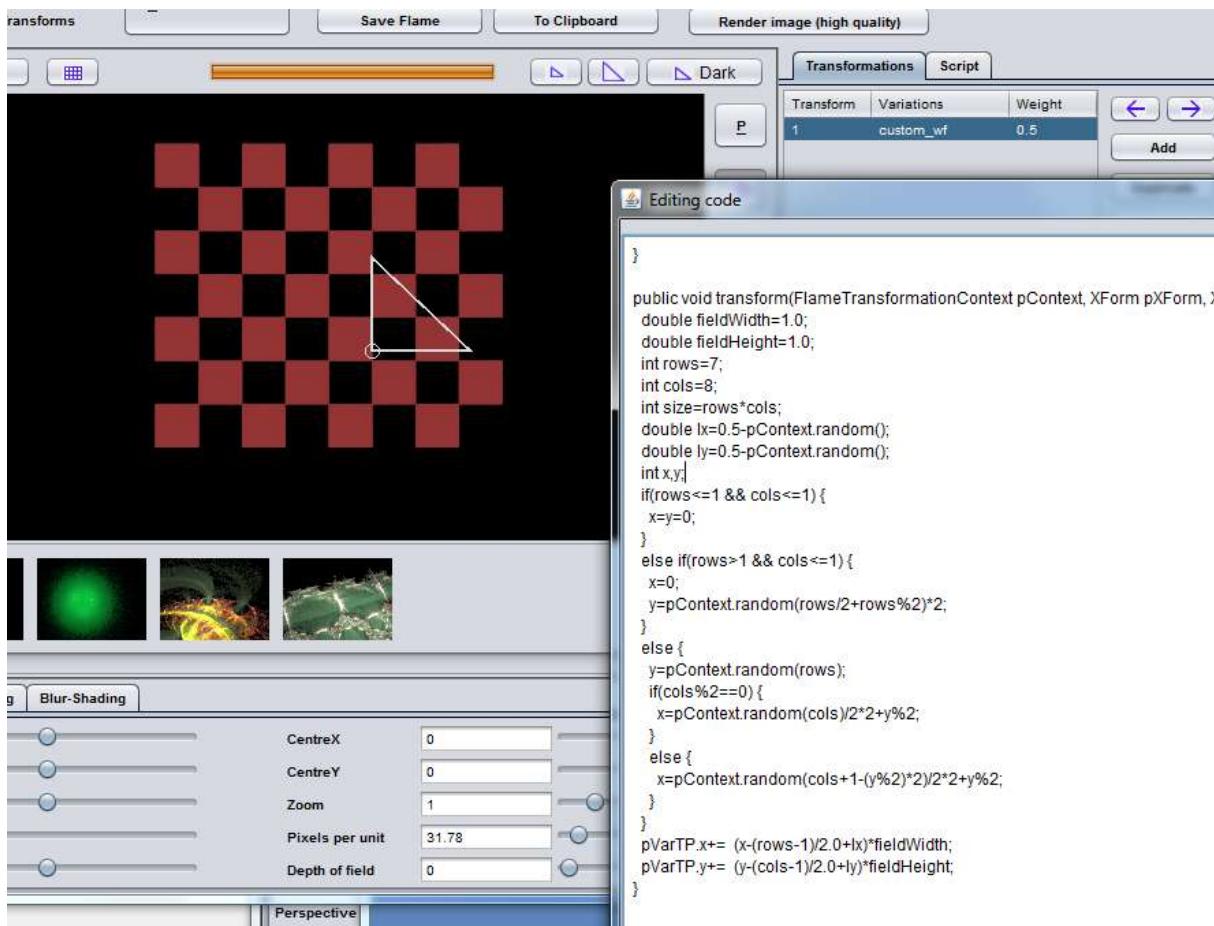
    double lx=0.5-pContext.random();

    double ly=0.5-pContext.random();

    int x,y;

```

```
if(rows<=1 && cols<=1) {  
    x=y=0;  
}  
  
else if(rows>1 && cols<=1) {  
    x=0;  
    y=pContext.random(rows/2+rows%2)*2;  
}  
  
else {  
    y=pContext.random(rows);  
    if(cols%2==0) {  
        x=pContext.random(cols)/2*2+y%2;  
    }  
    else {  
        x=pContext.random(cols+1-(y%2)*2)/2*2+y%2;  
    }  
}  
  
pVarTP.x+= (x-(rows-1)/2.0+lx)*fieldWidth;  
pVarTP.y+= (y-(cols-1)/2.0+ly)*fieldHeight;  
}
```

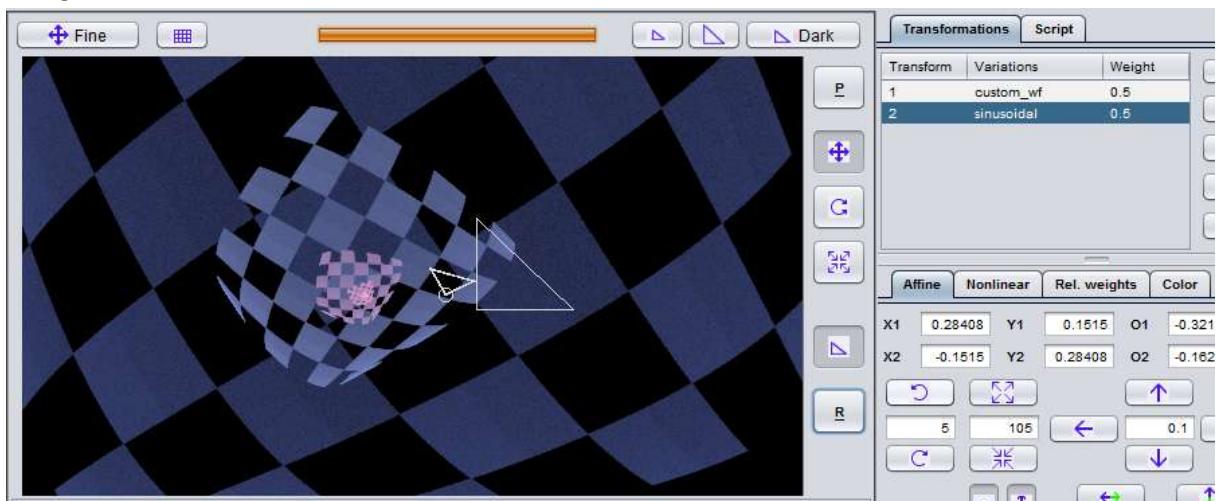


3.3 Using the checkerboard

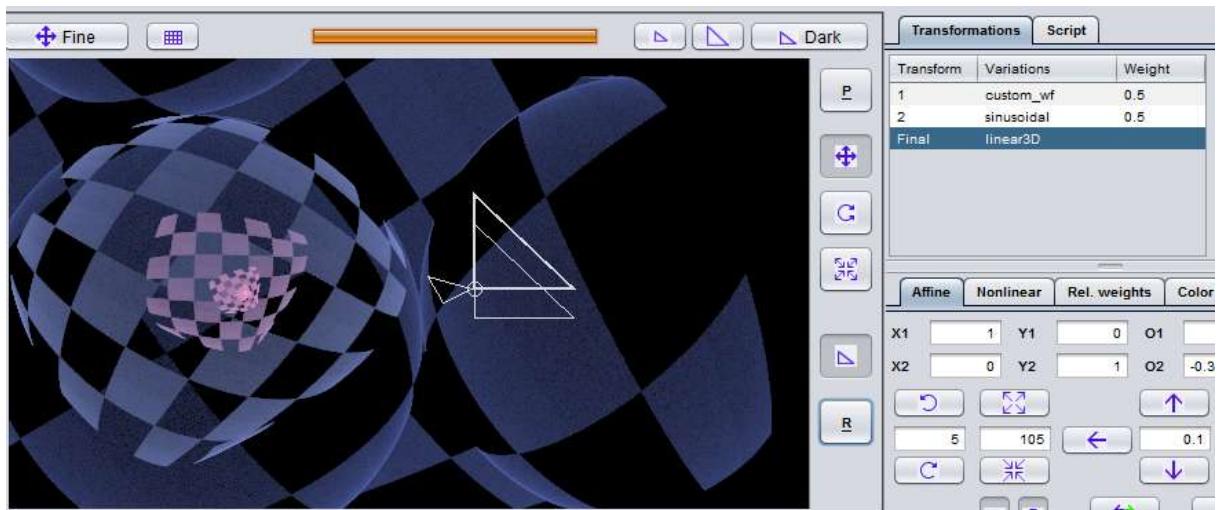
You can now use the checkerboard as any other variation of this type (e. g. to replace a square variation in a old fractal of yours).

Or just play around with it:

- Create a new transform by pressing the "Add" button from the "Transformations" tab
- Change "Variation 1" of this new transform to "Sinusoidal" and drag, size and rotate the triangle a little bit around:



- Create some nice bubbles effect by adding a final transformation by pressing the "Add final" button on the transformations tab and changing "Variation 1" to "bwraps7"
- Modify a triangle of the final transform until you receive an interesting effect



That's it so far. All code samples (inside the flame files) are supplied in the appendix. Have fun to try your own things! :-)

4 Appendix

4.1 Mathematical functions and symbols

4.1.1 Static functions and symbols

The following mathematical functions and symbols are available "out of the box":

- double EPSILON = 0.00000001;
- int TRUE = 1;
- int FALSE = 0;
- double M_PI = Math.PI;
- double M_PI_2 = M_PI * 0.5;
- double M_PI_4 = M_PI * 0.25;
- double M_1_PI = 1.0 / M_PI;
- double M_2_PI = 2.0 / M_PI;
- double M_2PI = 2.0 * M_PI;

- int abs(int var);
- double fabs(double var);
- int sign(double val)
- double sin(double a);
- double cos(double a);
- double tan(double a);
- double fmod(double a, double b);
- double sqr(double a);
- double atan2(double y, double x);
- double exp(double a);
- double sqrt(double a);
- double pow(double value, double power);
- double floor(double value);
- double round(double value);
- double log10(double value);
- double log(double value);
- double sinh(double value);
- double cosh(double value);
- double tanh(double value);
- double min(double a, double b);
- double max(double a, double b);
- double atan(double value);
- double acos(double value);
- double asin(double value);
- double rint(double value);

4.1.2 Methods of the FlameTransformationsContext

- double random();
(returns a random number greater equal 0.0 and smaller than 1.0)

- int random(int pMax);
(returns an integer random number greater equal zero and smaller pMax)

4.2 Flame (with inlined sourcecode) of the first example

```

<flame name="JWildfire" version="0.42 (31.03.2012)" size="581 327" center="0.0 0.0"
scale="140.0" rotate="85.0" oversample="1" color_oversample="1" filter="1.2" quality="100.0"
background="0.0 0.0 0.0" brightness="4.0" gamma="4.0" gamma_threshold="0.04"
estimator_radius="9" estimator_minimum="0" estimator_curve="0.4" temporal_samples="1.0"
cam_zoom="1.0" cam_pitch="0.0" cam_yaw="0.0" cam_persp="0.0" cam_zpos="0.0" cam_dof="0.0"
shading_shading="FLAT" >

    <xform weight="0.5" color="0.0" opacity="0.0" symmetry="0.0" square="1.0" coeffs="1.0 0.0 0.0
1.0 0.0 0.0" chaos="0.0 1.0" />

        <xform weight="0.5" color="0.0" symmetry="0.0" custom_wf="1.0" custom_wf_a="0.0"
custom_wf_b="0.0" custom_wf_c="0.0" custom_wf_d="0.0" custom_wf_e="0.0" custom_wf_f="0.0"
custom_wf_g="0.0"
custom_wf_code="696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E626173652E58
466F726D3B0D0A696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E626173652E5859
5A506F696E743B0D0A696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E7661726961
74696F6E2E466C616D655472616E73666F726D6174696F6E436F6E746578743B0D0A696D706F727420737461746963
206F72672E6A77696C64666972652E626173652E4D6174684C69622E2A3B0D0A0D0A20207075626C696320766F6964
20696E697428466C616D655472616E73666F726D6174696F6E436F6E746578742070436F6E746578742C2058466F72
6D207058466F726D29207B0D0A0D0A20207D0D0A0D0A20207075626C696320766F6964207472616E73666F726D2846
6C616D655472616E73666F726D6174696F6E436F6E746578742070436F6E746578742C2058466F726D207058466F72
6D2C2058595A506F696E74207041666696E6554502C2058595A506F696E74207056617254502C20646F75626C6520
70416D6F756E7429207B0D0A20202020207056617254502E79202B3D2070416D6F756E74202A207041666696E655450
2E792A73696E287041666696E6554502E782A3230292A657870282D7041666696E6554502E782A32293B0A20207D
0D0A" coeffs="1.0 0.0 0.0 1.0 0.0 0.0" chaos="1.0 0.0" />

    <finalxform weight="0.0" color="0.0" symmetry="0.0" julia="1.0" coeffs="1.0 0.0 0.0 1.0 0.0
0.0" chaos="1.0 1.0" />

    <palette count="256" format="RGB" >

BB2B2FB83034B63539B33A3DB03F42AE4347AB484CA84D51A55255A3575AA05C5F9D6164

9B6669986B6D95707293747790797C8D7E818A838588888A858D8F8292948097997D9C9E

7AA1A278A5A775AAC72AFB170B4B66DB9BA6ABEBF67C3C465C8C962CDCE5FD2D25DD6D7

5ADBD57E0E155E5E652EAEA4FEFEF4CF4F44AF9F949FAFB4BF6F64EF1F250EDEE53E8EA

55E3E658DFE15ADADD5DD5D95FD1D561CCD064C7CC66C3C869BEC46BB9C06EB5BB70B0B7

73ACB375A7AF78A2AA7A9EA67D99A27F949E81909A848B9586869189828D8B7D898E7984

907480936F7C956B789866749A616F9C5D6B9F5867A15363A44F5FA64A5AA94556AB4152

AE3C4EB03849AF3848AC3C48A94048A74448A44748A14B499E4F499B5249985649965A49

935E499061498D65498A6949876C4985704982744A7F774A7C7B4A797F4A76834A74864A

718A4A6E8E4A6B914A68954A65994B639C4B60A04B5DA44B5AA84B57AB4B54AF4B52B34B

4FB64B4CBA4B49BE4B46C14C43C54C41C94C3ECD4C3BD04C38D44C3AD04B3BCC4A3DC949

3EC54840C14842BD4743BA4645B64546B24448AE434AAA424BA7414DA3404E9F40509B3F

52983E53943D55903C568C3B58893A5985395B81385D7D385E7937607636617235636E34

656A33666732686331695F306B5B306D572F6E542E70502D714C2C73482B75452A764129

783D287939277B36277C34267D38277D3B277E3F287F42287F4629804929804D2A81502A

81542B82582B835B2C835F2C84622D84662D85692E866D2E86702F87742F877730887B30

```

```

897F318982318A86318A89328B8D328B90338C94338D97348D9B348E9E358EA2358FA536

90A93690AD3791B03791B43892B73892BB3993BE3994C23A94C53A95C93B96C73B98C43B

9AC03B9CBC3B9EB83BA0B43BA2B13BA4AD3BA6A93BA8A53BAAA23BAC9E3BAE9A3BB0963B

B2923BB38F3BB58B3BB7873BB9833BBB7F3BBD7C3BBF783CC1743CC3703CC56C3CC7693C

C9653CCB613CCD5D3CCF593CD1563CD3523CD54E3CD64A3CD8463CDA433CDC3F3CDE3B3C

E0373CE2333CE4303CE62C3C</palette>

</flame>

```

4.3 Flame (with inlined sourcecode) of the second example

```

<flame name="JWildfire" version="0.42 (31.03.2012)" size="1920 1080" center="-0.42 -0.02"
scale="1070.3991920905366" rotate="-95.0" oversample="1" color_oversample="1" filter="1.2"
quality="100.0" background="0.0 0.0 0.0" brightness="4.0" gamma="4.0" gamma_threshold="0.04"
estimator_radius="9" estimator_minimum="0" estimator_curve="0.4" temporal_samples="1.0"
cam_zoom="1.0" cam_pitch="0.0" cam_yaw="0.0" cam_persp="0.0" cam_zpos="0.0" cam_dof="0.0"
shading_shading="FLAT" >

    <xform weight="0.5" color="0.0" opacity="0.0" symmetry="0.0" custom_wf="1.0"
custom_wf_a="0.0" custom_wf_b="0.0" custom_wf_c="0.0" custom_wf_d="0.0" custom_wf_e="0.0"
custom_wf_f="0.0" custom_wf_g="0.0"
custom_wf_code="696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E626173652E58
466F726D3B0D0A696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E626173652E5859
5A506F696E743B0D0A696D706F7274206F72672E6A77696C64666972652E6372656174652E74696E612E7661726961
74696F6E2E466C616D655472616E73666F726D6174696F6E436F6E746578743B0D0A696D706F727420737461746963
206F72672E6A77696C64666972652E626173652E4D6174684C69622E2A3B0D0A0D0A7075626C696320766F69642069
6E697428466C616D655472616E73666F726D6174696F6E436F6E746578742070436F6E746578742C2058466F726D20
7058466F726D29207B0D0A0D0A7075626C696320766F6964207472616E73666F726D28466C616D655472
616E73666F726D6174696F6E436F6E746578742070436F6E746578742C2058466F726D207058466F726D2C2058595A
506F696E742070416666696E6554502C2058595A506F696E74207056617254502C20646F75626C652070416D6F756E
7429207B0D0A2020646F75626C65206669656C6457696474683D312E303B0A2020646F75626C65206669656C644865
696768743D312E303B0A2020696E7420726F77733D373B0A2020696E7420636F6C733D373B0A2020696E742073697A
653D726F77732A636F6C733B0A2020646F75626C65206C783D302E352D70436F6E746578742E72616E646F6D28293B
0A2020646F75626C65206C793D302E352D70436F6E746578742E72616E646F6D28293B0A2020696E7420782C793B0A
2020696628726F77733C3D3120262620636F6C733C3D3129207B0A20202020783D793D303B0A20207D0A2020656C73
6520696628726F77733E3120262620636F6C733C3D3129207B0A20202020783D303B0A20202020793D70436F6E7465
78742E72616E646F6D28726F77732F322B726F77732532292A323B0A20207D0A2020656C7365207B0A20202020793D
70436F6E746578742E72616E646F6D28726F7773293B0A20202020696628636F6C7325323D3029207B0A20202020
2020783D70436F6E746578742E72616E646F6D28636F6C73292F322A322B7925323B0A202020207D0A20202020656C
7365207B0A2020202020783D70436F6E746578742E72616E646F6D28636F6C732B312D28792532292A32292F322A
322B7925323B0A202020207D0A20207056617254502E782B3D202028782D28726F77732D31292F322E302B
6C78292A6669656C6457696474683B0A20207056617254502E792B3D202028792D28636F6C732D31292F322E302B6C
79292A6669656C644865696768743B0A7D0D0A" coeffs="1.0 0.0 0.0 1.0 0.0 0.0" chaos="1.0 1.0" />

    <xform weight="0.5" color="0.28" symmetry="0.0" sinusoidal="1.0" coeffs="0.34463024803248105
0.140681639442366 -0.140681639442366 0.34463024803248105 -0.2793181351406864
0.32578397212543575" chaos="1.0 1.0" />

    <finalxform weight="0.0" color="0.0" symmetry="0.0" bwraps7="1.0" bwraps7_cellsize="0.9"
bwraps7_space="0.0" bwraps7_gain="2.3000000000000003" bwraps7_inner_twist="1.0"
bwraps7_outer_twist="-0.2" coeffs="1.0 0.0 0.0 1.0 0.0 0.0" chaos="1.0 1.0" />

    <palette count="256" format="RGB" >

5F1B475A19465417464F15454A13454411443F0F443A0D43350B432F09422A0741250541

2003401C03401D04421E06431F08441F0A46200C47210E48220F4922114B23134C24154D

25174F251950261A51271C52281E542820552922562A24582B25592B275A2C295C2D2B5D

2E2D5E2E5F2F3061303262313463313665323866333967343B68343D6A353F6B36416C

37436E37446F3846703948713A4A733A4C743B4E753C4F773E5076424F75464F734A4F72

```

4D4F70514E6F554E6D594E6C5C4D6A604D69644D67674C666B4C646F4C63734C61764B60
7A4B5E7E4B5D824A5B854A5A894A588D49579049569449549848539C48519F4850A3484E
A7474DAB474BAE474AB24648B64647B94645BD4544C14542C54541C8453FCC443ED0443C
D4443BD74339DB4338D94237D84135D64034D43F33D33E31D13D30D03C2FCE3B2ECC3A2C
CB382BC9372AC73628C63527C43426C23324C13223BF3122BD3020BC2F1FBA2E1EB92D1C
B72C1BB52B1AB42A19B22917B02816AF2715AD2513AB2412AA2311A8220FA7210EA5200D
A31F0BA21E0AA01D099E1C079D1B069B1A059919049818029617019416019117048D1806
8A1909861A0C831B0F801C117C1D14791E17761F1A72201C6F211F6B2222682325652427
61252A5E262D5B2730572832542935502A384D2B3B4A2B3D462C40432D43402E453C2F48
39304B35314E3232502F33532B345628355925365B21375E1E38611A3964173A66143B69
103C6C0D3D6F0A3E71063F740740770941790C427C0E447F114581134684164787184989
1B4A8C1D4B8F204D91234E94254F972850992A529C2D539E2F54A13255A43457A63758A9
3959AC3C5AAE3E5CB1415DB4435EB6465FB94961BC4B62BE4E63C15065C45366C65567C9
5868CC5A6ACE5D6BD15F6CD3626DD6646FD96770DB6971DE6C72E16E74E3FD5757F85556
F25356ED5155E84F55E34D54DD4B54D84953D34753CE4552C84352C34151BE3F51B83D50
B33B4FAE394FA9374EA3354E9E334D99314D942F4C8E2D4C892B4B84294B7E274A79254A
7423496F2148691F48641D47</palette>
</flame>