Computergrafik/Visualisierung I - WiSe 2020/2021 Exam replacement – research paper

Oleg, Bissing, Matr.-Nr.: 47309

***Abstract*—Topic of this research is "Hopalong Fractal". Hopalong is one of the fractals, which are presented in the world around us, not only in technic. Hopalong is a mathematical algorithm, which is used to create a complicated picture, that repeats itself. In this article hopalong algorithm is programmed using Java language, provides a visual output of four different hopalongs with randomized colors. Coding approach is detailed presented with sufficient comments and explanations.**

I. Motivation and Introduction

"I find the ideas in the fractals, both as a body of knowledge and as a metaphor, an incredibly important way of looking at the world." Vice President of the USA and Nobel Laureate Al Gore, New York Times, Wednesday, June 21, 2000, discussing some of the "big think" questions that intrigue him.

Computer graphics as a subject to study provides us with a vast amount of challenging tasks. One of them is to implement complicated mathematical formulas using programming languages. Fractals in this context are special - they seem to exist to be programmed: they are endless, perfect algorithm to create some magic on the screen.

Hopalongs, like other fractals, give the programmer an opportunity to be creative: it is a matter of a few parameter changes to give a hopalong completely other form and shape. It can be zoomed in and out, rotated and moved to exactly match all the wishes. There is also a room to play with colors, and this room is as endless as hapalong itself.

II. Theory and related works

A fractal is a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole. Fractals are generally self-similar and independent of scale.

There are many mathematical structures that are fractals; e.g. Sierpinski triangle, Koch snowflake, Peano curve, Mandelbrot set, and Lorenz attractor. Fractals also describe many real-world objects, such as clouds, mountains, turbulence, and coastlines, that do not correspond to simple geometric shapes.

Discovered around 1980, the Mandelbrot fractal set may well be the most familiar image produced by the mathematics of the last century. Its status as a cultural icon needs no support[4].

According to Mandelbrot, who invented the word "fractal": "I coined ‘fractal’ from the Latin adjective ‘fractus’. The corresponding Latin verb ‘frangere’ means "to break": to create irregular fragents. It is therefore sensible - and how appropriate for our needs! - that, in addition to "fragmented" (as in ‘fraction’ or ‘refraction’), ‘fractus’ should also mean "irregular", both meanings being preserved in ‘fragment’"([1], P. 4).

"Fractals prove to have many uses in technical areas of mathematics and science" ([2], P. 21).

"It starts with a formula so simple that no one could possibly have expected so much from it. You program this silly little formula into your trusty personal computer or workstation, and suddenly everything breaks loose. Astronomy described simple rules and simple effects, while history described complicated rules and complicated effects. Fractal geometry has revealed simple rules and complicated effects. The complication one sees is not only most extraordinary but is also spontaneously attractive, and often breathtakingly beautiful" ([2], P. 26).

The name Hopalong is derived from the fact, that such an image is built of points hopping along on an elliptical path starting from one point in the centre[5].

Hopalong fractal is one of so-called "Orbit fractals". These applets work on the principal of tracing out orbits of some function. Given a starting point x_n, the next point is calculated like so:

$$x_{n+1} = f(x_n)$$

This process is then repeated.

While hopalong’s applet is just an extension of gingerbread’s applet it makes sense to mention it as well. It’s feature is that the orbital formula is only applied to one starting point to create the image. This formula is:

$$x_{n+1} = 1 - y_n + |x_n|$$

$$y_{n+1} = x_n$$

This fractal uses $x_0 = -0.1$ and $y_0 = 0$. These values can be changed by setting the parameters xstart and ystart.

And now the hopalong applet. It was originally discovered by Barry Martin of Aston University. It basically just replaces the formulas with these:

$$x_{n+1} = y_n - sg(x_n) * sqrt|b * x_n - c|$$

$$y_{n+1} = a - x_n$$

In this fractal, a, b and c are constants which, by default are initialised to -55, -1 and -42 respectively, although these can be changed[3].

Normally the image doesn’t depend on the first point. These attractors have the butterfly effect. That means if you change the parameters a bit, you’ll get a total new image with very little similarity to the first one[5].

III. Hopalong-Fractal

A. From theory to code

One of the most important steps in the beginning is to think over the way to put theoretical ideas into the lines of the code. While there is a complicated mathematical theory behind it practical approach should be careful.

B. Coding

This program is written in Java language. Most of the classes use Java build-in libraries .awt and .swing.

1) Visual output: Program is made within four classes. The first one called HopalongVisualisation does nothing except of creating the next one - MyFrame. MyFrame class extends JFrame class and creates MyPanel class. It is also needed for some basic settings like closing operator, visibility, size and location of the window. Those are written in the constructor:

```
MyPanel panel;

public MyFrame() {
   panel = new MyPanel();
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.add(panel);
   this.setSize(600, 650);
   this.setLocationRelativeTo(null);
   setVisible(true);
}
```

MyPanel class extends JPanel class and implements ActionListener. It creates within constructor the next class (MyPicture), a panel and buttons. There are five buttons. To create each of them following code is used:

```
button1 = new JButton("Hopalong 1");
button1.addActionListener(this);
button1.setFocusable(false);
```

First is created a button itself, then an ActionListener is added to this new button. Last command deletes a focus from this button.

Buttons 1-4 creates new hopalong. Last of the buttons called "Randomize colors" and it creates the same hopalong with the new color schema which is always randomly generated (see Picture 1). Detailed information about implementation follows.

Next what MyPanel class constructor does is creating a buttons bar (using JPanel class), with nessesary settings (background color, size, layout) and adds buttons to it. For the buttons bar a FlowLayout is used to allow the program put components in a row, sized at their preferred size.

```
buttonsBar = new JPanel();
buttonsBar.setBackground(Color.black);
buttonsBar.setPreferredSize(new Dimension(100, 35));
buttonsBar.setLayout(new FlowLayout());
buttonsBar.add(button1);
...
buttonsBar.add(button5);
```

At last, MyPanel class constructor set the size of the panel itself, its layout, adds panel and buttons bar, sets visibility. This time a BorderLayout is used to arrange the components to fit in the regions needed (in this case CENTER and SOUTH).

```
this.setSize(600, 650);
this.setLayout(new BorderLayout(0, 5));
this.add(picture, BorderLayout.CENTER);
this.add(buttonsBar, BorderLayout.SOUTH);
this.setVisible(true);
```

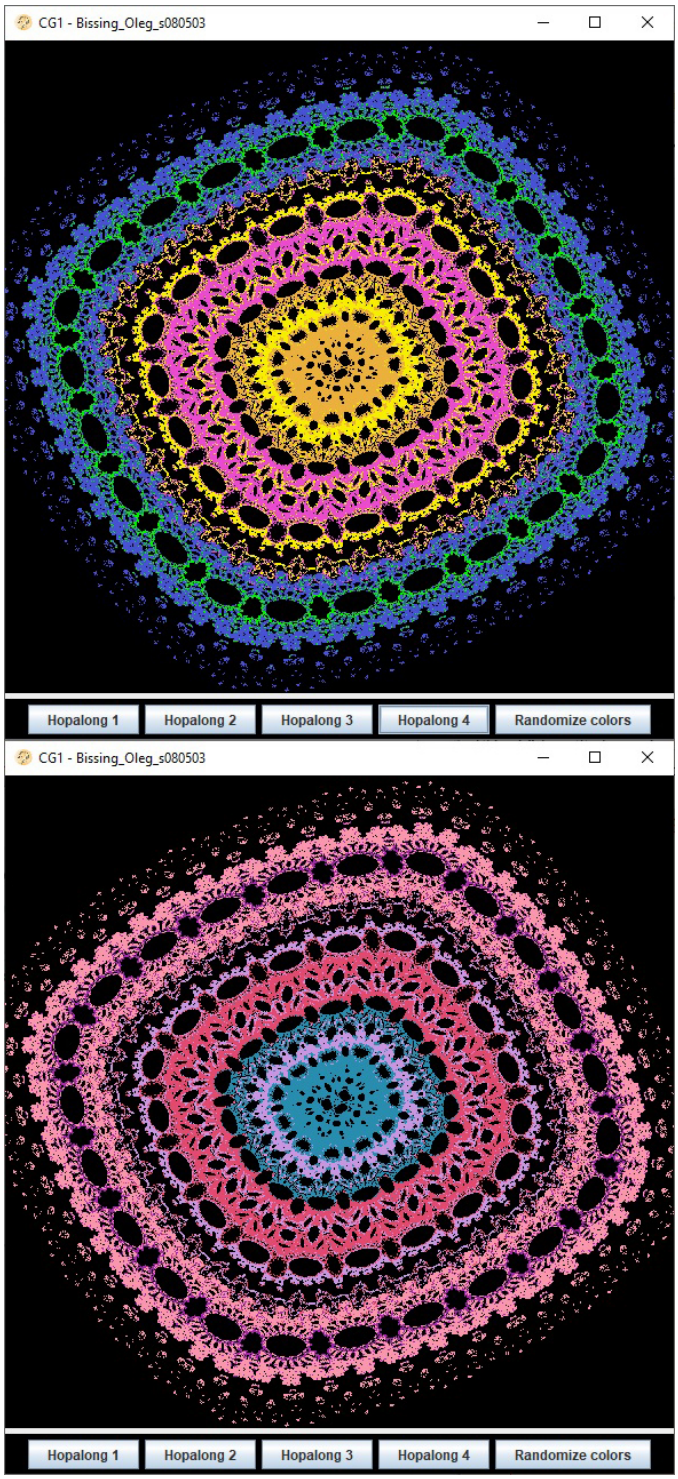

Figure 1. Example of changing color schema by using button "Randomize colors"

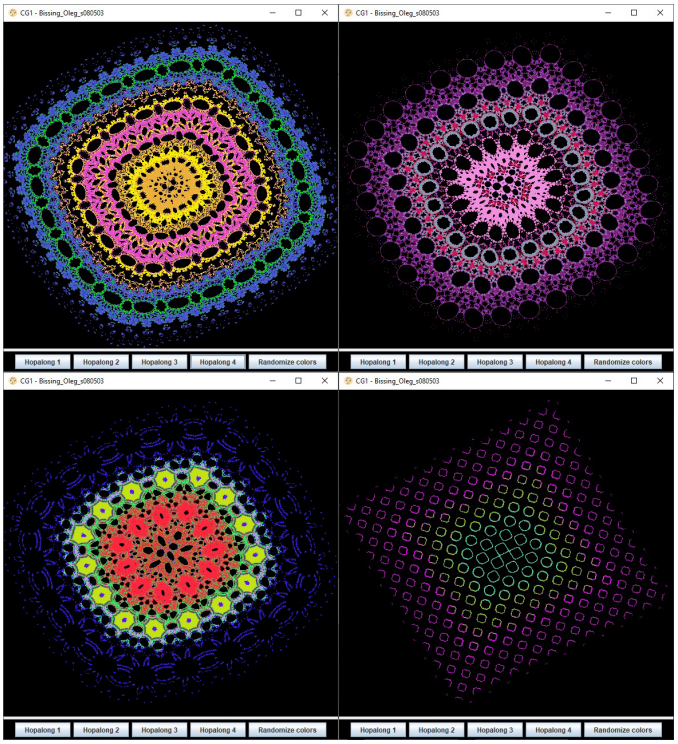

Figure 2. Example of four hopalongs with different settings (buttons "Hopalong 1" - "Hopalong 4")

As already mentioned, MyPanel class implements ActionListener. It is needed to interact with the user using buttons. There are four buttons to paint four hopalongs with different settings using constructor in MyPicture class(see Picture 2). It works wie follow: when user clicks the button an existing JPanel is deleted and new one is created at the same place.

```
@Override
public void actionPerformed(ActionEvent e) {
   if(e.getSource() == button1) {
      this.remove(picture);
      picture = new MyPicture(1.1, -0.5, 1, 500000, 1);
      this.add(picture);
      SwingUtilities.updateComponentTreeUI(this);
   }
   if(e.getSource() == button2) {
      this.remove(picture);
      picture = new MyPicture(1.1, 0.5, 1, 2500000, 2);
      this.add(picture);
      SwingUtilities.updateComponentTreeUI(this);
   }
   ...
   if(e.getSource() == button5) {
      repaint();
   }
}
```

Last class of this program called MyPicture and it extends JPanel. It initializes some important variables: height, width and some values, needed for the calculation of fractals (hopalongs); creates some variables as well. Those created variables are first initialized in the constructor. Important here is that these are default values and used to create the first hopalong which is respectively the default one.

Then the constructor is overridden with parameters, given from the outside of this class. This is a point, where ActionListener connected with buttons in MyPanel class comes to the scene. As mentioned before, when user clicks the button an existing JPanel is deleted and new one is created at the same place. This new one is always created using this overridden method and transferred from MyPanel class values are assigned to those default values, initialized by default constructor.

Values of	d	e	f	dots
Hopalong 1	1.1	-0.5	1	500_000
Hopalong 2	1.1	0.5	1	2_500_000
Hopalong 3	5	1	20	500_000
Hopalong 4	1	0.1	20	1_000_000

Table I
COMPARISON OF THE VALUES OF HOPALONG'S PARAMETERS

```
public MyPicture() {
   this.setPreferredSize(new Dimension(w, h));
   this.d = 1.1;
   this.e = -0.5;
   this.f = 1;
   this.dots = 500000;
   this.scale = 1;
}

public MyPicture(double d, double e, double f, long dots
     , int scale) {
   this.setPreferredSize(new Dimension(w, h));
   this.d = d;
   this.e = e;
   this.f = f;
   this.dots = dots;
   this.scale = scale;
}
```

2) Calculations: According to James Henstridge, a, b and c are constants which, by default are initialised to -55, -1 and -42 respectively, although these can be changed. Java algorithm of hopalong in this program is based on his implementation of the formulas from part 2 with some changes and optimizations. Among them those constants are changed. They are represented by other letters, so that d = a, e = b, c = f. Values of these renamed variables are changed as well. As you can see in the code passage above, by default d = 1.1, e = -0.5, f = 1. As also was mentioned before, these values are changed by the user to get new hopalongs (see Table I). Also changes an amount of dots - parameter which is practically responsible for the size of the hopalong.

One last parameter, given in the constructor is made for scaling. This is needed to fit all different hopalongs with different size into frame of the same size.

```
if (scale == 1) {
   graph.scale(0.04, 0.04);
   graph.rotate(-6, -19000, 28000);
} else if (scale == 2) {
   graph.scale(0.06, 0.06);
   graph.rotate(-6, -11500, 18000);
} else if (scale == 3) {
   ...
}
```

As you can see there is not only scaling but rotating and positioning as well to paint the picture exactly in the middle of the frame. This is a part of the method paint, which is perhaps the most important one - it practically creates these hopalongs. To be able to use mentioned above scaling and rotations Graphics class must be casted into a Graphics2D class.

```
public void paint(Graphics g) {
   super.paint(g);
   Graphics2D graph = (Graphics2D) g;
   ...
}
```

The picture, in our case - hopalong, is not painted as a whole, but pixel by pixel. For this purpose is used a drawLine method inside a for-loop.

Here you can see a practical need of the parameter dots: by changing this parameter an amount of dots painted would be changed, which directly impacts to the size of the hopalong. But it is not that simple: while new dots are not necessary painted on the perimeter of the hopalong, their amount can also affect the shape or detailing of the hopalong, but not its size.

```
for (i = 0; i < dots; i++) {
   xnew = newX(x, y);
   ynew = newY(x);
   x = xnew;
   y = ynew;
   graph.drawLine(transX(x), transY(y), transX(x),
        transY(y));
   if (i % 100000 == 0) {
      graph.setColor(randomColor());
   }
}
```

Within if-statement a value of 100000 to be found. It is responsible for change of the painting color to a new one every 100000 dots. This parameter can also be changed to change a color scheme of the hopalong.

Paint method uses four other methods newX, newY, transX, transY. This is where all the math is made: newX and newY methods represent hopalong applet formulas from the part 2, parameter sgn is responsible for the sign of x.

```
public double newX(double x, double y) {
   int sgn = (x > 0) ? 1 : (x < 0) ? -1 : 0;
   return (y - sgn * Math.sqrt(Math.abs((e * x - f))));
}

public double newY(double x) {
   return d - x;
}
```

TransX and transY are part of the implementation of the gingerbread fractal, extension of which is the hopalong fractal. These classes are used to make some calculations of the fractal based on formulas and using constant (r, l, t, b) and given (x, y, w, h) variables.

```
public int transX(double x)     {
   return (int)((x - l) / (r - l) * w);
}

public int transY(double y)     {
   return (int)((y - t) / (b - t) * h);
}
```

Constant variables, when changed, affect different characteristics of the fractal: compression, tension, shape in general, etc. Given variables width (w) and height (h) are responsible for correlation between the size of fractal and the size of the output window.

Last method to mention is randomColor which uses an build-in Java classes Random and Color. As mentioned before, it is needed to change a color schema of hopalong when the button "Randomize" is pressed.

```
public Color randomColor() {
   Random rand = new Random();
   int r = rand.nextInt(255);
   int g = rand.nextInt(255);
   int b = rand.nextInt(255);
   return new Color(r, g, b);
}
```

This method creates a new color using random integer numbers in range between 0 and 255 as RGB parameters (each of them is represented by 256 shades) and returns it to the other method or class.

IV. Summary and Outlook

Computer graphics is a challenging subject which is tight bound to the mathematics. Without use of mathematics no computer graphics would ever work. In mathematical theory exist a lot of algorithms and formulas, which a not necessary, but extremely interesting to implement in computer graphics. One of them are fractals.

The term "fractal" is invented by Benoit Mandelbrot. The geometric characterization of the fractals is self-similarity: the shape is made of smaller copies of itself. The copies are similar to the whole: same shape but different size. There lots of different fractals, hopalong is only one of them. Like all the fractals it has its own mathematical formula which is given in part 2.

Hopalong's formula is implemented using Java programming language. Visual output is based on Java build-in libraries .awt and .swing as well as ActionListener class implementation. Screenshots of a launched program are added to demonstrate some examples of the visual output.

Powerful tools of the object-oriented programming within Java are used. Among them are overridden methods, different build-in classes (Random, Graphics2D, etc.).

A dozen small code passages with comments and explanations are added to clarify every step of this research. They cover not only formula implementation, but all the important details of the program to show the reader step-by-step the code part of the research.

While hopalongs noticeably differ within tiny parameter changes a table is added along screenshots to show it. It is enough to change a sign of a value or multiply amount of dots to get a completely new pattern of the hopalong. Four noticeably different hopalongs because of a little changed parameters are presented to prove that.

Hopalongs are one of the perfect tools to show beauty of mathematics using computer graphics. They give a programmer a lot to work with: not only a possibility to play endless with parameters to change its shape, form and color, but also an opportunity to bring more life in play, perhaps by using animations, zoom/unzoom it, ect.

References

[1] Mandelbrot B.B.: *"The Fractal Geometry of Nature"*, Times Books, 1982

[2] Frame M.L., Mandelbrot B.B.: *"Fractals, Graphics, and Mathematics Education"*, Mathematical Association of America, 2002

[3] Website Henstridge J.: http://www.jamesh.id.au/.

[4] Website of the Yale University, Department of Mathematics: https://users.math.yale.edu/public_html/People/frame/Fractals/

[5] Website Fraktalwelt: http://www.fraktalwelt.de/myhome/simpiter2.htm