Groux Marcel Jean Jacques 227630
Platzer Casimir Benjamin 228352

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Report of improvements

Contents

Iterative Calculation of the Flame:

Idea:

Since computing a fractal in a good quality takes a lot of time to compute, one idea would be not to calculate it all at the same time. Without this feature the program would "block" every time you are computing a large fractal. This would block the event-dispatch thread which normally is used for painting swing components.

The differences you can see with this feature is, that when the fractal is computed, only a few points are drawn to the screen. Every few milliseconds there are more points added to the screen, so the basic shape of the fractal is in some cases given from the beginning. This allows editing the fractal on a much easier and of course faster way.

Please feel free to experiment with it by fully expanding the window.

Implementation:

We added compute() with different arguments to the class Flame, since it must continue computing from an existing FlameAccumulator.Builder. The method takes a FlameAccumulator.Builder and the amount of points to compute. Then it computes twenty points without adding them to the builder. After that it starts computing as much points as specified and adds them to the builder. The builder is then returned. In the class FlameBuilderPreviewComponent the method paintComponent() was modified. Basically that method is always running, as long the flame is still being computed. Which is why we have to keep track of what has been modified using if-clauses. For example whenever the builder is still equal to null, the component's dimensions changed or the rectangle changed, the rectangle and the temporary rectangle with the same aspectRatio as the component have to be modified then it has to recompute the fractal again.

There is another if-clause checking whether the Flame.Builder i.e. the fractal itself has changed. If it's not the case the image where the fractal is drawn to is drawn to the screen.

If it is the case, we first check whether or not there is already a running calculation. If there is one we have to stop it. Otherwise many Timers will be started and the program will use up all resources fast. If there isn't already a running calculation, a resettable timer is created which calculates a specified amount of points until it reaches the maximum amount, given by the width, height and density.

Rectangle Resize:

Idea:

After having modified an existing flame fractal, it is strongly possible, that it isn't in the original rectangle anymore. So instead of trying to fit it in the rectangle, it would be much easier to be able to resize and move the rectangle. So the idea is to implement a feature allowing the user to move and resize the rectangle. This would definitely be possible by using buttons, but we thought it might be more interesting to use the mouse when hovering over the component itself.

When the user wants to move the rectangle he simply has to left click on the preview component of the fractal. The cursor will then change to the default move cursor, indicating what mode is activated. By pressing both mouse buttons, you can move the rectangle but not resize it. For resizing the rectangle you have to right click the component. The cursor will then change to the default resizing cursor for north-south resizing. If you don't move the mouse, nothing will happen. The further up you will drag the mouse the faster it will zoom in, the further down the faster it will zoom out. For easier usage with a mouse it is also possible to zoom using the mouse wheel.

Implementation:

The class FlameBuilderPreviewComponent contains an Interface and a MouseAdapter. We made an own class, which is a MouseAdapter. As an argument it takes the model, which has to be of the type of our interface: ResetableRectangle. The interface has a method called setRectangle(Rectangle rectangle). This allows us to make the model modify its rectangle. The rectangle is observed by the components using it for display: FlameBuilderPreviewComponent and AffineTransformationsComponent. These two implement our interface, so that we must override the method setRectangle(). Our MouseAdapter is then added to the right component as MouseListener, MouseMotionListener and MouseWheelListener. When someone then presses one of the button, the cursor of the mouse is changed, depending on which one has been pressed, using a method called setCursors(). It checks what buttons are pressed and sets the cursor for the mode.

When moving the rectangle, it calculates the difference of the last coordinates to the current ones and modifies the rectangle. When using the mousewheel for scaling, it scales the rectangle by a constant factor, depending on the rotation. When using the mousebutton it's different. It does calculate the difference from the current coordinates and the first ones. So when going up with the mouse you zoom in fast and when you go down with your mouse you zoom out slower. For constant scrolling when getting no user input there's a timer, which resets as soon another mouse is moved for resizing.