Il Frattalatore

CAPR group:

Claudio, Eutizi (ID: 10812073)
Andrés, Bertazzi (ID: 10488849)
Pierluigi, Tartabini(ID: 10845797)
Riccardo, Martinelli (ID: 10456202)

Juce Homework assignment #4
Computer Music: Languages & Systems
Politecnico di Milano

May 18, 2022

Contents

1 Introduction

The **Frattalatore** is a Synthesizer based on fractal mathematical systems, implemented using **Juce** and external projects with **C++**.
The synthesizer applies a sum of 5 oscillators, related each other by a fractal set of parameters. All the oscillators take the same MIDI note input given by a piano keyboard but they are set with different parameters.

2 What can the user do?

The user has to open 2 projects: the external project and the Synthesizer.
In the external project the user has to choose which fractal wants to play with the Synthesizer.
A window appears with the image of the fractal and the user can choose a point from that. In the same moment the oscillators on the GUI Synthesizer will be set with different values taken from the chosen fractal point; by default they can control and modify the parameters of the oscillators using different knobs and features.
The user can play notes with a MIDI keyboard and the final result will be a sum of the fundamental note and the 5 sound's oscillators modulated previously.

3 GUIs

The GUIs of the **Frattalatore** have been made from different projects and communicate each other through **Osc Messages**.
In particular the GUI in the **C++** part is based on the `SFML` library that allows to manage events and complex mathematical iterations.

3.1 Menu Window

The first window is a Menu where the user can choose the type of Fractal. The choice must be done with the keyboard's arrows and then selected with Enter button.

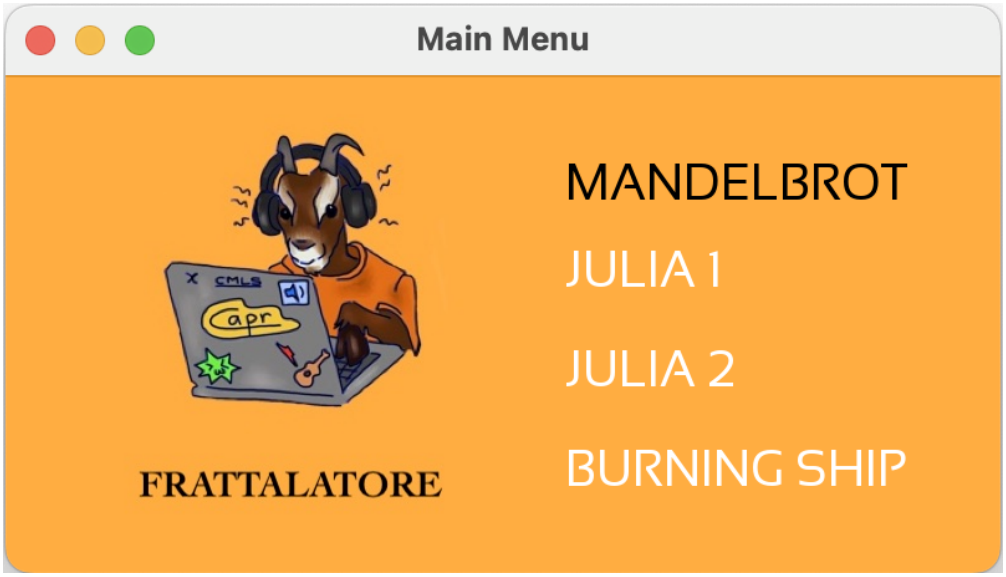
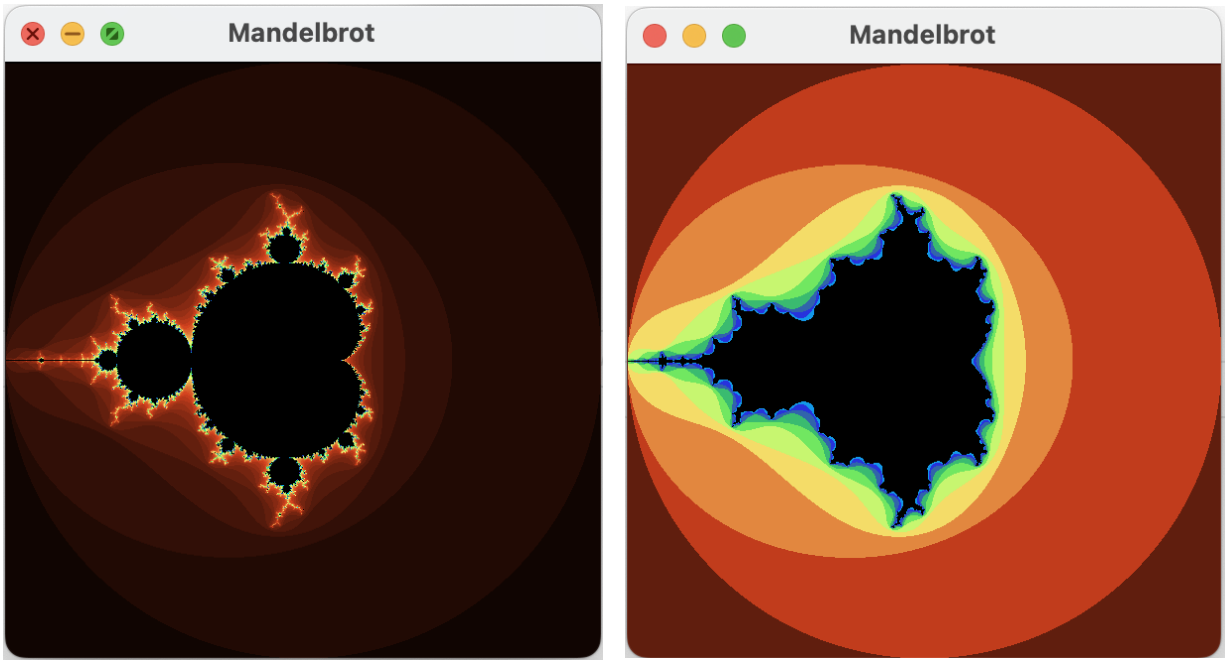


Figure 1: Main Menu Window

3.2 Fractal Images

After the previous choice, the user will see the second window where is drawn the fractal image.
The user can do many actions using the mouse or the computer keyboard:

- **Enter**
  The main important event is select a point on the figure with the enter button in order to generate the sequence of coordinates associated to the parameters of the oscillators.
- **Wheel Scroll**
  The user can increase/decrease the number of iterations so that he can see more details of the fractal by scrolling the wheel of the mouse.



(a) Fractal Window

(b) Fractal Window with less iterations

Figure 2: Fractal Images

3.3 Synth Window

The window has a background image which is modified with an **Osc Messages** inside the `PlugInEditor.cpp`. Each image corresponds to the Fractal selected inside the Fractal window. The design of the knobs of the Synthetizer is made with `JKnobMan` which creates an image of different knob's frames. Finally the image is imported and each frame is read inside the `CustomLookAndFeel` files. The background of the GUI changes according to the chosen fractal image.
The structure of the Synth is made of:

- **Oscillators**
  The Frattalore has five different oscillators.
  Each one has a selector of the type of wave (**Sine**, **Saw**, **Square**) and 4 different knobs that controls **Gain**, **Pitch**, **FM Freq** and **FM Depth**. The last two regards on the FM modulation and select the frequency and the amplitude of it.

- **Filters**
  The Filter section allows to select three different types of filters:

  1. **HighPassFilter** (HPF)
  2. **BandPassFilter** (BPF)
  3. **LowPassFilter** (LPF)

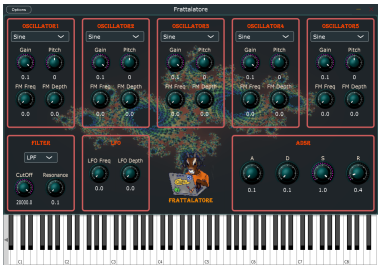
  It has also two parameters that allows to select the **CutOff frequency** and **Resonance** of the filter.

- **LFO**
  This section allows to modulate the output signal by selecting the **LFO Frequency** and **LFO Depth** of the modulation.

- **ADSR**
  This is a section which allows the user to modify the envelope of the signal by using 4 parameters: **Attack** (A), **Decay** (D), **Sustain** (S), **Release** (R).

- **Keyboard**
  It is a **MIDI Keyboard** which can be used both with the laptop keyboard and directly on the GUI with the mouse. The Plugin also can receive MIDI inputs by an external MIDI keyboard.

(a) Mandelbrot



(b) Julia 1

(c) Julia 2

(d) Burning ship

Figure 3: Synth GUI

4 Fractals

A geometrical fractal is *"a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole"*, as described by Professor Mandelbrot in his publications.
Iterating a specific recurrence relation at each point in the complex space, an image is drawn more and more detailed increasing the number of iterations. To clarify, this concept can be seen also in nature where big clouds are composed with little "infinite" clouds or the mountain shapes are made with little "infinite" shapes,...
Each point of the space (composed by a Real part and an Imaginary part) applies an iterative process using the fractal formula and creates a sequence of points. As long as the sequence observes a particular belonging condition, is possible to continue to iterate the sequence until the infinite (in this particular case until it reaches the maximum number of iterations). If an element of the sequence is out from the belonging condition, the iteration ends (the sequence diverges). So each point belongs to the fractal with a particular weight, measured with the number of each point iterations.
Visually, the interesting thing is that is possible to zoom the fractal "until the infinite" in particular points in order to visualize the same or a similar starting image (as seen in the following images).

4.1 Mandelbrot set

Mathematically the Mandelbrot fractal is created by the sequences generated for each point of the complex domain. For each complex point (in this case equal to p), the first element of the sequence is the point itself because z_0 is equal to 0. Than we iterate the result into z_n and the p remains the initial complex point.

$$f_p(z) = \begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + p \end{cases}$$

4.2 Julia set

We can notice that the formula of the Julia sets is the same of Mandelbrot set. The difference depends on how we set the initial conditions, but the iteration process is the same.
For each complex point p, the first element of the sequence is generated by putting the point coordinates to z_n and sum the preset constant c. Then we can iterate the result into z_n and c remains the initial constant.We chose 2 different Julia fractals setting 2 different constants.

4.2.1 Julia set 1

Setting the constant $c = -0.7269 + 0.1889i$, we have the following fractal equation:

$$f_p(z) = z_{n+1} = z_n^2 + c$$

4.2.2 Julia set 2

Setting the constant $c = 0.285 + 0.01i$, we have the following fractal equation:

$$f_p(z) = z_{n+1} = z_n^2 + c$$

4.3 Burning ship fractal

Mathematically the Burning ship fractal applies the same process of the Mandelbrot fractal. c is each complex point of the complex space.

$$f_c(z) = \begin{cases} z_0 = 0 \\ z_{n+1} = (|\Re(z_n)| + i\,|\Im(z_n)|)^2 + c \end{cases}$$

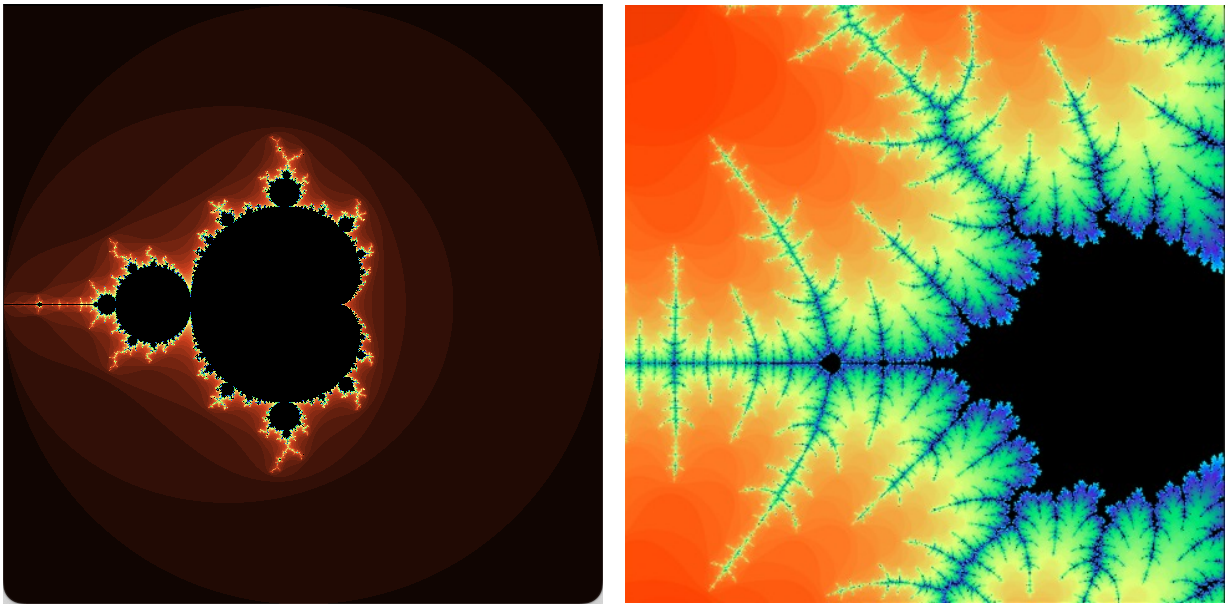
4.4 Fractal design and code

To implement these formula in the code it was created two classes called `Fractal` and `threeValuesArray` with their related main method types:

- `int ricorsioniMandelbrot(double cr, double ci, int max_iterations);`
- `std::vector <threeValues> calcoloMandelbrot(double cr, double ci, int iterazioni_user, int iter);`

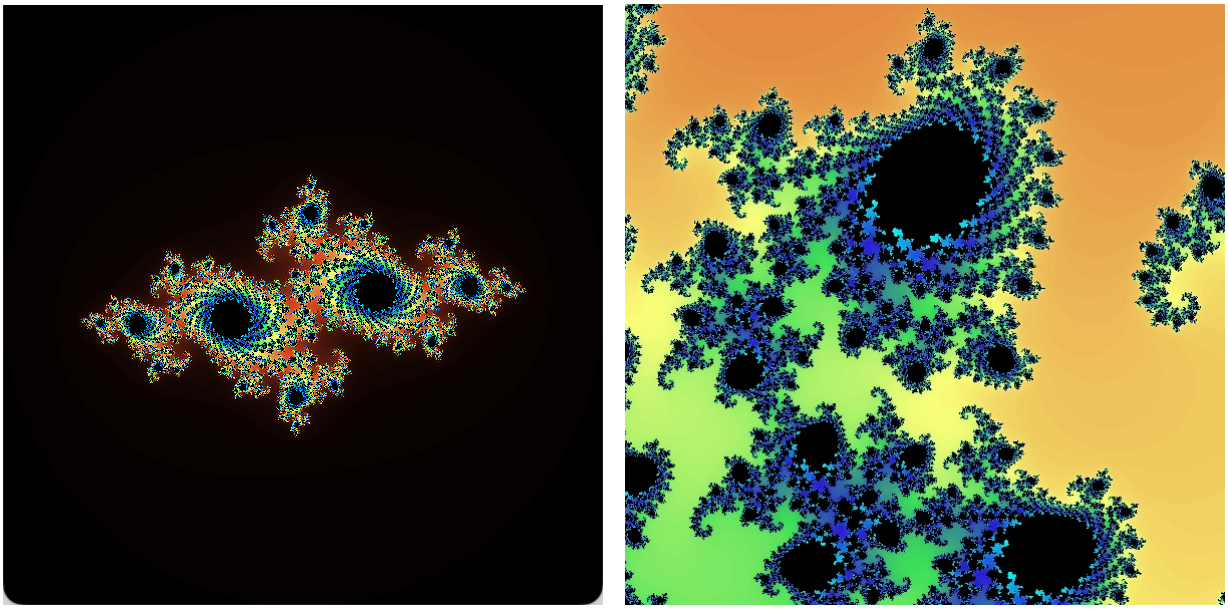
The first one is used inside two `for` cycles in which are scanned all the pixel (x,y) of the points belonging to the window. The returning `int` of the method is the number of iterations of the current point.
To draw the images it's set a high number of iterations that each point of the domain could be computed to create its relative sequence. Theoretically the number of iterations is infinite. Thus each point has a grade of divergence of the sequence: high grade correspond to a great belonging to the fractal, low grade correspond to a less belonging.
Each value is used to set the RGB of the each window pixel and to draw the fractal image. Finally it is modified the color set with a particular interpolation that emphasizes the boundaries of the figure, using Color paint_fractal(int n, int max_iter);.
It's useful to remember that the 4 fractal images are represented in a complex plane and each point has 2 coordinates (Real one and Imaginary one). All fractals have a unique condition: the modulus of the complex elements of the sequence has to be less than 2 units: $\sqrt{(\Re(z_n))^2 + (\Im(z_n))^2} < 2$. For this reason the complex domain and the fractal images are inside the square centered in the origin of complex space, with height and weight equal to 4.
Drawing the Fractal images with this method was be possible using `SFML` library, that optimizes the graphic.

The second method is important to send the values to **Juce** with the **Osc protocol** (as will see in the next paragraph). It returns a vector of 5 position; for each position there are 3 elements: x coordinate, y coordinate and number of iteration.
The first position is related to the point that the user chose from the window with the Enter keyboard button; the following are the points of the first x,y coordinates of the fractal sequence.
Each couple of coordinates has a proper sequence more or less convergent to the fractal depending on the first choosing point.



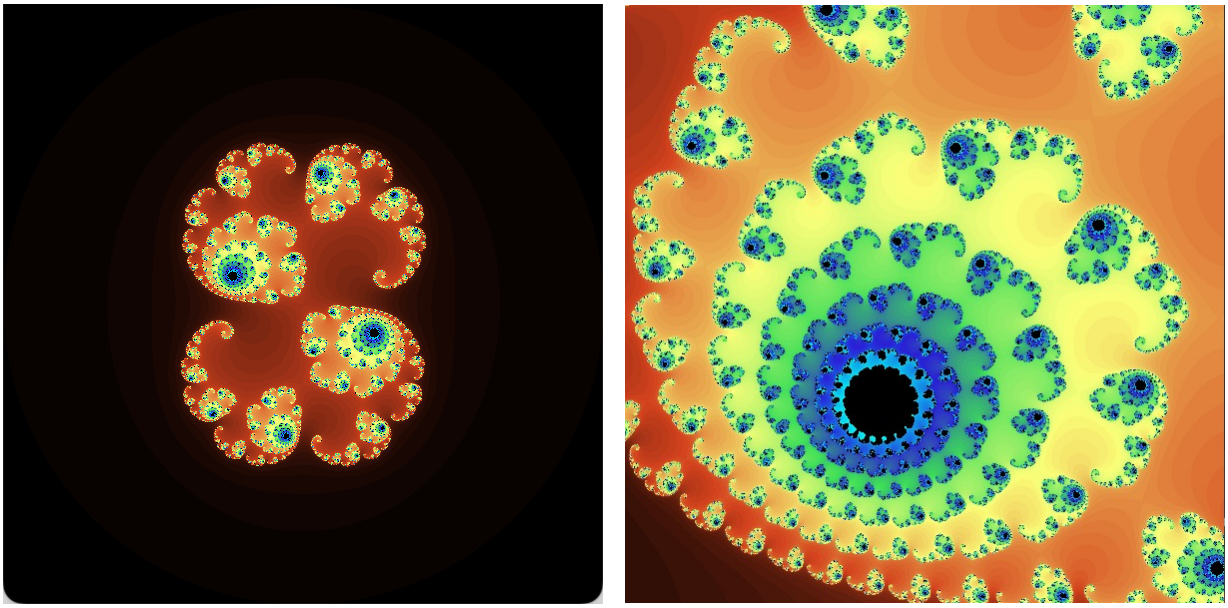
(a) Mandelbrot fractal (b) Zoomed detail

Figure 4: MANDELBROT



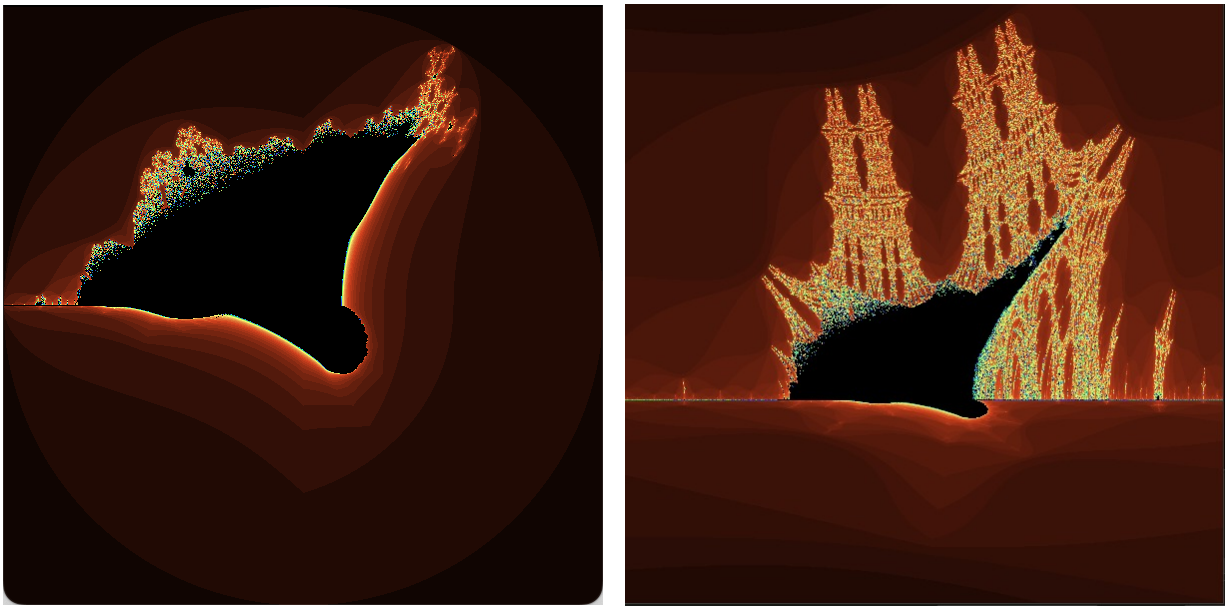
(a) Julia 1 fractal (b) Zoomed detail

Figure 5: JULIA 1



(a) Julia 2 fractal (b) Zoomed detail

Figure 6: JULIA 2



(a) Burning ship fractal (b) Zoomed detail

Figure 7: BURNING SHIP

5 OSC Messages

To allow the communication between the **C++** project and the **Juce** Synthesizer it was implemented the OSC protocol (OSC is the acronyms of *Open Sound Control*), that supports sound synthesis through multimedia devices.
The communication is like Server and Client prototype and goes through the follow local address and port of the device.

```
#define ADDRESS "127.0.0.1"
#define PORT 7000
```

The Server is the **C++** project: it initialises an output buffer size (`#define OUTPUT_BUFFER_SIZE 1024`) that will contain the values that the **Juce** Synthesizer (the Client) uses to set the initial conditions of the oscillators.
In particular the sent values are:

1. x and y coordinates of the chosen points of the fractal
2. number of iterations of each sequence generated from each point

Juce receives a message for each values and sets the initial conditions of the oscillators' parameters.

6 Synth structure

The user can choose a point of the Fractal image that iterate 5 times. Each iteration corresponds to an oscillator. All the oscillators play simultaneously.
The user can decide which point of the image wants to iterate and it's created an array of 5 couple of coordinates corresponding to the first complex elements (Real part, Imaginary part) of the fractal sequence.
Each point of this array is associated to a single oscillator and it can be itself the first element of a new fractal sequence.
As we said in the previous paragraphs the number of iterations of a sequence of a single point in the complex space corresponds to the "grade" of divergence of this fractal point. We associate the grade of divergence of each sequence (each oscillator) to the level [0,1] of the **FM depth** that modulate the input sound, that the user can play using a keyboard. If the grade of divergence is high (theoretically infinite), the sequence doesn't diverge and there's not modulation: FM depth is equal to 0. Instead, a smaller grade corresponds to an higher FM depth, until a maximum of 1 if the point diverges immediately.
Moreover we can change the **FM frequency** of each oscillator. We use the modulus of the complex number ($\sqrt{\Re(p)^2 + \Im(p)^2}$) to modulate the FM frequency.
If the point chosen by the user belongs to a hypothetically infinite fractal sequence, the output sounds good. Instead the sequence diverges out of the fractal condition, so the sound is distorted.

6.1 Oscillators

6.2 Filters

6.3 ADSR

7 Future Developments

7.1 Fractal Image

- **Zoom**
  Graphically the user could zoom in and zoom out the fractal and find the infinite loop in particular points of the complex space.

- **Move**
  The user could move on the image using the keyboard arrow: left, right, up and down.

Many thanks to ***Marco Di Mambro*** *for the CAPR logo design and drawing.*