# Practical Training Project

Vienna University of Technology

# Rational Piano

Fabian Ehrentraud

student ID 0725639, study code 535

e0725639@student.tuwien.ac.at

Vienna, July 30, 2010

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Overview

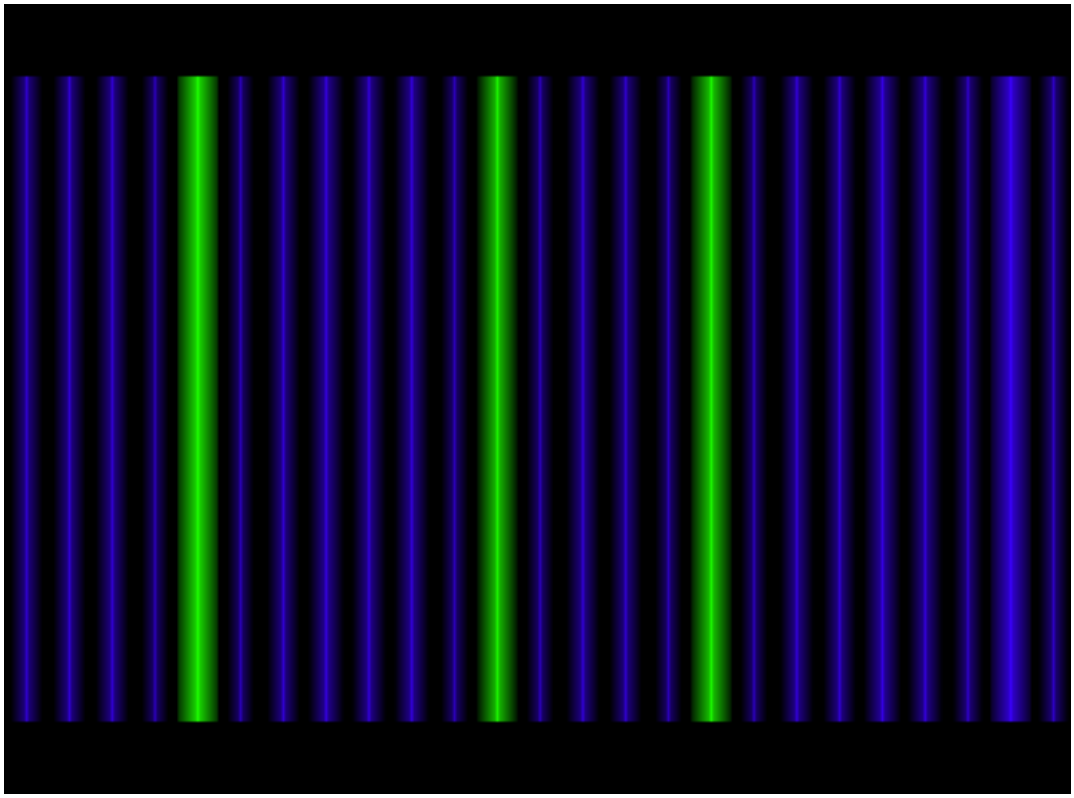

Figure 1.1: The Interface of Rational Piano in action

*Rational Piano* is a piano interface for TUIO [26] enabled multitouch tables. As shown in Fig.1.1 the interface is very basic. The application displays playable notes (or keys)[1] as vertical lines. Pressing and releasing those lines sends note messages

[1] in the 12TET scale - other scales are planned, see chapter 6

via MIDI [2] or OSC[2] [3] to external[3] synthesizer applications.

For all displayed keys respectively, its *consonance* [1] is displayed by a varying line width. Consonance expresses how well each key would fit to the played keys and is calculated in real time with the framerate. For an explanation of the thoughts and algorithms behind the consonance calculation, see chapter 2. If you just want to start playing around instantly, you can skip to chapter 3.

The application was named "Rational Piano" because on the one hand, the consonance calculation is based on Rational Numbers and on the other hand, it provides a rational way for playing with melodies and harmonies.

## 1.1 Target Audience

This application is targeted at **Musical Novices**.

Necessary thinking ballast should be reduced (which tone or chord fits now) and the coupling of the auditive and visual senses should get invigorated. It should be a helping tool to get a feeling for harmonious laws without the need to think about or even to understand the theories behind them. This is also what traditional learning methods aim at but with a higher entry step.

## 1.2 Pictures

Most of the pictures in this document were drawn with Wolfram Mathematica®[13]. The source is included in the file *doc/images/images.nb*. The Interaction diagram in Fig.5.1 was drawn with Dia [14], its source is available in file *doc/images/control_interaction.dia*.

## 1.3 License

Source Code, images and other original work including this documentation is licensed under the **Open Software License version 3.0** [30]. The full license text also is available in file *"LICENSE.txt"*.

Libraries in the subfolder "lib" are licensed under the original licenses - for more information on these, see chapter 5.2.

[2] with namespace SynOSCopy [27]

[3] external to the Rational Piano application, it can still run on the same computer

# Chapter 2

# Consonance and Dissonance

## 2.1 The Concept

So what is consonance? There exist many different definitions [11]. Consonance expresses how well some sounds fit together. That is dependent of the single sounds' **base frequencies**, their **spectra** and also what the listener is **accustomed** to[1]. In this application only base frequencies are considered[2] as it is a good approximation for most harmonic spectra.

### 2.1.1 Ratio of two Frequencies

You are probably familiar with the term *Octave*. Imagine (or try out) playing a tone on an instrument like the piano and at the same time play the octave[3] above. Fig.2.1 shows playing an A with 440Hz plus its octave above with 880Hz. Playing that *interval* sounds quite good.

Now, in contrast, play the same first tone and as the second tone use one Semitone **below** the octave - that interval is called a *Major Seventh*. Fig.2.2 shows the keys of that interval where again the lower key is an A with 440Hz and the upper key is a G sharp with (approximately) 825Hz[4]. It does not sound that good anymore[5], more **tensionful**, as if the two tones would **beat** each other.

[1] and probably some other things too

[2] If you are interested into spectrum considerations, look up the works of William Sethares [4, 5]

[3] In our western music system we most commonly use the *Equal Temperament* [6] which divides one octave into 12 *Semitones*, it is also called *12TET*. So here the octave is 12 Semitones above any tone.

[4] Which is a major seventh in Just Intonation [7], in 12TET it is actually $440 \cdot 2^{\frac{11}{12}} = 830.609Hz$ which has not got a ratio as it is an irrational number, but we will get to this later

[5] mileage may vary

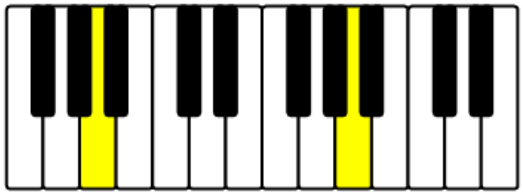

Figure 2.1: An *Octave* played on a keyboard, base tone is an A with 440Hz

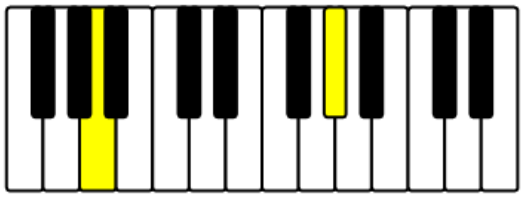

Figure 2.2: The interval *Major Seventh* played on a keyboard, base tone is an A with 440Hz

So why is that? The (upper) octave of a tone has double its base frequency. So if the lower tone has the (base) frequency $f_{lower}$ and the upper tone has the (base) frequency $f_{higher}$, the simple mathematical relationship is $f_{higher} = 2 \cdot f_{lower}$. The *ratio* of the frequencies is $\frac{f_{higher}}{f_{lower}} = \frac{880Hz}{440Hz} = \frac{2}{1}$. The exact frequencies are of no concern[6] as they shorten out in the fraction, only the shortened ratio of the frequencies is important. Looking at Fig.2.3 one can see that the two base frequencies "meet again" very soon, this fact correnspponds to the low ratio of the frequencies.

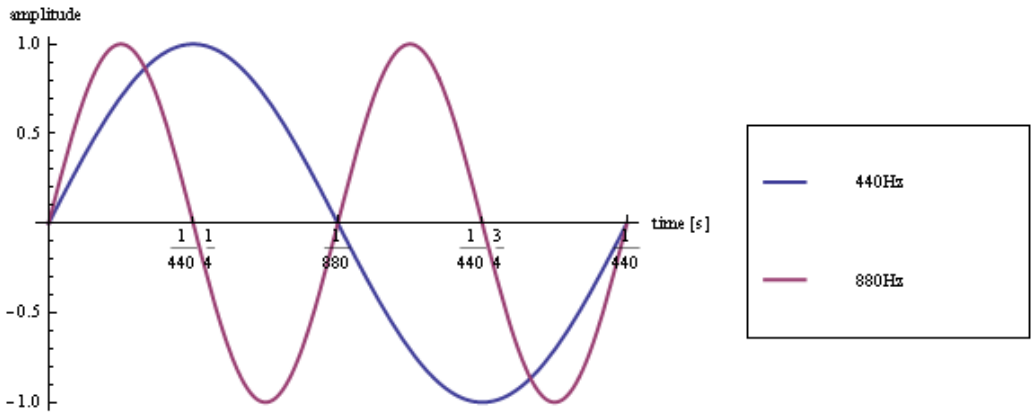


Figure 2.3: Two Sine waves representing the base frequencies $440Hz$ and $880Hz$

Now we look at the second example with the Major Seventh. Here the mathe-

[6] as long as we stay in the human hearing range

matical relation is $\frac{f_{higher}}{f_{lower}} = \frac{825Hz}{440Hz} = \frac{15}{8}$. In Fig.2.4 one can see that it takes some more cycles for the two base frequencies to "meet again", compared to the octave, which correnspponds to the higher ratio of the frequencies.

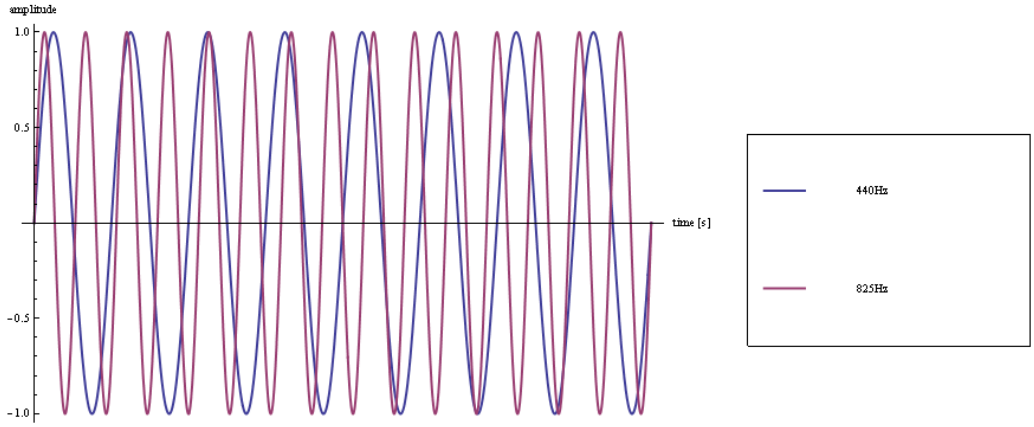


Figure 2.4: Two Sine waves representing the base frequencies $440Hz$ and $\frac{15}{8} \cdot 440Hz = 825Hz$

Comparing those two fractions and more of other intervals shown in Table 2.1, one can see that fractions composed of a **low numerator and denominator** expresses a **more consonant interval** than a fraction that is composed of a high numerator and/or denominator[7]. So the lower the result of the **multiplication of the numerator with the denominator**, the higher the consonance, or in other terms, the lower the dissonance[8].

To quantify that mathematically, we can say that $Dissonance(\frac{n}{d}) := n \cdot d$ of the **shortened** fraction $\frac{n}{d}$, so the *Greatest Common Divisor* of the numerator and denominator is 1: $GCD(n, d) = 1$.

This function is *commutative* as the multiplication is commutative: $Dissonance(\frac{n}{d}) = n \cdot d = d \cdot n = Dissonance(\frac{d}{n})$.

As all interesting numerator and denominator values are positive and non-zero, the result of the Dissonance fuction will always be at least one. As $Consonance = \frac{1}{Dissonance}$, the result of the Consonance function will always lie between 0 and 1.

### 2.1.2 Inexact Fractions

While working with rational numbers now seems easy, we have got a **problem** when working with ratios of the Equal Temperament. As you can see in Table 2.1, its

[7] This observation was already made by many, among them Pythagoras [11]

[8] Dissonance in this context is used as anti-Consonance, or mathematically $Consonance = \frac{1}{Dissonance}$

| Number of Semitones | Interval Name | Just Intonation | Equal Temperament |
|---|---|---|---|
| 0 | unison | $\frac{1}{1}$ | $2^{\frac{0}{12}}$ |
| 1 | semitone | $\frac{16}{15}$ | $2^{\frac{1}{12}}$ |
| 2 | whole tone | $\frac{9}{8}$ | $2^{\frac{2}{12}}$ |
| 3 | minor third | $\frac{6}{5}$ | $2^{\frac{3}{12}}$ |
| 4 | major third | $\frac{5}{4}$ | $2^{\frac{4}{12}}$ |
| 5 | perfect fourth | $\frac{4}{3}$ | $2^{\frac{5}{12}}$ |
| 6 | tritone | $\frac{7}{5}$ | $2^{\frac{6}{12}}$ |
| 7 | perfect fifth | $\frac{3}{2}$ | $2^{\frac{7}{12}}$ |
| 8 | minor sixth | $\frac{8}{5}$ | $2^{\frac{8}{12}}$ |
| 9 | major sixth | $\frac{5}{3}$ | $2^{\frac{9}{12}}$ |
| 10 | minor seventh | $\frac{9}{5}$ | $2^{\frac{10}{12}}$ |
| 11 | major seventh | $\frac{15}{8}$ | $2^{\frac{11}{12}}$ |
| 12 | octave | $\frac{2}{1}$ | $2^{\frac{12}{12}}$ |

Table 2.1: Intervals in an octave and their ratios according to Just Intonation and non-rational multiples according to Equal Temperament

tones' ratios consist of roots of the number two. All of those (exept to octaves) are *Irrational Numbers* that have got **no fractional representation**. That means if we have given any two frequencies that have no rational ratio, in the mathematic sense, we can not calculate the consonance.

Even with perfect ratios, there is a **problem**: if there are two tones playing with a ratio that has got a high numerator and denominator, and therefore a high dissonance, it still could be that the interval sounds consonant. That is the case if there is a nearby fraction with a low numerator and denominator. As you can see in Fig.2.5, the two ratios are just (approximately) 1 *Cent*[9] apart, which is the lower bound of hearing difference of the human's ear, meaning the **two ratios cannot be distinguished** even though their dissonances differ about many orders of magnitude.

| The human ear tends to hear more consonant ratios, even if they are some cent apart. |
|---|

But the farther apart, the less good it sounds.

To both fill the "irrational holes" and take proximate and more consonant frac-

[9] 1 Cent equals the ratio $2^{\frac{1}{1200}}$ [9].

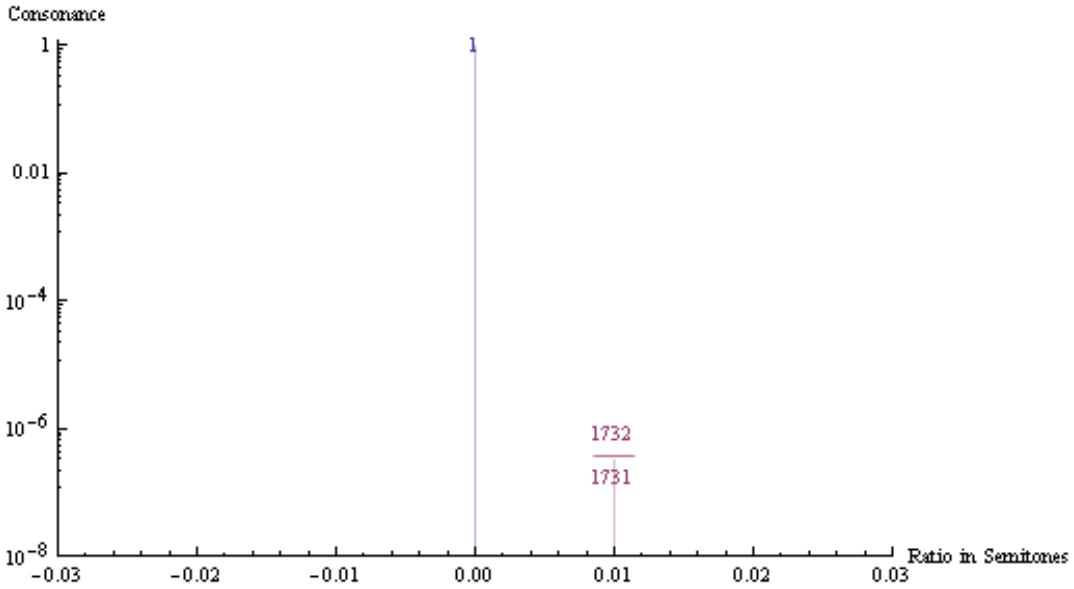


Figure 2.5: Nearby ratios that are indistinguishable

tions into consideration, each fraction gets "fuzzified" by a *Gaussian Bell Curve* like shown in Fig.4.4 [8]. In Fig.2.6 you can see all ratios from Just Intonation with a bell curve around them. Note that the left half of the figure was added by taking the inverse of every ratio. You can see that consonance calculation of a ratio is a commutative function as it is symmetric.

To get the heard fraction for any given (maybe irrational) ratio, the maximum consonance of every point on the ratio axis is taken like seen in Fig.2.7.

To get the *Semitone Difference* between two tones, the formula $12 \cdot log_2(\frac{f_1}{f_2})$ is used[10].

### 2.1.3 Considered Fractions

The consonance curve of the last section allows to calculate the consonance of any two sounding tones[11] with given base frequencies, considering the fractions of Just Intonation.

But there are more interesting (meaning more consonant) fractions than those in Just Intonation. How to calculate those?

[10] Later, when calculating the consonances, the absolute value will be taken as the curve is symmetric and only one side is needed

[11] with equal volume, but more about that later

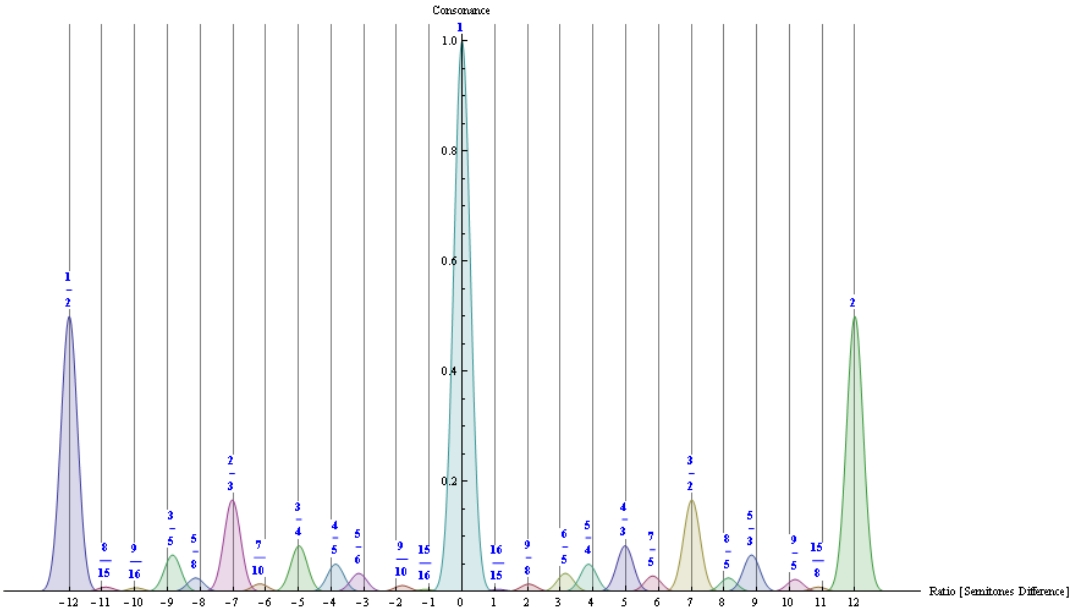


Figure 2.6: The "fuzzified" Ratios of Just Intonation from Table 2.1, note that there was added an octave below by taking the inverse of each fraction

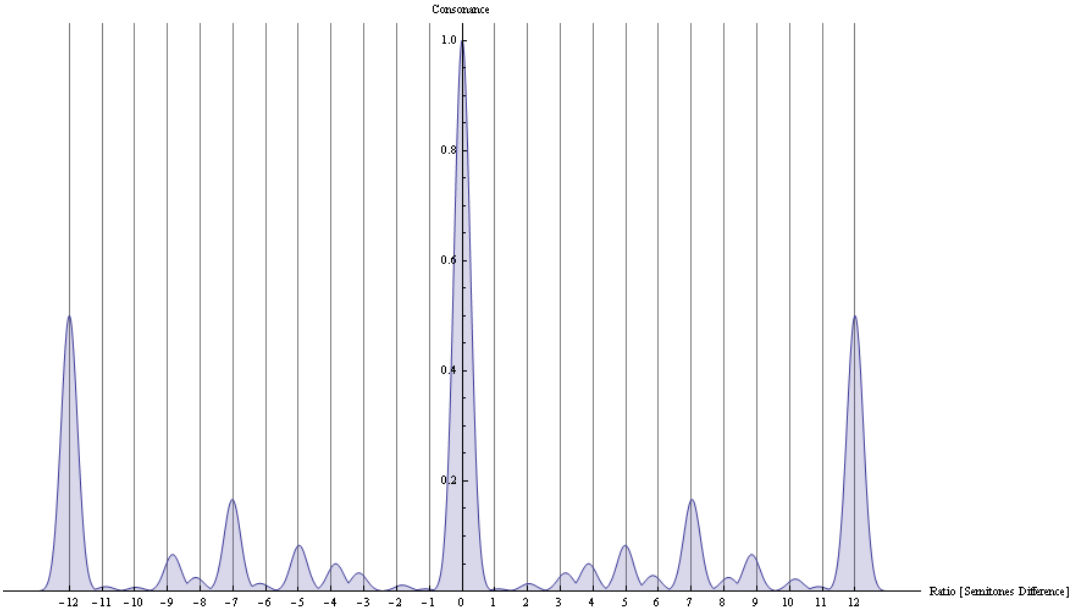


Figure 2.7: The Maximum Consonance of each Point on the Ratio Axis is heard, so this is the only interesting value

According to the formula

$$Dissonance(\frac{n}{d}) = n \cdot d : GCD(n, d) = 1$$

there exists at least one fraction for **any integer value** of the Dissonance function. We are only interested in positive non-zero results (and numerators and denominators) as negative, zero and infinite frequency values do not make much sense[12]. The first few Dissonance values with their according fractions are listed in Table 2.2.

| Dissonance Value<br>and its prime factorization | Shortened Fractions<br>and their prime factorizations |
|---|---|
| $1$ | $\frac{1}{1}$ |
| $2$ | $\frac{2}{1}, \frac{1}{2}$ |
| $3$ | $\frac{3}{1}, \frac{1}{3}$ |
| $4 = 2^2$ | $\frac{4}{1} = \frac{2^2}{1}, \frac{1}{4} = \frac{1}{2^2}$ |
| $5$ | $\frac{5}{1}, \frac{1}{5}$ |
| $6 = 2 \cdot 3$ | $\frac{6}{1} = \frac{2 \cdot 3}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{6} = \frac{1}{2 \cdot 3}$ |
| $7$ | $\frac{7}{1}, \frac{1}{7}$ |
| $8 = 2^3$ | $\frac{8}{1} = \frac{2^3}{1}, \frac{1}{8} = \frac{1}{2^3}$ |
| $9 = 3^2$ | $\frac{9}{1} = \frac{3^2}{1}, \frac{1}{9} = \frac{1}{3^2}$ |
| $10 = 2 \cdot 5$ | $\frac{10}{1} = \frac{2 \cdot 5}{1}, \frac{5}{2}, \frac{2}{5}, \frac{1}{10} = \frac{1}{2 \cdot 5}$ |
| $11$ | $\frac{11}{1}, \frac{1}{11}$ |
| $12 = 2^2 \cdot 3$ | $\frac{12}{1} = \frac{2^2 \cdot 3}{1}, \frac{4}{3} = \frac{2^2}{3}, \frac{3}{4} = \frac{3}{2^2}, \frac{1}{12} = \frac{1}{2^2 \cdot 3}$ |

Table 2.2: All (positive non-zero) Dissonance Values with their corresponding Fractions and their Prime Factorization

Examining the shortened fractions and their prime factorizations [10], one can see that all prime facors of the Dissonance value are distributed on each side of the fraction bar. Each prime factor stays with its powers, e.g. a $3^2$ will be placed either in the numerator or the denominator, but it will not get torn apart into $\frac{3 \cdot x}{3 \cdot y}$ as that would shorten out and lead to a lower Dissonance value[13]. For the prime factors of the Dissonance value, *all possibilities* of putting its prime factors on the left or right side of the fraction bar are exploited.

That way, given a maximum wanted Dissonance value[14], we can calculate all fractions that are more consonant than this. In Fig.2.8, all fractions with a Dissonance value lower or equal to *157* are shown. As the numbers collide a lot, Fig.2.9 shows the same fractions in a logarithmical measure.

[12] here

[13] that has already been covered by the emerging algorithm

[14] called *maxfrac* in the settings of Rational Piano 4.4

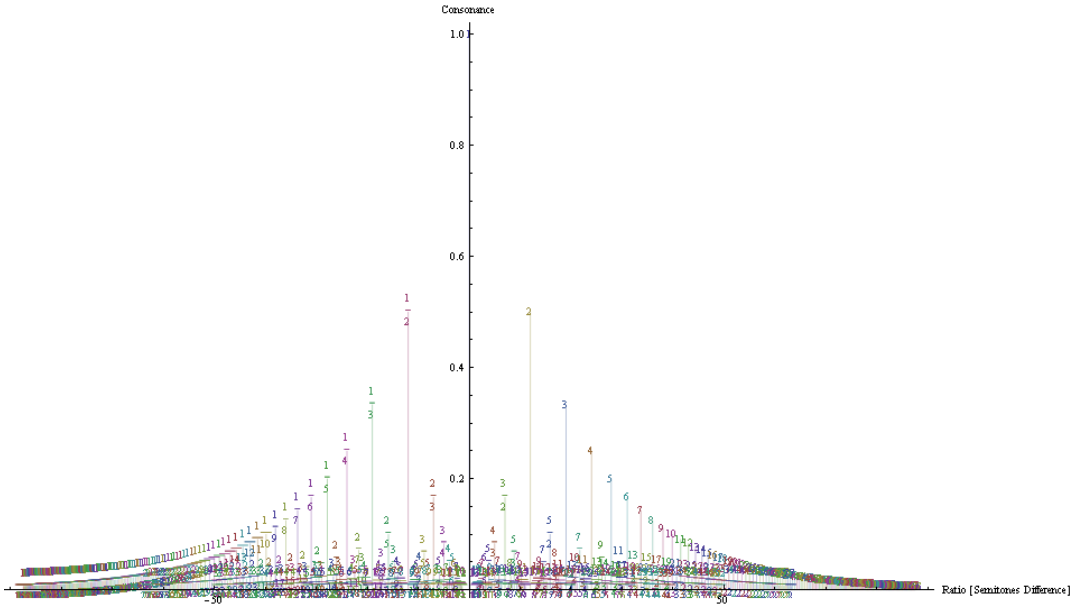


Figure 2.8: All Fractions with a Dissonance value lower or equal to 157

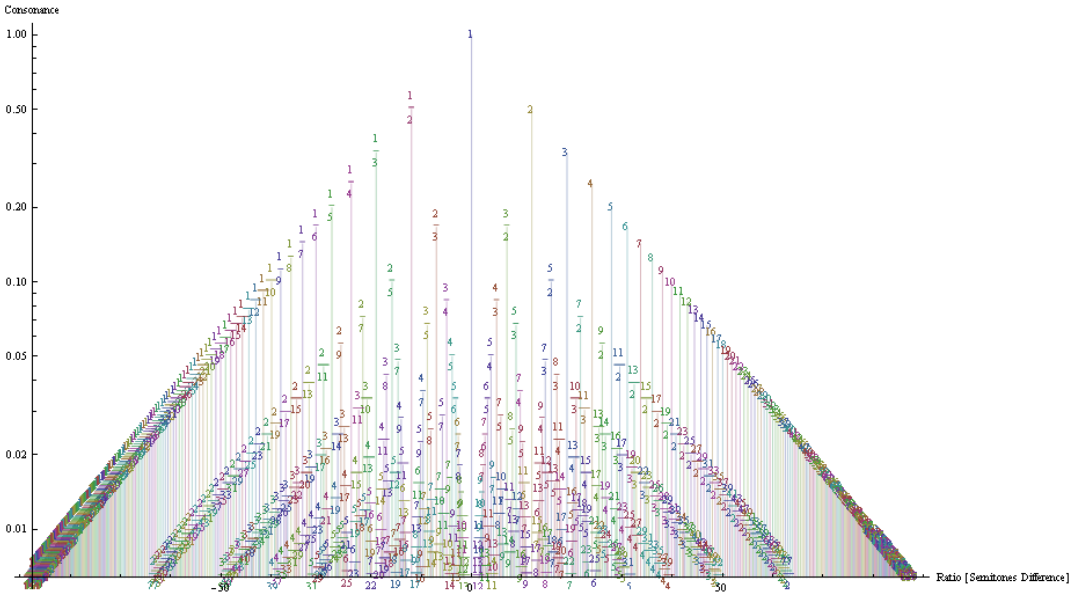


Figure 2.9: All Fractions with a Dissonance value lower or equal to 157 in a Logarithmical Measure

As we have noticed earlier, the Dissonance Function is commutative, thus we can ignore the left side of the plot when we take the absolute value of the semitone difference instead.

In the same way as in the last section, we can now "fuzzify" each fraction with a

bell curve and then, for each point on the ratio axis, calculate the maximum consonance. Fig.2.10 shows the result and Fig.2.11 the same with a logarithmical measure. Both only show a part of the the right half of the full range.

The *Harmonic Entropy Model* [12] is based on similar thoughts and leads to a very similar figure.

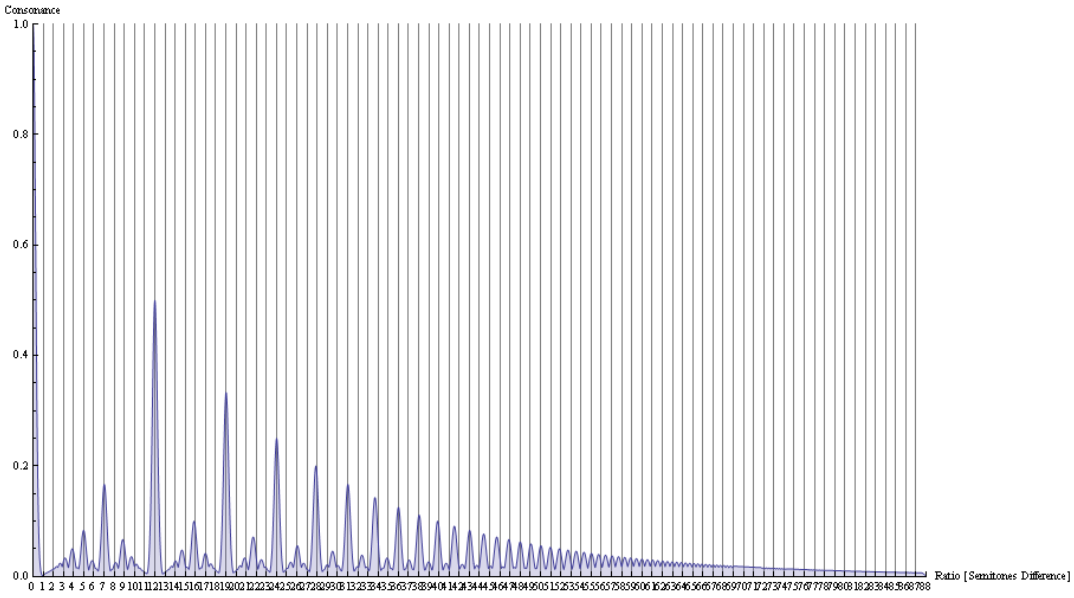


Figure 2.10: Consonance for Frequency Ratios of up to 4 Octaves

The Dissonance of any two arbitrary frequencies then is:

$DissonanceF(frequency_1, frequency_2) :=$
$DissonanceCurve(12 \cdot log_2(\frac{frequency_1}{frequency_2})) =$
$DissonanceCurve(|12 \cdot log_2(\frac{frequency_1}{frequency_2})|)$

The logarithm translates the frequency ratio to a semitone difference.

### 2.1.4 Taking account for the Tones' Volumes

With the calculated curves from the previous section, it is possible to calculate the consonance of any two frequencies **with the same volumes**. To grasp why this is, imagine two very dissonant tones playing with the same volume. When one of the two gets quieter and the other stays at its volume level, the perceived dissonance vanishes as the quieter tone does not "disturb" the louder one anymore.

Assuming the tone's volumes are in the range $[0, 1]$[15], we can multiply the calculated Dissonance value with the two volumes which makes it smaller and thus makes

[15]For real tones, there is no upper limit on the volume, but for the calculation stated here, they would have to get cropped or scaled.

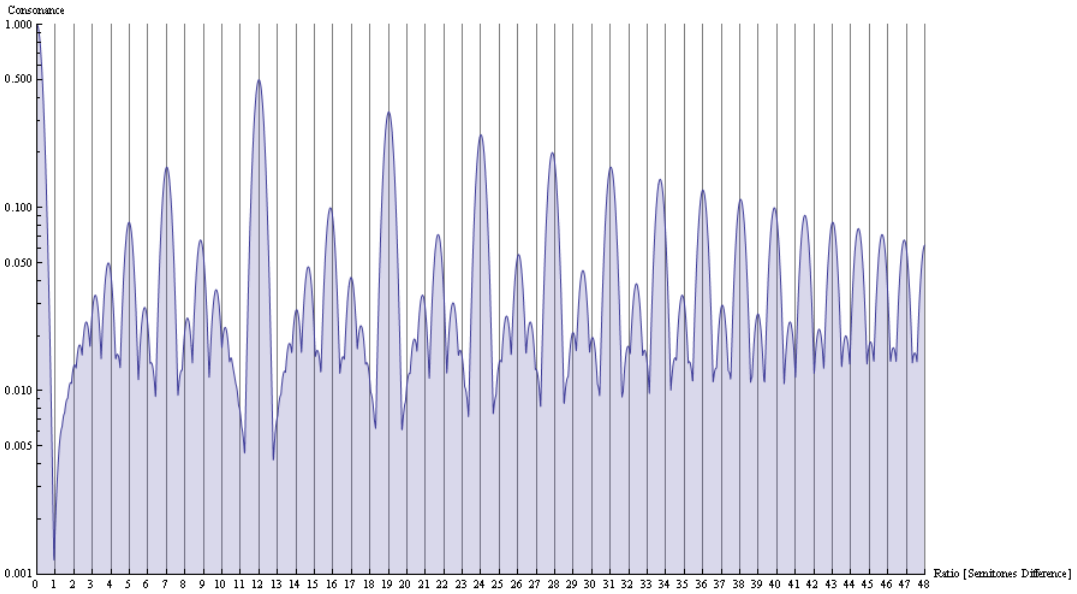


Figure 2.11: Consonance for Frequency Ratios of up to 4 Octaves in a Logarithmical Measure

the Consonance value bigger:

$$DissonanceFV(frequency_1, volume_1, frequency_2, volume_2) := Dissonance(volume_1 \cdot volume_2 \cdot DissonanceF(frequency_1, frequency_2))$$

**Note:** We have to guarantee that the modified Dissonance value stays greater or equal to 1! Otherwise, the Consonance value will no more be in range $[0, 1]$.

If we want to know how a not yet playing frequency would fit to a playing one, we use the very same formula, but we have to assume a volume for the hypothesized frequency[16].

### 2.1.5 Chords consisting of more than two Tones

Now it gets interesting as just the last brick is missing. If there are several tones playing and we want to calculate the Dissonance of any $frequency_{wanted}$ with $volume_{wanted}$, the single dissonances with the playing tones are simply added up:

$$DissonanceFVMulti(frequency_{wanted}, volume_{wanted}, frequency_1, volume_1, frequency_2, volume_2, \cdots) := 1 + DissonanceFV(volume_{wanted}, volume_1, frequency_{wanted}, frequency_1) + DissonanceFV(volume_{wanted}, volume_2, frequency_{wanted}, frequency_2) + \cdots$$

[16]With Rational Piano, always a volume of 1 is assumed. This has to do with the fact that smaller values just would dim the output, but give the same (but scaled) result. Also, the given Multitouch Table has no pressure detection, and every presses' volume is assumed as 1.

Adding 1 serves the fact that the overall Dissonance should always be greater or equal to 1. Again, this helps keeping the Consonance value in the range $[0, 1]$.

**Disclaimer:** Dissonance calculation probably is no linear function in reality, so this approach is only an idea that satisfies several goals as fast computability, continuous volume response, etc.

That is it. That is all behind the Consonance calculation in Rational Piano. Well, all exept to the actual computation described in the next section.

## 2.2 The Computation

The whole consonance calculation takes place in the source file *src/RationalPiano/-ConsonanceCalculation/Consonance.java*.

**Note:** You might also want to have a look at the Mathematica®notebook *doc/images/images.nb* which contains the consonance curve calculation, used for the images in this document, in a functional programming style. The following algorithm description is based on the Java version.

At the initialization phase, the dissonance curve gets calculated and in realtime, the dissonances of the active voices get added up for every key.

### 2.2.1 Initialization Phase

When the object *Consonance* gets constructed, all prime factorizations up to the maximum wanted dissonance get calculated, then all possibilities of distributing them on both sides of the fraction bar are iterated and finally the minimum dissonance for each semitone-difference[17] is calculated from the "bellified" fractions.

Parameters are:

1. the note range *notecount* to consider, this is the maximum semitone difference,
2. the maximum dissonance value *maxfrac* to calculate the factorization for,
3. and the width *bellWidth* of the Gaussian bell curve with that each fraction should get approximated.

[17] In this version of Rational Piano, the 12TET scale is used. This means we have fixed frequencies for which we can calculate the ratio (semitone difference) for.

### Prime Factorizations

To calculate prime factorizations, we need primes first. Those are calculated up to the value *maxfrac* as this is the maximum we could possibly need. As we need all primes, they can be calculated incrementally from 2 to *maxfrac*. That means each tested integer does not have to be divided by every lower integer, but only by every lower prime that has been found.

After we have got the prime numbers, the combined factorizations can get calculated. That also happens incrementally for every integer from 1 to *maxfrac* separately.

**Note:** "Combined Factorizations" means that all prime factors of the single primes get multiplied back together again, that means instead of returning $[2, 2, 3, 3, 5]$, the algorithm returns $[4, 9, 5]$.

The factorization algorithm tries to divide the wanted integer by all primes and keeps the factors that have no remainder.

The result is a list of factorization lists for every integer up to *maxfrac*.

### Sublists

Now, all possibilities of putting the combined prime factors on each side of the fraction bar are iterated. This happens for every factorization list of integers contained in the list separately.

The distribution iteration works with a trick: An integer is used where a binary position denotes one of the combined fractions of the given factorization. Not all binary positions are needed. So the algorithm just walks through all numbers from 0 to $1 << count(combined\_fractions)$. For each number, all used binary digits get examined: if it is 1, the combined fraction corresponding with that digit gets multiplied with the numerator; it it is 0, with the denominator. That way all possible fractions get calculated and stored as a *Rational* in a list.

**Note:** It is necessary for the integer to have at least as much binary digits as there are combined fractions in the list. There is no reason for concern here as the first whole number that has got more combined prime factors than 32 is the multiplication of the first 32 primes and that is a lot bigger than the maximum 32 bit integer: $525896479052627740771371797072411912900610967452630$.

### Calculate Minimum Dissonance for each Note

The last initialization step is to calculate the minimum dissonance for each semitone-difference in the note range.

For each semitone-difference[18], all Rationals get iterated and for each the value of the bell curve at the current position gets calculated. The minimum value gets chosen.

The result is a list where on position 0, the dissonance value for a ratio with the semitone difference of 0 is stored; on position 1 the dissonance value for a semitone difference of 1, and so on.

**Note:** By doing that step, only ratios with full semitone-differences can get taken into consideration. For other scales than 12TET, many more ratios have to get considered. When aiming at a *Continuous Scale* (see Chapter 6), there are no ratios that can be precalculated.

### 2.2.2 Realtime Consonance Calculation

There is not much left to do in realtime. The consonance calculation function gets a list with all active voices (keys) and their volumes. Then it walks through all *output* keys in the note range - that are those that are not yet playing and we want to know the consonance for - and adds up the dissonances of the active keys which it derives from the key distance and the according value in the dissonance list and the volume of that voice.

The result for each *output* key is incremented by 1 to make sure that the dissonance value is greater or equal to 1, and then it is inverted to retrieve consonance.

And that is how it works!

[18] which is an absolute value

# Chapter 3

# Running the Rational Piano

The Application should run on Windows, Linux and OSX but it was only tested on Windows.

**Additional Requirements:**

- At first you will need a *Java Runtime Environment 1.6* or later (1.5 will probably work, too) [15].
- You will also need a *virtual MIDI cable* if you are using MIDI and not SynOSCopy. See Section 3.2 for details.
- As this application just generates note output but no sound, you will also need a *Synthesizer* understanding MIDI or SynOSCopy. See Section 3.2 for more info.
- Also you will need *CCV (Community Core Vision)* for Multitouch Input (and of course a connected Camera-Beamer-Table-Installation). See section 3.3 for more.

## 3.1 A first glance

To run the precompiled JAR package, run the following command from the directory containing *RationalPiano.jar*:

```
java -jar RationalPiano.jar
```

**Note:** You can also run it with *javaw* instead, but that will prevent the displaying of the debug output.

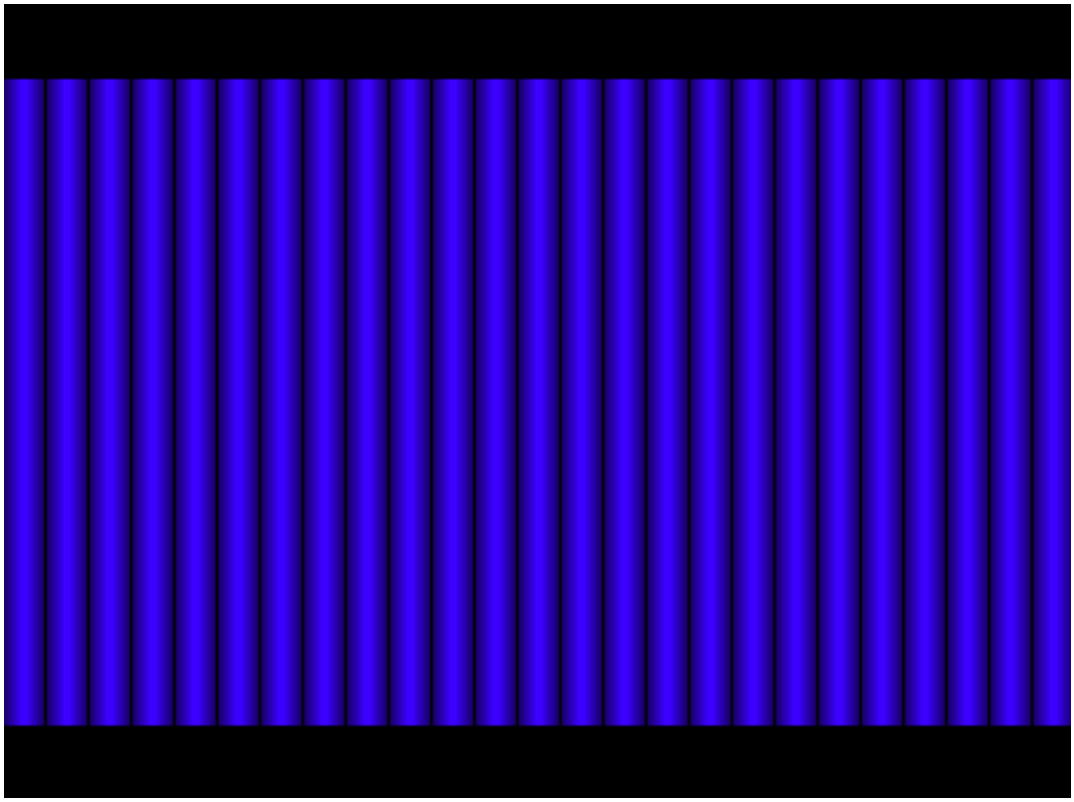

Figure 3.1: The Interface of Rational Piano after startup

**Troubleshooting:** If the *java* executable could not be found, but Java is installed, it is probably not in your PATH environment. Instead of just typing *"java"*, you can enter the full path to your *java* executable.

After the applet has been loaded, you see the interface like in Fig.3.1.

Now you can start playing around (but yet without sound or multitouch input): either click on the single lines or press your alphanumeric keyboard keys.

**Note:** The JAR has been complied for a **German Keyboard Layout**, if you want keyboard input for your differing keyboard layout, you need to recompile the application. But you are probably after Multitouch Input, so the keyboard layout does not matter anyway, it is just there for debugging purposes.

You should see a screen similar to Fig.1.1 when you press some keys. As long as a key is pressed, it is highlighted in a different color[1].

[1] green by default

## 3.2 Setting up Sound Output

To get Note Messages out of the Applet, one can either use MIDI or OSC (with namespace SynOSCopy). In either case a synthesizer application[2] is needed on the other side of that channel.

MIDI is the preferred way because nearly all software synthesizers understand it. OSC (and especially the SynOSCopy namespace) is not widely adopted for note messages to synthesizers.

### 3.2.1 Using MIDI

When using this applet via MIDI with a software synthesizer, one needs a *Virtual MIDI Cable* to connect the Applet with the Synthesizer.

*Maple Virtual Midi Cable* is a free one for Windows [17]. This specific virtual MIDI cable works simply: it has four ports that are registered as MIDI outputs and four ports that are registered as MIDI inputs. All data fed into one of those MIDI output ports gets forwarded to the corresponding MIDI input port.

By default, the MIDI output port *"Maple Midi Out: Port 4"* is chosen. You can adjust that to your needs in the configuration file, for more details see Chapter 4.

That means, in your synthesizer application you need to setup the MIDI input to *"Maple Midi In: Port 4"* or similar.

**Note:** If you have an **external** MIDI synthesizer connected to your soundcard via a physical MIDI cable, you *do not* need the virtual MIDI cable. You just need to setup the hardware MIDI output port in the configuration file, for more details see chapter 4.

#### Help with the synthesizer application

If you are desperate and do not know how to find a synthesizer application understanding MIDI, here is a little help. You also might head over to KVR [19] and browse some free (or non-free) synthesizers. Note that for most of those, you will need a *host application*.

A free VSTi[3] for Windows and OSX is 4front Piano [18].

But then you will still need a VST host application. A minimalistic free VST host is SAVIHost [21], but it is only available for Windows. See its website for information on how to load a VST plugin (like 4front Piano) in SAVIHost.

[2] or a host hosting VSTi plugins or whatever else understands one of the two communication protocols

[3] synthesizer plugin

To set the correct MIDI port in SAVIHost, in the Menu choose *Devices* → *MIDI* and as *Input Port* the entry *"Maple Midi In: Port 4"* (or whatever your configured port is).

**Now you should be able to hear your playing from Rational Piano.** Do not forget to restart the application after installing the virtual MIDI cable.

**Info:** Furtherwise, under Windows, it is adviseable (but not necessary) to use an ASIO driver for the soundcard to achieve lower latency of the audio generating part. If you have a consumer audio card that does not ship with ASIO drivers, you can use ASIO4ALL [20]. Unfortunately SAVIHost does not support ASIO, but with other hosts you can minimize your audio latency that way.

**Alternative** to 4front Piano running in SAVIHost: You could also use PureData [22], but do not be too sad about poor sounding synths[4]. Start up PureData, activate the *compute audio* checkbox, choose menu *Help* → *Browser ...* then *7.stuff/* → *synth* → *1.poly.synth.pd* which you need to double-click. Then choose menu *Media* → *MIDI settings...* and as *input device 1* choose *Maple Midi In: Port 4*, press *OK*. Now check that *Edit* mode is *deactivated* in menu *Edit* → *Edit mode* (there should be no check mark next to the menu entry). Load a preset by clicking the circle next to *recall* in one of the *preset* boxes. Last thing to do is find the box next to *MUTE* at the bottom of the patch and drag the number up to *100*. That should be enough to let PureData receive MIDI messages from Rational Piano. If not, try activating the box on the top left of the patch under the text *random note tester* which play some random notes. If not, check your audio settings in menu *Media* → *Audio settings...*.

### 3.2.2 Using OSC

When using OSC, one has to use a host or synthesizer that understands OSC and the SynOSCopy namespace. You can use PureData [22] with a custom patch for that task.

By default, OSC Voice messages get sent to UDP port *12000* of *localhost*.

**Note:** Rational Piano's Voice Management for SynOSCopy is not implemented yet due to a conceptual problem with SynOSCopy[5]. Right now, every note gets sent to the voice with the same number as the MIDI note number.

There is no polyphonic SynOSCopy patch for PureData yet. But you can try a monophonic one. Load up the *Keyboard Example* from *SynOscP5* [29]. In Edit

[4] You can make it sound good, but most simple synths just sound retro

[5] If a voice gets turned off, the sender has no idea whether the synthesized voice has already faded out when it turns on the voice with the same number again. This could cut a fading-out voice. An idea for improvement for SynOSCopy would be to specify a parameter with the voice-on message which allows to choose whether the voice should be cut or not. The receiver then must have its own voice management.

mode[6], change the box containing *"routeOSC /V1"* to *"routeOSC /*"*. Activate audio if you have not already by clicking menu *Media → audio ON*. You can also turn up the volume with the green slider (with Edit mode deactivated).

**Now you should be able to hear your playing from Rational Piano.**

## 3.3 Enabling Multitouch Input

Now, right after following the instructions of the last section, you should already be able to hear sounds when you press the typing keyboard keys.

Now we want to setup Multitouch Input from your multitouch table. You will need (or maybe already have) the program CCV (Community Core Vision) [23]. Rational Piano was tested with CCV 1.3.

In CCV, adjust the camera recognition settings to your needs, and **activate TUIO UDP output** in the right panel in the section *Communication*. This will by default map to TUIO port 3333 which is preconfigured in Rational Piano's settings.

This should be enough to make the Rational Piano application receive the multitouch messages generated by fingers pressing onto the surface. If it does not work as expected, examine the recognized blobs in CCV and try to adjust its settings.

[6] in menu *Edit → Edit mode* there should be a check mark next to the menu entry

# Chapter 4

# Tweaking the Settings

When Rational Piano is run the first time, it generates a configuration file with filename *"RationalPianoSettings.cfg"*. It is text-based and contains an explanation for each parameter which should be self-explanatory. You do not need to worry to mess anything up, the configuration file gets read everytime the application is started and immediately is written back in the original format again. All non-readable or garbage lines get sorted out. Parameters out of range get corrected, the minimum and maximum values are mentioned in the comments in the configuration file.

**Info:** For Double or Float values without a fractional, the dot is optional.

## 4.1 Window Settings

Settings to adjust window properties:

**fullscreen**
*Description:* Whether the application should get run full screen or in a window.
*Note:* In full screen mode, the window width and height settings are ignored and the screen resolution is used instead.
*Type:* Boolean.
*Default value:* true.

**width**
*Description:* Width of the application window's content.
*Note:* Only in windowed mode.
*Type:* Integer
*Bounds:* Minimum = 100, Maximum = 10000.
*Default value:* 400.

**height**
*Description:* Height of the application window's content.

*Note:* Only in windowed mode.
*Type:* Integer
*Bounds:* Minimum = 100, Maximum = 10000.
*Default value:* 300.

**framerate**
*Description:* How often the image should get redrawn per second. The consonance calculation also takes place on every frame.
*Type:* Float
*Bounds:* Minimum = 1.0, Maximum = 200.0.
*Default value:* 60.0.

## 4.2 Line Settings

**notecount**
*Description:* Counts of notes to display as lines.
*Note:* The maximum MIDI note number is 127, so notestart + notecount - 1 should be smaller than 128.
*Type:* Integer
*Bounds:* Minimum = 1, Maximum = 128.
*Default value:* 25.

**notestart**
*Description:* MIDI note number of the first note.
*Note:* The maximum MIDI note number is 127, so notestart + notecount - 1 should be smaller than 128.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 127.
*Default value:* 60 (C4).

**lineBend**
*Description:* Nonlinear distortion of the line width translation, 1 means no distortion, smaller values mean stronger distortion where the lines have a tendency to be more wide.
*Note:* For a visual explanation how input values get translated to output values in dependence of this parameter, see Fig.4.1.
*Type:* Double
*Bounds:* Minimum = 0, Maximum = 1.
*Default value:* 0.25.

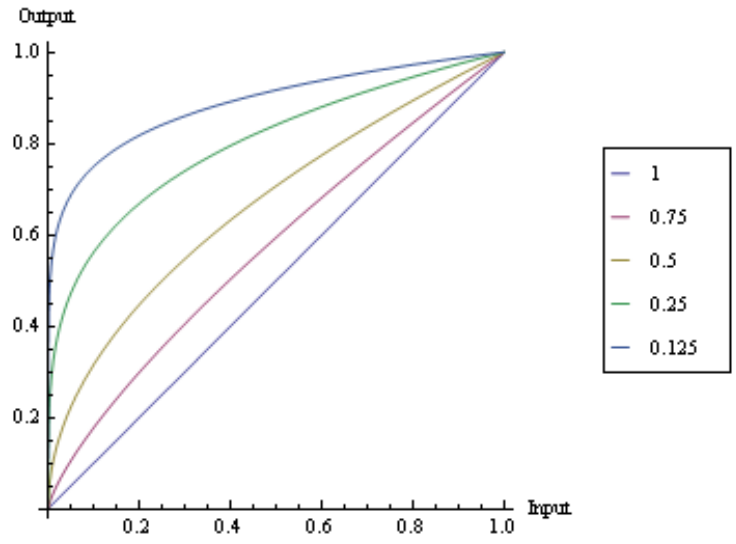


Figure 4.1: Nonlinear distortion of the line width translation

## 4.3 Note Output Settings

**outputMode**
*Description:* Choose wheter Note messages should get sent via MIDI, OSC (with the SynOSCopy namespace), or both.
*Type:* Enum RationalPiano.NoteOut.outputModes
*Values:* MIDI_ONLY, OSC_ONLY, MIDI_AND_OSC.
*Default value:* MIDI_AND_OSC.

**oscport**
*Description:* UDP port to send OSC voice messages (with the SynOSCopy namespace) to.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 65535.
*Default value:* 12000.

**midiDevice**
*Description:* Part of the string of the MIDI port to send note messages to.
*Note:* If you do not know the exact name of the port, just run the application and watch the debug output where all ports get listed.
*Type:* String
*Default value:* Maple Midi Out: Port 4.

**midiChannel**
*Description:* MIDI channel to use for note messages that get sent to the chosen MIDI port.

*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 15.
*Default value:* 0.

## 4.4 Consonance Calculation Settings

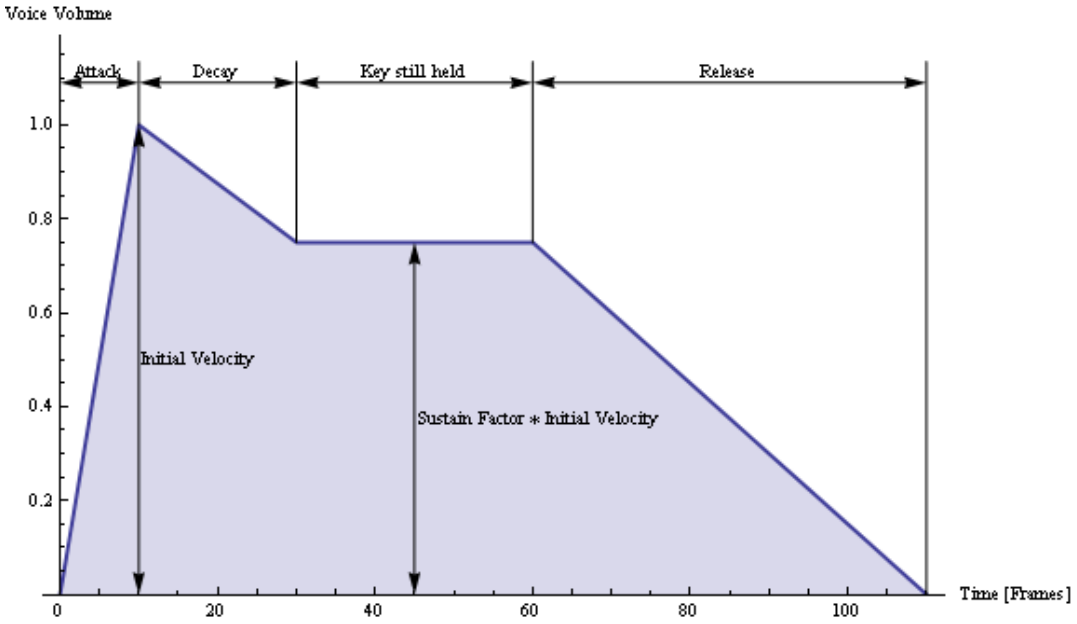


Figure 4.2: An ADSR (Attack Decay Sustain Release) curve in holdSustain mode where the key gets hold after the decay phase

For all pressed notes, internal volumes are tracked within the application for visualization purposes. To allow smooth visual transistions between note activations and deactivations, the volume of each activated note gets approximated by an ADSR (Attack Decay Sustain Release) curve. For two example ADSR curves, see Fig.4.2 and Fig.4.3. The volume curve is **not** meant to approximate the played sound of the synthesizer. It should approximate the fade-in and fade-out of the **felt presence** of a base frequency. So even when a note's sound has already been faded out physically, the human auditory system still tries to find consonant tones to that sound.

This is just for consonance calculation and does **not** impact note output.

**attack**
*Description:* Attack time of new voices in seconds.
*Type:* Double
*Bounds:* Minimum = 0, Maximum = 60.
*Default value:* 0.15.

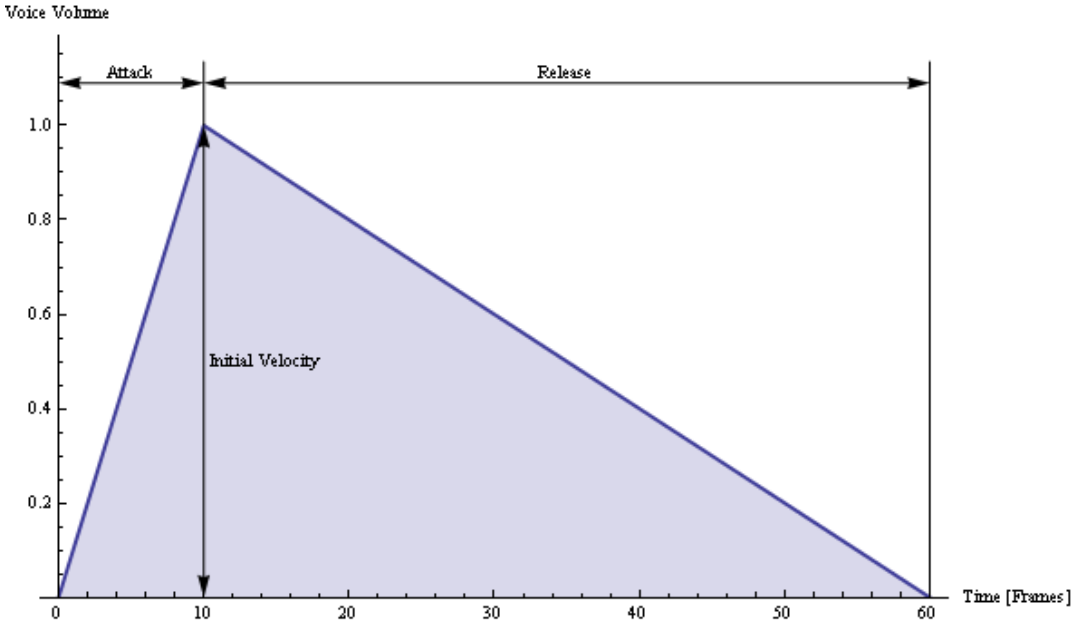


Figure 4.3: An ADSR (Attack Decay Sustain Release) curve with deactivated holdSustain mode (pluck mode) where after the attack phase, the release phase follows immediately, no matter whether the key is still held

**decay**
*Description:* Decay time of held voices in seconds.
*Note:* Only applicable if holdSustain is true.
*Type:* Double
*Bounds:* Minimum = 0, Maximum = 60.
*Default value:* 0.5.

**sustain**
*Description:* Sustain level of held voices as a fraction of the initial velocity.
*Note:* Only applicable if holdSustain is true, in that case the volume stays at the sustain level after the attack + decay phase.
*Type:* Double
*Bounds:* Minimum = 0, Maximum = 1.
*Default value:* 0.65.

**release**
*Description:* Release time of held voices in seconds.
*Type:* Double
*Bounds:* Minimum = 0, Maximum = 60.
*Default value:* 1.0.

**holdSustain**
*Description:* If true, hold the voices' volumes after attack + decay phase at the

sustain level. If false, volumes are not held and release starts right after the attack phase (pluck mode).
*Type:* Boolean
*Bounds:* Minimum = 100, Maximum = 10000.
*Default value:* true.

**maxfrac**
*Description:* Maximum dissonance to calculate a fraction for.
*Note:* For details, see Chapter 2.
*Type:* Integer
*Bounds:* Minimum = 1, Maximum = 2147483647.
*Default value:* 157, as with that value for 88 all of the notes of a full piano keyboard there is a fitting fraction in the range of $\pm 50$ semitones.

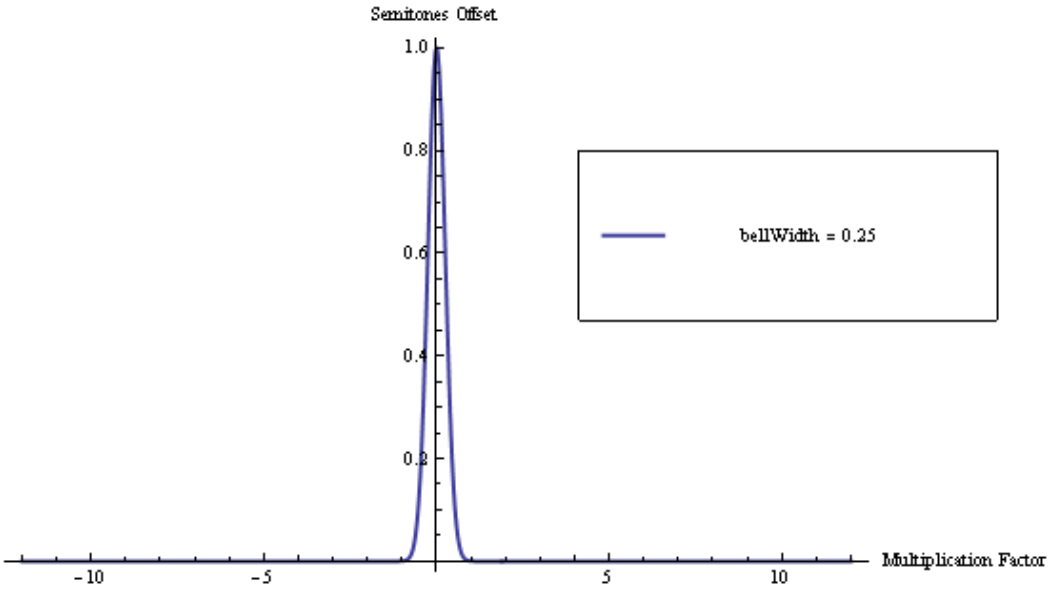


Figure 4.4: Bell Curve with width = 0.25

**bellWidth**
*Description:* Width of the bell curve which gets drawn around each fraction when calculating dissonance.
*Note:* The bell curve has the function $e^{-\frac{(-12 \cdot Log_2(fraction)+semitoneDifference)^2}{2 \cdot bellWidth^2}}$ like shown in Fig.4.4.
*Type:* Double
*Bounds:* Minimum = 0.01, Maximum = 1.
*Default value:* 0.25.

## 4.5 Multitouch Input Settings

**tuioPort**
*Description:* UDP port to listen at for TUIO cursor messages.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 65535.
*Default value:* 3333.

## 4.6 Color Settings

The colors are setup in HSB format, so one color is expressed by the three 8-bit properties Hue, Saturation and Brightness.

**backgroundColorHue**
*Description:* Hue of the Background Color.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 0.

**backgroundColorSaturation**
*Description:* Saturation of the Background Color.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 0.

**backgroundColorBrightness**
*Description:* Brightness of the Background Color.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 0.

**lineColorHueInactive**
*Description:* Hue of an inactive line.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 180.

**lineColorHueActive**
*Description:* Hue of an active line.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 80.

**lineColorSaturation**
*Description:* Saturation of the color of a line.

*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 255.

**lineColorBrightness**
*Description:* Brightness of the color of a line.
*Type:* Integer
*Bounds:* Minimum = 0, Maximum = 255.
*Default value:* 255.

# Chapter 5

# Understanding and Building the Sourcecode

All source files are well documented[1] with Javadoc. To read the generated Javadoc, open *doc/javadoc/index.html*.

## 5.1 Classes and how they work together

To get a grasp of how the classes work together, have a look at Fig.5.1[2].

Most classes are just instantiated once (in Fig.5.1 all classes that have *no* arrow with an "n" pointing at it).

### 5.1.1 RationalPiano.Run

**RationalPiano.Run.RationalPiano** is the main class whose *main()* method gets called to start the application. It is a Processing applet that gets called from itself with the method PApplet.main(). The advantage is that it can run in *present* mode[3] and as a native Java application.

RationalPiano.Run.RationalPiano initializes the logger, loads the configuration, starts up the Processing applet and creates interconnected objects of GraphicControls, NoteOutput, Voices and InputDevs. Furthermore, it forwards Processing calls to draw() to the objects of GraphicControls (for drawing) and Voices (for consonance calculation). It also forwards keyboard and mouse events to the instance of InputDevs.

[1] hopefully
[2] Notice that this is not UML
[3] fullscreen mode

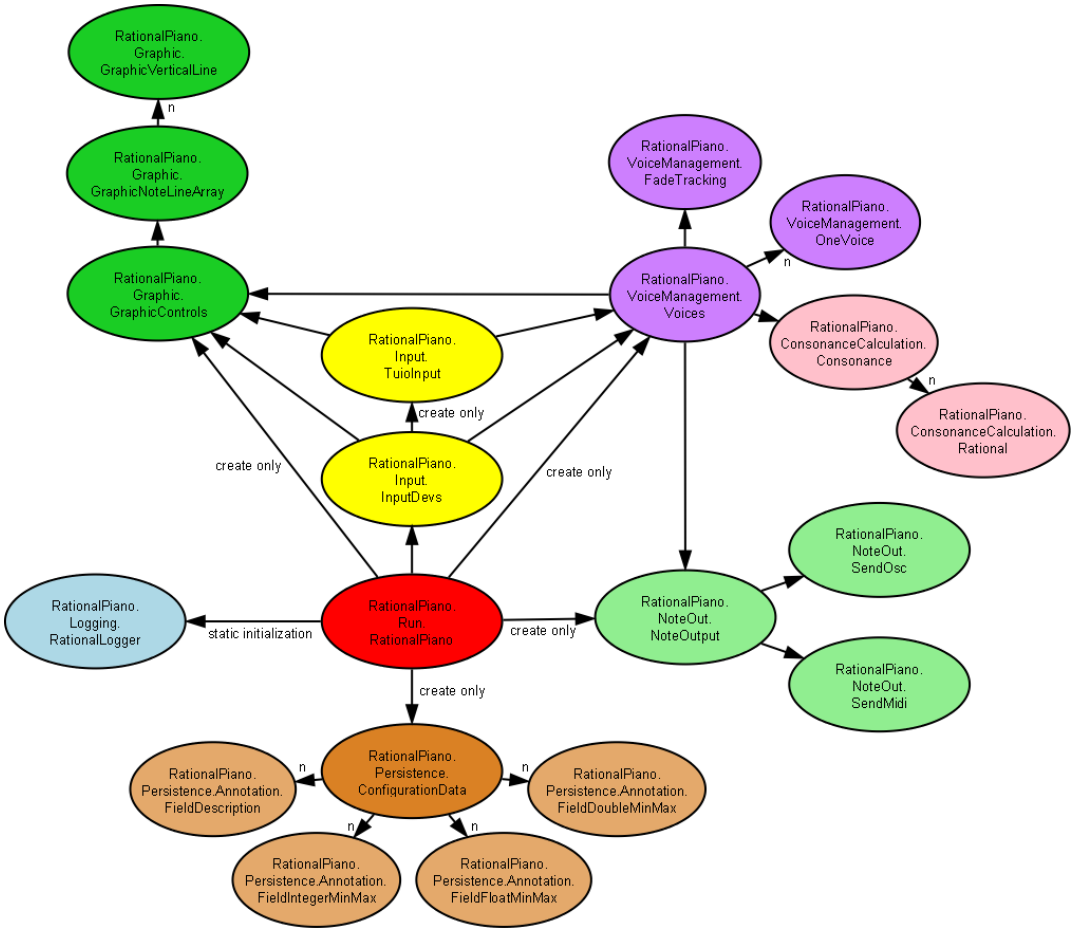


Figure 5.1: Interaction of the Classes

### 5.1.2 RationalPiano.Logging

**RationalPiano.Logging.RationalLogger** provides a method to initialize the top logger which gets (indirectly[4]) used for all debug output of the application. It is also possible to write all output to a logfile by initializing the logger with a different parameter.

### 5.1.3 RationalPiano.Persistence

**RationalPiano.Persistence.ConfigurationData** has several public fields that are the initialization parameters for the generated objects mentioned in section 5.1.1. For an explanation of all fields, have a look at chapter 4. The class reads in the configuration from a file and writes the configuration to it again.

[4]The loggers of all other classes get routed through the top logger

### 5.1.4 RationalPiano.Persistence.Annotations

This package contains the annotations **FieldDescription**, **FieldDoubleMinMax**, **FieldFloatMinMax** and **FieldIntegerMinMax** which are used to comment the fields of RationalPiano.Persistence.ConfigurationData in a way that it is possible to write the comments to the configuration file.

### 5.1.5 RationalPiano.VoiceManagement

**RationalPiano.VoiceManagement.Voices** manages active voices. It provides methods to add and remove voices which get called by an object of InputDevs. Calls to that method get forwarded to NoteOut. Voices also keeps an instance of Consonance with which it calculates the consonance for all active voices every frame. The resulting consonances get sent to the object of GraphicControls to set the line widths.

Every active voice is an object of type **RationalPiano.VoiceManagement.OneVoice** that allows to store various parameters like initial velocity or hold time individually for each active voice.

**RationalPiano.VoiceManagement.FadeTracking** makes it possible to use an ADSR envelope[5] for each voice. The Voices object uses an object of this class to calculate a current velocity for each active voice.

### 5.1.6 RationalPiano.ConsonanceCalculation

**RationalPiano.ConsonanceCalculation.Consonance** provides a method to calculate the consonance for all keys in range for a set of voice velocities. The Voices object initializes and keeps an instance of this class. For an explanation of the computation, see section 2.2.

**RationalPiano.ConsonanceCalculation.Rational** represents a rational number and is used for the initialization of Consonance.

### 5.1.7 RationalPiano.Graphic

**RationalPiano.Graphic.GraphicControls** manages the screen elements[6] and allows to draw them with a provided method. It holds an instance of GraphicNoteLineArray and provides methods to get line parameters and to set line widths and activation status from an object of Voices. It also provides a function for getting the

[5] see section 4.4

[6] right now just lines

corresponding line for an x-y-coordinate for use with an object of InputDevs and TuioInput.

**RationalPiano.Graphic.GraphicNoteLineArray** holds all GraphicVerticalLine objects drawn on the screen. It distributes them across the screen and manages their corresponding MIDI keys. Most line functions of GraphicControls are just wrappers around methods of this class.

**RationalPiano.Graphic.GraphicVerticalLine** represents a vertical line that has a changeable width and activation state. It provides a method to get drawn on screen.

### 5.1.8 RationalPiano.Input

**RationalPiano.Input.InputDevs** manages input from mouse and keyboard. For multitouch it creates an object of TuioInput. For each new input it looks up the corresponding MIDI key in the GraphicControls object and then activates or deactivates the according voice with a call to the Voice object.

**RationalPiano.Input.TuioInput** manages TUIO input and processes it the same way as InputDevs.

### 5.1.9 RationalPiano.NoteOut

**RationalPiano.NoteOut.NoteOutput** sets up MIDI and SynOSCopy output according to the configuration. It provides functions to activate and deactivate a voice which get called from the Voices object. It forwards those calls to SendMIDI and/or SendOSC, depending on the configuration.

**RationalPiano.NoteOut.SendMidi** opens the configured MIDI device and forwards note messages to it.

**RationalPiano.NoteOut.SendOsc** forwards note messages to the configured UDP port as a SynOSCopy message.

## 5.2 Library Dependencies

The application relies on Java, Processing and some additional libraries for MIDI, SynOSCopy and TUIO support.

The necessary libraries[7] are included in the subdirectory *lib*. All of them are open source. For the exact licenses, see the following sections.

[7] except to the JDK

### 5.2.1 Java Development Kit

You will need a Java SE Development Kit [16] with version 6 or later. Version 5 probably would work too, but it was not tested.

### 5.2.2 Processing

The library *core.jar* included with Processing is needed [24]. This application was tested with versions 1.0, 1.1 and 1.2.1, no changes were necessary to run it with the diverse versions.

The used export library is licensed under the GNU LGPL [31].

### 5.2.3 TUIO

For multitouch input, the TUIO library *libTUIO.jar* with version 1.4 is used [26].

The library is licenced under the GNU GPL version 2 [32].

### 5.2.4 RWMidi

For MIDI in- and output, the library *rwmidi.jar* with version 0.1c is used [25].

It is licensed under the GNU LGPL [31].

### 5.2.5 SynOscP5

For SynOSCopy output, the libraries *javaosc.jar* and *synOscP5.jar* with versions from 12 September 2009 are used [28].

The library *synOscP5.jar* is licensed under the Artistic License [33].

The exact copyright notice and license text for the library *javaosc.jar* is stated in file *lib/javaosc_license.txt*.

# Chapter 6

# Prospects and Future Development Ideas

## 6.1 Bugs

There are some lines marked in the sourcecode with the keywords **TODO**, **FIXME** and maybe also **XXX**. Next to TODO or FIXME there should be a description for what is to do or to fix. XXX usually stands for a missing Javadoc comment.

- **Problem:** The RWMidi Library does not throw exceptions. So when opening an unavailable MIDI device, a stack trace gets printed out, but the program continues normally, unaware of the failed attempt. Everytime a MIDI message gets sent to the unopened device, a new stack trace gets printed.
  **Solution:** Modify the open source library RWMidi and throw exceptions instead of catching them and printing stack traces.
- **Problem:** If two blobs get too close, they merge and generate a new blob with a new note-on message.

## 6.2 Expansions

### 6.2.1 Display and Interface

- Make the consonance display of the lines **more clear** - the width is not that good recognizeable for smaller ones, e.g. show the consonance additionally with a triangle with another color and a variable height.
- Provide a **Menu** to adjust parameters online.

- Calculate **Overall Consonance** of the playing and display it e.g. by a varying background color.
- Provide an additional **Visual Output** of active voices' nearest rational numbers in a prime factor view or in divisor tree view.

### 6.2.2 Input

- Make **Keyboard-Input localized** and not just for the German Layout.
- It would also be thinkable to let away the multitouch input and directly **Input from a MIDI Keyboard**.
  The visual output could be differed too, so that with a beamer the consonances could be displayed next to the input keys. That would not be possible with a continuous scale of course (except with a Continuum Fingerboard [34]).
- Allow **MIDI Input** which just impacts consonance calculation, but does not get routed to the output.
- An expansion of the previous idea: A **"Tutor" Mode** where musical pieces or harmony series get played back that also do not get routed to the output.
- **Polyphonic Pitch-Shift** where a moving finger changes the frequency of the corresponding note. The frequencies where the note snaps could either be keys of a fixed scale or follow different algorithms (see next idea).
  **Problems:** It is hard for the framework to do the tracking of crossing fingers, errors are possible. MIDI messages would need pitch bend which only works with a channel hack.
- **Snap Setting** for polyphonic pitch-shift - for example the voices could snap to at least equal-consonant frequencies as the first touch, or another example is a controller (like a foot pedal) to adjust the snap width.
- Introduce the possibility of an **Interconnection of several Multotouch Tables** which are all separate keyboard controllers by themselves, but whose consonance calculation takes account for active voices of the other tables.

### 6.2.3 Consonance Calculation

- The consonance calculation formula could get tweaked a little so that the fade-out looks smoother at the end.

- ADSR values could be relative delta-values.
  **Benefit:** The attack-/decay-/release-**rate** and not the time is the same for all voices.
  **Problem:** The voices then are no more stateless as they must know in which phase they are, there is no possibility to determine the phase just from the frame-age of the voice anymore.

- Take account for the **Spectrum of the Sound Generator** for calculating the consonance[1]. This could either be achieved with static samples of the synthesizer or with dynamically recorded ones.

- An expansion of the previous idea would be to record the environment and provide the possibility to play **consonant to the ambient**.

### 6.2.4 Scales

- Use different (fixed) **Scales** than 12TET.
  **Difficulty:** MIDI Tuning messages or MIDI Pitch Bend messages would be necessary.

- Provide a **Continuous Scale**[2] as input. Pressed x-values "snap" to a next consonant ratio. The minimum consonance could be set in real time by a parameter (controlled e.g. by a foot pedal).
  **Difficulty:** MIDI messages would need pitch bend which only works with a channel hack.

### 6.2.5 Dreams

- An **Automatic Mode** where a very harmonic playing or accompaning takes place would be an ultimate goal. This is very hard as also the dimension of rhythm has to be considered.

[1] William Sethares developed such an approach with his "Adaptive Tuning"
[2] A bit like the Continuum Fingerboard [34]

# Appendix A

# History of Changes

**2010-07-30**

First public release.

# Bibliography


[1] *Consonance* Wikipedia entry. http://en.wikipedia.org/wiki/Consonance.

[2] *MIDI* Wikipedia entry. http://en.wikipedia.org/wiki/Midi.

[3] *OSC*. http://opensoundcontrol.org/.

[4] William Sethares, *Wikipedia entry*. http://en.wikipedia.org/wiki/William_Sethares.

[5] William Sethares, *Adaptive Tuning*. http://eceserv0.ece.wisc.edu/~sethares/paperspdf/adaptun.pdf.

[6] *Equal Temperament* Wikipedia entry. http://en.wikipedia.org/wiki/Equal_temperament.

[7] *Just Intonation* Wikipedia entry. http://en.wikipedia.org/wiki/Just_intonation.

[8] *Gaussian Bell Curve* Wikipedia entry. http://en.wikipedia.org/wiki/Gaussian_function.

[9] *Musical Cent* Wikipedia entry. http://en.wikipedia.org/wiki/Cent_(music).

[10] *Prime Factorization* Wikipedia entry. http://en.wikipedia.org/wiki/Prime_factorization.

[11] *Consonance and Dissonance* Wikipedia entry. http://en.wikipedia.org/wiki/Consonance_and_dissonance.

[12] *Harmonic Entropy* Model. http://launch.groups.yahoo.com/group/harmonic_entropy/.

[13] *Wolfram Mathematica* Homepage. http://www.wolfram.com/products/mathematica/index.html.

[14] *Dia* Homepage. http://live.gnome.org/Dia.

[15] *Java Runtime Environment (JRE) Download*. http://www.java.com/en/download/manual.jsp.

[16] *Java SE Development Kit (JDK) 6 Download*. http://java.sun.com/javase/downloads/widget/jdk6.jsp.

[17] *Maple Virtual Midi Cable*. http://www.maplemidi.com/Maple_driver.html.

[18] *4front Piano VSTi*. http://www.yohng.com/piano.html.

[19] *KVR*. http://www.kvraudio.com/.

[20] *ASIO4ALL*. http://www.tippach.net/asio4all.

[21] *SAVIHost* free VST Host. http://www.hermannseib.com/english/savihost.htm.

[22] *PureData*. http://puredata.info/downloads.

[23] *CCV (Community Core Vision)*. http://ccv.nuigroup.com/.

[24] *Processing Download*. http://processing.org/download/.

[25] *RWMidi*. http://ruinwesen.com/files, http://ruinwesen.com/support-files/rwmidi-0.1c.zip. Documentation: http://ruinwesen.com/support-files/rwmidi/documentation/RWMidi.html.

[26] *TUIO Download*. http://www.tuio.org/?software, Direct Download at http://prdownloads.sourceforge.net/reactivision/TUIO_JAVA-1.4.zip?download, Javadoc at http://www.tuio.org/?java.

[27] *SynOSCopy Namespace Proposal*. http://synosc.wetpaint.com/.

[28] *SynOSCopy*. http://code.google.com/p/synoscp5/ and http://51858360.de.strato-hosting.eu/synoscp5/.

[29] *SynOSCopy Keyboard Example*. http://51858360.de.strato-hosting.eu/synoscp5/keyboard.html and a direct link to the PureData patch: http://51858360.de.strato-hosting.eu/synoscp5/examples/keyboard/keyboard.pd.

[30] *Open Software License ("OSL") v. 3.0*. http://www.opensource.org/licenses/osl-3.0.php.

[31] *GNU Lesser General Public License*. `http://www.opensource.org/licenses/lgpl-license.php`.

[32] *GNU General Public License version 2*. `http://www.opensource.org/licenses/gpl-2.0.php`.

[33] *Artistic License*. `http://dev.perl.org/licenses/artistic.html`.

[34] *Continuum Fingerboard*. `http://www.cerlsoundgroup.org/Continuum/`.