Interactive Music Science Collaborative Activities

Team Teaching for STEAM Education

Deliverable 4.4
First Version of Computational models for sound and music generation for virtual instruments

Date:	05/10/2017
Author(s):	Robert Piechaud (IRCAM), Quentin Lamerand (IRCAM)
Contributor(s):	
Quality Assuror(s):	Fotini Simistira (UNIFRI), Carlos Acosta (LEOPOLY)
Dissemination level:	PU
Work package	WP4 – Core enabling technologies modules of iMuSciCA
Version:	1.0
Keywords:	sound generation, virtual musical instrument
Description:	First version of the computational models for sound and music generation for virtual instruments.

H2020-ICT-22-2016 Technologies for Learning and Skills
iMuSciCA (Interactive Music Science Collaborative Activities)
Project No. 731861
Project Runtime: January 2017 – June 2019

Executive Summary

In this deliverable we present the first version of the computational models for sound and music generation for virtual instruments. The virtual instrument models are based on physics and embodied in Ircam's Modalys technology. As part of the iMuSciCA project, Modalys has been ported from C++ to the browser HTML5 context in order to function like any other workbench module. The result library is called modalys.js. First we present the Modalys iMuSciCA API, and then we will expose some standalone examples. Finally and briefly, we will talk about CPU performances.

The following urls will be come across in this document:
Root url:
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/

Instruments test urls:
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings.html
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings_with_snail.html
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/xylo.html
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/plate.html
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/simpledrum.html
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings2channels.html

Version Log			
Date	**Version No.**	**Author**	**Change**
04-09-2017	0.1	Robert Piechaud (IRCAM)	Initial content
02-10-2017	0.2	Robert Piechaud (IRCAM), Quentin Lamerand (IRCAM)	Document finalization, submission for quality assurance
04-10-2017	0.3	Robert Piechaud (IRCAM)	Incorporation of comments of quality assurors
05-10-2017	1.0	Vassilis Katsouros (ATHENA)	Submission to the EU

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation	Description
JS	Javascript
WASM	Web Assembly
asm.js	Strict subset of Javascript
API	Application Programming Interface
CPU	Central Processing Unit
WP	Work Package
PU	Public document
ATHENA	ATHENA RESEARCH AND INNOVATION CENTER IN INFORMATION COMMUNICATION & KNOWLEDGE TECHNOLOGIES
UCLL	UC LIMBURG
EA	ELLINOGERMANIKI AGOGI SCHOLI PANAGEA SAVVA AE
IRCAM	INSTITUT DE RECHERCHE ET DE COORDINATION ACOUSTIQUE MUSIQUE
LEOPOLY	3D FOR ALL SZAMITASTECHNIKAI FEJLESZTO KFT
CABRI	Cabrilog SAS
WIRIS	MATHS FOR MORE SL
UNIFRI	UNIVERSITE DE FRIBOURG

1. Introduction

The Modalys[1] virtual music instrument technology is embedded into **modalys.js**, a single-file javascript library. Modalys owns two types of sounding objects: i) primitives, such as strings, tubes, plates or membranes, and ii) 3D objects defined by their geometry expressed in a mesh in .obj format. In the second case, the mesh can either contains solids[2], or surface elements representing the neutral fiber[3] of the object.
modalys.js, through its message-based communication API, contains all that is required to instantiate, test and finally perform virtual musical instruments based on physical models.

2. Installation and technical requirements for modalys.js

2.1. Installation

modalys.js physical model engine is activated just by including the javascript library in the html code:

```
<script src="script/modalys.js"></script>
```

2.2 Requirements

modalys.js is built using emscripten[4] with heavy optimizations. It produced asm.js[5] code, as special, strict subset of javascript. As such, it can only run properly on Firefox and Edge "nightly" (alpha) build.
That said, we intend to target WebAssembly[6] in the final version of modalys.js for iMuSciCA, no more restricting ourselves to Firefox and Edge development builds, and thus allowing a much broader browser compatibility with even better performances.

3. Description and use of modalys.js

As a core sound library, modalys.js doesn't ***display*** anything (apart from log messages within the browser's developer console). As such, modalys.js works very much as a server, responding to client's request to test or perform virtual instruments.

[1] http://forumnet.ircam.fr/product/modalys-en/
[2] out of iMuSciCA's scope.
[3] or neutral axis: https://en.wikipedia.org/wiki/Neutral_axis
[4] https://en.wikipedia.org/wiki/Emscripten
[5] https://en.wikipedia.org/wiki/Asm.js
[6] http://webassembly.org/

3.1. modalys.js API

3.1.1. Communication context

iMuSciCA core modules communicate with each other through an asynchronous client-side messaging service: postal.js, which implements a channel/topic(.subtopic) paradigm.

- We use the modalys channel.
- Each module has a dedicated topic and subtopics
- To 'talk' to the Modalys module, another module will publish on a topic from the modalys channel, named after the requested action (ex: play)
- Module can send notifications by publishing on the modalys > notification topic.
- A module will therefore subscribe to modalys > notification to receive Modalys' feedbacks.

3.1.2. Messages *to* Modalys

Upon initialization modalys.js subscribes to all possible modalys topics :

```
var channel = postal.channel("modalys");

channel.subscribe("#", function(data, envelope) { // subscribe to all topics
  // dispatch to matching method of Modalys
});
```

Modalys' subtopics currently are:

- modalys > **try**: test the sound of an instrument being designed.
- modalys > **play**: real time performance of a virtual instrument.
- modalys > **updateParameter**: update some parameter (pluck position etc) in real time while playing.
- modalys > **pause**: put a virtual instrument in hold (play mode)
- modalys > **resume**: resume playing the instrument after hold.
- modalys > **stop**: stop a virtual instrument.

3.1.3. Messages *from* Modalys

modalys.js fires notifications on the modalys > notification topic when necessary.

3.1.4. Trying a virtual music instrument at design stage

It is a leopoly ↔ modalys communication situation.
Below are the types of messages that should be sent upon instrument design stage, when testing how an instrument sounds.

Primitive-based instruments

Example for a 2-string instrument:

```
channel.publish("try", {
  "requestId": "[unique ID provided by Leopoly]",
  "instrumentType": "pluckedString",
  "name": "My plucked bichord",
  "parts": [
    {
      physicalParameters : {
        "attachmentPoints": [{
          "x": 0.001,
          "y": 0.002,
          "z": 0.09
        }, {
          "x": 1.0,
          "y": 0.002,
          "z": 0.09
        }],
        "radius": 0.001,
        "tension": 10.5,
        "material": "steel"
      }
    },
    {
      physicalParameters : {
        "attachmentPoints": [{
          "x": 0.001,
          "y": 0.012,
          "z": 0.09
        }, {
          "x": 1.0,
          "y": 0.012,
          "z": 0.09
        }],
        "radius": 0.0018,
        "tension": 9.3,
        "material": "nylon"
      }
    }
  ]
});
```

⚠ ***The attachment points for a string will be interpreted by Modalys as between the nut and the bridge.***

Parts

Some instruments are made of one single object (ex: a bell), others from multiple similar objects called “parts” (ex: a simplified violin made made with 4 strings, a xylophone made of 12 bars etc.). In some other iMuSciCA documents, parts are sometimes called “subobjects”.

Mesh-based instruments

Example for a bell:

```
channel.publish("try", {
  "requestId": "[unique ID provided by Leopoly]",
  "instrumentType": "3dBell”
  "name": "My 3D Bell",
  "parts": [{
    "physicalParameters": {
      "thickness": 0.01,
      "material": "bronze"
    }
  }],
  "Mesh": "o Bell\nv 0 0 0\n...[surface mesh in .obj format]...f 243 543 653\n"
});
```

In return, modalys should send this message upon reception:

```
channel.publish("notification", {
        "requestId": "[the ID that was provided in the request]",
        "sessionId": "[a unique ID provided by Modalys]",
        "status": "preparing"
});
```

And then, when ready:

```
channel.publish("notification", {
        "requestId": "xxxxxx",
        "sessionId": "yyyyyy",
        "status": "playing"
});
```

Finally, when the sound testing is over, modalys terminates the instruments and sends this:

```
channel.publish("notification", {
        "requestId": "xxxxxx",
        "sessionId": "yyyyyy",
        "status": "over"
});
```

⚠ ***Modalys determines when to stop the test sound, depending on the instrument type.***

NB: the **requestID** should be generated by the workbench. Available to every module, there must be a global function:

```
function getUniqueID() {
                //get a unique ID some way
                var uniqueID = ...
```

```
                         return uniqueID;
}
```

3.1.5. Playing a virtual music instrument

It is a Athena ↔ modalys communication situation.
Below are the main types of messages that are sent in performance situation.

3.1.5.1. Preparing for performance

When performance mode is initialized, the following message must be sent to Modalys to prepare for performance mode:

```
channel.publish("play", {
   "requestId": "[unique ID provided by Αθηνά]",
   "instrumentType": "pluckedString",
   "name": "My plucked bichord",
   "uniqueInstrumentId": "2342343", // created by the design environment, this id
                              // is registered on the server as well.
   "parts": [{
         "partId": "A",
         "physicalParameters": {
            "attachmentPoints": [{
               "x": 0.001,
               "y": 0.002,
               "z": 0.09
            }, {
               "x": 1.0,
               "y": 0.002,
               "z": 0.09
            }],
            "radius": 0.001,
            "tension": 10.5,
            "material": "steel"
         }
      },
      {
         "partId": "B",
         "physicalParameters": {
            "attachmentPoints": [{
               "x": 0.001,
               "y": 0.012,
               "z": 0.09
            }, {
               "x": 1.0,
               "y": 0.012,
               "z": 0.09
            }],
            "radius": 0.0018,
            "tension": 9.3,
            "material": "nylon"
          }
        }
    ]
});
```

You can see that each part (string here) has an ID (unique among parts). This ID will be used for mapping the right gesture to the right part (ex: which string is being plucked in a guitar).

3.1.5.2. Error notification

If a problem occurred while instantiating the instrument, modalys fires an error notification. For instance:

```
channel.publish("notification", {
    "requestId": "xxxxxx"
    "status": "error",
    "message": "Error instantiating instrument (request ID xxxxxx): missing physical parameter."
});
```

3.1.5.3. Notification when ready to play

When the instrument is ready to be performed, modalys.js sends a notification which includes the parameter introspection of the instrument:

```
channel.publish("notification", {
    "requestId": "xxxxxx",
    "sessionId": "yyyyyy",
    "status": "ready",
    "partsNumber": 2,
    "performanceParameters": [{
          "name": "plectrumPosition",  /* the 'vertical' position of the plectrum towards the string*/
          "type": "float",
          "min": -1.0,
          "max": 1.0
      }, {
          "name": "stringPluckPoint",  /* the interaction point along the string*/
          "type": "float",
          "min": 0.01,
          "max": 0.99
      }
    ]
});
```

3.1.5.4. Performing

During the performance, gesture data are being sent to modalys.js in real time, and several parameters can be changed at once:

```
channel.publish("updateParameter", {
  "sessionId": "yyyyyy",
  "parameters": [{
    "name": "plectrumPosition",
    "partId": "A",
    "value": 0.03,
    "when": 0.01

    }, {
    "name": "stringPluckPoint",
    "partId": "A",
    "value": 0.49
    }
  ]
});
```

The **when** field is optional. It creates a linear interpolation between the current value and the new one, during the specified time (in sec.). If absent, the parameter change is instant.

3.1.5.5. Pausing, resuming or ending a performance

A session can be paused or resumed instantly ; the effect in modalys.js will be to disconnect/reconnect the web audio node attached to the virtual instrument. But the current state of this instrument is kept "frozen". This following request must be sent:

```
channel.publish("pause [or resume]", {
  "sessionID": "yyyyyy"
});
```

To end a performance session, send the following request:

```
channel.publish("stop", {
  "sessionID": "yyyyyy"
});
```

And finally, once the instrument is actually terminated, modalys fires this:

```
channel.publish("notification", {
  "sessionID": "yyyyyy",
  "status": "over"
});
```

3.1.6. The server side eigenvalue solver for 3D objects

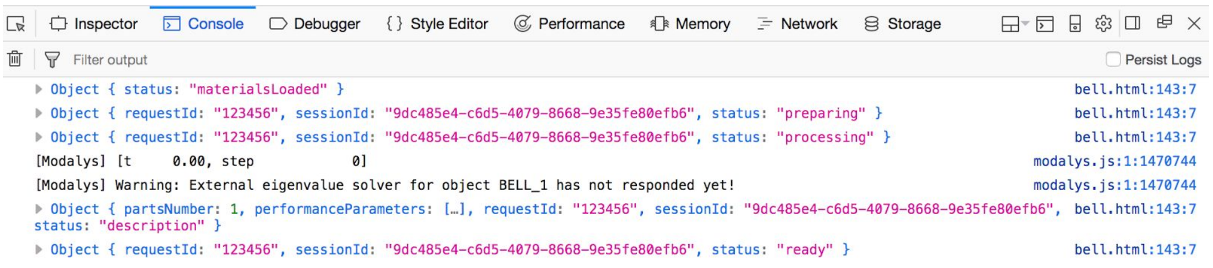
The ability to create sounding objects from 3D mesh is one of Modalys' particularities. This is, however, the most demanding mathematical aspects of the core engine.

The static mode computation increases exponentially upon the mesh's granularity, with a generalized eigenvalue problem under the hood. For that reason, this very CPU-intensive initial phase is processed server-side by the **eigenvalueproblemsolver** command line (based on Lapack[7] and Superlu[8]) with asynchronous client requests (AJAX) .

3.1.6.1. 3D object instantiation sequence

There are essentially 4 steps when instantiating a 3D object:

1. A musical instrument creation request is received by another iMuSciCA module. Status is **preparing**.
2. It is then processed, and an asynchronous AJAX request is sent to the eigenvalueproblemsolver. Status is **processing**.
3. The instrument's parameter description is broadcast for other modules to know.
4. Once the mode computation is received by the client, the instrument is ready to be performed. Status is **processing**.



```
▸ Object { status: "materialsLoaded" }                                                              bell.html:143:7
▸ Object { requestId: "123456", sessionId: "9dc485e4-c6d5-4079-8668-9e35fe80efb6", status: "preparing" }     bell.html:143:7
▸ Object { requestId: "123456", sessionId: "9dc485e4-c6d5-4079-8668-9e35fe80efb6", status: "processing" }    bell.html:143:7
[Modalys] [t      0.00, step           0]                                                       modalys.js:1:1470744
[Modalys] Warning: External eigenvalue solver for object BELL_1 has not responded yet!           modalys.js:1:1470744
▸ Object { partsNumber: 1, performanceParameters: […], requestId: "123456", sessionId: "9dc485e4-c6d5-4079-8668-9e35fe80efb6",  bell.html:143:7
status: "description" }
▸ Object { requestId: "123456", sessionId: "9dc485e4-c6d5-4079-8668-9e35fe80efb6", status: "ready" }         bell.html:143:7
```

3.1.6.2. Server platform

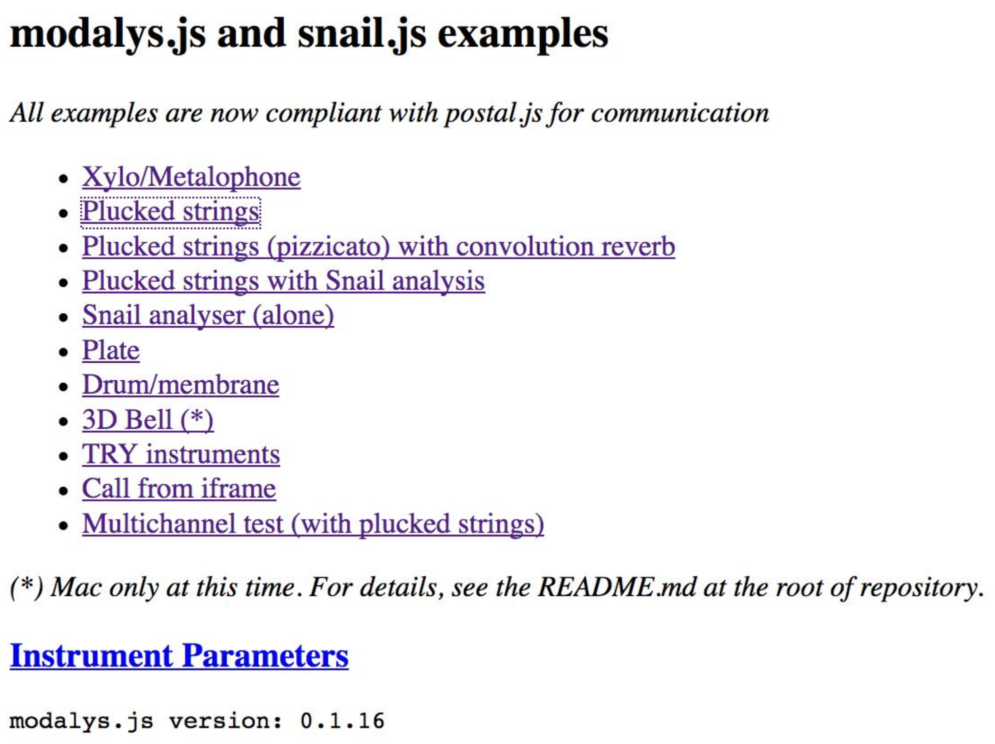
The eigenvalueproblemsolver executable is currently available for Mac OS server only ; it has been successfully tested, but only locally at Ircam. We intend to make the command line available for Linux servers shortly, so it can run on the iMuSciCA server to be tested and experimented by everyone.

3.2. Standalone modalys.js musical instrument prototypes

Independently from iMuSciCA workbench, we provide with several simple modalys.js example showing the other developers how to implement and use the physical model virtual instrument technology.
All example instrument are equipped with some basic control, emulating a "gesture" - for instance a slider for the normal distance of a mallet to a drum. In actual play situation, the gesture parameters are much more natural way.
Each page gives a summary of the available construction (physical) and performance parameters.

http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/

[7] http://www.netlib.org/lapack/
[8] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/


modalys.js and snail.js examples

All examples are now compliant with postal.js for communication

- Xylo/Metalophone
- Plucked strings
- Plucked strings (pizzicato) with convolution reverb
- Plucked strings with Snail analysis
- Snail analyser (alone)
- Plate
- Drum/membrane
- 3D Bell (*)
- TRY instruments
- Call from iframe
- Multichannel test (with plucked strings)

(*) Mac only at this time. For details, see the README.md at the root of repository.

Instrument Parameters

```
modalys.js version: 0.1.16
```

3.2.1. Plucked strings

This example presents a 4-string instrument, closed in sound to an acoustic guitar. The tuning is in fifths : low F-C-G-D. It features a “pluck” type of interaction.

http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings.html

Controls

Each string is equipped with individual controls:

Plucking a string

Press the key 1...4 to pluck the string F, C, G, D respectively (like a viola pitched one fifth lower).

Changing the pluck contact point

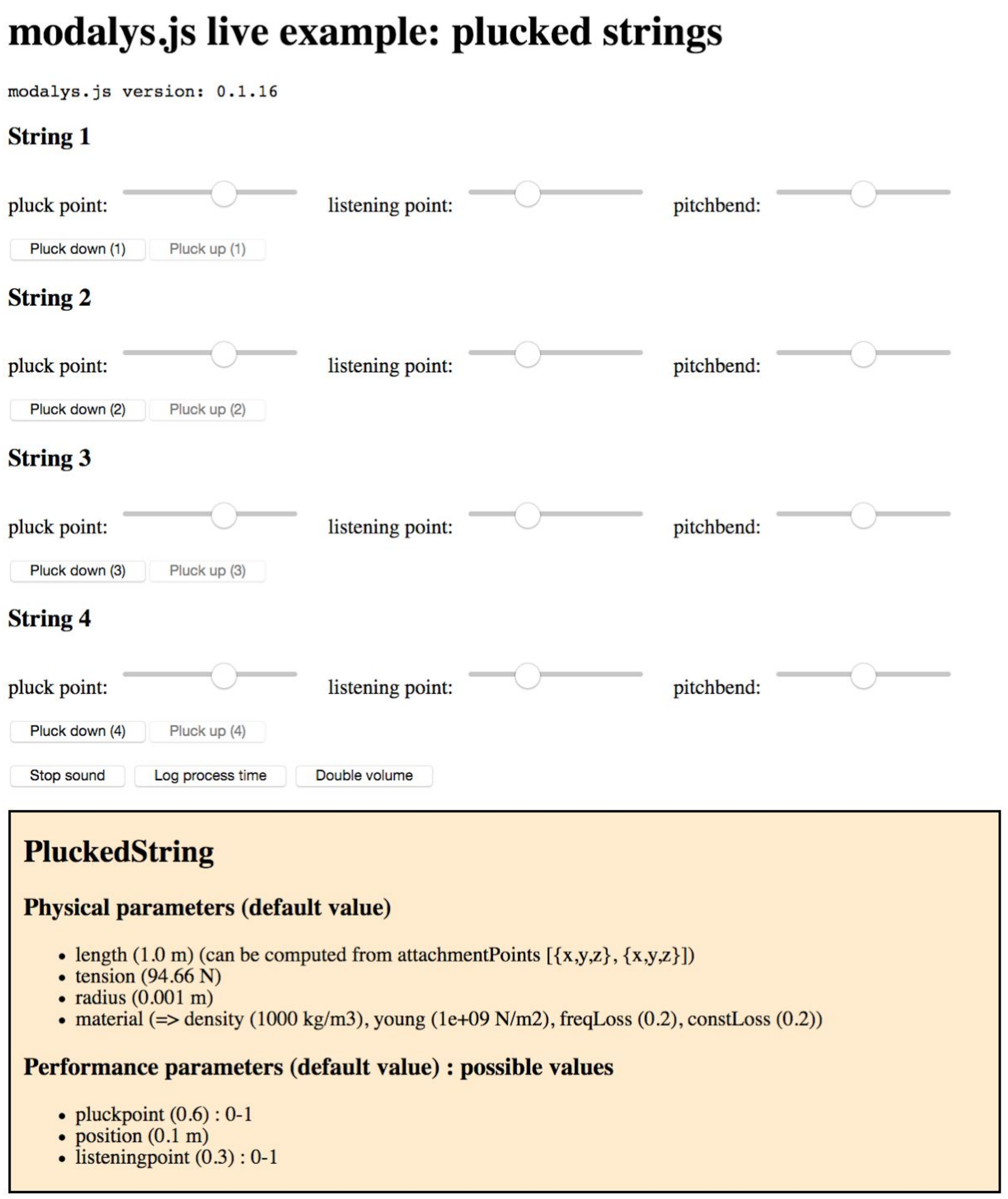
Move the corresponding slider, representing the total length of the string (a value between 0 and 1).

Changing the listening point

In a real-life string instrument, the string’s energy is propagated to the soundboard via the bridge ; the energy is then transferred to the soundboard which makes the air vibrate, and this vibration finally reaches our ears - which we call “sound”. In Modalys, it is simplified in most cases : we attach a listening point to the string directly, as we were pressing the ear against it.
This control allows to change the listening point position along the string (a value, between 0 and 1).

Altering the pitch slightly

This slider provides with a fine control over the pitch, expressed in cents. The values are in [-200...200], that is, making the pitch vary in +/- one whole tone.



PluckedString

Physical parameters (default value)

- length (1.0 m) (can be computed from attachmentPoints [{x,y,z}, {x,y,z}])
- tension (94.66 N)
- radius (0.001 m)
- material (=> density (1000 kg/m3), young (1e+09 N/m2), freqLoss (0.2), constLoss (0.2))

Performance parameters (default value) : possible values

- pluckpoint (0.6) : 0-1
- position (0.1 m)
- listeningpoint (0.3) : 0-1

3.2.2 Plucked strings with convolution reverb

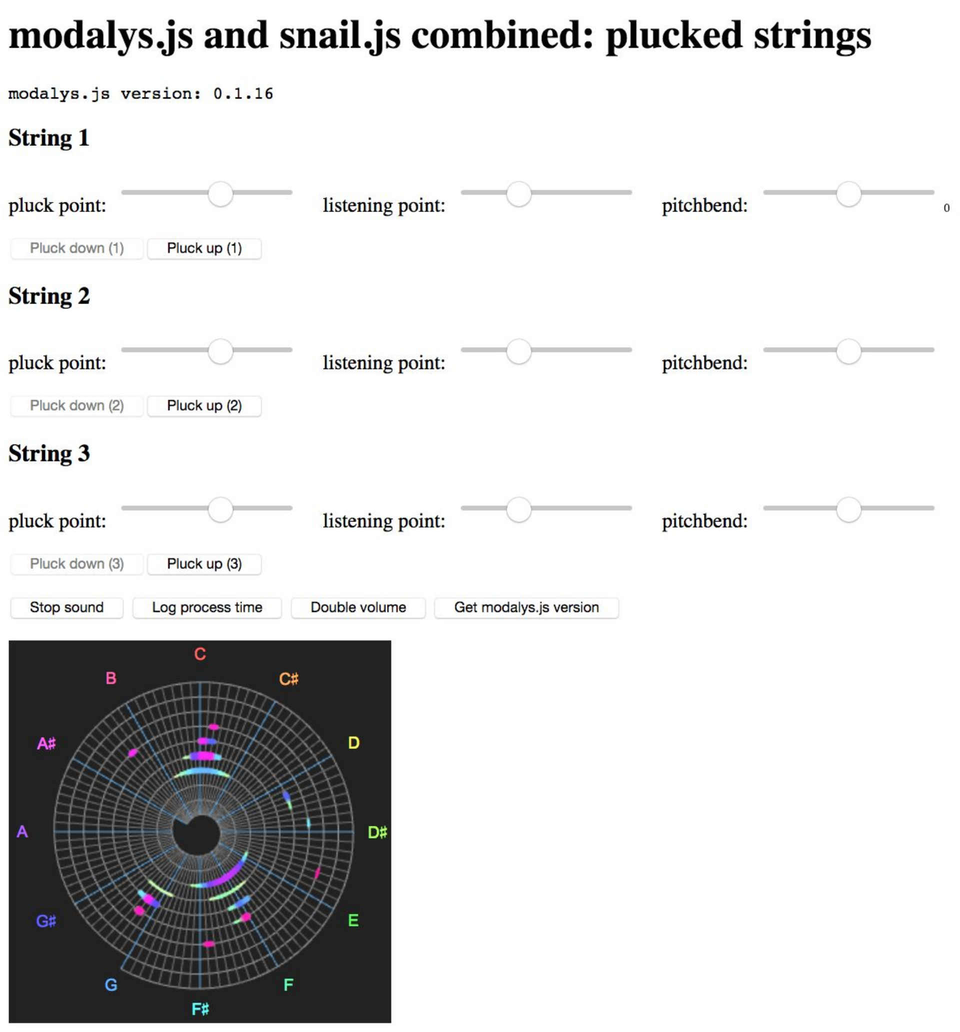
This is the same example of a 4-string instrument, but the strings have a somehow quicker decay, simulating a violin family pizzicato sound. In this example, we also experimented with web audio's built-in convolution reverb, using a real, expensive violin's impulse response. Here is how to play this instrument:

- ➔ Play with the 1,2,3 and 4 keyboard keys (same as previous example)
- ➔ Press on the button "add convolution reverb" to get the effect.
- ➔ Play again with the keys to hear the difference.

NB: web audio's convolution reverb is quite CPU-demanding, so we will probably not include it into the final workbench.

3.2.3. Plucked strings with Snail analysis

In this example, we combine both Modalys and Snail technologies. This is again the same plucked string instrument, reduced to 3 strings (tuned on low F, C and G) and equipped with a real time Snail analyzer. The instrument can be performed using the key 1,2, and 3.
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings_with_snail.html



modalys.js and snail.js combined: plucked strings

modalys.js version: 0.1.16

String 1

pluck point: listening point: pitchbend: 0

String 2

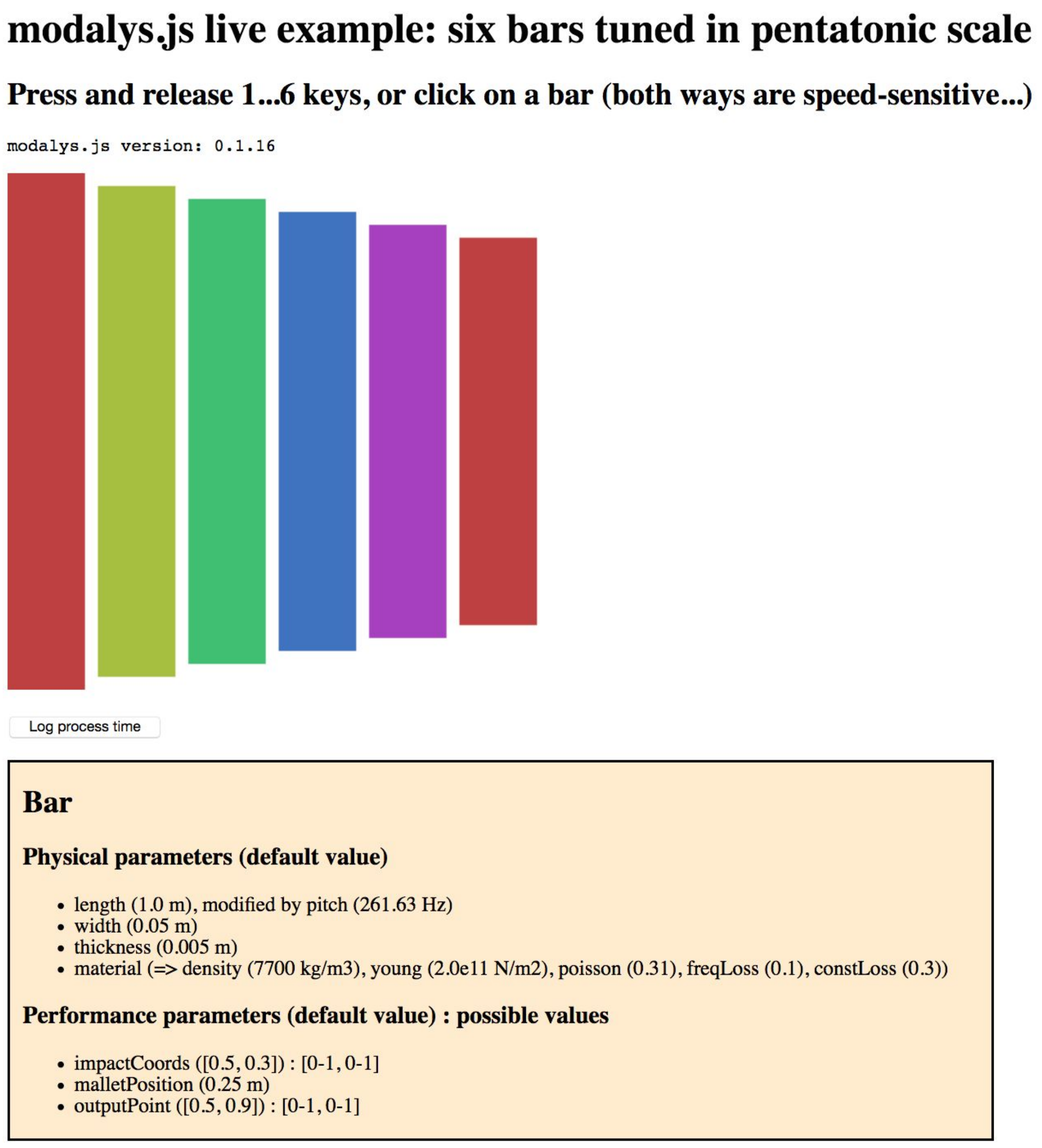
pluck point: listening point: pitchbend:

String 3

pluck point: listening point: pitchbend:

3.2.4. Xylo-/metallophone

This is pentatonic metallophone - C, D, E, G, A, C that is sensitive to the speed with which it is "hit" with either the mouse or the 1...6 keyboard keys.
It is also a good example where a fundamental object + interaction is replicated into several "parts", pitched differently. We later intend to have this kind instrument react to MIDI on-off message.
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/xylo.html



modalys.js live example: six bars tuned in pentatonic scale

Press and release 1...6 keys, or click on a bar (both ways are speed-sensitive...)

```
modalys.js version: 0.1.16
```

Bar

Physical parameters (default value)

- length (1.0 m), modified by pitch (261.63 Hz)
- width (0.05 m)
- thickness (0.005 m)
- material (=> density (7700 kg/m3), young (2.0e11 N/m2), poisson (0.31), freqLoss (0.1), constLoss (0.3))

Performance parameters (default value) : possible values

- impactCoords ([0.5, 0.3]) : [0-1, 0-1]
- malletPosition (0.25 m)
- outputPoint ([0.5, 0.9]) : [0-1, 0-1]

3.2.5. Percussion: metal plate

This example simulates a rectangular metal plate hit with a felt mallet. The interaction, called “felt connection”, is not a simple hit, but rather emulates the complex hysteresis phenomenon found in felt hammers or mallets.
The example is equipped with a real time oscilloscope displaying the waveform.
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/plate.html

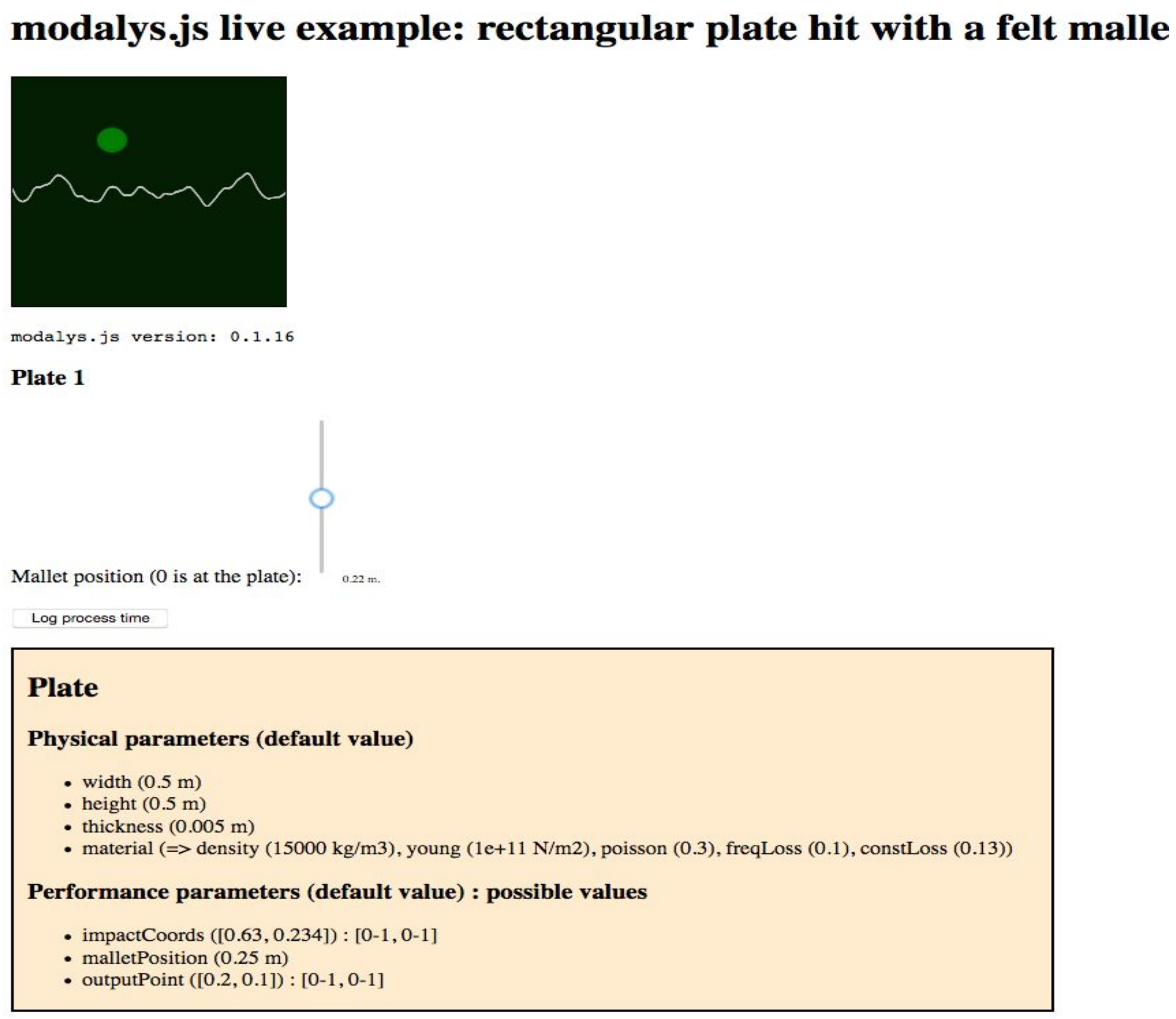
Controls

Mallet vertical position slider

This represents the vertical position (expressed in meters) of the mallet above the membrane. Move the slider to and fro (the lower the closer to the plate). If you stay around zero too long, the membrane will quickly get damped, producing a rather “ugly” sound, as on a real instrument.

Hitting point

Click on the square to change the mallet-plate 2D contact point. The sound may vary a great deal depending on this location.



modalys.js live example: rectangular plate hit with a felt malle

`modalys.js version: 0.1.16`

Plate 1

Mallet position (0 is at the plate): 0.22 m.

Plate

Physical parameters (default value)

- width (0.5 m)
- height (0.5 m)
- thickness (0.005 m)
- material (=> density (15000 kg/m3), young (1e+11 N/m2), poisson (0.3), freqLoss (0.1), constLoss (0.13))

Performance parameters (default value) : possible values

- impactCoords ([0.63, 0.234]) : [0-1, 0-1]
- malletPosition (0.25 m)
- outputPoint ([0.2, 0.1]) : [0-1, 0-1]

3.2.6. Percussion: membranophone

This example simulates a membranophone (here a bass drum) hit with a felt mallet. As with the metal plate, the interaction includes the hysteresis phenomenon.
The example is also equipped with a real time oscilloscope displaying the waveform.
http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/simpledrum.html

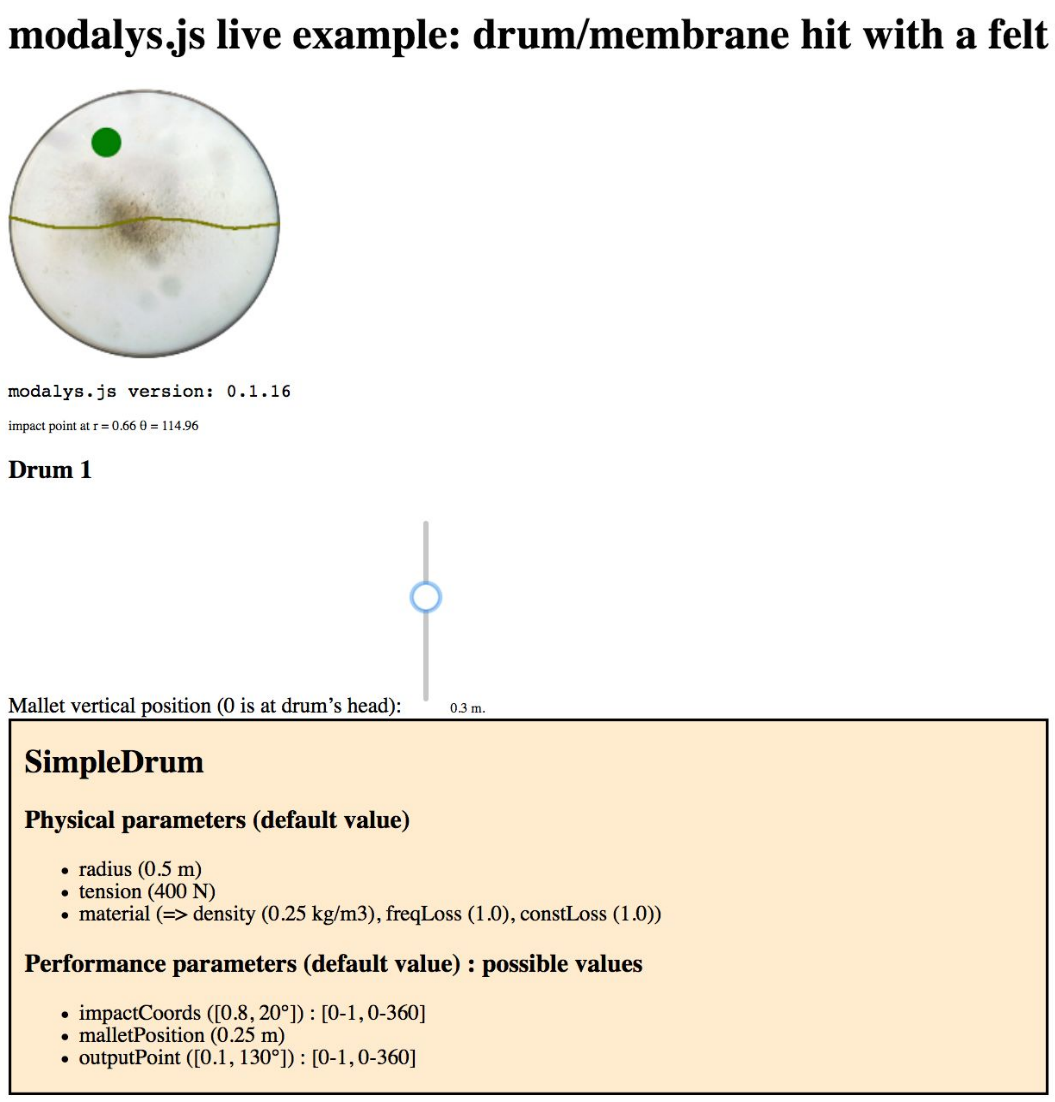
Controls

Mallet vertical position slider

This represents the vertical position (expressed in meters) of the mallet above the membrane. Move the slider to and fro (the lower the closer to the membrane). If you stay around zero too long, you'll get a somewhat ugly, damped sound.

Hitting point

Click on the drum round picture to change the mallet-membrane 2D contact point. The sound will vary in tone depending on the location.



modalys.js live example: drum/membrane hit with a felt mallet

`modalys.js version: 0.1.16`

impact point at r = 0.66 θ = 114.96

Drum 1

Mallet vertical position (0 is at drum's head): 0.3 m.

SimpleDrum

Physical parameters (default value)

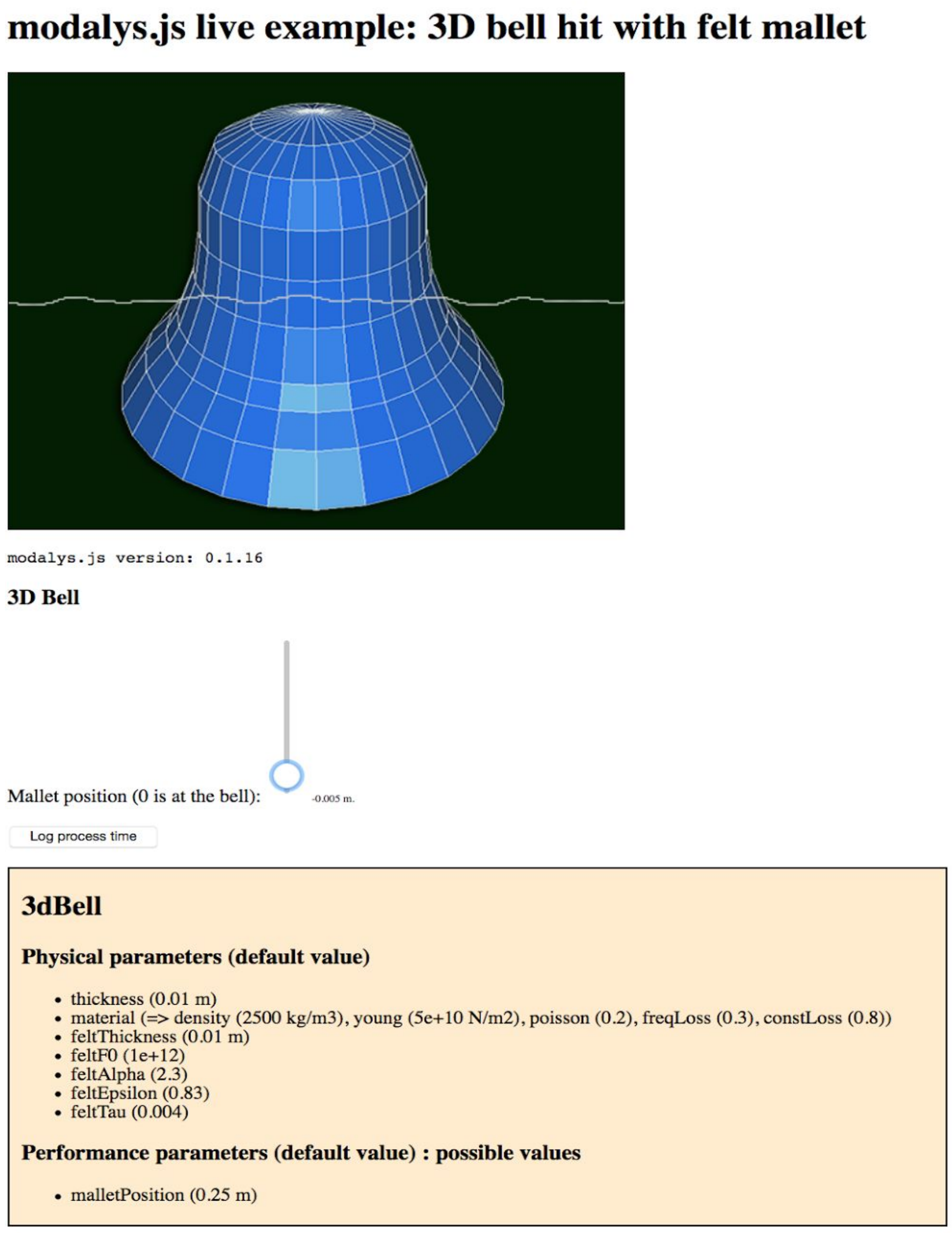
- radius (0.5 m)
- tension (400 N)
- material (=> density (0.25 kg/m3), freqLoss (1.0), constLoss (1.0))

Performance parameters (default value) : possible values

- impactCoords ([0.8, 20°]) : [0-1, 0-360]
- malletPosition (0.25 m)
- outputPoint ([0.1, 130°]) : [0-1, 0-360]

3.2.7. 3D bell

In this experimental example, we create a 3D bell out of a mesh (in .obj format). The mesh, mechanically well-defined[9], is 2D surface embedded in the 3D space, representing the neutral axis of a church bell. To instantiate the final object, the 2D surface is equipped with a thickness and some physical material properties (those of bronze).
The static mode computation increases exponentially according to the mesh's granularity, with a generalized eigenvalue problem under the hood. For that reason, this initial phase is processed server-side by the eigenvalueproblemsolver command line, the client request being asynchronous. Once the mode computation is received, the instrument is ready to be performed.



modalys.js live example: 3D bell hit with felt mallet

```
modalys.js version: 0.1.16
```

3D Bell

Mallet position (0 is at the bell): -0.005 m.

Log process time

3dBell

Physical parameters (default value)

- thickness (0.01 m)
- material (=> density (2500 kg/m3), young (5e+10 N/m2), poisson (0.2), freqLoss (0.3), constLoss (0.8))
- feltThickness (0.01 m)
- feltF0 (1e+12)
- feltAlpha (2.3)
- feltEpsilon (0.83)
- feltTau (0.004)

Performance parameters (default value) : possible values

- malletPosition (0.25 m)

[9] It is a common belief that any type of mesh could result in consistent mechanical computation. To the contrary, a mesh must follow very strict geometrical constraints in order, for instance, to create a sounding object.

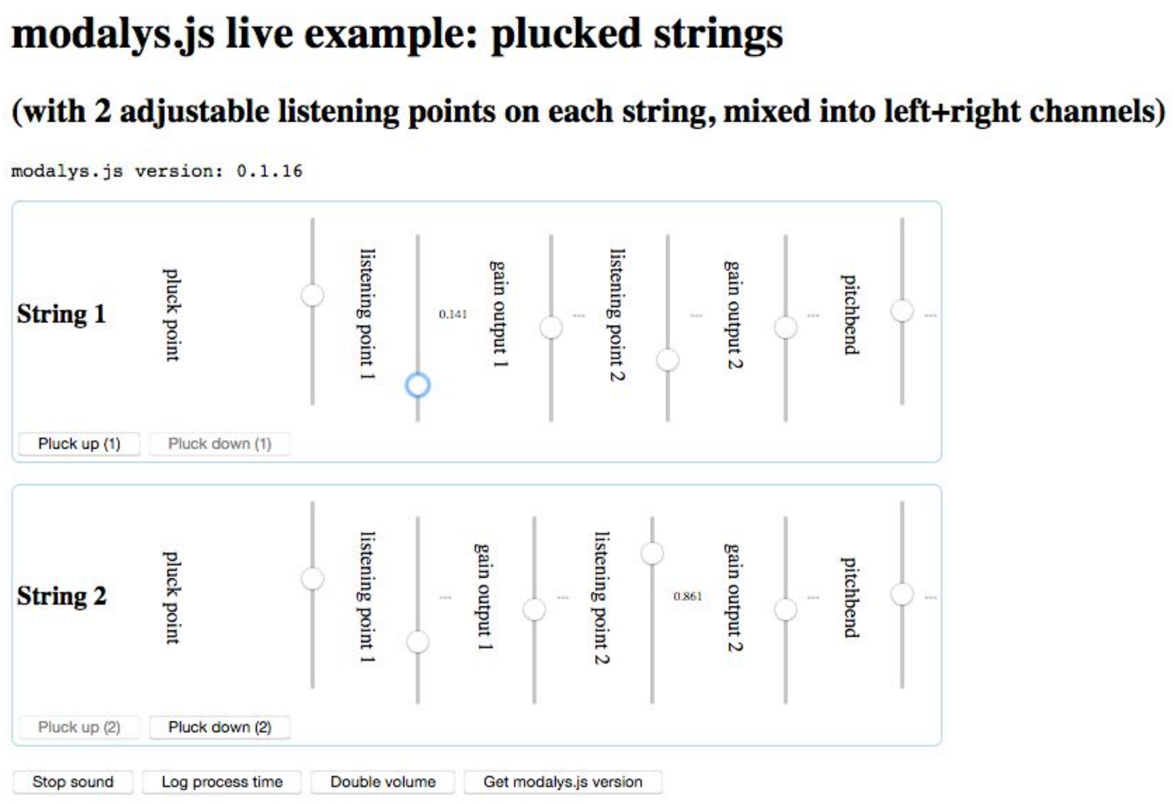
3.2.8. Multi-channel test

A Modalys object can be listened to with any number of listening points, each being routed to a specific channel (left or right). In the previous instruments, only channel 1 is used, and this last example shows a multi-channel situation.

Controls

Controls are like the first plucked string example, except that each string is equipped with 2 independent listening points, each routed to channel 1 (R) and 2 (L) respectively.

http://devtest.leopoly.com/3d-interaction-repository-V1/modalys/public/pluckedstrings2channels.html



modalys.js live example: plucked strings

(with 2 adjustable listening points on each string, mixed into left+right channels)

modalys.js version: 0.1.16

4. Performances

4.1. Current results

With technologies as CPU-demanding as Ircam or Leopoly, performances within the browser are critical in iMuSciCA. Version after version, we have been getting more and more encouraging performances for modalys.js with the following relatively modest test machine:

- Macbook pro early 2013, 2,4 GHz, 8GB RAM

- Mac OS 10.10
- Firefox Nightly 58.a1

The performance depends a great deal on the instrument's complexity (number of modes, number of parts etc.) To give an idea, the 4-plucked string instrument, with 30 modes for each string, performs at **10 ms per 1024 sample frame**, at a sample rate of 44100 kHz. So for now modalys.js takes less than half of the available CPU resource.

4.2. Strategy to improve performances

As previously stated, we intend to target WebAssembly in our compilation, and we believe we will get a CPU boost from it.
Also, the number of modes can be slightly lowered, without compromising the sound quality too much.