29.10.2007 / Lauri Savioja

The EVERTims room simulation software

1. Main principle

The EVERTims software is a beam tracing system designed for efficient detection of image sources with a moving listener. The software can handle dynamic geometry and sound sources as well, but the performance is not as good as with dynamic listeners. The system is able to handle multiple sound sources and multiple listeners at the same time.

2. User manual

The program is command-line driven, and there are no setup-files. If you need to change something, you should go into the code and make the change ☺.

2.1. Compiling

There is 'Makefile', but no 'autoconf'. The selection between OSX and Linux must be made by hand in the Makefile. After that just say 'make' (and hope for the best).

2.2. Command Line

The following options are available in the command line:

`-s <port>,` the <port> for the Reader to listen to the messages.
`-p <pattern>,` the dummy terminal print writer.
`-v <pattern/host:port>,` the <host> and <port> to write in the "VirChor" protocol
`-a <pattern/host:port>,` the <host> and <port> to write reflection data for auralization in the "Markus" protocol
`-m <materialfile>,` the name of the material file.
`-d <mindepth>`, the minimun order of reflections
`-D <maxdepth>,` the maximum order of reflections
`-g,` use the internal graphics module for drawing the result. BROKEN AT THE MOMENT. DO NOT USE!
`-f <roomfile>,` read the room geometry from the <roomfile>. DOESN'T WORK AT THE MOMENT.

In the above pattern is a regular expression and is matched against the name of the solution formed as 'source_name-listener_name'. If the pattern is omitted everything matches. Such that command line:

```
./ims –p 'listener_0/' –a waves:10234 –v
'source_[03].*listener_1/localhost:1980'
```

would make the following:

- print the solution for all the sound sources for listener_0
- send to auralization all the reflection paths for all the listeners. The messages are sent to the port 10234 of computer named ‘waves’.
- send to VirChor the reflection paths of sound sources ‘source_0’ and ‘source_3’ for listener_1. The messages are sent to port 1980 at the local computer.

2.3. Reader

The Reader (reader.cc) is listening to the given socket, and whenever there is a message the Reader reacts accordingly. The following messages are supported:

```
/face id material_id p0_x p0_y p0_z p1_x p1_y … p3_y p3_z
/facefinished
/source id 4x4-float-matrix
/listener id 4x4-float-matrix
```

All the faces have a material, and the handling of materials is described below in section “Materials”.

2.4. Writers

At this time there are two different Writer-implementations (writer.cc), and they are:

- VirchorWriter for visualizing the reflection paths with VirChor.
- MarkusWriter for auralization with the Markus’s auralization engine.

Both of them communicate via a socket connection, and their protocols are described below. Each time there is a change in the scene, all the reflection paths are updated. The reason to a change can be any of the following: listener moved, sound source moved, some surface moved, some material changed.

2.4.1. VirchorWriter

In VirChor-scene there should be a set of lines for visualizing the reflection paths. For each change all the lines are redrawn and if the number of line segments has decreased, the rest of them are marked invisible. Messages of the following types are sent to VirChor:

```
/line_on id x0 y0 z0 x1 y1 z1
/line_off id
```

2.4.2. MarkusWriter

The reflection data for each change is packed to one OSC bundle, and is as follows:

```
#bundle
time_tag 0.0
sizeof(EVERT_SOURCE_MSG)
EVERT_SOURCE_MSG

sizeof(EVERT_LISTENER_MSG)
EVERT_LISTENER_MSG

sizeof(EVERT_MSG_1)
EVERT_MSG_1

...

sizeof(EVERT_MSG_N)
EVERT_MSG_N
```

EVERT_SOURCE_MSG and EVERT_LISTENER_MSG are the following:

```
/source ,sfff source_id x y z
/listener ,sfff listener_id x y z
```

EVERT_MSG_i can be any one of the following:

```
/in ,iiffffffffffffffffff path_id order r1_x r1_y r1_z
rN_x rN_y rN_z dist abs_0 abs_1 ... abs_9

/upd ,iiffffffffffffffffff path_id order r1_x r1_y r1_z
rN_x rN_y rN_z dist abs_0 abs_1 ... abs_9

/out ,i path_id
```

In the above:

- path_id is a path identifier. It is an integer value, and is guaranteed to be the same for a reflection path while it is visible. If it gets invisible and then visible again, the path_id might change.
- order is the reflection order, for the direct sound the order == 0, and the rN_X and abs_M are irrelevant
- rN_X is the Nth order reflection point
- dist is the length of the whole reflection path
- abs_M is the absorption coefficient for the octave band M

The order of the messages in the bundle is regulated according to the following rules:

1. The `/source` and `/listener` messages are always present and precede the other messages telling to which source-receiver pair all the following messages in this bundle relate to.
2. All the `/out` messages precede all the `/in and /upd` messages

2.5. Materials

EVERTims reads the material-data from one file in the startup. The file is line-based ASCII, and its structure is as follows:

```
material_id abs0 abs1 … abs9 diff
```

where

- material_id is the name of the material, and the Reader refers to materials by these names.
- absX is the absorption coefficient for each octave band. The correspondence of band X and actual frequency is such that abs0 corresponds to Y Hz.
- diff is the frequency independent diffusion coefficient

3. Implementation

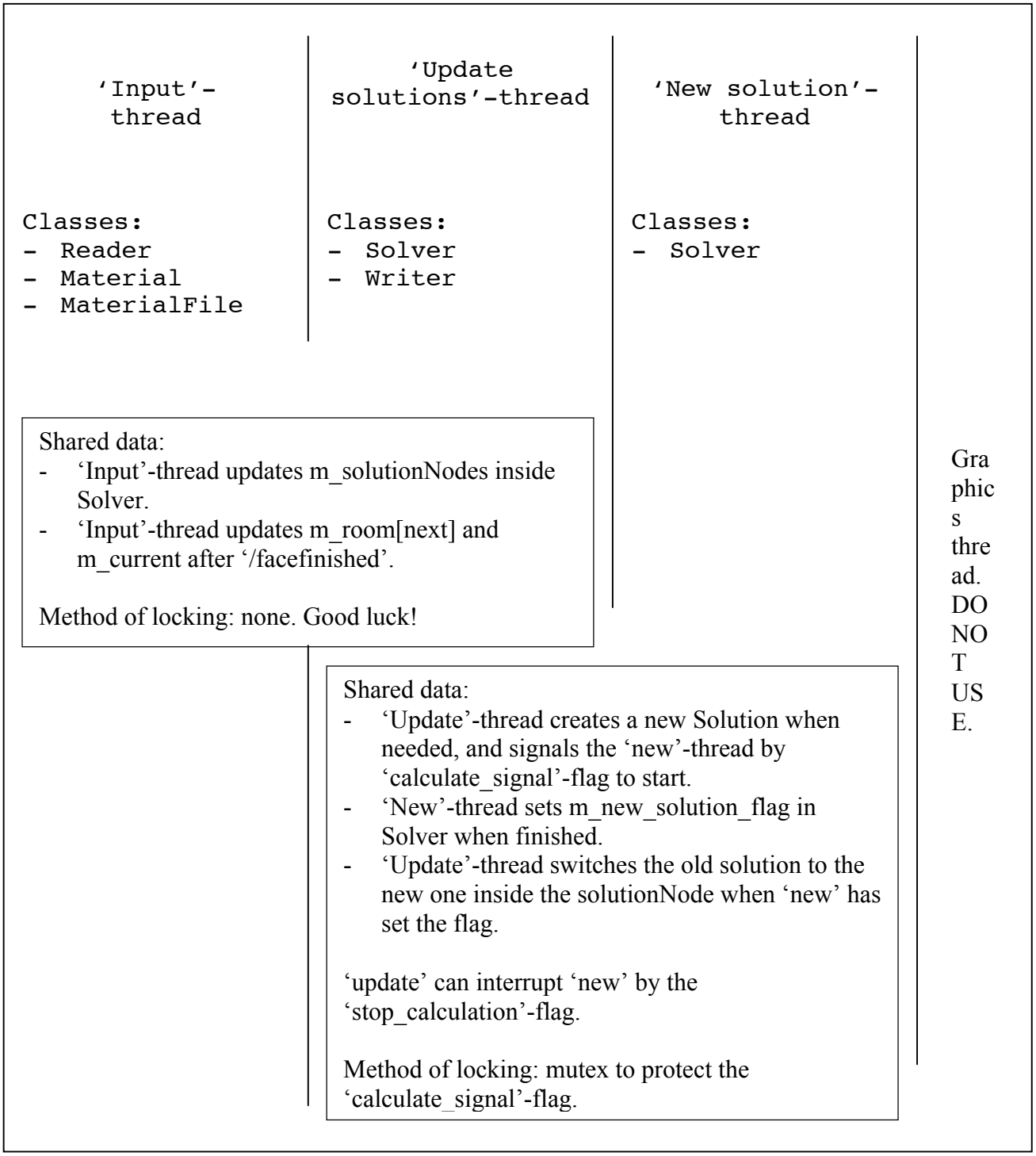
The implementation consists of two parts, the EVERT library, and the actual multithreaded simulation software utilizing the EVERT library.

3.1. EVERT-library

The core of the system is the EVERT-library. It is described in detail in a journal paper under review at the time of writing this document.

3.2. Threads

The software is run on multiple threads as illustrated below.



3.2.1. Structure solutionNode

The communication between the 'Input'- and 'Update'-threads is implemented with the struct solutionNode. The same structure is the main driver of the 'Update'-thread in general, as well. There is one solutionNode for each 'Source'-'Receiver'-pair. Each node has it's own m_solution representing the actual beam-tracing solution. The complete contents of the struct can be found in 'solver.h'.

3.2.2. ‘Input’-thread

Init:
- Read material database.
- Open input socket
- Block everything else until first `/facefinished` message received.

Main loop:
- Wait for a message
- With a new message first find the type of the message and then switch accordingly

New faces are added to the element map emap (safe operation).

After ‘/facefinished’ a new EL::Room is created from the emap, and this is updated into Solver. This should be safe, but not guaranteed. In addition ‘stop_calculation’ flag is set.

For each ‘/source’-message, it is checked if the source is a new one or was it known before. For a new source new solutionNodes are created for each listener. For a known source, the existing solutionNodes are updated.

For each ‘/listener’ message, the same applies as with the ‘source (just swap the words ‘listener’ and ‘source’ in the previous paragraph).

3.2.3. ‘Update’-thread

Main loop:

1) Check if we have got new solutionNodes and map them to the solutionMap to be taken into account in further computation.
2) Check if we have a new solution ready from the ‘new’-thread, and replace m_solution with m_new_solution in corresponding solutionNode. If the ‘new’-thread is running just skip the following tests and jump to the listener updates
3) Check if we have unprocessed geometry changes in any solutionNode. In the case ask for a new solution from the ‘new’-thread.
4) Check if we have unprocessed source changes in any solutionNode. In the case ask for a new solution from the ‘new’-thread.
5) Check if we have solutionNodes with m_depth smaller than MAX_DEPTH. In the case ask for a new solution from the ‘new’-thread.
6) Update all the solutions with listener changes, and call the corresponding writers.

3.2.4. ‘New’-thread

Main loop:

1) Wait for ‘calculate_signal’-flag to be set.
2) Compute the new solution.

3) Set the ‘m_next_solution’-flag.
4) Unset the ‘calculate_signal’-flag.

If there is ‘stop_signal’-flag detected at any time stop immediately and discard the solution under computation.