The FMB Algorithm

Pascal Baillehache
bayashipascal@gmail.com

April 23, 2020

Abstract

This paper introduces how to perform intersection detection of pair of static/dynamic cuboid/tetrahedron in 2D/3D by using the Fourier-Motzkin elimination method.
It includes the mathematical definition of the problem, its mathematical resolution with the Fourier-Motzkin elimination method, the resulting algorithm and its implementation in C, and its validation and qualification against the SAT algorithm. Results are commented and show that the FMB algorithm can be in average up to 4.8 times faster than the SAT algorithm.

Contents

1 The problem as a system of linear inequations **5**
- 1.1 Notations and definitions . . . 5
- 1.2 Static case . . . 5
- 1.3 Dynamic case . . . 8

2 Resolution of the problem by Fourier-Motzkin method **12**
- 2.1 The Fourier-Motzkin elimination method . . . 12
- 2.2 Application of the Fourier-Motzkin method to the intersection problem . . . 14
- 2.3 About the size of the system of linear inequations . . . 14

3 Algorithms of the solution **16**
- 3.1 2D static . . . 17
- 3.2 3D static . . . 23
- 3.3 2D dynamic . . . 30
- 3.4 3D dynamic . . . 37

4 Implementation of the algorithms in C **46**
- 4.1 Frames . . . 46
  - 4.1.1 Header . . . 46
  - 4.1.2 Body . . . 49
- 4.2 FMB . . . 74
  - 4.2.1 2D static . . . 74
  - 4.2.2 3D static . . . 84
  - 4.2.3 2D dynamic . . . 95
  - 4.2.4 3D dynamic . . . 106

5 Minimal example of use **118**
- 5.1 2D static . . . 118
- 5.2 3D static . . . 120
- 5.3 2D dynamic . . . 121
- 5.4 3D dynamic . . . 123

6 Unit tests **125**
- 6.1 Code . . . 125
  - 6.1.1 2D static . . . 125
  - 6.1.2 3D static . . . 138
  - 6.1.3 2D dynamic . . . 146
  - 6.1.4 3D dynamic . . . 152
- 6.2 Results . . . 158

6.2.1 2D static . . . 158
6.2.2 3D static . . . 162
6.2.3 2D dynamic . . . 165
6.2.4 3D dynamic . . . 166

7 Validation against SAT **169**
7.1 Code . . . 169
7.1.1 2D static . . . 169
7.1.2 3D static . . . 172
7.1.3 2D dynamic . . . 176
7.1.4 3D dynamic . . . 180
7.2 Results . . . 183
7.2.1 Failures . . . 183
7.2.2 2D static . . . 184
7.2.3 2D dynamic . . . 184
7.2.4 3D static . . . 184
7.2.5 3D dynamic . . . 185

8 Qualification against SAT **185**
8.1 Code . . . 185
8.1.1 2D static . . . 185
8.1.2 3D static . . . 199
8.1.3 2D dynamic . . . 214
8.1.4 3D dynamic . . . 228
8.2 Results . . . 242
8.2.1 2D static . . . 242
8.2.2 2D static (near case only) . . . 247
8.2.3 3D static . . . 252
8.2.4 3D static (near case only) . . . 257
8.2.5 2D dynamic . . . 262
8.2.6 3D dynamic . . . 267

9 Comments about the qualification results **272**

10 Conclusion **276**

11 Annex **277**
11.1 Runtime environment . . . 277
11.2 SAT implementation . . . 279
11.2.1 Header . . . 279
11.2.2 Body . . . 280

11.3 Makefile 302
11.3.1 2D static 305
11.3.2 3D static 306
11.3.3 2D dynamic 307
11.3.4 3D dynamic 308
11.3.5 Doc 308
11.4 Dynamic analysis 308
11.5 Static analysis 330

Introduction

This paper introduces the FMB (Fourier-Motzkin-Baillehache) algorithm which can be used to perform intersection detection of moving and resting parallelepipeds and triangles in 2D, and cuboids and tetrahedrons in 3D.

The detection result is returned has a boolean (intersection / no intersection), and if there is intersection, a bounding box of the intersection.

The two first sections introduce how the problem can be expressed as a system of linear inequation, and its resolution using the Fourier-Motzkin method.

The algorithm of the solution and its implementation in the C programming language are detailed in the three following sections.

The three next sections introduce the unit tests, the validation and the qualification in term of relative performance of the FMB algorithm against the SAT algorithm.

Finally, the last section contains comments about the qualification results.

All the algorithms, the code, the results and this doc are available on GitHub at:

https://github.com/BayashiPascal/FMB

1 The problem as a system of linear inequations

1.1 Notations and definitions

- $[M]_{r,c}$ is the component at column c and row r of the matrix M
- $[V]_r$ is the r-th component of the vector $\overrightarrow{V}$
- the term "Frame" is used indifferently for parallelepiped, triangle, cuboid and tetrahedron.

1.2 Static case

The two Frames are represented as a vector origin and a number of component vectors equal to the dimension D of the space where live the Frames. Each vector is of dimension equal to D.

Let's call $\mathbb{A}$ and $\mathbb{B}$ the two Frames tested for intersection. If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\mathbb{A} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{A}}.\overrightarrow{X} \end{array} \right\} \tag{1}$$

$$\mathbb{B} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \overrightarrow{O_{\mathbb{B}}} + C_{\mathbb{B}}.\overrightarrow{X} \end{array} \right\} \tag{2}$$

where $\overrightarrow{O_{\mathbb{A}}}$ is the origin of $\mathbb{A}$ and $C_{\mathbb{A}}$ is the matrix of the components of $\mathbb{A}$ (one component per column). Idem for $\overrightarrow{O_{\mathbb{B}}}$ and $C_{\mathbb{B}}$.

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\mathbb{A} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{A}}.\overrightarrow{X} \end{array} \right\} \tag{3}$$

$$\mathbb{B} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \overrightarrow{O_{\mathbb{B}}} + C_{\mathbb{B}}.\overrightarrow{X} \end{array} \right\} \tag{4}$$

I'll assume the Frames are well formed, i.e. their components matrix is invertible. It is then possible to express $\mathbb{B}$ in $\mathbb{A}$'s coordinates system, noted

as $\mathbb{B}_{\mathbb{A}}$. If $\mathbb{B}$ is a cuboid:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1}.(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}) \end{array} \right\} \tag{5}$$

If $\mathbb{B}$ is a tetrahedron:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}) \end{array} \right\} \tag{6}$$

$\mathbb{A}$ in its own coordinates system becomes, for a cuboid:

$$\mathbb{A}_{\mathbb{A}} = \left\{ \overrightarrow{X} \in [0.0, 1.0]^D \right\} \tag{7}$$

and for a tetrahedron:

$$\mathbb{A}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \end{array} \right\} \tag{8}$$

The intersection of $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{A}$'s coordinates sytem, $\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}}$,can then be expressed as follow.

If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}\right) \cap [0.0, 1.0]^D \end{array} \right\} \tag{9}$$

If $\mathbb{A}$ is a cuboid and $\mathbb{B}$ is a tetrahedron:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}\right) \cap [0.0, 1.0]^D \end{array} \right\} \tag{10}$$

If $\mathbb{A}$ is a tetrahedron and $\mathbb{B}$ is a cuboid:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}\right) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}\right)\right]_i \leq 1.0 \end{array} \right\} \tag{11}$$

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X}) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1}. \left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} \right) \right]_i \leq 1.0 \end{array} \right\} \tag{12}$$

These can in turn be expressed as systems of linear inequations as follow, given the two shortcuts $\overrightarrow{O_{\mathbb{B}_{\mathbb{A}}}} = C_{\mathbb{A}}^{-1}.(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}})$ and $C_{\mathbb{B}_{\mathbb{A}}} = C_{\mathbb{A}}^{-1}.C_{\mathbb{B}}$.

If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\left\{ \begin{array}{rcl} [X]_0 & \leq & 1.0 \\ & \ldots & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ & \ldots & \\ -[X]_{D-1} & \leq & 0.0 \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} . [X]_i & \leq & 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ & \ldots & \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} . [X]_i & \leq & 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} . [X]_i & \leq & [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ & \ldots & \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} . [X]_i & \leq & [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \end{array} \right. \tag{13}$$

If $\mathbb{A}$ is a cuboid and $\mathbb{B}$ is a tetrahedron:

$$\left\{ \begin{array}{rcl} -[X]_0 & \leq & 0.0 \\ & \ldots & \\ -[X]_{D-1} & \leq & 0.0 \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} . [X]_i & \leq & 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ & \ldots & \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} . [X]_i & \leq & 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} . [X]_i & \leq & [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ & \ldots & \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} . [X]_i & \leq & [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq & 1.0 \end{array} \right. \tag{14}$$

If $\mathbb{A}$ is a tetrahedron and $\mathbb{B}$ is a cuboid:

$$\left\{\begin{array}{rcl} [X]_0 & \leq & 1.0 \\ \ldots & & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ \ldots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{0,i}.[X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \ldots & & \\ -\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{D-1,i}.[X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ \sum_{j=0}^{D-1}\left(\left(\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{j,i}\right).[X]_i\right) & \leq & 1.0-\sum_{j=0}^{D-1}[O_{\mathbb{B}_\mathbb{A}}]_j \end{array}\right. \tag{15}$$

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\left\{\begin{array}{rcl} -[X]_0 & \leq & 0.0 \\ \ldots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{0,i}.[X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \ldots & & \\ -\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{D-1,i}.[X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ \sum_{i=0}^{D-1}[X]_i & \leq & 1.0 \\ \sum_{j=0}^{D-1}\left(\left(\sum_{i=0}^{D-1}[C_{\mathbb{B}_\mathbb{A}}]_{j,i}\right).[X]_i\right) & \leq & 1.0-\sum_{j=0}^{D-1}[O_{\mathbb{B}_\mathbb{A}}]_j \end{array}\right. \tag{16}$$

1.3 Dynamic case

If the frames $\mathbb{A}$ and $\mathbb{B}$ are moving linearly along the vectors $\overrightarrow{V_\mathbb{A}}$ and $\overrightarrow{V_\mathbb{B}}$ respectively during the interval of time $t \in [0.0, 1.0]$, the above definition of the problem is modified as follow.

If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\mathbb{A} = \left\{\begin{array}{c} \overrightarrow{X} \in [0.0,1.0]^D \\ t \in [0.0,1.0] \\ \overrightarrow{O_\mathbb{A}} + C_\mathbb{A}.\overrightarrow{X} + \overrightarrow{V_\mathbb{A}}.t \end{array}\right\} \tag{17}$$

$$\mathbb{B} = \left\{\begin{array}{c} \overrightarrow{X} \in [0.0,1.0]^D \\ t \in [0.0,1.0] \\ \overrightarrow{O_\mathbb{B}} + C_\mathbb{B}.\overrightarrow{X} + \overrightarrow{V_\mathbb{B}}.t \end{array}\right\} \tag{18}$$

where $\overrightarrow{O_{\mathbb{A}}}$ is the origin of $\mathbb{A}$ and $C_{\mathbb{A}}$ is the matrix of the components of $\mathbb{A}$ (one component per column). Idem for $\overrightarrow{O_{\mathbb{B}}}$ and $C_{\mathbb{B}}$.

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\mathbb{A} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{A}}.\overrightarrow{X} + \overrightarrow{V_{\mathbb{A}}}.t \end{array} \right\} \tag{19}$$

$$\mathbb{B} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \overrightarrow{O_{\mathbb{B}}} + C_{\mathbb{B}}.\overrightarrow{X} + \overrightarrow{V_{\mathbb{B}}}.t \end{array} \right\} \tag{20}$$

If $\mathbb{B}$ is a cuboid, $\mathbb{B}_{\mathbb{A}}$ becomes:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \end{array} \right\} \tag{21}$$

If $\mathbb{B}$ is a tetrahedron, $\mathbb{B}_{\mathbb{A}}$ becomes:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \end{array} \right\} \tag{22}$$

$\mathbb{A}$ in its own coordinates system has the same definition as in the static case. For a cuboid:

$$\mathbb{A}_{\mathbb{A}} = \left\{ \overrightarrow{X} \in [0.0, 1.0]^D \right\} \tag{23}$$

and for a tetrahedron:

$$\mathbb{A}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \end{array} \right\} \tag{24}$$

The intersection of $\mathbb{A}$ and $\mathbb{B}$ in $\mathbb{A}$'s coordinates sytem, $\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}}$, can then be expressed as follow.

If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \cap [0.0, 1.0]^D \end{array} \right\} \quad (25)$$

If $\mathbb{A}$ is a cuboid and $\mathbb{B}$ is a tetrahedron:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \cap [0.0, 1.0]^D \end{array} \right\} \quad (26)$$

If $\mathbb{A}$ is a tetrahedron and $\mathbb{B}$ is a cuboid:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right)\right]_i \leq 1.0 \end{array} \right\} \quad (27)$$

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \overrightarrow{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1}.\left(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}} + C_{\mathbb{B}}.\overrightarrow{X} + \left(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}}\right).t\right)\right]_i \leq 1.0 \end{array} \right\} \quad (28)$$

These lead to the following systems of linear inequations, given the three shortcuts $\overrightarrow{O_{\mathbb{B}_{\mathbb{A}}}} = C_{\mathbb{A}}^{-1}.(\overrightarrow{O_{\mathbb{B}}} - \overrightarrow{O_{\mathbb{A}}})$, $\overrightarrow{V_{\mathbb{B}_{\mathbb{A}}}} = C_{\mathbb{A}}^{-1}.(\overrightarrow{V_{\mathbb{B}}} - \overrightarrow{V_{\mathbb{A}}})$ and $C_{\mathbb{B}_{\mathbb{A}}} = C_{\mathbb{A}}^{-1}.C_{\mathbb{B}}$.

If $\mathbb{A}$ and $\mathbb{B}$ are two cuboids:

$$\begin{cases} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ [X]_0 & \leq & 1.0 \\ \dots & & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ [V_{\mathbb{B}_\mathbb{A}}]_0 .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ [V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ -[V_{\mathbb{B}_\mathbb{A}}]_0 .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \end{cases} \tag{29}$$

If $\mathbb{A}$ is a cuboid and $\mathbb{B}$ is a tetrahedron:

$$\begin{cases} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ [V_{\mathbb{B}_\mathbb{A}}]_0 .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ [V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ -[V_{\mathbb{B}_\mathbb{A}}]_0 .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq & 1.0 \end{cases} \tag{30}$$

If $\mathbb{A}$ is a tetrahedron and $\mathbb{B}$ is a cuboid:

$$\left\{\begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ [X]_0 & \leq & 1.0 \\ \dots & & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -[V_{\mathbb{B}_\mathbb{A}}]_0 .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ \sum_{j=0}^{D-1} \left([V_{\mathbb{B}_\mathbb{A}}]_j .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{j,i} [X]_i \right) & \leq & 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{B}_\mathbb{A}}]_j \end{array}\right. \tag{31}$$

If $\mathbb{A}$ and $\mathbb{B}$ are two tetrahedrons:

$$\left\{\begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -[V_{\mathbb{B}_\mathbb{A}}]_0 .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{0,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_\mathbb{A}}]_{D-1} .t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{D-1,i} [X]_i & \leq & [O_{\mathbb{B}_\mathbb{A}}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq & 1.0 \\ \sum_{j=0}^{D-1} \left([V_{\mathbb{B}_\mathbb{A}}]_j .t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_\mathbb{A}}]_{j,i} [X]_i \right) & \leq & 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{B}_\mathbb{A}}]_j \end{array}\right. \tag{32}$$

2 Resolution of the problem by Fourier-Motzkin method

2.1 The Fourier-Motzkin elimination method

The Fourier-Motzkin elimination method has been introduced by J.J.-B. Fourier in 1827 [1], and described in the Ph.D. thesis of T.S. Motzkin in 1936 [2]. This is a generalization of the Gaussian elimination method to linear systems of inequalities. This method consists of eliminating one variable of the system and rewrite a new system accordingly. Then the elimination operation is repeated on another variable in the new system, and so on until we obtain a trivial system with only one variable. From there, a solution

for each variable can be obtained if it exists. The variable elimination is performed as follow.
Lets write the linear system $\mathcal{I}$ of m inequalities and n variables as

$$\left\{\begin{array}{lllll} a_{11}.x_1 & +a_{12}.x_2 & +\cdots & +a_{1n}.x_n & \leq b_1 \\ a_{21}.x_1 & +a_{22}.x_2 & +\cdots & +a_{2n}.x_n & \leq b_2 \\ & & \vdots & & \\ a_{m1}.x_1 & +a_{m2}.x_2 & +\cdots & +a_{mn}.x_n & \leq b_m \end{array}\right. \tag{33}$$

with

$$\begin{array}{c} i \in 1, 2, ..., m \\ j \in 1, 2, ..., n \\ x_i \in \mathbb{R} \\ a_{ij} \in \mathbb{R} \\ b_j \in \mathbb{R} \end{array} \tag{34}$$

To eliminate the first variable x_1, lets multiply each inequality by $1.0/|a_{i1}|$ where $a_{i1} \neq 0.0$. The system becomes

$$\left\{\begin{array}{lllll} x_1 & +a'_{i2}.x_2 & +\cdots & +a'_{in}.x_n & \leq b'_i \quad (i \in \mathcal{I}_+) \\ & a_{i2}.x_2 & +\cdots & +a_{in}.x_n & \leq b_i \quad (i \in \mathcal{I}_0) \\ -x_1 & +a'_{i2}.x_2 & +\cdots & +a'_{in}.x_n & \leq b'_i \quad (i \in \mathcal{I}_-) \end{array}\right. \tag{35}$$

where

$$\begin{array}{c} \mathcal{I}_+ = \{i : a_{i1} > 0.0\} \\ \mathcal{I}_0 = \{i : a_{i1} = 0.0\} \\ \mathcal{I}_- = \{i : a_{i1} < 0.0\} \\ a'_{ij} = a_{ij}/|a_{i1}| \\ b'_i = b_i/|a_{i1}| \end{array}$$

Then $x_1, x_2, \cdots, x_n \in \mathbb{R}^n$ is a solution of $\mathcal{I}$ if and only if

$$\left\{\begin{array}{ll} \sum_{j=2}^{n}((a'_{kj} + a'_{lj}).x_j) \leq b'_k + b'_l & (k \in \mathcal{I}_+, l \in \mathcal{I}_-) \\ \sum_{j=2}^{n}(a_{ij}.x_j) \leq b_i & i \in \mathcal{I}_0 \end{array}\right. \tag{36}$$

and

$$\max_{l \in \mathcal{I}_-}(\sum_{j=2}^{n}(a'_{lj}.x_j) - b'_l) \leq x_1 \leq \min_{k \in \mathcal{I}_+}(b'_k - \sum_{j=2}^{n}(a'_{kj}.x_j)) \tag{37}$$

The same method is then applied on this new system to eliminate the second variable x_2, and so on until we reach the inequality

$$\max_{l \in \mathcal{I}''^{\cdots'}_-}(-b''^{\cdots'}_l) \leq x_n \leq \min_{k \in \mathcal{I}''^{\cdots'}_+}(b''^{\cdots'}_k) \tag{38}$$

If this inequality has no solution, then neither the system $\mathcal{I}$. If it has a solution, the minimum and maximum are the bounding values for the variable x_n. One can get a particular solution to the system $\mathcal{I}$ by choosing a value for x_n between these bounding values, which allows to set a particular value for the variable x_{n-1}, and so on back up to x_1.

2.2 Application of the Fourier-Motzkin method to the intersection problem

The Fourier-Motzkin method can be directly applied to the inequality systems of the previous section, to obtain the bounding box of the intersection, if the system has a solution. If the system has no solution, the method will eventually reach an inconsistent inequality, meaning there is no intersection between the two Frames.

One coordinate $\overrightarrow{S}$, or $(\overrightarrow{S}, t)$ in dynamic case, within the bounds obtained by the resolution of the system is expressed in the Frame $\mathbb{B}$'s coordinates system. One can get the equivalent coordinates $\overrightarrow{S'}$, or $(\overrightarrow{S'}, t)$, in the real world's coordinates system as follow:

$$\overrightarrow{S'} = \overrightarrow{O_{\mathbb{B}}} + C_{\mathbb{B}}.\overrightarrow{S} \tag{39}$$

$$(\overrightarrow{S'}, t) = \left(\overrightarrow{O_{\mathbb{B}}} + C_{\mathbb{B}}.\overrightarrow{S} + \overrightarrow{V}_{\mathbb{B}}.t, t\right) \tag{40}$$

Only one inconsistent inequality is sufficient to prove the absence of solution, and then the non intersection of the Frames. Thus, one shall check the inconsistence of each inequality as soon as possible during the resolution of the system to optimize the speed of the algorithm.

A sufficient condition for one inequality $\sum_i a_i X_i \leq Y$ to be inconsistent is, given that $\forall i, X_i \in [0.0, 1.0]$:

$$Y < \sum_{i \in I^-} a_i \tag{41}$$

where $I^- = \{i, a_i < 0.0\}$.

2.3 About the size of the system of linear inequations

During implementation in languages where the developper needs to manage memory itself the size of the systems (35) resulting from variable elimination

is necessary but cannot be forecasted. Instead, a maximum size can be calculated as follow.

Let's call n_-, n_+ and n_0, each in $[0, \mathbb{N}]$, the size of, respectively, $\mathcal{I}_-$, $\mathcal{I}_+$ and $\mathcal{I}_0$, and N the number of inequalities in the original system and N' the number inequalities in the resulting system. We have:

$$n_- + n_+ + n_0 = N \tag{42}$$

and

$$n_-.n_+ + n_0 = N' \tag{43}$$

Now let's define $K = N - n_0$, then we have:

$$n_- + n_+ = K \tag{44}$$

then,

$$n_-.n_+ = n_-.(K - n_-) \tag{45}$$

then,

$$n_-.n_+ = K.n_- - n_-^2 \tag{46}$$

The right part is a polynomial whose maximum is reached for $n_- = K/2$. Then,

$$n_-.n_+ \leq K^2/2 - K^2/4 \tag{47}$$

or,

$$n_-.n_+ \leq K^2/4 \tag{48}$$

and putting back the definition of K

$$n_-.n_+ \leq (N - n_0)^2/4 \tag{49}$$

which is also

$$n_-.n_+ \leq N^2/4 \tag{50}$$

From (43) we get,

$$N' \leq N^2/4 + n_0 \tag{51}$$

and finally,

$$N' \leq N^2/4 + N \tag{52}$$

The maximum number of inequations in the initial system is defined for each case (2D/3D, static/dynamic) in the previous section. This leads to the following maximum number of inequations:

	N	N'	N''	N'''
$2Dstatic$	8	24		
$2Ddynamic$	10	35	342	
$3Dstatic$	12	48	624	
$3Ddynamic$	14	63	1056	279840

However, these values are much higher than the ones encountered in the case of the systems correpsonding to the intersection problem. It can be noticed that n_0 can be better estimated as the inequations corresponding to the constraints $0.0 \leq x \leq 1.0$ leads to, for N', $n_0 \in \{D-1, 2(D-1)\}$ in static case and $n_0 \in \{D+1, 2D+1\}$ in dynamic case. Thus we can reduce N' to:

	N	N'
$2Dstatic$	8	14
$2Ddynamic$	10	16
$3Dstatic$	12	27
$3Ddynamic$	14	29

and so on for N'' and N'''. In pratice, the maximum number of inequations encountered during validation were:

	N	N'	N''	N'''
$2Dstatic$	8	11		
$2Ddynamic$	10	13	21	
$3Dstatic$	12	20	55	
$3Ddynamic$	14	22	57	560

3 Algorithms of the solution

In this section I introduce the algorithms of the solution of the previous section for each case (static/dynamic and 2D/3D), and the algorithms to manipulate the structure used to represent the Frames.

Algorithms are given in pseudo code, and consequently without any optimization based on properties of one given language. One can refer to the C implementation in the following section for possible optimization in this language.

Algorithms are also given independantly from each other. Code commonalization may be possible if one plans to use several cases together, but this

is dependant of the implementation and thus left to the developper responsibility.

3.1 2D static

```
ENUM FrameType
  FrameCuboid,
  FrameTetrahedron
END ENUM

STRUCT AABB2D
  // x,y
  real min[2]
  real max[2]
END STRUCT

STRUCT Frame2D
  FrameType type
  real orig[2]
  // comp[iComp][iAxis]
  real comp[2][2]
  AABB2D bdgBox
  real invComp[2][2]
END STRUCT

FUNCTION powi(base, exp)
  res = 1
  FOR i=0 TO (exp - 1)
    res = res * base
  END FOR
  RETURN res
END FUNCTION

FUNCTION Frame2DPrint(that)
  IF that.type == FrameTetrahedron
    PRINT "T"
  ELSE IF that.type == FrameCuboid
    PRINT "C"
  END IF
  PRINT "o("
  FOR i = 0 TO 1
    PRINT that.orig[i]
    IF i < 1
      PRINT ","
    END IF
  END FOR
  comp = ["x","y"]
  FOR j = 0 TO 1
    PRINT ") ", comp[j], "("
    FOR i = 0 TO 1
      PRINT that.comp[j][i]
      IF i < 1
        PRINT ","
      END IF
    END FOR
  END FOR
  PRINT ")"
END FUNCTION
```

```
FUNCTION AABB2DPrint(that)
  PRINT "minXY("
  FOR i = 0 TO 1
    PRINT that.min[i]
    IF i < 1
      PRINT ","
    END IF
  END FOR
  PRINT ")-maxXY("
  FOR i = 0 TO 1
    PRINT that.max[i]
    IF i < 1
      PRINT ","
    END IF
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION Frame2DExportBdgBox(that, bdgBox, bdgBoxProj)
  FOR i = 0 TO 1
    bdgBoxProj.max[i] = that.orig[i]
    FOR j = 0 TO 1
      bdgBoxProj.max[i] =
        bdgBoxProj.max[i] + that.comp[j][i] * bdgBox.min[j]
    END FOR
    bdgBoxProj.min[i] = bdgBoxProj.max[i]
  END FOR
  nbVertices = powi(2, 2)
  FOR iVertex = 1 TO (nbVertices - 1)
    FOR i = 0 TO 1
      IF BITWISEAND(iVertex, powi(2, i)) <> 0
        v[i] = bdgBox.max[i]
      ELSE
        v[i] = bdgBox.min[i]
      END IF
    END FOR
    FOR i = 0 TO 1
      w[i] = that.orig[i]
      FOR j = 0 TO 1
        w[i] = w[i] + that.comp[j][i] * v[j]
      END FOR
    END FOR
    FOR i = 0 TO 1
      IF bdgBoxProj.min[i] > w[i]
        bdgBoxProj.min[i] = w[i]
      END IF
      IF bdgBoxProj.max[i] < w[i]
        bdgBoxProj.max[i] = w[i]
      END IF
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame2DImportFrame(P, Q, Qp)
  FOR i = 0 TO 1
    v[i] = Q.orig[i] - P.orig[i]
  END FOR
  FOR i = 0 TO 1
    Qp.orig[i] = 0.0
    FOR j = 0 TO 1
      Qp.orig[i] = Qp.orig[i] + P.invComp[j][i] * v[j]
```

```
      Qp.comp[j][i] = 0.0
      FOR k = 0 TO 1
        Qp.comp[j][i] = Qp.comp[j][i] + P.invComp[k][i] * Q.comp[j][k]
      END FOR
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame2DUpdateInv(that)
  det = that.comp[0][0] * that.comp[1][1] -
    that.comp[1][0] * that.comp[0][1]
  that.invComp[0][0] = that.comp[1][1] / det
  that.invComp[0][1] = -that.comp[0][1] / det
  that.invComp[1][0] = -that.comp[1][0] / det
  that.invComp[1][1] = that.comp[0][0] / det
END FUNCTION

FUNCTION Frame2DCreateStatic(type, orig, comp)
  that.type = type
  FOR iAxis = 0 TO 1
    that.orig[iAxis] = orig[iAxis]
    FOR iComp = 0 TO 1
      that.comp[iComp][iAxis] = comp[iComp][iAxis]
    END FOR
  END FOR
  FOR iAxis = 0 TO 1
    min = orig[iAxis]
    max = orig[iAxis]
    FOR iComp = 0 TO 1
      IF that.type == FrameCuboid
        IF that.comp[iComp][iAxis] < 0.0
          min = min + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0
          max = max + that.comp[iComp][iAxis]
        END IF
      ELSE IF that.type == FrameTetrahedron
        IF that.comp[iComp][iAxis] < 0.0 AND
          min > orig[iAxis] + that.comp[iComp][iAxis]
          min = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0 AND
          max < orig[iAxis] + that.comp[iComp][iAxis]
          max = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
      END IF
    END FOR
    that.bdgBox.min[iAxis] = min
    that.bdgBox.max[iAxis] = max
  END FOR
  Frame2DUpdateInv(that)
  RETURN that
END FUNCTION

FUNCTION Sgn(v)
  IF 0.0 < v
    a = 1
  ELSE
    a = 0
  END IF
  IF v < 0.0
    b = 1
```

```
  ELSE
    b = 0
  END IF
  RETURN A - B
END FUNCTION

FUNCTION Neg(x)
  IF x < 0.0
    RETURN x
  ELSE
    RETURN 0.0
  END IF
END FUNCTION

FST_VAR = 0
SND_VAR = 1

FUNCTION ElimVar2D(M, Y, nbRows, nbCols, Mp, Yp, nbRemainRows)
  nbRemainRows = 0
  FOR iRow = 0 TO (nbRows - 2)
    IF M[iRow][0] <> 0.0
      FOR jRow = (iRow + 1) TO (nbRows - 1)
        IF sgn(M[iRow][0]) <> sgn(M[jRow][0]) AND
            M[jRow][0] <> 0.0
          sumNegCoeff = 0.0
          jCol = 0
          FOR iCol = 1 TO (nbCols - 1)
            Mp[nbRemainRows][jCol] =
              M[iRow][iCol] / ABS(M[iRow][0]) +
              M[jRow][iCol] / ABS(M[jRow][0])
            sumNegCoeff = sumNegCoeff + neg(Mp[nbRemainRows][jCol])
            jCol = jCol + 1
          END FOR
          Yp[nbRemainRows] =
            Y[iRow] / ABS(M[iRow][0]) +
            Y[jRow] / ABS(M[jRow][0])
          IF Yp[nbRemainRows] < sumNegCoeff
            RETURN TRUE
          END IF
          nbRemainRows = nbRemainRows + 1
        END IF
      END FOR
    END IF
  END FOR
  FOR iRow = 0 TO (nbRows - 1)
    IF M[iRow][0] == 0.0
      jCol = 0
      FOR iCol = 1 TO (nbCols - 1)
        Mp[nbRemainRows][jCol] = M[iRow][iCol]
        jCol = jCol + 1
      END FOR
      Yp[nbRemainRows] = Y[iRow]
      nbRemainRows = nbRemainRows + 1
    END IF
  END FOR
  RETURN FALSE
END FUNCTION

FUNCTION GetBoundLastVar2D(iVar, M, Y, nbRows, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR jRow = 0 TO (nbRows - 1)
```

```
    IF M[jRow][0] > 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.max[iVar] > y
        bdgBox.max[iVar] = y
      END IF
    ELSE IF M[jRow][0] < 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.min[iVar] < y
        bdgBox.min[iVar] = y
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION GetBoundVar2D(iVar, M, Y, nbRows, nbCols, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR iRow = 0 .. TO (nbRows - 1)
    IF M[iRow][0] <> 0.0
      min = -1.0 * Y[iRow]
      max = Y[iRow]
      FOR iCol = 1 .. TO (nbCols - 1)
        IF M[iRow][iCol] > 0.0
          min = min + M[iRow][iCol] * bdgBox.min[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.min[iCol + iVar]
        ELSE IF M[iRow][iCol] < 0.0
          min = min + M[iRow][iCol] * bdgBox.max[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.max[iCol + iVar]
        END IF
      END FOR
      min = min / (-1.0 * M[iRow][0])
      max = max / M[iRow][0]
      IF bdgBox.min[iVar] > min
        bdgBox.min[iVar] = min
      END IF
      IF bdgBox.max[iVar] < max
        bdgBox.max[iVar] = max
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION FMBTestIntersection2D(that, tho, bdgBox)
  Frame2DImportFrame(that, tho, thoProj)
  M[0][0] = -thoProj.comp[0][0]
  M[0][1] = -thoProj.comp[1][0]
  Y[0] = thoProj.orig[0]
  IF Y[0] < neg(M[0][0]) + neg(M[0][1])
    RETURN FALSE
  END IF
  M[1][0] = -thoProj.comp[0][1]
  M[1][1] = -thoProj.comp[1][1]
  Y[1] = thoProj.orig[1]
  IF Y[1] < neg(M[1][0]) + neg(M[1][1])
    RETURN FALSE
  END IF
  nbRows = 2
  IF that.type == FrameCuboid
    M[nbRows][0] = thoProj.comp[0][0]
    M[nbRows][1] = thoProj.comp[1][0]
    Y[nbRows] = 1.0 - thoProj.orig[0]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])
```

```
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][1]
    M[nbRows][1] = thoProj.comp[1][1]
    Y[nbRows] = 1.0 - thoProj.orig[1]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
  ELSE
    M[nbRows][0] = thoProj.comp[0][0] + thoProj.comp[0][1]
    M[nbRows][1] = thoProj.comp[1][0] + thoProj.comp[1][1]
    Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
  END IF
  IF tho.type == FrameCuboid
    M[nbRows][0] = 1.0
    M[nbRows][1] = 0.0
    Y[nbRows] = 1.0
    nbRows = nbRows + 1
    M[nbRows][0] = 0.0
    M[nbRows][1] = 1.0
    Y[nbRows] = 1.0
    nbRows = nbRows + 1
  ELSE
    M[nbRows][0] = 1.0
    M[nbRows][1] = 1.0
    Y[nbRows] = 1.0
    nbRows = nbRows + 1
  END IF
  M[nbRows][0] = -1.0
  M[nbRows][1] = 0.0
  Y[nbRows] = 0.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = -1.0
  Y[nbRows] = 0.0
  nbRows = nbRows + 1
  inconsistency = ElimVar2D(M, Y, nbRows, 2, Mp, Yp, nbRowsP)
  IF inconsistency == TRUE
    RETURN FALSE
  END IF
  GetBoundLastVar2D(SND_VAR, Mp, Yp, nbRowsP, bdgBoxLocal)
  IF bdgBoxLocal.min[SND_VAR] >= bdgBoxLocal.max[SND_VAR]
    RETURN FALSE
  ELSE
    GetBoundVar2D(FST_VAR, M, Y, nbRows, 2, bdgBoxLocal)
    bdgBox = bdgBoxLocal
  END IF
  RETURN TRUE
END FUNCTION

origP2D = [0.0, 0.0]
compP2D = [
  [1.0, 0.0],
  [0.0, 1.0]]
P2D = Frame2DCreateStatic(FrameCuboid, origP2D, compP2D)
origQ2D = [0.0, 0.0]
```

```
compQ2D = [
  [1.0, 0.0],
  [0.0, 1.0]]
Q2D = Frame2DCreateStatic(FrameCuboid, origQ2D, compQ2D)
isIntersecting2D = FMBTestIntersection2D(P2D, Q2D, bdgBox2DLocal)
IF isIntersecting2D == TRUE
  PRINT "Intersection detected."
  Frame2DExportBdgBox(Q2D, bdgBox2DLocal, bdgBox2D);
  AABB2DPrint(bdgBox2D)
ELSE
  PRINT "No intersection."
END IF
```

3.2 3D static

```
ENUM FrameType
  FrameCuboid,
  FrameTetrahedron
END ENUM

STRUCT AABB3D
  // x,y,z
  real min[3]
  real max[3]
END STRUCT

STRUCT Frame3D
  FrameType type
  real orig[3]
  // comp[iComp][iAxis]
  real comp[3][3]
  AABB3D bdgBox
  real invComp[3][3]
END STRUCT

FUNCTION powi(base, exp)
  res = 1
  FOR i=0 TO (exp - 1)
    res = res * base
  END FOR
  RETURN res
END FUNCTION

FUNCTION Frame3DPrint(that)
  IF that.type == FrameTetrahedron
    PRINT "T"
  ELSE IF that.type == FrameCuboid
    PRINT "C"
  END IF
  PRINT "o("
  FOR i = 0 TO 2
    PRINT that.orig[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  comp = ["x","y","z"]
  FOR j = 0 TO 2
    PRINT ") ", comp[j], "("
    FOR i = 0 TO 2
      PRINT that.comp[j][i]
```

```
      IF i < 2
        PRINT ","
      END IF
    END FOR
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION AABB3DPrint(that)
  PRINT "minXYZ("
  FOR i = 0 TO 2
    PRINT that.min[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  PRINT ")-maxXYZ("
  FOR i = 0 TO 2
    PRINT that.max[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION Frame3DExportBdgBox(that, bdgBox, bdgBoxProj)
  FOR i = 0 TO 2
    bdgBoxProj.max[i] = that.orig[i]
    FOR j = 0 TO 2
      bdgBoxProj.max[i] =
        bdgBoxProj.max[i] + that.comp[j][i] * bdgBox.min[j]
    END FOR
    bdgBoxProj.min[i] = bdgBoxProj.max[i]
  END FOR
  nbVertices = powi(2, 3)
  FOR iVertex = 1 TO (nbVertices - 1)
    FOR i = 0 TO 2
      IF BITWISEAND(iVertex, powi(2, i)) <> 0
        v[i] = bdgBox.max[i]
      ELSE
        v[i] = bdgBox.min[i]
      END IF
    END FOR
    FOR i = 0 TO 2
      w[i] = that.orig[i]
      FOR j = 0 TO 2
        w[i] = w[i] + that.comp[j][i] * v[j]
      END FOR
    END FOR
    FOR i = 0 TO 2
      IF bdgBoxProj.min[i] > w[i]
        bdgBoxProj.min[i] = w[i]
      END IF
      IF bdgBoxProj.max[i] < w[i]
        bdgBoxProj.max[i] = w[i]
      END IF
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame3DImPortFrame(P, Q, Qp)
```

```
  FOR i = 0 TO 2
    v[i] = Q.orig[i] - P.orig[i]
  END FOR
  FOR i = 0 TO 2
    Qp.orig[i] = 0.0
    FOR j = 0 TO 2
      Qp.orig[i] = Qp.orig[i] + P.invComp[j][i] * v[j]
      Qp.comp[j][i] = 0.0
      FOR k = 0 TO 2
        Qp.comp[j][i] = Qp.comp[j][i] + P.invComp[k][i] * Q.comp[j][k]
      END FOR
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame3DUpdateInv(that)
  det =
    that.comp[0][0] * (that.comp[1][1] * that.comp[2][2] -
    that.comp[1][2] * that.comp[2][1]) -
    that.comp[1][0] * (that.comp[0][1] * that.comp[2][2] -
    that.comp[0][2] * that.comp[2][1]) +
    that.comp[2][0] * (that.comp[0][1] * that.comp[1][2] -
    that.comp[0][2] * that.comp[1][1])
  that.invComp[0][0] = (that.comp[1][1] * that.comp[2][2] -
    that.comp[2][1] * that.comp[1][2]) / det
  that.invComp[0][1] = (that.comp[2][1] * that.comp[0][2] -
    that.comp[2][2] * that.comp[0][1]) / det
  that.invComp[0][2] = (that.comp[0][1] * that.comp[1][2] -
    that.comp[0][2] * that.comp[1][1]) / det
  that.invComp[1][0] = (that.comp[2][0] * that.comp[1][2] -
    that.comp[2][2] * that.comp[1][0]) / det
  that.invComp[1][1] = (that.comp[0][0] * that.comp[2][2] -
    that.comp[2][0] * that.comp[0][2]) / det
  that.invComp[1][2] = (that.comp[0][2] * that.comp[1][0] -
    that.comp[1][2] * that.comp[0][0]) / det
  that.invComp[2][0] = (that.comp[1][0] * that.comp[2][1] -
    that.comp[2][0] * that.comp[1][1]) / det
  that.invComp[2][1] = (that.comp[0][1] * that.comp[2][0] -
    that.comp[2][1] * that.comp[0][0]) / det
  that.invComp[2][2] = (that.comp[0][0] * that.comp[1][1] -
    that.comp[1][0] * that.comp[0][1]) / det
END FUNCTION

FUNCTION Frame3DCreateStatic(type, orig, comp)
  that.type = type
  FOR iAxis = 0 TO 2
    that.orig[iAxis] = orig[iAxis]
    FOR iComp = 0 TO 2
      that.comp[iComp][iAxis] = comp[iComp][iAxis]
    END FOR
  END FOR
  FOR iAxis = 0 TO 2
    min = orig[iAxis]
    max = orig[iAxis]
    FOR iComp = 0 TO 2
      IF that.type == FrameCuboid
        IF that.comp[iComp][iAxis] < 0.0
          min = min + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0
          max = max + that.comp[iComp][iAxis]
        END IF
```

```
      ELSE IF that.type == FrameTetrahedron
        IF that.comp[iComp][iAxis] < 0.0 AND
          min > orig[iAxis] + that.comp[iComp][iAxis]
          min = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0 AND
          max < orig[iAxis] + that.comp[iComp][iAxis]
          max = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
      END IF
    END FOR
    that.bdgBox.min[iAxis] = min
    that.bdgBox.max[iAxis] = max
  END FOR
  Frame3DUpdateInv(that)
  RETURN that
END FUNCTION

FUNCTION Sgn(v)
  IF 0.0 < v
    a = 1
  ELSE
    a = 0
  END IF
  IF v < 0.0
    b = 1
  ELSE
    b = 0
  END IF
  RETURN A - B
END FUNCTION

FUNCTION Neg(x)
  IF x < 0.0
    RETURN x
  ELSE
    RETURN 0.0
  END IF
END FUNCTION

FST_VAR = 0
SND_VAR = 1
THD_VAR = 2

FUNCTION ElimVar3D(M, Y, nbRows, nbCols, Mp, Yp, nbRemainRows)
  nbRemainRows = 0
  FOR iRow = 0 TO (nbRows - 2)
    IF M[iRow][0] <> 0.0
      FOR jRow = (iRow + 1) TO (nbRows - 1)
        IF sgn(M[iRow][0]) <> sgn(M[jRow][0]) AND
            M[jRow][0] <> 0.0
          sumNegCoeff = 0.0
          jCol = 0
          FOR iCol = 1 TO (nbCols - 1)
            Mp[nbRemainRows][jCol] =
              M[iRow][iCol] / ABS(M[iRow][0]) +
              M[jRow][iCol] / ABS(M[jRow][0])
            sumNegCoeff = sumNegCoeff + neg(Mp[nbRemainRows][jCol])
            jCol = jCol + 1
          END FOR
          Yp[nbRemainRows] =
            Y[iRow] / ABS(M[iRow][0]) +
```

```
            Y[jRow] / ABS(M[jRow][0])
          IF Yp[nbRemainRows] < sumNegCoeff
            RETURN TRUE
          END IF
          nbRemainRows = nbRemainRows + 1
        END IF
      END FOR
    END IF
  END FOR
  FOR iRow = 0 TO (nbRows - 1)
    IF M[iRow][0] == 0.0
      jCol = 0
      FOR iCol = 1 TO (nbCols - 1)
        Mp[nbRemainRows][jCol] = M[iRow][iCol]
        jCol = jCol + 1
      END FOR
      Yp[nbRemainRows] = Y[iRow]
      nbRemainRows = nbRemainRows + 1
    END IF
  END FOR
  RETURN FALSE
END FUNCTION

FUNCTION GetBoundLastVar3D(iVar, M, Y, nbRows, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR jRow = 0 TO (nbRows - 1)
    IF M[jRow][0] > 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.max[iVar] > y
        bdgBox.max[iVar] = y
      END IF
    ELSE IF M[jRow][0] < 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.min[iVar] < y
        bdgBox.min[iVar] = y
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION GetBoundVar3D(iVar, M, Y, nbRows, nbCols, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR iRow = 0 .. TO (nbRows - 1)
    IF M[iRow][0] <> 0.0
      min = -1.0 * Y[iRow]
      max = Y[iRow]
      FOR iCol = 1 .. TO (nbCols - 1)
        IF M[iRow][iCol] > 0.0
          min = min + M[iRow][iCol] * bdgBox.min[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.min[iCol + iVar]
        ELSE IF M[iRow][iCol] < 0.0
          min = min + M[iRow][iCol] * bdgBox.max[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.max[iCol + iVar]
        END IF
      END FOR
      min = min / (-1.0 * M[iRow][0])
      max = max / M[iRow][0]
      IF bdgBox.min[iVar] > min
        bdgBox.min[iVar] = min
      END IF
```

```
      IF bdgBox.max[iVar] < max
        bdgBox.max[iVar] = max
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION FMBTestIntersection3D(that, tho, bdgBox)
  Frame3DImportFrame(that, tho, thoProj)
  M[0][0] = -thoProj.comp[0][0]
  M[0][1] = -thoProj.comp[1][0]
  M[0][2] = -thoProj.comp[2][0]
  Y[0] = thoProj.orig[0]
  IF Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2])
    RETURN FALSE
  END IF
  M[1][0] = -thoProj.comp[0][1]
  M[1][1] = -thoProj.comp[1][1]
  M[1][2] = -thoProj.comp[2][1]
  Y[1] = thoProj.orig[1]
  IF Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2])
    RETURN FALSE
  END IF
  M[2][0] = -thoProj.comp[0][2]
  M[2][1] = -thoProj.comp[1][2]
  M[2][2] = -thoProj.comp[2][2]
  Y[2] = thoProj.orig[2]
  IF Y[2] < neg(M[2][0]) + neg(M[2][1]) + neg(M[2][2])
    RETURN FALSE
  END IF
  nbRows = 3
  IF that.type == FrameCuboid
    M[nbRows][0] = thoProj.comp[0][0]
    M[nbRows][1] = thoProj.comp[1][0]
    M[nbRows][2] = thoProj.comp[2][0]
    Y[nbRows] = 1.0 - thoProj.orig[0]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][1]
    M[nbRows][1] = thoProj.comp[1][1]
    M[nbRows][2] = thoProj.comp[2][1]
    Y[nbRows] = 1.0 - thoProj.orig[1]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][2]
    M[nbRows][1] = thoProj.comp[1][2]
    M[nbRows][2] = thoProj.comp[2][2]
    Y[nbRows] = 1.0 - thoProj.orig[2]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
  ELSE
    M[nbRows][0] =
      thoProj.comp[0][0] + thoProj.comp[0][1] + thoProj.comp[0][2]
```

```
  M[nbRows][1] =
    thoProj.comp[1][0] + thoProj.comp[1][1] + thoProj.comp[1][2]
  M[nbRows][2] =
    thoProj.comp[2][0] + thoProj.comp[2][1] + thoProj.comp[2][2]
  Y[nbRows] =
    1.0 - thoProj.orig[0] - thoProj.orig[1] - thoProj.orig[2]
  IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                 neg(M[nbRows][2])
    RETURN FALSE
  END IF
  nbRows = nbRows + 1
END IF
IF tho.type == FrameCuboid
  M[nbRows][0] = 1.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 1.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
ELSE
  M[nbRows][0] = 1.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 1.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
END IF
M[nbRows][0] = -1.0
M[nbRows][1] = 0.0
M[nbRows][2] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = -1.0
M[nbRows][2] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = 0.0
M[nbRows][2] = -1.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
inconsistency =
  ElimVar3D(M, Y, nbRows, 3, Mp, Yp, nbRowsP)
IF inconsistency == TRUE
  RETURN FALSE
END IF
inconsistency =
  ElimVar3D(Mp, Yp, nbRowsP, 2, Mpp, Ypp, nbRowsPP)
IF inconsistency == TRUE
  RETURN FALSE
END IF
GetBoundLastVar3D(THD_VAR, Mpp, Ypp, nbRowsPP, bdgBoxLocal)
IF bdgBoxLocal.min[THD_VAR] >= bdgBoxLocal.max[THD_VAR]
```

```
    RETURN FALSE
  ELSE
    GetBoundVar3D(SND_VAR, Mp, Yp, nbRowsP, 2, bdgBoxLocal)
    GetBoundVar3D(FST_VAR, M, Y, nbRows, 3, bdgBoxLocal)
    bdgBox = bdgBoxLocal
  END IF
  RETURN TRUE
END FUNCTION

origP3D = [0.0, 0.0, 0.0]
compP3D = [
  [1.0, 0.0, 0.0],
  [0.0, 1.0, 0.0],
  [0.0, 0.0, 1.0]]
P3D = Frame3DCreateStatic(FrameTetrahedron, origP3D, compP3D)
origQ3D = [0.0, 0.0, 0.0]
compQ3D = [
  [1.0, 0.0, 0.0],
  [0.0, 1.0, 0.0],
  [0.0, 0.0, 1.0]]
Q3D = Frame3DCreateStatic(FrameTetrahedron, origQ3D, compQ3D)
isIntersecting3D = FMBTestIntersection3D(P3D, Q3D, bdgBox3DLocal)
IF isIntersecting3D == TRUE
  PRINT "Intersection detected."
  Frame3DExportBdgBox(Q3D, bdgBox3DLocal, bdgBox3D)
  AABB3DPrint(bdgBox3D)
ELSE
  PRINT "No intersection."
END IF
```

3.3 2D dynamic

```
ENUM FrameType
  FrameCuboid,
  FrameTetrahedron
END ENUM

STRUCT AABB2DTime
  // x,y,t
  real min[3]
  real max[3]
END STRUCT

STRUCT Frame2DTime
  FrameType type
  real orig[2]
  // comp[iComp][iAxis]
  real comp[2][2]
  AABB2DTime bdgBox
  real invComp[2][2]
  real speed[2]
END STRUCT

FUNCTION powi(base, exp)
  res = 1
  FOR i=0 TO (exp - 1)
    res = res * base
  END FOR
  RETURN res
END FUNCTION
```

```
FUNCTION Frame2DTimePrint(that)
  IF that.type == FrameTetrahedron
    PRINT "T"
  ELSE IF that.type == FrameCuboid
    PRINT "C"
  END IF
  PRINT "o("
  FOR i = 0 TO 1
    PRINT that.orig[i]
    IF i < 1
      PRINT ","
    END IF
  END FOR
  PRINT ") s("
  FOR i = 0 TO 1
    PRINT that.speed[i]
    IF i < 1
      PRINT ","
    END IF
  END FOR
  comp = ["x", "y"]
  FOR j = 0 TO 1
    PRINT ") ", comp[j], "("
    FOR i = 0 TO 1
      PRINT that.comp[j][i]
      IF i < 1
        PRINT ","
      END IF
    END FOR
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION AABB2DTimePrint(that)
  PRINT "minXYT("
  FOR i = 0 TO 2
    PRINT that.min[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  PRINT ")-maxXYT("
  FOR i = 0 TO 2
    PRINT that.max[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION Frame2DTimeExportBdgBox(that, bdgBox, bdgBoxProj)
  bdgBoxProj.min[2] = bdgBox.min[2]
  bdgBoxProj.max[2] = bdgBox.max[2]
  FOR i = 0 TO 1
    bdgBoxProj.max[i] = that.orig[i] + that.speed[i] * bdgBox.min[2]
    FOR j = 0 TO 1
      bdgBoxProj.max[i] =
        bdgBoxProj.max[i] + that.comp[j][i] * bdgBox.min[j]
    END FOR
    bdgBoxProj.min[i] = bdgBoxProj.max[i]
  END FOR
```

```
  nbVertices = powi(2, 2)
  FOR iVertex = 1 TO (nbVertices - 1)
    FOR i = 0 TO 1
      IF BITWISEAND(iVertex, powi(2, i)) <> 0
        v[i] = bdgBox.max[i]
      ELSE
        v[i] = bdgBox.min[i]
      END IF
    END FOR
    FOR i = 0 TO 1
      w[i] = that.orig[i]
      FOR j = 0 TO 1
        w[i] = w[i] + that.comp[j][i] * v[j]
      END FOR
    END FOR
    FOR i = 0 TO 1
      IF bdgBoxProj.min[i] > w[i] + that.speed[i] * bdgBox.min[2]
        bdgBoxProj.min[i] = w[i] + that.speed[i] * bdgBox.min[2]
      END IF
      IF bdgBoxProj.min[i] > w[i] + that.speed[i] * bdgBox.max[2]
        bdgBoxProj.min[i] = w[i] + that.speed[i] * bdgBox.max[2]
      END IF
      IF bdgBoxProj.max[i] < w[i] + that.speed[i] * bdgBox.min[2]
        bdgBoxProj.max[i] = w[i] + that.speed[i] * bdgBox.min[2]
      END IF
      IF bdgBoxProj.max[i] < w[i] + that.speed[i] * bdgBox.max[2]
        bdgBoxProj.max[i] = w[i] + that.speed[i] * bdgBox.max[2]
      END IF
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame2DTimeImPortFrame(P, Q, Qp)
  FOR i = 0 TO 1
    v[i] = Q.orig[i] - P.orig[i]
    s[i] = Q.speed[i] - P.speed[i]
  END FOR
  FOR i = 0 TO 1
    Qp.orig[i] = 0.0
    Qp.speed[i] = 0.0
    FOR j = 0 TO 1
      Qp.orig[i] = Qp.orig[i] + P.invComp[j][i] * v[j]
      Qp.speed[i] = Qp.speed[i] + P.invComp[j][i] * s[j]
      Qp.comp[j][i] = 0.0
      FOR k = 0 TO 1
        Qp.comp[j][i] = Qp.comp[j][i] + P.invComp[k][i] * Q.comp[j][k]
      END FOR
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame2DTimeUpdateInv(that)
  det = that.comp[0][0] * that.comp[1][1] -
    that.comp[1][0] * that.comp[0][1]
  that.invComp[0][0] = that.comp[1][1] / det
  that.invComp[0][1] = -that.comp[0][1] / det
  that.invComp[1][0] = -that.comp[1][0] / det
  that.invComp[1][1] = that.comp[0][0] / det
END FUNCTION

FUNCTION Frame2DTimeCreateStatic(type, orig, comp)
  that.type = type
```

```
  FOR iAxis = 0 TO 1
    that.orig[iAxis] = orig[iAxis]
    that.speed[iAxis] = speed[iAxis]
    FOR iComp = 0 TO 1
      that.comp[iComp][iAxis] = comp[iComp][iAxis]
    END FOR
  END FOR
  FOR iAxis = 0 TO 1
    min = orig[iAxis]
    max = orig[iAxis]
    FOR iComp = 0 TO 1
      IF that.type == FrameCuboid
        IF that.comp[iComp][iAxis] < 0.0
          min = min + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0
          max = max + that.comp[iComp][iAxis]
        END IF
      ELSE IF that.type == FrameTetrahedron
        IF that.comp[iComp][iAxis] < 0.0 AND
          min > orig[iAxis] + that.comp[iComp][iAxis]
          min = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0 AND
          max < orig[iAxis] + that.comp[iComp][iAxis]
          max = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
      END IF
    END FOR
    IF that.speed[iAxis] < 0.0
      min = min + that.speed[iAxis]
    END IF
    IF that.speed[iAxis] > 0.0
      max = max + that.speed[iAxis]
    END IF
    that.bdgBox.min[iAxis] = min
    that.bdgBox.max[iAxis] = max
  END FOR
  that.bdgBox.min[2] = 0.0
  that.bdgBox.max[2] = 1.0
  Frame2DTimeUpdateInv(that)
  RETURN that
END FUNCTION

FUNCTION Sgn(v)
  IF 0.0 < v
    a = 1
  ELSE
    a = 0
  END IF
  IF v < 0.0
    b = 1
  ELSE
    b = 0
  END IF
  RETURN A - B
END FUNCTION

FUNCTION Neg(x)
  IF x < 0.0
    RETURN x
  ELSE
```

```
    RETURN 0.0
  END IF
END FUNCTION

FST_VAR = 0
SND_VAR = 1
THD_VAR = 2

FUNCTION ElimVar2DTime(M, Y, nbRows, nbCols, Mp, Yp, nbRemainRows)
  nbRemainRows = 0
  FOR iRow = 0 TO (nbRows - 2)
    IF M[iRow][0] <> 0.0
      FOR jRow = (iRow + 1) TO (nbRows - 1)
        IF sgn(M[iRow][0]) <> sgn(M[jRow][0]) AND
            M[jRow][0] <> 0.0
          sumNegCoeff = 0.0
          jCol = 0
          FOR iCol = 1 TO (nbCols - 1)
            Mp[nbRemainRows][jCol] =
              M[iRow][iCol] / ABS(M[iRow][0]) +
              M[jRow][iCol] / ABS(M[jRow][0])
            sumNegCoeff = sumNegCoeff + neg(Mp[nbRemainRows][jCol])
            jCol = jCol + 1
          END FOR
          Yp[nbRemainRows] =
            Y[iRow] / ABS(M[iRow][0]) +
            Y[jRow] / ABS(M[jRow][0])
          IF Yp[nbRemainRows] < sumNegCoeff
            RETURN TRUE
          END IF
          nbRemainRows = nbRemainRows + 1
        END IF
      END FOR
    END IF
  END FOR
  FOR iRow = 0 TO (nbRows - 1)
    IF M[iRow][0] == 0.0
      jCol = 0
      FOR iCol = 1 TO (nbCols - 1)
        Mp[nbRemainRows][jCol] = M[iRow][iCol]
        jCol = jCol + 1
      END FOR
      Yp[nbRemainRows] = Y[iRow]
      nbRemainRows = nbRemainRows + 1
    END IF
  END FOR
  RETURN FALSE
END FUNCTION

FUNCTION GetBoundLastVar2DTime(iVar, M, Y, nbRows, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR jRow = 0 TO (nbRows - 1)
    IF M[jRow][0] > 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.max[iVar] > y
        bdgBox.max[iVar] = y
      END IF
    ELSE IF M[jRow][0] < 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.min[iVar] < y
        bdgBox.min[iVar] = y
```

```
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION GetBoundVar2DTime(iVar, M, Y, nbRows, nbCols, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR iRow = 0 .. TO (nbRows - 1)
    IF M[iRow][0] <> 0.0
      min = -1.0 * Y[iRow]
      max = Y[iRow]
      FOR iCol = 1 .. TO (nbCols - 1)
        IF M[iRow][iCol] > 0.0
          min = min + M[iRow][iCol] * bdgBox.min[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.min[iCol + iVar]
        ELSE IF M[iRow][iCol] < 0.0
          min = min + M[iRow][iCol] * bdgBox.max[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.max[iCol + iVar]
        END IF
      END FOR
      min = min / (-1.0 * M[iRow][0])
      max = max / M[iRow][0]
      IF bdgBox.min[iVar] > min
        bdgBox.min[iVar] = min
      END IF
      IF bdgBox.max[iVar] < max
        bdgBox.max[iVar] = max
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION FMBTestIntersection2DTime(that, tho, bdgBox)
  Frame2DTimeImportFrame(that, tho, thoProj)
  M[0][0] = -thoProj.comp[0][0]
  M[0][1] = -thoProj.comp[1][0]
  M[0][2] = -thoProj.speed[0]
  Y[0] = thoProj.orig[0]
  IF (Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2]))
    RETURN FALSE
  END IF
  M[1][0] = -thoProj.comp[0][1]
  M[1][1] = -thoProj.comp[1][1]
  M[1][2] = -thoProj.speed[1]
  Y[1] = thoProj.orig[1]
  IF (Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2]))
    RETURN FALSE
  END IF
  nbRows = 2
  IF that.type == FrameCuboid
    M[nbRows][0] = thoProj.comp[0][0]
    M[nbRows][1] = thoProj.comp[1][0]
    M[nbRows][2] = thoProj.speed[0]
    Y[nbRows] = 1.0 - thoProj.orig[0]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][1]
    M[nbRows][1] = thoProj.comp[1][1]
```

```
  M[nbRows][2] = thoProj.speed[1]
  Y[nbRows] = 1.0 - thoProj.orig[1]
  IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                 neg(M[nbRows][2])
    RETURN FALSE
  END IF
  nbRows = nbRows + 1
ELSE
  M[nbRows][0] = thoProj.comp[0][0] + thoProj.comp[0][1]
  M[nbRows][1] = thoProj.comp[1][0] + thoProj.comp[1][1]
  M[nbRows][2] = thoProj.speed[0] + thoProj.speed[1]
  Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1]
  IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                 neg(M[nbRows][2])
    RETURN FALSE
  END IF
  nbRows = nbRows + 1
END IF
IF tho.type == FrameCuboid
  M[nbRows][0] = 1.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
ELSE
  M[nbRows][0] = 1.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
END IF
M[nbRows][0] = -1.0
M[nbRows][1] = 0.0
M[nbRows][2] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = -1.0
M[nbRows][2] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = 0.0
M[nbRows][2] = 1.0
Y[nbRows] = 1.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = 0.0
M[nbRows][2] = -1.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
inconsistency =
  ElimVar2DTime(M, Y, nbRows, 3, Mp, Yp, nbRowsP)
IF inconsistency == TRUE
  RETURN FALSE
END IF
inconsistency =
```

```
    ElimVar2DTime(Mp, Yp, nbRowsP, 2, Mpp, Ypp, nbRowsPP)
  IF inconsistency == TRUE
    RETURN FALSE
  END IF
  GetBoundLastVar2DTime(THD_VAR, Mpp, Ypp, nbRowsPP, bdgBoxLocal)
  IF bdgBoxLocal.min[THD_VAR] >= bdgBoxLocal.max[THD_VAR]
    RETURN FALSE
  ELSE
    GetBoundVar2DTime(SND_VAR, Mp, Yp, nbRowsP, 2, bdgBoxLocal)
    GetBoundVar2DTime(FST_VAR, M, Y, nbRows, 3, bdgBoxLocal)
    bdgBox = bdgBoxLocal
  END IF
  RETURN TRUE
END FUNCTION

origP2DTime = [0.0, 0.0]
speedP2DTime = [0.0, 0.0]
compP2DTime = [
  [1.0, 0.0],
  [0.0, 1.0]]
P2DTime =
  Frame2DTimeCreateStatic(
    FrameCuboid, origP2DTime, speedP2DTime, compP2DTime)
origQ2DTime = [0.0,0.0]
speedQ2DTime = [0.0,0.0]
compQ2DTime = [
  [1.0, 0.0],
  [0.0, 1.0]]
Q2DTime =
  Frame2DTimeCreateStatic(
    FrameCuboid, origQ2DTime, speedQ2DTime, compQ2DTime)
isIntersecting2DTime =
  FMBTestIntersection2DTime(P2DTime, Q2DTime, bdgBox2DTimeLocal)
IF isIntersecting2DTime == TRUE
  PRINT "Intersection detected."
  Frame2DTimeExportBdgBox(Q2DTime, bdgBox2DTimeLocal, bdgBox2DTime)
  AABB2DTimePrint(bdgBox2DTime)
ELSE
  PRINT "No intersection."
END IF
```

3.4 3D dynamic

```
ENUM FrameType
  FrameCuboid,
  FrameTetrahedron
END ENUM

STRUCT AABB3DTime
  // x,y,z,t
  real min[4]
  real max[4]
END STRUCT

STRUCT Frame3DTime
  FrameType type
  real orig[3]
  // comp[iComp][iAxis]
  real comp[3][3]
  AABB3DTime bdgBox
  real invComp[3][3]
```

```
  real speed[3]
END STRUCT

FUNCTION powi(base, exp)
  res = 1
  FOR i=0 TO (exp - 1)
    res = res * base
  END FOR
  RETURN res
END FUNCTION

FUNCTION Frame3DTimePrint(that)
  IF that.type == FrameTetrahedron
    PRINT "T"
  ELSE IF that.type == FrameCuboid
    PRINT "C"
  END IF
  PRINT "o("
  FOR i = 0 TO 2
    PRINT that.orig[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  PRINT " s("
  FOR i = 0 TO 2
    PRINT that.speed[i]
    IF i < 2
      PRINT ","
    END IF
  END FOR
  comp = ["x", "y", "z"]
  FOR j = 0 TO 2
    PRINT " ", comp[j], "("
    FOR i = 0 TO 2
      PRINT that.comp[j][i]
      IF i < 2
        PRINT ","
      END IF
    END FOR
  END FOR
  PRINT ")"
END FUNCTION

FUNCTION AABB3DTimePrint(that)
  PRINT "minXYZT("
  FOR i = 0 TO 3
    PRINT that.min[i]
    IF i < 3
      PRINT ","
    END IF
  END FOR
  PRINT ")-maxXYZT("
  FOR i = 0 TO 3
    PRINT that.max[i]
    IF i < 3
      PRINT ","
    END IF
  END FOR
  PRINT ")"
END FUNCTION
```

```
FUNCTION Frame3DTimeExportBdgBox(that, bdgBox, bdgBoxProj)
  bdgBoxProj.min[3] = bdgBox.min[3]
  bdgBoxProj.max[3] = bdgBox.max[3]
  FOR i = 0 TO 2
    bdgBoxProj.max[i] = that.orig[i] + that.speed[i] * bdgBox.min[3]
    FOR j = 0 TO 2
      bdgBoxProj.max[i] =
        bdgBoxProj.max[i] + that.comp[j][i] * bdgBox.min[j]
    END FOR
    bdgBoxProj.min[i] = bdgBoxProj.max[i]
  END FOR
  nbVertices = powi(2, 3)
  FOR iVertex = 1 TO (nbVertices - 1)
    FOR i = 0 TO 2
      IF BITWISEAND(iVertex, powi(2, i)) <> 0
        v[i] = bdgBox.max[i]
      ELSE
        v[i] = bdgBox.min[i]
      END IF
    END FOR
    FOR i = 0 TO 2
      w[i] = that.orig[i]
      FOR j = 0 TO 2
        w[i] = w[i] + that.comp[j][i] * v[j]
      END FOR
    END FOR
    FOR i = 0 TO 2
      IF bdgBoxProj.min[i] > w[i] + that.speed[i] * bdgBox.min[3]
        bdgBoxProj.min[i] = w[i] + that.speed[i] * bdgBox.min[3]
      END IF
      IF bdgBoxProj.min[i] > w[i] + that.speed[i] * bdgBox.max[3]
        bdgBoxProj.min[i] = w[i] + that.speed[i] * bdgBox.max[3]
      END IF
      IF bdgBoxProj.max[i] < w[i] + that.speed[i] * bdgBox.min[3]
        bdgBoxProj.max[i] = w[i] + that.speed[i] * bdgBox.min[3]
      END IF
      IF bdgBoxProj.max[i] < w[i] + that.speed[i] * bdgBox.max[3]
        bdgBoxProj.max[i] = w[i] + that.speed[i] * bdgBox.max[3]
      END IF
    END FOR
  END FOR
END FUNCTION

FUNCTION Frame3DTimeImPortFrame(P, Q, Qp)
  FOR i = 0 TO 2
    v[i] = Q.orig[i] - P.orig[i]
    s[i] = Q.speed[i] - P.speed[i]
  END FOR
  FOR i = 0 TO 2
    Qp.orig[i] = 0.0
    Qp.speed[i] = 0.0
    FOR j = 0 TO 2
      Qp.orig[i] = Qp.orig[i] + P.invComp[j][i] * v[j]
      Qp.speed[i] = Qp.speed[i] + P.invComp[j][i] * s[j]
      Qp.comp[j][i] = 0.0
      FOR k = 0 TO 2
        Qp.comp[j][i] = Qp.comp[j][i] + P.invComp[k][i] * Q.comp[j][k]
      END FOR
    END FOR
  END FOR
END FUNCTION
```

```
FUNCTION Frame3DTimeUpdateInv(that)
  det =
    that.comp[0][0] *
    (that.comp[1][1] * that.comp[2][2] - that.comp[1][2] * that.comp[2][1])
        -
    that.comp[1][0] *
    (that.comp[0][1] * that.comp[2][2] - that.comp[0][2] * that.comp[2][1])
        +
    that.comp[2][0] *
    (that.comp[0][1] * that.comp[1][2] - that.comp[0][2] * that.comp[1][1])
  that.invComp[0][0] = (that.comp[1][1] * that.comp[2][2] -
    that.comp[2][1] * that.comp[1][2]) / det
  that.invComp[0][1] = (that.comp[2][1] * that.comp[0][2] -
    that.comp[2][2] * that.comp[0][1]) / det
  that.invComp[0][2] = (that.comp[0][1] * that.comp[1][2] -
    that.comp[0][2] * that.comp[1][1]) / det
  that.invComp[1][0] = (that.comp[2][0] * that.comp[1][2] -
    that.comp[2][2] * that.comp[1][0]) / det
  that.invComp[1][1] = (that.comp[0][0] * that.comp[2][2] -
    that.comp[2][0] * that.comp[0][2]) / det
  that.invComp[1][2] = (that.comp[0][2] * that.comp[1][0] -
    that.comp[1][2] * that.comp[0][0]) / det
  that.invComp[2][0] = (that.comp[1][0] * that.comp[2][1] -
    that.comp[2][0] * that.comp[1][1]) / det
  that.invComp[2][1] = (that.comp[0][1] * that.comp[2][0] -
    that.comp[2][1] * that.comp[0][0]) / det
  that.invComp[2][2] = (that.comp[0][0] * that.comp[1][1] -
    that.comp[1][0] * that.comp[0][1]) / det
END FUNCTION

FUNCTION Frame3DTimeCreateStatic(type, orig, comp)
  that.type = type
  FOR iAxis = 0 TO 2
    that.orig[iAxis] = orig[iAxis]
    that.speed[iAxis] = speed[iAxis]
    FOR iComp = 0 TO 2
      that.comp[iComp][iAxis] = comp[iComp][iAxis]
    END FOR
  END FOR
  FOR iAxis = 0 TO 2
    min = orig[iAxis]
    max = orig[iAxis]
    FOR iComp = 0 TO 2
      IF that.type == FrameCuboid
        IF that.comp[iComp][iAxis] < 0.0
          min = min + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0
          max = max + that.comp[iComp][iAxis]
        END IF
      ELSE IF that.type == FrameTetrahedron
        IF that.comp[iComp][iAxis] < 0.0 AND
          min > orig[iAxis] + that.comp[iComp][iAxis]
          min = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
        IF that.comp[iComp][iAxis] > 0.0 AND
          max < orig[iAxis] + that.comp[iComp][iAxis]
          max = orig[iAxis] + that.comp[iComp][iAxis]
        END IF
      END IF
    END FOR
    IF that.speed[iAxis] < 0.0
```

```
      min = min + that.speed[iAxis]
    END IF
    IF that.speed[iAxis] > 0.0
      max = max + that.speed[iAxis]
    END IF
    that.bdgBox.min[iAxis] = min
    that.bdgBox.max[iAxis] = max
  END FOR
  that.bdgBox.min[3] = 0.0
  that.bdgBox.max[3] = 1.0
  Frame3DTimeUpdateInv(that)
  RETURN that
END FUNCTION

FUNCTION Sgn(v)
  IF 0.0 < v
    a = 1
  ELSE
    a = 0
  END IF
  IF v < 0.0
    b = 1
  ELSE
    b = 0
  END IF
  RETURN A - B
END FUNCTION

FUNCTION Neg(x)
  IF x < 0.0
    RETURN x
  ELSE
    RETURN 0.0
  END IF
END FUNCTION

FST_VAR = 0
SND_VAR = 1
THD_VAR = 2
FOR_VAR = 3

FUNCTION ElimVar3DTime(M, Y, nbRows, nbCols, Mp, Yp, nbRemainRows)
  nbRemainRows = 0
  FOR iRow = 0 TO (nbRows - 2)
    IF M[iRow][0] <> 0.0
      FOR jRow = (iRow + 1) TO (nbRows - 1)
        IF sgn(M[iRow][0]) <> sgn(M[jRow][0]) AND
            M[jRow][0] <> 0.0
          sumNegCoeff = 0.0
          jCol = 0
          FOR iCol = 1 TO (nbCols - 1)
            Mp[nbRemainRows][jCol] =
              M[iRow][iCol] / ABS(M[iRow][0]) +
              M[jRow][iCol] / ABS(M[jRow][0])
            sumNegCoeff = sumNegCoeff + neg(Mp[nbRemainRows][jCol])
            jCol = jCol + 1
          END FOR
          Yp[nbRemainRows] =
            Y[iRow] / ABS(M[iRow][0]) +
            Y[jRow] / ABS(M[jRow][0])
          IF Yp[nbRemainRows] < sumNegCoeff
            RETURN TRUE
```

```
          END IF
          nbRemainRows = nbRemainRows + 1
        END IF
      END FOR
    END IF
  END FOR
  FOR iRow = 0 TO (nbRows - 1)
    IF M[iRow][0] == 0.0
      jCol = 0
      FOR iCol = 1 TO (nbCols - 1)
        Mp[nbRemainRows][jCol] = M[iRow][iCol]
        jCol = jCol + 1
      END FOR
      Yp[nbRemainRows] = Y[iRow]
      nbRemainRows = nbRemainRows + 1
    END IF
  END FOR
  RETURN FALSE
END FUNCTION

FUNCTION GetBoundLastVar3DTime(iVar, M, Y, nbRows, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR jRow = 0 TO (nbRows - 1)
    IF M[jRow][0] > 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.max[iVar] > y
        bdgBox.max[iVar] = y
      END IF
    ELSE IF M[jRow][0] < 0.0
      y = Y[jRow] / M[jRow][0]
      IF bdgBox.min[iVar] < y
        bdgBox.min[iVar] = y
      END IF
    END IF
  END FOR
END FUNCTION

FUNCTION GetBoundVar3DTime(iVar, M, Y, nbRows, nbCols, bdgBox)
  bdgBox.min[iVar] = 0.0
  bdgBox.max[iVar] = 1.0
  FOR iRow = 0 .. TO (nbRows - 1)
    IF M[iRow][0] <> 0.0
      min = -1.0 * Y[iRow]
      max = Y[iRow]
      FOR iCol = 1 .. TO (nbCols - 1)
        IF M[iRow][iCol] > 0.0
          min = min + M[iRow][iCol] * bdgBox.min[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.min[iCol + iVar]
        ELSE IF M[iRow][iCol] < 0.0
          min = min + M[iRow][iCol] * bdgBox.max[iCol + iVar]
          max = max - M[iRow][iCol] * bdgBox.max[iCol + iVar]
        END IF
      END FOR
      min = min / (-1.0 * M[iRow][0])
      max = max / M[iRow][0]
      IF bdgBox.min[iVar] > min
        bdgBox.min[iVar] = min
      END IF
      IF bdgBox.max[iVar] < max
        bdgBox.max[iVar] = max
      END IF
```

```
    END IF
  END FOR
END FUNCTION

FUNCTION FMBTestIntersection3DTime(that, tho, bdgBox)
  Frame3DTimeImportFrame(that, tho, thoProj)
  M[0][0] = -thoProj.comp[0][0]
  M[0][1] = -thoProj.comp[1][0]
  M[0][2] = -thoProj.comp[2][0]
  M[0][3] = -thoProj.speed[0]
  Y[0] = thoProj.orig[0]
  IF Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2]) + neg(M[0][3])
    RETURN FALSE
  END IF
  M[1][0] = -thoProj.comp[0][1]
  M[1][1] = -thoProj.comp[1][1]
  M[1][2] = -thoProj.comp[2][1]
  M[1][3] = -thoProj.speed[1]
  Y[1] = thoProj.orig[1]
  IF Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2]) + neg(M[1][3])
    RETURN FALSE
  END IF
  M[2][0] = -thoProj.comp[0][2]
  M[2][1] = -thoProj.comp[1][2]
  M[2][2] = -thoProj.comp[2][2]
  M[2][3] = -thoProj.speed[2]
  Y[2] = thoProj.orig[2]
  IF Y[2] < neg(M[2][0]) + neg(M[2][1]) + neg(M[2][2]) + neg(M[2][3])
    RETURN FALSE
  END IF
  nbRows = 3
  IF that.type == FrameCuboid
    M[nbRows][0] = thoProj.comp[0][0]
    M[nbRows][1] = thoProj.comp[1][0]
    M[nbRows][2] = thoProj.comp[2][0]
    M[nbRows][3] = thoProj.speed[0]
    Y[nbRows] = 1.0 - thoProj.orig[0]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2]) + neg(M[nbRows][3])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][1]
    M[nbRows][1] = thoProj.comp[1][1]
    M[nbRows][2] = thoProj.comp[2][1]
    M[nbRows][3] = thoProj.speed[1]
    Y[nbRows] = 1.0 - thoProj.orig[1]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2]) + neg(M[nbRows][3])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
    M[nbRows][0] = thoProj.comp[0][2]
    M[nbRows][1] = thoProj.comp[1][2]
    M[nbRows][2] = thoProj.comp[2][2]
    M[nbRows][3] = thoProj.speed[2]
    Y[nbRows] = 1.0 - thoProj.orig[2]
    IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                   neg(M[nbRows][2]) + neg(M[nbRows][3])
      RETURN FALSE
    END IF
    nbRows = nbRows + 1
```

```
ELSE
  M[nbRows][0] =
    thoProj.comp[0][0] + thoProj.comp[0][1] + thoProj.comp[0][2]
  M[nbRows][1] =
    thoProj.comp[1][0] + thoProj.comp[1][1] + thoProj.comp[1][2]
  M[nbRows][2] =
    thoProj.comp[2][0] + thoProj.comp[2][1] + thoProj.comp[2][2]
  M[nbRows][3] = thoProj.speed[0] + thoProj.speed[1] + thoProj.speed[2]
  Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1] - thoProj.orig[2]
  IF Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
                 neg(M[nbRows][2]) + neg(M[nbRows][3])
    RETURN FALSE
  END IF
  nbRows = nbRows + 1
END IF
IF tho.type == FrameCuboid
  M[nbRows][0] = 1.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 0.0
  M[nbRows][3] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 0.0
  M[nbRows][3] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 1.0
  M[nbRows][3] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
ELSE
  M[nbRows][0] = 1.0
  M[nbRows][1] = 1.0
  M[nbRows][2] = 1.0
  M[nbRows][3] = 0.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
END IF
M[nbRows][0] = -1.0
M[nbRows][1] = 0.0
M[nbRows][2] = 0.0
M[nbRows][3] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = -1.0
M[nbRows][2] = 0.0
M[nbRows][3] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = 0.0
M[nbRows][2] = -1.0
M[nbRows][3] = 0.0
Y[nbRows] = 0.0
nbRows = nbRows + 1
M[nbRows][0] = 0.0
M[nbRows][1] = 0.0
```

```
  M[nbRows][2] = 0.0
  M[nbRows][3] = 1.0
  Y[nbRows] = 1.0
  nbRows = nbRows + 1
  M[nbRows][0] = 0.0
  M[nbRows][1] = 0.0
  M[nbRows][2] = 0.0
  M[nbRows][3] = -1.0
  Y[nbRows] = 0.0
  nbRows = nbRows + 1
  inconsistency =
    ElimVar3DTime(M, Y, nbRows, 4, Mp, Yp, nbRowsP)
  IF inconsistency == TRUE
    RETURN FALSE
  END IF
  inconsistency =
    ElimVar3DTime(Mp, Yp, nbRowsP, 3, Mpp, Ypp, nbRowsPP)
  IF inconsistency == TRUE
    RETURN FALSE
  END IF
  inconsistency =
    ElimVar3DTime(Mpp, Ypp, nbRowsPP, 2, Mppp, Yppp, nbRowsPPP)
  IF inconsistency == TRUE
    RETURN FALSE
  END IF
  GetBoundLastVar3DTime(FOR_VAR, Mppp, Yppp, nbRowsPPP, bdgBoxLocal)
  IF bdgBoxLocal.min[FOR_VAR] >= bdgBoxLocal.max[FOR_VAR]
    RETURN FALSE
  ELSE
    GetBoundVar3DTime(THD_VAR, Mpp, Ypp, nbRowsPP, 2, bdgBoxLocal)
    GetBoundVar3DTime(SND_VAR, Mp, Yp, nbRowsP, 3, bdgBoxLocal)
    GetBoundVar3DTime(FST_VAR, M, Y, nbRows, 4, bdgBoxLocal)
    bdgBox = bdgBoxLocal
  END IF
  RETURN TRUE
END FUNCTION

origP3DTime = [0.0, 0.0, 0.0]
speedP3DTime = [0.0, 0.0, 0.0]
compP3DTime = [
  [1.0, 0.0, 0.0],
  [0.0, 1.0, 0.0],
  [0.0, 0.0, 1.0]]
P3DTime =
  Frame3DTimeCreateStatic(
    FrameCuboid, origP3DTime, speedP3DTime, compP3DTime)
origQ3DTime = [0.0, 0.0, 0.0]
speedQ3DTime = [0.0, 0.0, 0.0]
compQ3DTime = [
  [1.0, 0.0, 0.0],
  [0.0, 1.0, 0.0],
  [0.0, 0.0, 1.0]]
Q3DTime =
  Frame3DTimeCreateStatic(
    FrameCuboid, origQ3DTime, speedQ3DTime, compQ3DTime)
isIntersecting3DTime =
  FMBTestIntersection3DTime(P3DTime, Q3DTime, bdgBox3DTimeLocal)
IF isIntersecting3DTime == TRUE
  PRINT "Intersection detected."
  Frame3DTimeExportBdgBox(Q3DTime, bdgBox3DTimeLocal, bdgBox3DTime)
  AABB3DTimePrint(bdgBox3DTime)
ELSE
```

```
  PRINT "No intersection."
END IF
```

4 Implementation of the algorithms in C

In this section I introduce an implementation of the algorithms of the previous section in the C language.

4.1 Frames

4.1.1 Header

```
#ifndef __FRAME_H_
#define __FRAME_H_

// ------------- Includes -------------

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// ------------- Macros -------------

// ------------- Enumerations -------------

typedef enum {

  FrameCuboid,
  FrameTetrahedron

} FrameType;

// ------------- Data structures -------------

// Axis aligned bounding box structure
typedef struct {

  // x, y
  double min[2];
  double max[2];

} AABB2D;

typedef struct {

  // x, y, z
  double min[3];
  double max[3];

} AABB3D;

typedef struct {

  // x, y, t
```

```
  double min[3];
  double max[3];

} AABB2DTime;

typedef struct {

  // x, y, z, t
  double min[4];
  double max[4];

} AABB3DTime;

// Axis unaligned cuboid and tetrahedron structure
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];

  // AABB of the frame
  AABB2D bdgBox;

  // Inverted components used during computation
  double invComp[2][2];

} Frame2D;

typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];

  // AABB of the frame
  AABB3D bdgBox;

  // Inverted components used during computation
  double invComp[3][3];

} Frame3D;

typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];

  // AABB of the frame
  AABB2DTime bdgBox;

  // Inverted components used during computation
  double invComp[2][2];
  double speed[2];

} Frame2DTime;

typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];
```

```
  // AABB of the frame
  AABB3DTime bdgBox;

  // Inverted components used during computation
  double invComp[3][3];
  double speed[3];

} Frame3DTime;

// ------------- Functions declaration -------------

// Print the AABB that on stdout
// Output format is
// (min[0], min[1], min[2], min[3])-(max[0], max[1], max[2], max[3])
void AABB2DPrint(const AABB2D* const that);
void AABB3DPrint(const AABB3D* const that);
void AABB2DTimePrint(const AABB2DTime* const that);
void AABB3DTimePrint(const AABB3DTime* const that);

// Print the Frame that on stdout
// Output format is
// T/C  <- type of Frame
// o(orig[0], orig[1], orig[2])
// s(speed[0], speed[1], speed[2])
// x(comp[0][0], comp[0][1], comp[0][2])
// y(comp[1][0], comp[1][1], comp[1][2])
// z(comp[2][0], comp[2][1], comp[2][2])
void Frame2DPrint(const Frame2D* const that);
void Frame3DPrint(const Frame3D* const that);
void Frame2DTimePrint(const Frame2DTime* const that);
void Frame3DTimePrint(const Frame3DTime* const that);

// Create a static Frame structure of FrameType type,
// at position orig with components comp ([iComp][iAxis])
// and speed
Frame2D Frame2DCreateStatic(
  const FrameType type,
  const double orig[2],
  const double comp[2][2]);
Frame3D Frame3DCreateStatic(
  const FrameType type,
  const double orig[3],
  const double comp[3][3]);
Frame2DTime Frame2DTimeCreateStatic(
  const FrameType type,
  const double orig[2],
  const double speed[2],
  const double comp[2][2]);
Frame3DTime Frame3DTimeCreateStatic(
  const FrameType type,
  const double orig[3],
  const double speed[3],
  const double comp[3][3]);

// Project the Frame Q in the Frame P's coordinates system and
// memorize the result in the Frame Qp
void Frame2DImportFrame(
  Frame2D* const P,
  const Frame2D* const Q,
  Frame2D* const Qp);
void Frame3DImportFrame(
```

```
  Frame3D* const P,
  const Frame3D* const Q,
  Frame3D* const Qp);
void Frame2DTimeImportFrame(
  Frame2DTime* const P,
  const Frame2DTime* const Q,
  Frame2DTime* const Qp);
void Frame3DTimeImportFrame(
  Frame3DTime* const P,
  const Frame3DTime* const Q,
  Frame3DTime* const Qp);

// Export the AABB bdgBox from that's coordinates system to
// the real coordinates system and update bdgBoxProj with the resulting
// AABB
void Frame2DExportBdgBox(
  const Frame2D* const that,
  const AABB2D* const bdgBox,
  AABB2D* const bdgBoxProj);
void Frame3DExportBdgBox(
  const Frame3D* const that,
  const AABB3D* const bdgBox,
  AABB3D* const bdgBoxProj);
void Frame2DTimeExportBdgBox(
  const Frame2DTime* const that,
  const AABB2DTime* const bdgBox,
  AABB2DTime* const bdgBoxProj);
void Frame3DTimeExportBdgBox(
  const Frame3DTime* const that,
  const AABB3DTime* const bdgBox,
  AABB3DTime* const bdgBoxProj);

// Check the intersection between two static AABB that and tho
bool AABBTestIntersection2D(
  const AABB2D* const that,
  const AABB2D* const tho);
bool AABBTestIntersection3D(
  const AABB3D* const that,
  const AABB3D* const tho);

// Power function for integer base and exponent
// Return base^exp
int powi(
  int base,
  unsigned int exp);

#endif
```

4.1.2 Body

```
#include "frame.h"

// ------------- Macros -------------

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Update the inverse components of the Frame that
void Frame2DUpdateInv(Frame2D* const that);
```

```
void Frame3DUpdateInv(Frame3D* const that);
void Frame2DTimeUpdateInv(Frame2DTime* const that);
void Frame3DTimeUpdateInv(Frame3DTime* const that);

// ------------- Functions implementation -------------

// Create a static Frame structure of FrameType type,
// at position orig with components comp and speed
// arrangement is comp[iComp][iAxis]
Frame2D Frame2DCreateStatic(
  const FrameType type,
  const double orig[2],
  const double comp[2][2]) {

  // Create the new Frame
  Frame2D that;
  that.type = type;
  for (
    int iAxis = 2;
    iAxis--;) {

    that.orig[iAxis] = orig[iAxis];

    for (
      int iComp = 2;
      iComp--;) {

      that.comp[iComp][iAxis] = comp[iComp][iAxis];

    }

  }

  // Create the bounding box
  for (
    int iAxis = 2;
    iAxis--;) {

    double min = orig[iAxis];
    double max = orig[iAxis];

    for (
      int iComp = 2;
      iComp--;) {

      if (that.type == FrameCuboid) {

        if (that.comp[iComp][iAxis] < 0.0) {

          min += that.comp[iComp][iAxis];

        }

        if (that.comp[iComp][iAxis] > 0.0) {

          max += that.comp[iComp][iAxis];

        }

      } else if (that.type == FrameTetrahedron) {

        if (
```

```
          that.comp[iComp][iAxis] < 0.0 &&
          min > orig[iAxis] + that.comp[iComp][iAxis]) {

          min = orig[iAxis] + that.comp[iComp][iAxis];

        }

        if (
          that.comp[iComp][iAxis] > 0.0 &&
          max < orig[iAxis] + that.comp[iComp][iAxis]) {

          max = orig[iAxis] + that.comp[iComp][iAxis];

        }

      }

    }

    that.bdgBox.min[iAxis] = min;
    that.bdgBox.max[iAxis] = max;

  }

  // Calculate the inverse matrix
  Frame2DUpdateInv(&that);

  // Return the new Frame
  return that;

}

Frame3D Frame3DCreateStatic(
  const FrameType type,
  const double orig[3],
  const double comp[3][3]) {

  // Create the new Frame
  Frame3D that;
  that.type = type;
  for (
    int iAxis = 3;
    iAxis--;) {

    that.orig[iAxis] = orig[iAxis];

    for (
      int iComp = 3;
      iComp--;) {

      that.comp[iComp][iAxis] = comp[iComp][iAxis];

    }

  }

  // Create the bounding box
  for (
    int iAxis = 3;
    iAxis--;) {

    double min = orig[iAxis];
```

```
    double max = orig[iAxis];

    for (
      int iComp = 3;
      iComp--;) {

      if (that.type == FrameCuboid) {

        if (that.comp[iComp][iAxis] < 0.0) {

          min += that.comp[iComp][iAxis];

        }

        if (that.comp[iComp][iAxis] > 0.0) {

          max += that.comp[iComp][iAxis];

        }

      } else if (that.type == FrameTetrahedron) {

        if (
          that.comp[iComp][iAxis] < 0.0 &&
          min > orig[iAxis] + that.comp[iComp][iAxis]) {

          min = orig[iAxis] + that.comp[iComp][iAxis];

        }

        if (
          that.comp[iComp][iAxis] > 0.0 &&
          max < orig[iAxis] + that.comp[iComp][iAxis]) {

          max = orig[iAxis] + that.comp[iComp][iAxis];

        }

      }

    }

    that.bdgBox.min[iAxis] = min;
    that.bdgBox.max[iAxis] = max;

  }

  // Calculate the inverse matrix
  Frame3DUpdateInv(&that);

  // Return the new Frame
  return that;

}

Frame2DTime Frame2DTimeCreateStatic(
  const FrameType type,
  const double orig[2],
  const double speed[2],
  const double comp[2][2]) {

  // Create the new Frame
```

```
Frame2DTime that;
that.type = type;
for (
  int iAxis = 2;
  iAxis--;) {

  that.orig[iAxis] = orig[iAxis];
  that.speed[iAxis] = speed[iAxis];

  for (
    int iComp = 2;
    iComp--;) {

    that.comp[iComp][iAxis] = comp[iComp][iAxis];

  }

}

// Create the bounding box
for (
  int iAxis = 2;
  iAxis--;) {

  double min = orig[iAxis];
  double max = orig[iAxis];

  for (
    int iComp = 2;
    iComp--;) {

    if (that.type == FrameCuboid) {

      if (that.comp[iComp][iAxis] < 0.0) {

        min += that.comp[iComp][iAxis];

      }

      if (that.comp[iComp][iAxis] > 0.0) {

        max += that.comp[iComp][iAxis];

      }

    } else if (that.type == FrameTetrahedron) {

      if (
        that.comp[iComp][iAxis] < 0.0 &&
        min > orig[iAxis] + that.comp[iComp][iAxis]) {

        min = orig[iAxis] + that.comp[iComp][iAxis];

      }

      if (
        that.comp[iComp][iAxis] > 0.0 &&
        max < orig[iAxis] + that.comp[iComp][iAxis]) {

        max = orig[iAxis] + that.comp[iComp][iAxis];

      }
```

```
      }

    }

    if (that.speed[iAxis] < 0.0) {

      min += that.speed[iAxis];

    }

    if (that.speed[iAxis] > 0.0) {

      max += that.speed[iAxis];

    }

    that.bdgBox.min[iAxis] = min;
    that.bdgBox.max[iAxis] = max;

  }

  that.bdgBox.min[2] = 0.0;
  that.bdgBox.max[2] = 1.0;

  // Calculate the inverse matrix
  Frame2DTimeUpdateInv(&that);

  // Return the new Frame
  return that;

}

Frame3DTime Frame3DTimeCreateStatic(
  const FrameType type,
  const double orig[3],
  const double speed[3],
  const double comp[3][3]) {

  // Create the new Frame
  Frame3DTime that;
  that.type = type;
  for (
    int iAxis = 3;
    iAxis--;) {

    that.orig[iAxis] = orig[iAxis];
    that.speed[iAxis] = speed[iAxis];

    for (
      int iComp = 3;
      iComp--;) {

      that.comp[iComp][iAxis] = comp[iComp][iAxis];

    }

  }

  // Create the bounding box
  for (
    int iAxis = 3;
```

```
  iAxis--;) {

  double min = orig[iAxis];
  double max = orig[iAxis];

  for (
    int iComp = 3;
    iComp--;) {

    if (that.type == FrameCuboid) {

      if (that.comp[iComp][iAxis] < 0.0) {

        min += that.comp[iComp][iAxis];

      }

      if (that.comp[iComp][iAxis] > 0.0) {

        max += that.comp[iComp][iAxis];

      }

    } else if (that.type == FrameTetrahedron) {

      if (
        that.comp[iComp][iAxis] < 0.0 &&
        min > orig[iAxis] + that.comp[iComp][iAxis]) {

        min = orig[iAxis] + that.comp[iComp][iAxis];

      }

      if (
        that.comp[iComp][iAxis] > 0.0 &&
        max < orig[iAxis] + that.comp[iComp][iAxis]) {

        max = orig[iAxis] + that.comp[iComp][iAxis];

      }

    }

  }

  if (that.speed[iAxis] < 0.0) {

    min += that.speed[iAxis];

  }

  if (that.speed[iAxis] > 0.0) {

    max += that.speed[iAxis];

  }

  that.bdgBox.min[iAxis] = min;
  that.bdgBox.max[iAxis] = max;

}
```

```
  that.bdgBox.min[3] = 0.0;
  that.bdgBox.max[3] = 1.0;

  // Calculate the inverse matrix
  Frame3DTimeUpdateInv(&that);

  // Return the new Frame
  return that;

}

// Update the inverse components of the Frame that
void Frame2DUpdateInv(Frame2D* const that) {

  // Shortcuts
  double (*tc)[2] = that->comp;
  double (*tic)[2] = that->invComp;

  double det = tc[0][0] * tc[1][1] - tc[1][0] * tc[0][1];
  if (fabs(det) < EPSILON) {

    fprintf(
      stderr,
      "FrameUpdateInv: det == 0.0\n");
    exit(1);

  }

  tic[0][0] = tc[1][1] / det;
  tic[0][1] = -tc[0][1] / det;
  tic[1][0] = -tc[1][0] / det;
  tic[1][1] = tc[0][0] / det;

}

void Frame3DUpdateInv(Frame3D* const that) {

  // Shortcuts
  double (*tc)[3] = that->comp;
  double (*tic)[3] = that->invComp;

  // Update the inverse components
  double det =
    tc[0][0] * (tc[1][1] * tc[2][2] - tc[1][2] * tc[2][1]) -
    tc[1][0] * (tc[0][1] * tc[2][2] - tc[0][2] * tc[2][1]) +
    tc[2][0] * (tc[0][1] * tc[1][2] - tc[0][2] * tc[1][1]);
  if (fabs(det) < EPSILON) {

    fprintf(
      stderr,
      "FrameUpdateInv: det == 0.0\n");
    exit(1);

  }

  tic[0][0] = (tc[1][1] * tc[2][2] - tc[2][1] * tc[1][2]) / det;
  tic[0][1] = (tc[2][1] * tc[0][2] - tc[2][2] * tc[0][1]) / det;
  tic[0][2] = (tc[0][1] * tc[1][2] - tc[0][2] * tc[1][1]) / det;
  tic[1][0] = (tc[2][0] * tc[1][2] - tc[2][2] * tc[1][0]) / det;
  tic[1][1] = (tc[0][0] * tc[2][2] - tc[2][0] * tc[0][2]) / det;
  tic[1][2] = (tc[0][2] * tc[1][0] - tc[1][2] * tc[0][0]) / det;
  tic[2][0] = (tc[1][0] * tc[2][1] - tc[2][0] * tc[1][1]) / det;
```

```
  tic[2][1] = (tc[0][1] * tc[2][0] - tc[2][1] * tc[0][0]) / det;
  tic[2][2] = (tc[0][0] * tc[1][1] - tc[1][0] * tc[0][1]) / det;

}

// Update the inverse components of the Frame that
void Frame2DTimeUpdateInv(Frame2DTime* const that) {

  // Shortcuts
  double (*tc)[2] = that->comp;
  double (*tic)[2] = that->invComp;

  double det = tc[0][0] * tc[1][1] - tc[1][0] * tc[0][1];
  if (fabs(det) < EPSILON) {

    fprintf(
      stderr,
      "FrameUpdateInv: det == 0.0\n");
    exit(1);

  }

  tic[0][0] = tc[1][1] / det;
  tic[0][1] = -tc[0][1] / det;
  tic[1][0] = -tc[1][0] / det;
  tic[1][1] = tc[0][0] / det;

}

void Frame3DTimeUpdateInv(Frame3DTime* const that) {

  // Shortcuts
  double (*tc)[3] = that->comp;
  double (*tic)[3] = that->invComp;

  // Update the inverse components
  double det =
    tc[0][0] * (tc[1][1] * tc[2][2] - tc[1][2] * tc[2][1]) -
    tc[1][0] * (tc[0][1] * tc[2][2] - tc[0][2] * tc[2][1]) +
    tc[2][0] * (tc[0][1] * tc[1][2] - tc[0][2] * tc[1][1]);
  if (fabs(det) < EPSILON) {

    fprintf(
      stderr,
      "FrameUpdateInv: det == 0.0\n");
    exit(1);

  }

  tic[0][0] = (tc[1][1] * tc[2][2] - tc[2][1] * tc[1][2]) / det;
  tic[0][1] = (tc[2][1] * tc[0][2] - tc[2][2] * tc[0][1]) / det;
  tic[0][2] = (tc[0][1] * tc[1][2] - tc[0][2] * tc[1][1]) / det;
  tic[1][0] = (tc[2][0] * tc[1][2] - tc[2][2] * tc[1][0]) / det;
  tic[1][1] = (tc[0][0] * tc[2][2] - tc[2][0] * tc[0][2]) / det;
  tic[1][2] = (tc[0][2] * tc[1][0] - tc[1][2] * tc[0][0]) / det;
  tic[2][0] = (tc[1][0] * tc[2][1] - tc[2][0] * tc[1][1]) / det;
  tic[2][1] = (tc[0][1] * tc[2][0] - tc[2][1] * tc[0][0]) / det;
  tic[2][2] = (tc[0][0] * tc[1][1] - tc[1][0] * tc[0][1]) / det;

}

// Project the Frame Q in the Frame P's coordinates system and
```

```
// memorize the result in the Frame Qp
void Frame2DImportFrame(
  Frame2D* const P,
  const Frame2D* const Q,
  Frame2D* const Qp) {

  // Update the inverse matrix of P
  Frame2DUpdateInv(P);

  // Shortcuts
  const double*  qo  = Q->orig;
  double*  qpo = Qp->orig;
  const double*  po  = P->orig;

  const double  (*pi)[2] = P->invComp;
  double (*qpc)[2] = Qp->comp;
  const double  (*qc)[2] = Q->comp;

  // Calculate the projection
  double v[2];
  for (
    int i = 2;
    i--;) {

    v[i] = qo[i] - po[i];

  }

  for (
    int i = 2;
    i--;) {

    qpo[i] = 0.0;

    for (
      int j = 2;
      j--;) {

      qpo[i] += pi[j][i] * v[j];
      qpc[j][i] = 0.0;

      for (
        int k = 2;
        k--;) {

        qpc[j][i] += pi[k][i] * qc[j][k];

      }

    }

  }

}

void Frame3DImportFrame(
  Frame3D* const P,
  const Frame3D* const Q,
  Frame3D* const Qp) {

  // Update the inverse matrix of P
  Frame3DUpdateInv(P);
```

```
  // Shortcuts
  const double*  qo  = Q->orig;
  double*  qpo = Qp->orig;
  const double*  po  = P->orig;

  const double  (*pi)[3] = P->invComp;
  double (*qpc)[3] = Qp->comp;
  const double  (*qc)[3] = Q->comp;

  // Calculate the projection
  double v[3];
  for (
    int i = 3;
    i--;) {

    v[i] = qo[i] - po[i];

  }

  for (
    int i = 3;
    i--;) {

    qpo[i] = 0.0;

    for (
      int j = 3;
      j--;) {

      qpo[i] += pi[j][i] * v[j];
      qpc[j][i] = 0.0;

      for (
        int k = 3;
        k--;) {

        qpc[j][i] += pi[k][i] * qc[j][k];

      }

    }

  }

}

void Frame2DTimeImportFrame(
  Frame2DTime* const P,
  const Frame2DTime* const Q,
  Frame2DTime* const Qp) {

  // Update the inverse matrix of P
  Frame2DTimeUpdateInv(P);

  // Shortcuts
  const double*  qo  = Q->orig;
  double*  qpo = Qp->orig;
  const double*  po  = P->orig;

  const double*  qs  = Q->speed;
  double*  qps = Qp->speed;
```

```
  const double*  ps  = P->speed;

  const double  (*pi)[2] = P->invComp;
  double (*qpc)[2] = Qp->comp;
  const double  (*qc)[2] = Q->comp;

  // Calculate the projection
  double v[2];
  double s[2];
  for (
    int i = 2;
    i--;) {

    v[i] = qo[i] - po[i];
    s[i] = qs[i] - ps[i];

  }

  for (
    int i = 2;
    i--;) {

    qpo[i] = 0.0;
    qps[i] = 0.0;

    for (
      int j = 2;
      j--;) {

      qpo[i] += pi[j][i] * v[j];
      qps[i] += pi[j][i] * s[j];
      qpc[j][i] = 0.0;

      for (
        int k = 2;
        k--;) {

        qpc[j][i] += pi[k][i] * qc[j][k];

      }

    }

  }

}

void Frame3DTimeImportFrame(
  Frame3DTime* const P,
  const Frame3DTime* const Q,
  Frame3DTime* const Qp) {

  // Update the inverse matrix of P
  Frame3DTimeUpdateInv(P);

  // Shortcuts
  const double*  qo  = Q->orig;
  double*  qpo = Qp->orig;
  const double*  po  = P->orig;

  const double*  qs  = Q->speed;
  double*  qps = Qp->speed;
```

```
  const double*  ps  = P->speed;

  const double  (*pi)[3] = P->invComp;
  double (*qpc)[3] = Qp->comp;
  const double  (*qc)[3] = Q->comp;

  // Calculate the projection
  double v[3];
  double s[3];
  for (
    int i = 3;
    i--;) {

    v[i] = qo[i] - po[i];
    s[i] = qs[i] - ps[i];

  }

  for (
    int i = 3;
    i--;) {

    qpo[i] = 0.0;
    qps[i] = 0.0;

    for (
      int j = 3;
      j--;) {

      qpo[i] += pi[j][i] * v[j];
      qps[i] += pi[j][i] * s[j];
      qpc[j][i] = 0.0;

      for (
        int k = 3;
        k--;) {

        qpc[j][i] += pi[k][i] * qc[j][k];

      }

    }

  }

}

// Export the AABB bdgBox from that's coordinates system to
// the real coordinates system and update bdgBoxProj with the resulting
// AABB
void Frame2DExportBdgBox(
  const Frame2D* const that,
  const AABB2D* const bdgBox,
  AABB2D* const bdgBoxProj) {

  // Shortcuts
  const double* to    = that->orig;
  const double* bbmi  = bdgBox->min;
  const double* bbma  = bdgBox->max;
  double* bbpmi = bdgBoxProj->min;
  double* bbpma = bdgBoxProj->max;
```

```
const double (*tc)[2] = that->comp;

// Initialise the coordinates of the result AABB with the projection
// of the first corner of the AABB in argument
for (
  int i = 2;
  i--;) {

  bbpma[i] = to[i];

  for (
    int j = 2;
    j--;) {

    bbpma[i] += tc[j][i] * bbmi[j];

  }

  bbpmi[i] = bbpma[i];

}

// Loop on vertices of the AABB
// Skip the first vertex which is the origin already computed above
int nbVertices = powi(2, 2);
for (
  int iVertex = nbVertices;
  iVertex-- && iVertex;) {

  // Declare a variable to memorize the coordinates of the vertex in
  // that's coordinates system
  double v[2];

  // Calculate the coordinates of the vertex in
  // that's coordinates system
  for (
    int i = 2;
    i--;) {

    v[i] = ((iVertex & (1 << i)) ? bbma[i] : bbmi[i]);

  }

  // Declare a variable to memorize the projected coordinates
  // in real coordinates system
  double w[2];

  // Project the vertex to real coordinates system
  for (
    int i = 2;
    i--;) {

    w[i] = to[i];

    for (
      int j = 2;
      j--;) {

      w[i] += tc[j][i] * v[j];

    }
```

```
    }

    // Update the coordinates of the result AABB
    for (
      int i = 2;
      i--;) {

      if (bbpmi[i] > w[i]) {

        bbpmi[i] = w[i];

      }

      if (bbpma[i] < w[i]) {

        bbpma[i] = w[i];

      }

    }

  }

}

void Frame3DExportBdgBox(
  const Frame3D* const that,
  const AABB3D* const bdgBox,
  AABB3D* const bdgBoxProj) {

  // Shortcuts
  const double* to    = that->orig;
  const double* bbmi  = bdgBox->min;
  const double* bbma  = bdgBox->max;
  double* bbpmi = bdgBoxProj->min;
  double* bbpma = bdgBoxProj->max;

  const double (*tc)[3] = that->comp;

  // Initialise the coordinates of the result AABB with the projection
  // of the first corner of the AABB in argument
  for (
    int i = 3;
    i--;) {

    bbpma[i] = to[i];

    for (
      int j = 3;
      j--;) {

      bbpma[i] += tc[j][i] * bbmi[j];

    }

    bbpmi[i] = bbpma[i];

  }

  // Loop on vertices of the AABB
  // skip the first vertex which is the origin already computed above
  int nbVertices = powi(2, 3);
```

```
  for (
    int iVertex = nbVertices;
    iVertex-- && iVertex;) {

    // Declare a variable to memorize the coordinates of the vertex in
    // that's coordinates system
    double v[3];

    // Calculate the coordinates of the vertex in
    // that's coordinates system
    for (
      int i = 3;
      i--;) {

      v[i] = ((iVertex & (1 << i)) ? bbma[i] : bbmi[i]);

    }

    // Declare a variable to memorize the projected coordinates
    // in real coordinates system
    double w[3];

    // Project the vertex to real coordinates system
    for (
      int i = 3;
      i--;) {

      w[i] = to[i];

      for (
        int j = 3;
        j--;) {

        w[i] += tc[j][i] * v[j];

      }

    }

    // Update the coordinates of the result AABB
    for (
      int i = 3;
      i--;) {

      if (bbpmi[i] > w[i]) {

        bbpmi[i] = w[i];

      }

      if (bbpma[i] < w[i]) {

        bbpma[i] = w[i];

      }

    }

  }

}
```

```
void Frame2DTimeExportBdgBox(
  const Frame2DTime* const that,
  const AABB2DTime* const bdgBox,
  AABB2DTime* const bdgBoxProj) {

  // Shortcuts
  const double* to    = that->orig;
  const double* ts    = that->speed;
  const double* bbmi  = bdgBox->min;
  const double* bbma  = bdgBox->max;
  double* bbpmi = bdgBoxProj->min;
  double* bbpma = bdgBoxProj->max;
  const double (*tc)[2] = that->comp;

  // The time component is not affected
  bbpmi[2] = bbmi[2];
  bbpma[2] = bbma[2];

  // Initialise the coordinates of the result AABB with the projection
  // of the first corner of the AABB in argument
  for (
    int i = 2;
    i--;) {

    bbpma[i] = to[i] + ts[i] * bbmi[2];

    for (
      int j = 2;
      j--;) {

      bbpma[i] += tc[j][i] * bbmi[j];

    }

    bbpmi[i] = bbpma[i];

  }

  // Loop on vertices of the AABB
  // skip the first vertex which is the origin already computed above
  int nbVertices = powi(2, 2);
  for (
    int iVertex = nbVertices;
    iVertex-- && iVertex;) {

    // Declare a variable to memorize the coordinates of the vertex in
    // 'that' 's coordinates system
    double v[2];

    // Calculate the coordinates of the vertex in
    // 'that' 's coordinates system
    for (
      int i = 2;
      i--;) {

      v[i] = ((iVertex & (1 << i)) ? bbma[i] : bbmi[i]);

    }

    // Declare a variable to memorize the projected coordinates
    // in real coordinates system
    double w[2];
```

```
    // Project the vertex to real coordinates system
    for (
      int i = 2;
      i--;) {

      w[i] = to[i];

      for (
        int j = 2;
        j--;) {

        w[i] += tc[j][i] * v[j];

      }

    }

    // Update the coordinates of the result AABB
    for (
      int i = 2;
      i--;) {

      if (bbpmi[i] > w[i] + ts[i] * bbmi[2]) {

        bbpmi[i] = w[i] + ts[i] * bbmi[2];

      }

      if (bbpmi[i] > w[i] + ts[i] * bbma[2]) {

        bbpmi[i] = w[i] + ts[i] * bbma[2];

      }

      if (bbpma[i] < w[i] + ts[i] * bbmi[2]) {

        bbpma[i] = w[i] + ts[i] * bbmi[2];

      }

      if (bbpma[i] < w[i] + ts[i] * bbma[2]) {

        bbpma[i] = w[i] + ts[i] * bbma[2];

      }

    }

  }

}

void Frame3DTimeExportBdgBox(
  const Frame3DTime* const that,
  const AABB3DTime* const bdgBox,
  AABB3DTime* const bdgBoxProj) {

  // Shortcuts
  const double* to    = that->orig;
  const double* ts    = that->speed;
  const double* bbmi  = bdgBox->min;
```

```
const double* bbma  = bdgBox->max;
double* bbpmi = bdgBoxProj->min;
double* bbpma = bdgBoxProj->max;
const double (*tc)[3] = that->comp;

// The time component is not affected
bbpmi[3] = bbmi[3];
bbpma[3] = bbma[3];

// Initialise the coordinates of the result AABB with the projection
// of the first corner of the AABB in argument
for (
  int i = 3;
  i--;) {

  bbpma[i] = to[i] + ts[i] * bbmi[3];

  for (
    int j = 3;
    j--;) {

    bbpma[i] += tc[j][i] * bbmi[j];

  }

  bbpmi[i] = bbpma[i];

}

// Loop on vertices of the AABB
// skip the first vertex which is the origin already computed above
int nbVertices = powi(2, 3);
for (
  int iVertex = nbVertices;
  iVertex-- && iVertex;) {

  // Declare a variable to memorize the coordinates of the vertex in
  // that's coordinates system
  double v[3];

  // Calculate the coordinates of the vertex in
  // that's coordinates system
  for (
    int i = 3;
    i--;) {

    v[i] = ((iVertex & (1 << i)) ? bbma[i] : bbmi[i]);

  }

  // Declare a variable to memorize the projected coordinates
  // in real coordinates system
  double w[3];

  // Project the vertex to real coordinates system
  for (
    int i = 3;
    i--;) {

    w[i] = to[i];

    for (
```

```
        int j = 3;
        j--;) {

        w[i] += tc[j][i] * v[j];

      }

    }

    // Update the coordinates of the result AABB
    for (
      int i = 3;
      i--;) {

      if (bbpmi[i] > w[i] + ts[i] * bbmi[3]) {

        bbpmi[i] = w[i] + ts[i] * bbmi[3];

      }

      if (bbpmi[i] > w[i] + ts[i] * bbma[3]) {

        bbpmi[i] = w[i] + ts[i] * bbma[3];

      }

      if (bbpma[i] < w[i] + ts[i] * bbmi[3]) {

        bbpma[i] = w[i] + ts[i] * bbmi[3];

      }

      if (bbpma[i] < w[i] + ts[i] * bbma[3]) {

        bbpma[i] = w[i] + ts[i] * bbma[3];

      }

    }

  }

}

// Print the AABB that on stdout
// Output format is (min[0], min[1], ...)-(max[0], max[1], ...)
void AABB2DPrint(const AABB2D* const that) {

  printf("minXY(");
  for (
    int i = 0;
    i < 2;
    ++i) {

    printf("%f", that->min[i]);
    if (i < 1) printf(",");

  }

  printf(")-maxXY(");
  for (
    int i = 0;
```

```
    i < 2;
    ++i) {

    printf("%f", that->max[i]);
    if (i < 1) printf(",");

  }

  printf(")");

}

void AABB3DPrint(const AABB3D* const that) {

  printf("minXYZ(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->min[i]);
    if (i < 2) printf(",");

  }

  printf(")-maxXYZ(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->max[i]);
    if (i < 2) printf(",");

  }

  printf(")");

}

void AABB2DTimePrint(const AABB2DTime* const that) {

  printf("minXYT(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->min[i]);
    if (i < 2) printf(",");

  }

  printf(")-maxXYT(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->max[i]);
    if (i < 2) printf(",");
```

```
  }

  printf(")");

}

void AABB3DTimePrint(const AABB3DTime* const that) {

  printf("minXYZT(");
  for (
    int i = 0;
    i < 4;
    ++i) {

    printf("%f", that->min[i]);
    if (i < 3) printf(",");

  }

  printf(")-maxXYZT(");
  for (
    int i = 0;
    i < 4;
    ++i) {

    printf("%f", that->max[i]);
    if (i < 3) printf(",");

  }

  printf(")");

}

// Print the Frame that on stdout
// Output format is
// T/C  <- type of Frame
// o(orig[0], orig[1], orig[2])
// s(speed[0], speed[1], speed[2])
// x(comp[0][0], comp[0][1], comp[0][2])
// y(comp[1][0], comp[1][1], comp[1][2])
// z(comp[2][0], comp[2][1], comp[2][2])
void Frame2DPrint(const Frame2D* const that) {

  if (that->type == FrameTetrahedron) {

    printf("T");

  } else if (that->type == FrameCuboid) {

    printf("C");

  }

  printf("o(");
  for (
    int i = 0;
    i < 2;
    ++i) {

    printf("%f", that->orig[i]);
    if (i < 1) printf(",");
```

```
  }

  char comp[2] = {'x', 'y'};
  for (
    int j = 0;
    j < 2;
    ++j) {

    printf(") %c(", comp[j]);
    for (
      int i = 0;
      i < 2;
      ++i) {

      printf("%f", that->comp[j][i]);
      if (i < 1) printf(",");

    }

  }

  printf(")");

}

void Frame3DPrint(const Frame3D* const that) {

  if (that->type == FrameTetrahedron) {

    printf("T");

  } else if (that->type == FrameCuboid) {

    printf("C");

  }

  printf("o(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->orig[i]);
    if (i < 2) printf(",");

  }

  char comp[3] = {'x', 'y', 'z'};
  for (
    int j = 0;
    j < 3;
    ++j) {

    printf(") %c(", comp[j]);
    for (
      int i = 0;
      i < 3;
      ++i) {

      printf("%f", that->comp[j][i]);
```

```
      if (i < 2) printf(",");

    }

  }

  printf(")");

}

void Frame2DTimePrint(const Frame2DTime* const that) {

  if (that->type == FrameTetrahedron) {

    printf("T");

  } else if (that->type == FrameCuboid) {

    printf("C");

  }

  printf("o(");
  for (
    int i = 0;
    i < 2;
    ++i) {

    printf("%f", that->orig[i]);
    if (i < 1) printf(",");

  }

  printf(") s(");
  for (
    int i = 0;
    i < 2;
    ++i) {

    printf("%f", that->speed[i]);
    if (i < 1) printf(",");

  }

  char comp[2] = {'x', 'y'};
  for (
    int j = 0;
    j < 2;
    ++j) {

    printf(") %c(", comp[j]);
    for (
      int i = 0;
      i < 2;
      ++i) {

      printf("%f", that->comp[j][i]);
      if (i < 1) printf(",");

    }

  }
```

```
  printf(")");

}

void Frame3DTimePrint(const Frame3DTime* const that) {

  if (that->type == FrameTetrahedron) {

    printf("T");

  } else if (that->type == FrameCuboid) {

    printf("C");

  }

  printf("o(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->orig[i]);
    if (i < 2) printf(",");

  }

  printf(") s(");
  for (
    int i = 0;
    i < 3;
    ++i) {

    printf("%f", that->speed[i]);
    if (i < 2) printf(",");

  }

  char comp[3] = {'x', 'y', 'z'};
  for (
    int j = 0;
    j < 3;
    ++j) {

    printf(") %c(", comp[j]);
    for (
      int i = 0;
      i < 3;
      ++i) {

      printf("%f", that->comp[j][i]);
      if (i < 2) printf(",");

    }

  }

  printf(")");

}
```

```
// Check the intersection between two static AABB that and tho
bool AABBTestIntersection2D(
  const AABB2D* const that,
  const AABB2D* const tho) {

  if (
    that->min[0] > tho->max[0] ||
    that->min[1] > tho->max[1] ||
    tho->min[0] > that->max[0] ||
    tho->min[1] > that->max[1]) {

    return false;

  }

  return true;

}

bool AABBTestIntersection3D(
  const AABB3D* const that,
  const AABB3D* const tho) {

  if (
    that->min[0] > tho->max[0] ||
    that->min[1] > tho->max[1] ||
    that->min[2] > tho->max[2] ||
    tho->min[0] > that->max[0] ||
    tho->min[1] > that->max[1] ||
    tho->min[2] > that->max[2]) {

    return false;

  }

  return true;

}

// Power function for integer base and exponent
// Return base^exp
int powi(
  int base,
  unsigned int exp) {

  int res = 1;
  for (; exp; --exp) res *= base;

  return res;

}
```

4.2 FMB

4.2.1 2D static

Header

```
#ifndef __FMB2D_H_
#define __FMB2D_H_
```

```
#include <stdbool.h>
#include "frame.h"

// ------------- Functions declaration -------------

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection2D(
  Frame2D* const that,
  const Frame2D* const tho,
  AABB2D* const bdgBox);

#endif
```

Body

```
#include "fmb2d.h"

// ------------- Macros -------------

// Return 1.0 if v is positive, -1.0 if v is negative, 0.0 else
#define sgn(v) (((0.0 < (v)) ? 1 : 0) - (((v) < 0.0) ? 1 : 0))

// Return x if x is negative, 0.0 else
#define neg(x) (x < 0.0 ? x : 0.0)

#define FST_VAR 0
#define SND_VAR 1

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// Return false if the system becomes inconsistent during elimination,
// else return true
bool ElimVar2D(
  const double (*M)[2],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[2],
  double* Yp,
  int* const nbRemainRows);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
```

```
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar2D(
  const int iVar,
  const double (*M)[2],
  const double* Y,
  const int nbRows,
  AABB2D* const bdgBox);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2D(
  const int iVar,
  const double (*M)[2],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB2D* const bdgBox);

// ------------- Functions implementation -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// (M arrangement is [iRow][iCol])
// Return true if the system becomes inconsistent during elimination,
// else return false
bool ElimVar2D(
  const double (*M)[2],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[2],
  double* Yp,
  int* const nbRemainRows) {

  // Initialize the number of rows in the result system
  int nbResRows = 0;

  // First we process the rows where the eliminated variable is not null
  // For each row except the last one
  for (
    int iRow = 0;
    iRow < nbRows - 1;
    ++iRow) {

    // Shortcuts
    const double fabsMIRowIVar = fabs(M[iRow][0]);

    // If the coefficient for the eliminated variable is not null
    // in this row
    if (fabsMIRowIVar > EPSILON) {

      // Shortcuts
      const double* MiRow = M[iRow];
      const int sgnMIRowIVar = sgn(MiRow[0]);
      const double YIRowDivideByFabsMIRowIVar = Y[iRow] / fabsMIRowIVar;
```

```
  // For each following rows
  for (
    int jRow = iRow + 1;
    jRow < nbRows;
    ++jRow) {

    // If coefficients of the eliminated variable in the two rows have
    // different signs and are not null
    if (
      sgnMIRowIVar != sgn(M[jRow][0]) &&
      fabs(M[jRow][0]) > EPSILON) {

      // Shortcuts
      const double* MjRow = M[jRow];
      const double fabsMjRow = fabs(MjRow[0]);

      // Declare a variable to memorize the sum of the negative
      // coefficients in the row
      double sumNegCoeff = 0.0;

      // Add the sum of the two normed (relative to the eliminated
      // variable) rows into the result system. This actually
      // eliminate the variable while keeping the constraints on
      // others variables
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol ) {

        Mp[nbResRows][iCol - 1] =
          MiRow[iCol] / fabsMIRowIVar +
          MjRow[iCol] / fabsMjRow;

        // Update the sum of the negative coefficient
        sumNegCoeff += neg(Mp[nbResRows][iCol - 1]);

      }

      // Update the right side of the inequality
      Yp[nbResRows] =
        YIRowDivideByFabsMIRowIVar +
        Y[jRow] / fabsMjRow;

      // If the right side of the inequality is lower than the sum of
      // negative coefficients in the row
      // (Add epsilon for numerical imprecision)
      if (Yp[nbResRows] < sumNegCoeff - EPSILON) {

        // Given that X is in [0,1], the system is inconsistent
        return true;

      }

      // Increment the nb of rows into the result system
      ++nbResRows;

    }

  }

}
```

```
  }

  // Then we copy and compress the rows where the eliminated
  // variable is null
  // Loop on rows of the input system
  for (
    int iRow = 0;
    iRow < nbRows;
    ++iRow) {

    // Shortcut
    const double* MiRow = M[iRow];

    // If the coefficient of the eliminated variable is null on
    // this row
    if (fabs(MiRow[0]) < EPSILON) {

      // Shortcut
      double* MpnbResRows = Mp[nbResRows];

      // Copy this row into the result system excluding the eliminated
      // variable
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol) {

        MpnbResRows[iCol - 1] = MiRow[iCol];

      }

      Yp[nbResRows] = Y[iRow];

      // Increment the nb of rows into the result system
      ++nbResRows;

    }

  }

  // Memorize the number of rows in the result system
  *nbRemainRows = nbResRows;

  // If we reach here the system is not inconsistent
  return false;

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar2D(
  const int iVar,
  const double (*M)[2],
  const double* Y,
  const int nbRows,
  AABB2D* const bdgBox) {
```

```
  // Shortcuts
  double* min = bdgBox->min + iVar;
  double* max = bdgBox->max + iVar;

  // Initialize the bounds to their maximum maximum and minimum minimum
  *min = 0.0;
  *max = 1.0;

  // Loop on rows
  for (
    int jRow = 0;
    jRow < nbRows;
    ++jRow) {

    // Shortcut
    double MjRowiVar = M[jRow][0];

    // If this row has been reduced to the variable in argument
    // and it has a strictly positive coefficient
    if (MjRowiVar > EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is lower than the current maximum bound
      if (*max > y) {

        // Update the maximum bound
        *max = y;

      }

    // Else, if this row has been reduced to the variable in argument
    // and it has a strictly negative coefficient
    } else if (MjRowiVar < -EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is greater than the current minimum bound
      if (*min < y) {

        // Update the minimum bound
        *min = y;

      }

    }

  }

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2D(
  const int iVar,
  const double (*M)[2],
  const double* Y,
  const int nbRows,
```

```
const int nbCols,
AABB2D* const bdgBox) {

// Shortcuts
double* bdgBoxMin = bdgBox->min;
double* bdgBoxMax = bdgBox->max;

// Initialize the bounds
bdgBoxMin[iVar] = 0.0;
bdgBoxMax[iVar] = 1.0;

// Loop on the rows
for (
  int iRow = 0;
  iRow < nbRows;
  ++iRow) {

  // Shortcuts
  const double* MIRow = M[iRow];
  double fabsMIRowIVar = fabs(MIRow[0]);

  // If the coefficient of the first variable on this row is not null
  if (fabsMIRowIVar > EPSILON) {

    // Declare two variables to memorize the min and max of the
    // requested variable in this row
    double min = -1.0 * Y[iRow];
    double max = Y[iRow];

    // Loop on columns except the first one which is the one of the
    // requested variable
    for (
      int iCol = 1;
      iCol < nbCols;
      ++iCol) {

      if (MIRow[iCol] > EPSILON) {

        min += MIRow[iCol] * bdgBoxMin[iCol + iVar];
        max -= MIRow[iCol] * bdgBoxMin[iCol + iVar];

      } else if (MIRow[iCol] < EPSILON) {

        min += MIRow[iCol] * bdgBoxMax[iCol + iVar];
        max -= MIRow[iCol] * bdgBoxMax[iCol + iVar];

      }

    }

    min /= -1.0 * MIRow[0];
    max /= MIRow[0];
    if (bdgBoxMin[iVar] > min) {

      bdgBoxMin[iVar] = min;

    }

    if (bdgBoxMax[iVar] < max) {

      bdgBoxMax[iVar] = max;
```

```
      }

    }

  }

}

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection2D(
  Frame2D* const that,
  const Frame2D* const tho,
  AABB2D* const bdgBox) {

  // Get the projection of the Frame tho in Frame that coordinates
  // system
  Frame2D thoProj;
  Frame2DImportFrame(that, tho, &thoProj);

  // Declare two variables to memorize the system to be solved M.X <= Y
  // (M arrangement is [iRow][iCol])
  double M[8][2];
  double Y[8];

  // Create the inequality system
  // -sum_iC_j,iX_i<=O_j
  M[0][0] = -thoProj.comp[0][0];
  M[0][1] = -thoProj.comp[1][0];
  Y[0] = thoProj.orig[0];
  if (Y[0] < neg(M[0][0]) + neg(M[0][1])) {

    return false;

  }

  M[1][0] = -thoProj.comp[0][1];
  M[1][1] = -thoProj.comp[1][1];
  Y[1] = thoProj.orig[1];
  if (Y[1] < neg(M[1][0]) + neg(M[1][1])) {

    return false;

  }

  // Variable to memorize the nb of rows in the system
  int nbRows = 2;

  if (that->type == FrameCuboid) {

    // sum_iC_j,iX_i<=1.0-O_j
    M[nbRows][0] = thoProj.comp[0][0];
    M[nbRows][1] = thoProj.comp[1][0];
    Y[nbRows] = 1.0 - thoProj.orig[0];
```

```
  if (Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][1];
  Y[nbRows] = 1.0 - thoProj.orig[1];
  if (Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])) {

    return false;

  }

  ++nbRows;

} else {

  // sum_j(sum_iC_j,iX_i)<=1.0-sum_iO_i
  M[nbRows][0] = thoProj.comp[0][0] + thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][0] + thoProj.comp[1][1];
  Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1];
  if (Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1])) {

    return false;

  }

  ++nbRows;

}

if (tho->type == FrameCuboid) {

  // X_i <= 1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
  M[nbRows][1] = 1.0;
  Y[nbRows] = 1.0;
  ++nbRows;

} else {

  // sum_iX_i<=1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 1.0;
  Y[nbRows] = 1.0;
  ++nbRows;

}

// -X_i <= 0.0
M[nbRows][0] = -1.0;
M[nbRows][1] = 0.0;
Y[nbRows] = 0.0;
```

```
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = -1.0;
Y[nbRows] = 0.0;
++nbRows;

// Solve the system
// Declare a AABB to memorize the bounding box of the intersection
// in the coordinates system of tho
AABB2D bdgBoxLocal = {

  .min = {0.0, 0.0},
  .max = {0.0, 0.0}

};

// Declare variables to eliminate the first variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mp[24][2];
//double Yp[24];
double Mp[11][2];
double Yp[11];
int nbRowsP;

// Eliminate the first variable
bool inconsistency =
  ElimVar2D(
    M,
    Y,
    nbRows,
    2,
    Mp,
    Yp,
    &nbRowsP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Get the bounds for the remaining second variable
GetBoundLastVar2D(
  SND_VAR,
  Mp,
  Yp,
  nbRowsP,
  &bdgBoxLocal);

// If the bounds are inconsistent
if (bdgBoxLocal.min[SND_VAR] >= bdgBoxLocal.max[SND_VAR]) {

  // The two Frames are not in intersection
  return false;

// Else, if the bounds are consistent here it means
```

```
  // the two Frames are in intersection.
  // If the user has requested for the resulting bounding box
  } else if (bdgBox != NULL) {

    // Get the bounds of the first variable from the bounds of the
    // second one
    GetBoundVar2D(
      FST_VAR,
      M,
      Y,
      nbRows,
      2,
      &bdgBoxLocal);

    // Memorize the result
    *bdgBox = bdgBoxLocal;

  }

  // If we've reached here the two Frames are intersecting
  return true;

}
```

4.2.2 3D static

Header

```
#ifndef __FMB3D_H_
#define __FMB3D_H_

#include <stdbool.h>
#include "frame.h"

// ------------- Functions declaration -------------

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A, B) may be different
// of the resulting AABB of FMBTestIntersection(B, A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection3D(
  Frame3D* const that,
  const Frame3D* const tho,
  AABB3D* const bdgBox);

#endif
```

Body

```
#include "fmb3d.h"

// ------------- Macros -------------
```

```
// Return 1.0 if v is positive, -1.0 if v is negative, 0.0 else
#define sgn(v) (((0.0 < (v)) ? 1 : 0) - (((v) < 0.0) ? 1 : 0))

// Return x if x is negative, 0.0 else
#define neg(x) (x < 0.0 ? x : 0.0)

#define FST_VAR 0
#define SND_VAR 1
#define THD_VAR 2

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// Return false if the system becomes inconsistent during elimination,
// else return true
bool ElimVar3D(
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[3],
  double* Yp,
  int* const nbRemainRows);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar3D(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  AABB3D* const bdgBox);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3D(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB3D* const bdgBox);

// ------------- Functions implementation -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
```

```
// (M arrangement is [iRow][iCol])
// Return true if the system becomes inconsistent during elimination,
// else return false
bool ElimVar3D(
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[3],
  double* Yp,
  int* const nbRemainRows) {

  // Initialize the number of rows in the result system
  int nbResRows = 0;

  // First we process the rows where the eliminated variable is not null
  // For each row except the last one
  for (
    int iRow = 0;
    iRow < nbRows - 1;
    ++iRow) {

    // Shortcuts
    const double fabsMIRowIVar = fabs(M[iRow][0]);

    // If the coefficient for the eliminated variable is not null
    // in this row
    if (fabsMIRowIVar > EPSILON) {

      // Shortcuts
      const double* MiRow = M[iRow];
      const int sgnMIRowIVar = sgn(MiRow[0]);
      const double YIRowDivideByFabsMIRowIVar = Y[iRow] / fabsMIRowIVar;

      // For each following rows
      for (
        int jRow = iRow + 1;
        jRow < nbRows;
        ++jRow) {

        // If coefficients of the eliminated variable in the two rows have
        // different signs and are not null
        if (
          sgnMIRowIVar != sgn(M[jRow][0]) &&
          fabs(M[jRow][0]) > EPSILON) {

          // Shortcuts
          const double* MjRow = M[jRow];
          const double fabsMjRow = fabs(MjRow[0]);

          // Declare a variable to memorize the sum of the negative
          // coefficients in the row
          double sumNegCoeff = 0.0;

          // Add the sum of the two normed (relative to the eliminated
          // variable) rows into the result system. This actually
          // eliminate the variable while keeping the constraints on
          // others variables
          for (
            int iCol = 1;
            iCol < nbCols;
            ++iCol ) {
```

```
          Mp[nbResRows][iCol - 1] =
            MiRow[iCol] / fabsMIRowIVar +
            MjRow[iCol] / fabsMjRow;

          // Update the sum of the negative coefficient
          sumNegCoeff += neg(Mp[nbResRows][iCol - 1]);

        }

        // Update the right side of the inequality
        Yp[nbResRows] =
          YIRowDivideByFabsMIRowIVar +
          Y[jRow] / fabsMjRow;

        // If the right side of the inequality is lower than the sum of
        // negative coefficients in the row
        // (Add epsilon for numerical imprecision)
        if (Yp[nbResRows] < sumNegCoeff - EPSILON) {

          // Given that X is in [0,1], the system is inconsistent
          return true;

        }

        // Increment the nb of rows into the result system
        ++nbResRows;

      }

    }

  }

}

// Then we copy and compress the rows where the eliminated
// variable is null
// Loop on rows of the input system
for (
  int iRow = 0;
  iRow < nbRows;
  ++iRow) {

  // Shortcut
  const double* MiRow = M[iRow];

  // If the coefficient of the eliminated variable is null on
  // this row
  if (fabs(MiRow[0]) < EPSILON) {

    // Shortcut
    double* MpnbResRows = Mp[nbResRows];

    // Copy this row into the result system excluding the eliminated
    // variable
    for (
      int iCol = 1;
      iCol < nbCols;
      ++iCol) {

      MpnbResRows[iCol - 1] = MiRow[iCol];
```

```
      }

      Yp[nbResRows] = Y[iRow];

      // Increment the nb of rows into the result system
      ++nbResRows;

    }

  }

  // Memorize the number of rows in the result system
  *nbRemainRows = nbResRows;

  // If we reach here the system is not inconsistent
  return false;

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar3D(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  AABB3D* const bdgBox) {

  // Shortcuts
  double* min = bdgBox->min + iVar;
  double* max = bdgBox->max + iVar;

  // Initialize the bounds to their maximum maximum and minimum minimum
  *min = 0.0;
  *max = 1.0;

  // Loop on rows
  for (
    int jRow = 0;
    jRow < nbRows;
    ++jRow) {

    // Shortcut
    double MjRowiVar = M[jRow][0];

    // If this row has been reduced to the variable in argument
    // and it has a strictly positive coefficient
    if (MjRowiVar > EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is lower than the current maximum bound
      if (*max > y) {

        // Update the maximum bound
```

```
        *max = y;

      }

    // Else, if this row has been reduced to the variable in argument
    // and it has a strictly negative coefficient
    } else if (MjRowiVar < -EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is greater than the current minimum bound
      if (*min < y) {

        // Update the minimum bound
        *min = y;

      }

    }

  }

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3D(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB3D* const bdgBox) {

  // Shortcuts
  double* bdgBoxMin = bdgBox->min;
  double* bdgBoxMax = bdgBox->max;

  // Initialize the bounds
  bdgBoxMin[iVar] = 0.0;
  bdgBoxMax[iVar] = 1.0;

  // Loop on the rows
  for (
    int iRow = 0;
    iRow < nbRows;
    ++iRow) {

    // Shortcuts
    const double* MIRow = M[iRow];
    double fabsMIRowIVar = fabs(MIRow[0]);

    // If the coefficient of the first variable on this row is not null
    if (fabsMIRowIVar > EPSILON) {

      // Declare two variables to memorize the min and max of the
      // requested variable in this row
      double min = -1.0 * Y[iRow];
      double max = Y[iRow];
```

```
      // Loop on columns except the first one which is the one of the
      // requested variable
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol) {

        if (MIRow[iCol] > EPSILON) {

          min += MIRow[iCol] * bdgBoxMin[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMin[iCol + iVar];

        } else if (MIRow[iCol] < EPSILON) {

          min += MIRow[iCol] * bdgBoxMax[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMax[iCol + iVar];

        }

      }

      min /= -1.0 * MIRow[0];
      max /= MIRow[0];
      if (bdgBoxMin[iVar] > min) {

        bdgBoxMin[iVar] = min;

      }

      if (bdgBoxMax[iVar] < max) {

        bdgBoxMax[iVar] = max;

      }

    }

  }

}

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection3D(
  Frame3D* const that,
  const Frame3D* const tho,
  AABB3D* const bdgBox) {

  // Get the projection of the Frame tho in Frame that coordinates
  // system
  Frame3D thoProj;
  Frame3DImportFrame(that, tho, &thoProj);
```

```
// Declare two variables to memorize the system to be solved M.X <= Y
// (M arrangement is [iRow][iCol])
double M[12][3];
double Y[12];

// Create the inequality system
// -sum_iC_j,iX_i<=O_j
M[0][0] = -thoProj.comp[0][0];
M[0][1] = -thoProj.comp[1][0];
M[0][2] = -thoProj.comp[2][0];
Y[0] = thoProj.orig[0];
if (Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2])) {

  return false;

}

M[1][0] = -thoProj.comp[0][1];
M[1][1] = -thoProj.comp[1][1];
M[1][2] = -thoProj.comp[2][1];
Y[1] = thoProj.orig[1];
if (Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2])) {

  return false;

}

M[2][0] = -thoProj.comp[0][2];
M[2][1] = -thoProj.comp[1][2];
M[2][2] = -thoProj.comp[2][2];
Y[2] = thoProj.orig[2];
if (Y[2] < neg(M[2][0]) + neg(M[2][1]) + neg(M[2][2])) {

  return false;

}

// Variable to memorize the nb of rows in the system
int nbRows = 3;

if (that->type == FrameCuboid) {

  // sum_iC_j,iX_i<=1.0-O_j
  M[nbRows][0] = thoProj.comp[0][0];
  M[nbRows][1] = thoProj.comp[1][0];
  M[nbRows][2] = thoProj.comp[2][0];
  Y[nbRows] = 1.0 - thoProj.orig[0];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][1];
  M[nbRows][2] = thoProj.comp[2][1];
  Y[nbRows] = 1.0 - thoProj.orig[1];
  if (
```

```
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][2];
  M[nbRows][1] = thoProj.comp[1][2];
  M[nbRows][2] = thoProj.comp[2][2];
  Y[nbRows] = 1.0 - thoProj.orig[2];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

} else {

  // sum_j(sum_iC_j,iX_i)<=1.0-sum_iO_i
  M[nbRows][0] =
    thoProj.comp[0][0] + thoProj.comp[0][1] + thoProj.comp[0][2];
  M[nbRows][1] =
    thoProj.comp[1][0] + thoProj.comp[1][1] + thoProj.comp[1][2];
  M[nbRows][2] =
    thoProj.comp[2][0] + thoProj.comp[2][1] + thoProj.comp[2][2];
  Y[nbRows] =
    1.0 - thoProj.orig[0] - thoProj.orig[1] - thoProj.orig[2];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

}

if (tho->type == FrameCuboid) {

  // X_i <= 1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 0.0;
  M[nbRows][2] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
  M[nbRows][1] = 1.0;
  M[nbRows][2] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
```

```
  M[nbRows][1] = 0.0;
  M[nbRows][2] = 1.0;
  Y[nbRows] = 1.0;
  ++nbRows;

} else {

  // sum_iX_i<=1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 1.0;
  M[nbRows][2] = 1.0;
  Y[nbRows] = 1.0;
  ++nbRows;

}

// -X_i <= 0.0
M[nbRows][0] = -1.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = -1.0;
M[nbRows][2] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = -1.0;
Y[nbRows] = 0.0;
++nbRows;

// Solve the system
// Declare a AABB to memorize the bounding box of the intersection
// in the coordinates system of tho
AABB3D bdgBoxLocal = {

  .min = {0.0, 0.0, 0.0},
  .max = {0.0, 0.0, 0.0}

};

// Declare variables to eliminate the first variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mp[48][3];
//double Yp[48];
double Mp[20][3];
double Yp[20];
int nbRowsP;

// Eliminate the first variable in the original system
bool inconsistency =
  ElimVar3D(
    M,
    Y,
    nbRows,
```

```
    3,
    Mp,
    Yp,
    &nbRowsP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Declare variables to eliminate the second variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mpp[624][3];
//double Ypp[624];
double Mpp[55][3];
double Ypp[55];
int nbRowsPP;

// Eliminate the second variable (which is the first in the new system)
inconsistency =
  ElimVar3D(
    Mp,
    Yp,
    nbRowsP,
    2,
    Mpp,
    Ypp,
    &nbRowsPP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Get the bounds for the remaining third variable
GetBoundLastVar3D(
  THD_VAR,
  Mpp,
  Ypp,
  nbRowsPP,
  &bdgBoxLocal);

// If the bounds are inconsistent
if (bdgBoxLocal.min[THD_VAR] >= bdgBoxLocal.max[THD_VAR]) {

  // The two Frames are not in intersection
  return false;

// Else, if the bounds are consistent here it means
// the two Frames are in intersection.
// If the user has requested for the resulting bounding box
} else if (bdgBox != NULL) {
```

```
    // Get the bounds of the other variables
    GetBoundVar3D(
      SND_VAR,
      Mp,
      Yp,
      nbRowsP,
      2,
      &bdgBoxLocal);

    GetBoundVar3D(
      FST_VAR,
      M,
      Y,
      nbRows,
      3,
      &bdgBoxLocal);

    // Memorize the result
    *bdgBox = bdgBoxLocal;

  }

  // If we've reached here the two Frames are intersecting
  return true;

}
```

4.2.3 2D dynamic

Header

```
#ifndef __FMB2DT_H_
#define __FMB2DT_H_

#include <stdbool.h>
#include "frame.h"

// ------------- Functions declaration -------------

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection2DTime(
  Frame2DTime* const that,
  const Frame2DTime* const tho,
  AABB2DTime* const bdgBox);

#endif
```

Body

```
#include "fmb2dt.h"

// ------------- Macros -------------

// Return 1.0 if v is positive, -1.0 if v is negative, 0.0 else
#define sgn(v) (((0.0 < (v)) ? 1 : 0) - (((v) < 0.0) ? 1 : 0))

// Return x if x is negative, 0.0 else
#define neg(x) (x < 0.0 ? x : 0.0)

#define FST_VAR 0
#define SND_VAR 1
#define THD_VAR 2

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// Return false if the system becomes inconsistent during elimination,
// else return true
bool ElimVar2DTime(
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[3],
  double* Yp,
  int* const nbRemainRows);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar2DTime(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  AABB2DTime* const bdgBox);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2DTime(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB2DTime* const bdgBox);

// ------------- Functions implementation -------------

// Eliminate the first variable in the system M.X<=Y
```

```
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// (M arrangement is [iRow][iCol])
// Return true if the system becomes inconsistent during elimination,
// else return false
bool ElimVar2DTime(
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[3],
  double* Yp,
  int* const nbRemainRows) {

  // Initialize the number of rows in the result system
  int nbResRows = 0;

  // First we process the rows where the eliminated variable is not null
  // For each row except the last one
  for (
    int iRow = 0;
    iRow < nbRows - 1;
    ++iRow) {

    // Shortcuts
    const double fabsMIRowIVar = fabs(M[iRow][0]);

    // If the coefficient for the eliminated variable is not null
    // in this row
    if (fabsMIRowIVar > EPSILON) {

      // Shortcuts
      const double* MiRow = M[iRow];
      const int sgnMIRowIVar = sgn(MiRow[0]);
      const double YIRowDivideByFabsMIRowIVar = Y[iRow] / fabsMIRowIVar;

      // For each following rows
      for (
        int jRow = iRow + 1;
        jRow < nbRows;
        ++jRow) {

        // If coefficients of the eliminated variable in the two rows have
        // different signs and are not null
        if (
          sgnMIRowIVar != sgn(M[jRow][0]) &&
          fabs(M[jRow][0]) > EPSILON) {

          // Shortcuts
          const double* MjRow = M[jRow];
          const double fabsMjRow = fabs(MjRow[0]);

          // Declare a variable to memorize the sum of the negative
          // coefficients in the row
          double sumNegCoeff = 0.0;

          // Add the sum of the two normed (relative to the eliminated
          // variable) rows into the result system. This actually
          // eliminate the variable while keeping the constraints on
          // others variables
          for (
```

```
          int iCol = 1;
          iCol < nbCols;
          ++iCol ) {

          Mp[nbResRows][iCol - 1] =
            MiRow[iCol] / fabsMIRowIVar +
            MjRow[iCol] / fabsMjRow;

          // Update the sum of the negative coefficient
          sumNegCoeff += neg(Mp[nbResRows][iCol - 1]);

        }

        // Update the right side of the inequality
        Yp[nbResRows] =
          YIRowDivideByFabsMIRowIVar +
          Y[jRow] / fabsMjRow;

        // If the right side of the inequality is lower than the sum of
        // negative coefficients in the row
        // (Add epsilon for numerical imprecision)
        if (Yp[nbResRows] < sumNegCoeff - EPSILON) {

          // Given that X is in [0,1], the system is inconsistent
          return true;

        }

        // Increment the nb of rows into the result system
        ++nbResRows;

      }

    }

  }

}

// Then we copy and compress the rows where the eliminated
// variable is null
// Loop on rows of the input system
for (
  int iRow = 0;
  iRow < nbRows;
  ++iRow) {

  // Shortcut
  const double* MiRow = M[iRow];

  // If the coefficient of the eliminated variable is null on
  // this row
  if (fabs(MiRow[0]) < EPSILON) {

    // Shortcut
    double* MpnbResRows = Mp[nbResRows];

    // Copy this row into the result system excluding the eliminated
    // variable
    for (
      int iCol = 1;
      iCol < nbCols;
```

```
        ++iCol) {

        MpnbResRows[iCol - 1] = MiRow[iCol];

      }

      Yp[nbResRows] = Y[iRow];

      // Increment the nb of rows into the result system
      ++nbResRows;

    }

  }

  // Memorize the number of rows in the result system
  *nbRemainRows = nbResRows;

  // If we reach here the system is not inconsistent
  return false;

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar2DTime(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  AABB2DTime* const bdgBox) {

  // Shortcuts
  double* min = bdgBox->min + iVar;
  double* max = bdgBox->max + iVar;

  // Initialize the bounds to there maximum maximum and minimum minimum
  *min = 0.0;
  *max = 1.0;

  // Loop on rows
  for (
    int jRow = 0;
    jRow < nbRows;
    ++jRow) {

    // Shortcut
    double MjRowiVar = M[jRow][0];

    // If this row has been reduced to the variable in argument
    // and it has a strictly positive coefficient
    if (MjRowiVar > EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is lower than the current maximum bound
```

```
      if (*max > y) {

        // Update the maximum bound
        *max = y;

      }

    // Else, if this row has been reduced to the variable in argument
    // and it has a strictly negative coefficient
    } else if (MjRowiVar < -EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is greater than the current minimum bound
      if (*min < y) {

        // Update the minimum bound
        *min = y;

      }

    }

  }

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2DTime(
  const int iVar,
  const double (*M)[3],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB2DTime* const bdgBox) {

  // Shortcuts
  double* bdgBoxMin = bdgBox->min;
  double* bdgBoxMax = bdgBox->max;

  // Initialize the bounds
  bdgBoxMin[iVar] = 0.0;
  bdgBoxMax[iVar] = 1.0;

  // Loop on the rows
  for (
    int iRow = 0;
    iRow < nbRows;
    ++iRow) {

    // Shortcuts
    const double* MIRow = M[iRow];
    double fabsMIRowIVar = fabs(MIRow[0]);

    // If the coefficient of the first variable on this row is not null
    if (fabsMIRowIVar > EPSILON) {

      // Declare two variables to memorize the min and max of the
```

```
      // requested variable in this row
      double min = -1.0 * Y[iRow];
      double max = Y[iRow];

      // Loop on columns except the first one which is the one of the
      // requested variable
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol) {

        if (MIRow[iCol] > EPSILON) {

          min += MIRow[iCol] * bdgBoxMin[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMin[iCol + iVar];

        } else if (MIRow[iCol] < EPSILON) {

          min += MIRow[iCol] * bdgBoxMax[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMax[iCol + iVar];

        }

      }

      min /= -1.0 * MIRow[0];
      max /= MIRow[0];
      if (bdgBoxMin[iVar] > min) {

        bdgBoxMin[iVar] = min;

      }

      if (bdgBoxMax[iVar] < max) {

        bdgBoxMax[iVar] = max;

      }

    }

  }

}

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection2DTime(
  Frame2DTime* const that,
  const Frame2DTime* const tho,
  AABB2DTime* const bdgBox) {

  // Get the projection of the Frame tho in Frame that coordinates
  // system
```

```
Frame2DTime thoProj;
Frame2DTimeImportFrame(that, tho, &thoProj);

// Declare two variables to memorize the system to be solved M.X <= Y
// (M arrangement is [iRow][iCol])
double M[10][3];
double Y[10];

// Create the inequality system
// -V_jT-sum_iC_j,iX_i<=0_j
M[0][0] = -thoProj.comp[0][0];
M[0][1] = -thoProj.comp[1][0];
M[0][2] = -thoProj.speed[0];
Y[0] = thoProj.orig[0];
if (Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2])) {

  return false;

}

M[1][0] = -thoProj.comp[0][1];
M[1][1] = -thoProj.comp[1][1];
M[1][2] = -thoProj.speed[1];
Y[1] = thoProj.orig[1];
if (Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2])) {

  return false;

}

// Variable to memorize the nb of rows in the system
int nbRows = 2;

if (that->type == FrameCuboid) {

  // V_jT+sum_iC_j,iX_i<=1.0-0_j
  M[nbRows][0] = thoProj.comp[0][0];
  M[nbRows][1] = thoProj.comp[1][0];
  M[nbRows][2] = thoProj.speed[0];
  Y[nbRows] = 1.0 - thoProj.orig[0];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][1];
  M[nbRows][2] = thoProj.speed[1];
  Y[nbRows] = 1.0 - thoProj.orig[1];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }
```

```
  ++nbRows;

} else {

  // sum_j(V_jT+sum_iC_j,iX_i)<=1.0-sum_iO_i
  M[nbRows][0] = thoProj.comp[0][0] + thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][0] + thoProj.comp[1][1];
  M[nbRows][2] = thoProj.speed[0] + thoProj.speed[1];
  Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2])) {

    return false;

  }

  ++nbRows;

}

if (tho->type == FrameCuboid) {

  // X_i <= 1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 0.0;
  M[nbRows][2] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
  M[nbRows][1] = 1.0;
  M[nbRows][2] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

} else {

  // sum_iX_i<=1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 1.0;
  M[nbRows][2] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

}

// -X_i <= 0.0
M[nbRows][0] = -1.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = -1.0;
M[nbRows][2] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

// 0.0 <= t <= 1.0
M[nbRows][0] = 0.0;
```

```
M[nbRows][1] = 0.0;
M[nbRows][2] = 1.0;
Y[nbRows] = 1.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = -1.0;
Y[nbRows] = 0.0;
++nbRows;

// Solve the system
// Declare a AABB to memorize the bounding box of the intersection
// in the coordinates system of tho
AABB2DTime bdgBoxLocal = {

  .min = {0.0, 0.0, 0.0},
  .max = {0.0, 0.0, 0.0}

};

// Declare variables to eliminate the first variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mp[35][3];
//double Yp[35];
double Mp[13][3];
double Yp[13];
int nbRowsP;

// Eliminate the first variable in the original system
bool inconsistency =
  ElimVar2DTime(
    M,
    Y,
    nbRows,
    3,
    Mp,
    Yp,
    &nbRowsP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Declare variables to eliminate the second variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mpp[342][3];
//double Ypp[342];
double Mpp[21][3];
double Ypp[21];
int nbRowsPP;
```

```
// Eliminate the second variable (which is the first in the new system)
inconsistency =
  ElimVar2DTime(
    Mp,
    Yp,
    nbRowsP,
    2,
    Mpp,
    Ypp,
    &nbRowsPP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Get the bounds for the remaining third variable
GetBoundLastVar2DTime(
  THD_VAR,
  Mpp,
  Ypp,
  nbRowsPP,
  &bdgBoxLocal);

// If the bounds are inconsistent
if (bdgBoxLocal.min[THD_VAR] >= bdgBoxLocal.max[THD_VAR]) {

  // The two Frames are not in intersection
  return false;

// Else, if the bounds are consistent here it means
// the two Frames are in intersection.
// If the user has requested for the resulting bounding box
} else if (bdgBox != NULL) {

  // Get the bounds of the other variables
  GetBoundVar2DTime(
    SND_VAR,
    Mp,
    Yp,
    nbRowsP,
    2,
    &bdgBoxLocal);

  GetBoundVar2DTime(
    FST_VAR,
    M,
    Y,
    nbRows,
    3,
    &bdgBoxLocal);

  // Memorize the result
  *bdgBox = bdgBoxLocal;

}

// If we've reached here the two Frames are intersecting
return true;
```

```
}
```

4.2.4 3D dynamic

Header

```
#ifndef __FMB3DT_H_
#define __FMB3DT_H_

#include <stdbool.h>
#include "frame.h"

// ------------- Functions declaration -------------

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection3DTime(
  Frame3DTime* const that,
  const Frame3DTime* const tho,
  AABB3DTime* const bdgBox);

#endif
```

Body

```
#include "fmb3dt.h"

// ------------- Macros -------------

// Return 1.0 if v is positive, -1.0 if v is negative, 0.0 else
#define sgn(v) (((0.0 < (v)) ? 1 : 0) - (((v) < 0.0) ? 1 : 0))

// Return x if x is negative, 0.0 else
#define neg(x) (x < 0.0 ? x : 0.0)

#define FST_VAR 0
#define SND_VAR 1
#define THD_VAR 2
#define FOR_VAR 3

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// Return false if the system becomes inconsistent during elimination,
```

```
// else return true
bool ElimVar3DTime(
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[4],
  double* Yp,
  int* const nbRemainRows);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar3DTime(
  const int iVar,
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  AABB3DTime* const bdgBox);

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3DTime(
  const int iVar,
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB3DTime* const bdgBox);

// ------------- Functions implementation -------------

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// (M arrangement is [iRow][iCol])
// Return true if the system becomes inconsistent during elimination,
// else return false
bool ElimVar3DTime(
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  const int nbCols,
  double (*Mp)[4],
  double* Yp,
  int* const nbRemainRows) {

  // Initialize the number of rows in the result system
  int nbResRows = 0;

  // First we process the rows where the eliminated variable is not null
  // For each row except the last one
  for (
    int iRow = 0;
    iRow < nbRows - 1;
```

```
++iRow) {

// Shortcuts
const double fabsMIRowIVar = fabs(M[iRow][0]);

// If the coefficient for the eliminated variable is not null
// in this row
if (fabsMIRowIVar > EPSILON) {

  // Shortcuts
  const double* MiRow = M[iRow];
  const int sgnMIRowIVar = sgn(MiRow[0]);
  const double YIRowDivideByFabsMIRowIVar = Y[iRow] / fabsMIRowIVar;

  // For each following rows
  for (
    int jRow = iRow + 1;
    jRow < nbRows;
    ++jRow) {

    // If coefficients of the eliminated variable in the two rows have
    // different signs and are not null
    if (
      sgnMIRowIVar != sgn(M[jRow][0]) &&
      fabs(M[jRow][0]) > EPSILON) {

      // Shortcuts
      const double* MjRow = M[jRow];
      const double fabsMjRow = fabs(MjRow[0]);

      // Declare a variable to memorize the sum of the negative
      // coefficients in the row
      double sumNegCoeff = 0.0;

      // Add the sum of the two normed (relative to the eliminated
      // variable) rows into the result system. This actually
      // eliminate the variable while keeping the constraints on
      // others variables
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol ) {

        Mp[nbResRows][iCol - 1] =
          MiRow[iCol] / fabsMIRowIVar +
          MjRow[iCol] / fabsMjRow;

        // Update the sum of the negative coefficient
        sumNegCoeff += neg(Mp[nbResRows][iCol - 1]);

      }

      // Update the right side of the inequality
      Yp[nbResRows] =
        YIRowDivideByFabsMIRowIVar +
        Y[jRow] / fabsMjRow;

      // If the right side of the inequality is lower than the sum of
      // negative coefficients in the row
      // (Add epsilon for numerical imprecision)
      if (Yp[nbResRows] < sumNegCoeff - EPSILON) {
```

```
            // Given that X is in [0,1], the system is inconsistent
            return true;

          }

          // Increment the nb of rows into the result system
          ++nbResRows;

        }

      }

    }

  }

  // Then we copy and compress the rows where the eliminated
  // variable is null
  // Loop on rows of the input system
  for (
    int iRow = 0;
    iRow < nbRows;
    ++iRow) {

    // Shortcut
    const double* MiRow = M[iRow];

    // If the coefficient of the eliminated variable is null on
    // this row
    if (fabs(MiRow[0]) < EPSILON) {

      // Shortcut
      double* MpnbResRows = Mp[nbResRows];

      // Copy this row into the result system excluding the eliminated
      // variable
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol) {

        MpnbResRows[iCol - 1] = MiRow[iCol];

      }

      Yp[nbResRows] = Y[iRow];

      // Increment the nb of rows into the result system
      ++nbResRows;

    }

  }

  // Memorize the number of rows in the result system
  *nbRemainRows = nbResRows;

  // If we reach here the system is not inconsistent
  return false;

}
```

```
// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// The system is supposed to have been reduced to only one variable
// per row, the one in argument
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar3DTime(
  const int iVar,
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  AABB3DTime* const bdgBox) {

  // Shortcuts
  double* min = bdgBox->min + iVar;
  double* max = bdgBox->max + iVar;

  // Initialize the bounds to there maximum maximum and minimum minimum
  *min = 0.0;
  *max = 1.0;

  // Loop on rows
  for (
    int jRow = 0;
    jRow < nbRows;
    ++jRow) {

    // Shortcut
    double MjRowiVar = M[jRow][0];

    // If this row has been reduced to the variable in argument
    // and it has a strictly positive coefficient
    if (MjRowiVar > EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is lower than the current maximum bound
      if (*max > y) {

        // Update the maximum bound
        *max = y;

      }

    // Else, if this row has been reduced to the variable in argument
    // and it has a strictly negative coefficient
    } else if (MjRowiVar < -EPSILON) {

      // Get the scaled value of Y for this row
      double y = Y[jRow] / MjRowiVar;

      // If the value is greater than the current minimum bound
      if (*min < y) {

        // Update the minimum bound
        *min = y;

      }
```

```
    }

  }

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3DTime(
  const int iVar,
  const double (*M)[4],
  const double* Y,
  const int nbRows,
  const int nbCols,
  AABB3DTime* const bdgBox) {

  // Shortcuts
  double* bdgBoxMin = bdgBox->min;
  double* bdgBoxMax = bdgBox->max;

  // Initialize the bounds
  bdgBoxMin[iVar] = 0.0;
  bdgBoxMax[iVar] = 1.0;

  // Loop on the rows
  for (
    int iRow = 0;
    iRow < nbRows;
    ++iRow) {

    // Shortcuts
    const double* MIRow = M[iRow];
    double fabsMIRowIVar = fabs(MIRow[0]);

    // If the coefficient of the first variable on this row is not null
    if (fabsMIRowIVar > EPSILON) {

      // Declare two variables to memorize the min and max of the
      // requested variable in this row
      double min = -1.0 * Y[iRow];
      double max = Y[iRow];

      // Loop on columns except the first one which is the one of the
      // requested variable
      for (
        int iCol = 1;
        iCol < nbCols;
        ++iCol) {

        if (MIRow[iCol] > EPSILON) {

          min += MIRow[iCol] * bdgBoxMin[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMin[iCol + iVar];

        } else if (MIRow[iCol] < EPSILON) {

          min += MIRow[iCol] * bdgBoxMax[iCol + iVar];
          max -= MIRow[iCol] * bdgBoxMax[iCol + iVar];

        }
```

```
      }

      min /= -1.0 * MIRow[0];
      max /= MIRow[0];
      if (bdgBoxMin[iVar] > min) {

        bdgBoxMin[iVar] = min;

      }

      if (bdgBoxMax[iVar] < max) {

        bdgBoxMax[iVar] = max;

      }

    }

  }

}

// Test for intersection between Frame that and Frame tho
// Return true if the two Frames are intersecting, else false
// If the Frame are intersecting the AABB of the intersection
// is stored into bdgBox, else bdgBox is not modified
// If bdgBox is null, the result AABB is not memorized (to use if
// unnecessary and want to speed up the algorithm)
// The resulting AABB may be larger than the smallest possible AABB
// The resulting AABB of FMBTestIntersection(A,B) may be different
// of the resulting AABB of FMBTestIntersection(B,A)
// The resulting AABB is given in tho's local coordinates system
bool FMBTestIntersection3DTime(
  Frame3DTime* const that,
  const Frame3DTime* const tho,
  AABB3DTime* const bdgBox) {

  // Get the projection of the Frame tho in Frame that coordinates
  // system
  Frame3DTime thoProj;
  Frame3DTimeImportFrame(that, tho, &thoProj);

  // Declare two variables to memorize the system to be solved M.X <= Y
  // (M arrangement is [iRow][iCol])
  double M[14][4];
  double Y[14];

  // Create the inequality system
  // -V_jT-sum_iC_j,iX_i<=O_j
  M[0][0] = -thoProj.comp[0][0];
  M[0][1] = -thoProj.comp[1][0];
  M[0][2] = -thoProj.comp[2][0];
  M[0][3] = -thoProj.speed[0];
  Y[0] = thoProj.orig[0];
  if (Y[0] < neg(M[0][0]) + neg(M[0][1]) + neg(M[0][2]) + neg(M[0][3])) {

    return false;

  }

  M[1][0] = -thoProj.comp[0][1];
```

```
M[1][1] = -thoProj.comp[1][1];
M[1][2] = -thoProj.comp[2][1];
M[1][3] = -thoProj.speed[1];
Y[1] = thoProj.orig[1];
if (Y[1] < neg(M[1][0]) + neg(M[1][1]) + neg(M[1][2]) + neg(M[1][3])) {

  return false;

}

M[2][0] = -thoProj.comp[0][2];
M[2][1] = -thoProj.comp[1][2];
M[2][2] = -thoProj.comp[2][2];
M[2][3] = -thoProj.speed[2];
Y[2] = thoProj.orig[2];
if (Y[2] < neg(M[2][0]) + neg(M[2][1]) + neg(M[2][2]) + neg(M[2][3])) {

  return false;

}

// Variable to memorize the nb of rows in the system
int nbRows = 3;

if (that->type == FrameCuboid) {

  // V_jT+sum_iC_j,iX_i<=1.0-O_j
  M[nbRows][0] = thoProj.comp[0][0];
  M[nbRows][1] = thoProj.comp[1][0];
  M[nbRows][2] = thoProj.comp[2][0];
  M[nbRows][3] = thoProj.speed[0];
  Y[nbRows] = 1.0 - thoProj.orig[0];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2]) + neg(M[nbRows][3])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][1];
  M[nbRows][1] = thoProj.comp[1][1];
  M[nbRows][2] = thoProj.comp[2][1];
  M[nbRows][3] = thoProj.speed[1];
  Y[nbRows] = 1.0 - thoProj.orig[1];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2]) + neg(M[nbRows][3])) {

    return false;

  }

  ++nbRows;

  M[nbRows][0] = thoProj.comp[0][2];
  M[nbRows][1] = thoProj.comp[1][2];
  M[nbRows][2] = thoProj.comp[2][2];
  M[nbRows][3] = thoProj.speed[2];
  Y[nbRows] = 1.0 - thoProj.orig[2];
```

```
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2]) + neg(M[nbRows][3])) {

    return false;

  }

  ++nbRows;

} else {

  // sum_j(V_jT+sum_iC_j,iX_i)<=1.0-sum_iO_i
  M[nbRows][0] =
    thoProj.comp[0][0] + thoProj.comp[0][1] + thoProj.comp[0][2];
  M[nbRows][1] =
    thoProj.comp[1][0] + thoProj.comp[1][1] + thoProj.comp[1][2];
  M[nbRows][2] =
    thoProj.comp[2][0] + thoProj.comp[2][1] + thoProj.comp[2][2];
  M[nbRows][3] = thoProj.speed[0] + thoProj.speed[1] + thoProj.speed[2];
  Y[nbRows] = 1.0 - thoProj.orig[0] - thoProj.orig[1] - thoProj.orig[2];
  if (
    Y[nbRows] < neg(M[nbRows][0]) + neg(M[nbRows][1]) +
    neg(M[nbRows][2]) + neg(M[nbRows][3])) {

    return false;

  }

  ++nbRows;

}

if (tho->type == FrameCuboid) {

  // X_i <= 1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 0.0;
  M[nbRows][2] = 0.0;
  M[nbRows][3] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
  M[nbRows][1] = 1.0;
  M[nbRows][2] = 0.0;
  M[nbRows][3] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

  M[nbRows][0] = 0.0;
  M[nbRows][1] = 0.0;
  M[nbRows][2] = 1.0;
  M[nbRows][3] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

} else {

  // sum_iX_i<=1.0
  M[nbRows][0] = 1.0;
  M[nbRows][1] = 1.0;
```

```
  M[nbRows][2] = 1.0;
  M[nbRows][3] = 0.0;
  Y[nbRows] = 1.0;
  ++nbRows;

}

// -X_i <= 0.0
M[nbRows][0] = -1.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = -1.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = -1.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

// 0.0 <= t <= 1.0
M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 1.0;
Y[nbRows] = 1.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = -1.0;
Y[nbRows] = 0.0;
++nbRows;

// Solve the system
// Declare a AABB to memorize the bounding box of the intersection
// in the coordinates system of that
AABB3DTime bdgBoxLocal = {

  .min = {0.0, 0.0, 0.0, 0.0},
  .max = {0.0, 0.0, 0.0, 0.0}

};

// Declare variables to eliminate the first variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mp[63][4];
//double Yp[63];
double Mp[22][4];
```

```
double Yp[22];
int nbRowsP;

// Eliminate the first variable in the original system
bool inconsistency =
  ElimVar3DTime(
    M,
    Y,
    nbRows,
    4,
    Mp,
    Yp,
    &nbRowsP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Declare variables to eliminate the second variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mpp[1056][4];
//double Ypp[1056];
double Mpp[57][4];
double Ypp[57];
int nbRowsPP;

// Eliminate the second variable (which is the first in the new system)
inconsistency =
  ElimVar3DTime(
    Mp,
    Yp,
    nbRowsP,
    3,
    Mpp,
    Ypp,
    &nbRowsPP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Declare variables to eliminate the third variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mppp[279840][4];
//double Yppp[279840];
double Mppp[560][4];
double Yppp[560];
int nbRowsPPP;
```

```
// Eliminate the third variable (which is the first in the new system)
inconsistency =
  ElimVar3DTime(
    Mpp,
    Ypp,
    nbRowsPP,
    2,
    Mppp,
    Yppp,
    &nbRowsPPP);

// If the system is inconsistent
if (inconsistency == true) {

  // The two Frames are not in intersection
  return false;

}

// Get the bounds for the remaining fourth variable
GetBoundLastVar3DTime(
  FOR_VAR,
  Mppp,
  Yppp,
  nbRowsPPP,
  &bdgBoxLocal);

// If the bounds are inconsistent
if (bdgBoxLocal.min[FOR_VAR] >= bdgBoxLocal.max[FOR_VAR]) {

  // The two Frames are not in intersection
  return false;

// Else, if the bounds are consistent here it means
// the two Frames are in intersection.
// If the user has requested for the resulting bounding box
} else if (bdgBox != NULL) {

  // Get the bounds of the other variables
  GetBoundVar3DTime(
    THD_VAR,
    Mpp,
    Ypp,
    nbRowsPP,
    2,
    &bdgBoxLocal);

  GetBoundVar3DTime(
    SND_VAR,
    Mp,
    Yp,
    nbRowsP,
    3,
    &bdgBoxLocal);

  GetBoundVar3DTime(
    FST_VAR,
    M,
    Y,
    nbRows,
    4,
```

```
      &bdgBoxLocal);

    // Memorize the result
    *bdgBox = bdgBoxLocal;

  }

  // If we've reached here the two Frames are intersecting
  return true;

}
```

5 Minimal example of use

In this section I give a minimal example for each case of how to use the code given in the previous section.

5.1 2D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2d.h"

// Main function
int main(int argc, char** argv) {

  // Create the two objects to be tested for intersection
  double origP2D[2] = {0.0, 0.0};
  double compP2D[2][2] = {

    {1.0, 0.0},  // First component
    {0.0, 1.0}   // Second component

  };
  Frame2D P2D =
    Frame2DCreateStatic(
      FrameCuboid,
      origP2D,
      compP2D);

  double origQ2D[2] = {0.0, 0.0};
  double compQ2D[2][2] = {{1.0, 0.0}, {0.0, 1.0}};
  Frame2D Q2D =
    Frame2DCreateStatic(
      FrameCuboid,
      origQ2D,
      compQ2D);

  // Declare a variable to memorize the result of the intersection
  // detection
  AABB2D bdgBox2DLocal;
```

```
// Test for intersection between P and Q
bool isIntersecting2D =
  FMBTestIntersection2D(
    &P2D,
    &Q2D,
    &bdgBox2DLocal);

// If the two objects are intersecting
if (isIntersecting2D) {

  printf("Intersection detected in AABB ");

  // Export the local bounding box toward the real coordinates
  // system
  AABB2D bdgBox2D;
  Frame2DExportBdgBox(
    &Q2D,
    &bdgBox2DLocal,
    &bdgBox2D);

  // Clip with the AABB of 'Q2D' and 'P2D' to improve results
  for (
    int iAxis = 2;
    iAxis--;) {

    if (bdgBox2D.min[iAxis] < P2D.bdgBox.min[iAxis]) {

      bdgBox2D.min[iAxis] = P2D.bdgBox.min[iAxis];

    }

    if (bdgBox2D.max[iAxis] > P2D.bdgBox.max[iAxis]) {

      bdgBox2D.max[iAxis] = P2D.bdgBox.max[iAxis];

    }

    if (bdgBox2D.min[iAxis] < Q2D.bdgBox.min[iAxis]) {

      bdgBox2D.min[iAxis] = Q2D.bdgBox.min[iAxis];

    }

    if (bdgBox2D.max[iAxis] > Q2D.bdgBox.max[iAxis]) {

      bdgBox2D.max[iAxis] = Q2D.bdgBox.max[iAxis];

    }

  }

  AABB2DPrint(&bdgBox2D);
  printf("\n");

// Else, the two objects are not intersecting
} else {

  printf("No intersection.\n");

}
```

```
  return 0;

}
```

5.2 3D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb3d.h"

// Main function
int main(int argc, char** argv) {

  // Create the two objects to be tested for intersection
  double origP3D[3] = {0.0, 0.0, 0.0};
  double compP3D[3][3] = {

    {1.0, 0.0, 0.0},  // First component
    {0.0, 1.0, 0.0},  // Second component
    {0.0, 0.0, 1.0}   // Third component

  };
  Frame3D P3D =
    Frame3DCreateStatic(
      FrameTetrahedron,
      origP3D,
      compP3D);

  double origQ3D[3] = {0.0, 0.0, 0.0};
  double compQ3D[3][3] = {

    {1.0, 0.0, 0.0},
    {0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0}

  };
  Frame3D Q3D =
    Frame3DCreateStatic(
      FrameTetrahedron,
      origQ3D,
      compQ3D);

  // Declare a variable to memorize the result of the intersection
  // detection
  AABB3D bdgBox3DLocal;

  // Test for intersection between P and Q
  bool isIntersecting3D =
    FMBTestIntersection3D(
      &P3D,
      &Q3D,
      &bdgBox3DLocal);

  // If the two objects are intersecting
  if (isIntersecting3D) {
```

```
    printf("Intersection detected in AABB ");

    // Export the local bounding box toward the real coordinates
    // system
    AABB3D bdgBox3D;
    Frame3DExportBdgBox(
      &Q3D,
      &bdgBox3DLocal,
      &bdgBox3D);

    // Clip with the AABB of 'Q3D' and 'P3D' to improve results
    for (
      int iAxis = 3;
      iAxis--;) {

      if (bdgBox3D.min[iAxis] < P3D.bdgBox.min[iAxis]) {

        bdgBox3D.min[iAxis] = P3D.bdgBox.min[iAxis];

      }

      if (bdgBox3D.max[iAxis] > P3D.bdgBox.max[iAxis]) {

        bdgBox3D.max[iAxis] = P3D.bdgBox.max[iAxis];

      }

      if (bdgBox3D.min[iAxis] < Q3D.bdgBox.min[iAxis]) {

        bdgBox3D.min[iAxis] = Q3D.bdgBox.min[iAxis];

      }

      if (bdgBox3D.max[iAxis] > Q3D.bdgBox.max[iAxis]) {

        bdgBox3D.max[iAxis] = Q3D.bdgBox.max[iAxis];

      }

    }

    AABB3DPrint(&bdgBox3D);
    printf("\n");

  // Else, the two objects are not intersecting
  } else {

    printf("No intersection.\n");

  }

  return 0;

}
```

5.3 2D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
```

```
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2dt.h"

// Main function
int main(int argc, char** argv) {

  // Create the two objects to be tested for intersection
  double origP2DTime[2] = {0.0, 0.0};
  double speedP2DTime[2] = {0.0, 0.0};
  double compP2DTime[2][2] = {

    {1.0, 0.0},  // First component
    {0.0, 1.0}   // Second component

  };
  Frame2DTime P2DTime =
    Frame2DTimeCreateStatic(
      FrameCuboid,
      origP2DTime,
      speedP2DTime,
      compP2DTime);

  double origQ2DTime[2] = {0.0, 0.0};
  double speedQ2DTime[2] = {0.0, 0.0};
  double compQ2DTime[2][2] = {{1.0, 0.0}, {0.0, 1.0}};
  Frame2DTime Q2DTime =
    Frame2DTimeCreateStatic(
      FrameCuboid,
      origQ2DTime,
      speedQ2DTime,
      compQ2DTime);

  // Declare a variable to memorize the result of the intersection
  // detection
  AABB2DTime bdgBox2DTimeLocal;

  // Test for intersection between P and Q
  bool isIntersecting2DTime =
    FMBTestIntersection2DTime(
      &P2DTime,
      &Q2DTime,
      &bdgBox2DTimeLocal);

  // If the two objects are intersecting
  if (isIntersecting2DTime) {

    printf("Intersection detected in AABB ");

    // Export the local bounding box toward the real coordinates
    // system
    AABB2DTime bdgBox2DTime;
    Frame2DTimeExportBdgBox(
      &Q2DTime,
      &bdgBox2DTimeLocal,
      &bdgBox2DTime);

    // Clip with the AABB of 'Q2DTime' and 'P2DTime' to improve results
    for (
      int iAxis = 3;
      iAxis--;) {
```

```
      if (bdgBox2DTime.min[iAxis] < P2DTime.bdgBox.min[iAxis]) {

        bdgBox2DTime.min[iAxis] = P2DTime.bdgBox.min[iAxis];

      }

      if (bdgBox2DTime.max[iAxis] > P2DTime.bdgBox.max[iAxis]) {

        bdgBox2DTime.max[iAxis] = P2DTime.bdgBox.max[iAxis];

      }

      if (bdgBox2DTime.min[iAxis] < Q2DTime.bdgBox.min[iAxis]) {

        bdgBox2DTime.min[iAxis] = Q2DTime.bdgBox.min[iAxis];

      }

      if (bdgBox2DTime.max[iAxis] > Q2DTime.bdgBox.max[iAxis]) {

        bdgBox2DTime.max[iAxis] = Q2DTime.bdgBox.max[iAxis];

      }

    }

    AABB2DTimePrint(&bdgBox2DTime);
    printf("\n");

  // Else, the two objects are not intersecting
  } else {

    printf("No intersection.\n");

  }

  return 0;

}
```

5.4 3D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb3dt.h"

// Main function
int main(int argc, char** argv) {

  // Create the two objects to be tested for intersection
  double origP3DTime[3] = {0.0, 0.0, 0.0};
  double speedP3DTime[3] = {0.0, 0.0, 0.0};
  double compP3DTime[3][3] = {

    {1.0, 0.0, 0.0},  // First component
```

```
  {0.0, 1.0, 0.0},  // Second component
  {0.0, 0.0, 1.0}   // Third component

};
Frame3DTime P3DTime =
  Frame3DTimeCreateStatic(
    FrameCuboid,
    origP3DTime,
    speedP3DTime,
    compP3DTime);

double origQ3DTime[3] = {0.0, 0.0, 0.0};
double speedQ3DTime[3] = {0.0, 0.0, 0.0};
double compQ3DTime[3][3] =
  {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}};
Frame3DTime Q3DTime =
  Frame3DTimeCreateStatic(
    FrameCuboid,
    origQ3DTime,
    speedQ3DTime,
    compQ3DTime);

// Declare a variable to memorize the result of the intersection
// detection
AABB3DTime bdgBox3DTimeLocal;

// Test for intersection between P and Q
bool isIntersecting3DTime =
  FMBTestIntersection3DTime(
    &P3DTime,
    &Q3DTime,
    &bdgBox3DTimeLocal);

// If the two objects are intersecting
if (isIntersecting3DTime) {

  printf("Intersection detected in AABB ");

  // Export the local bounding box toward the real coordinates
  // system
  AABB3DTime bdgBox3DTime;
  Frame3DTimeExportBdgBox(
    &Q3DTime,
    &bdgBox3DTimeLocal,
    &bdgBox3DTime);

  // Clip with the AABB of 'Q3DTime' and 'P3DTime' to improve results
  for (
    int iAxis = 3;
    iAxis--;) {

    if (bdgBox3DTime.min[iAxis] < P3DTime.bdgBox.min[iAxis]) {

      bdgBox3DTime.min[iAxis] = P3DTime.bdgBox.min[iAxis];

    }

    if (bdgBox3DTime.max[iAxis] > P3DTime.bdgBox.max[iAxis]) {

      bdgBox3DTime.max[iAxis] = P3DTime.bdgBox.max[iAxis];

    }
```

```
      if (bdgBox3DTime.min[iAxis] < Q3DTime.bdgBox.min[iAxis]) {

        bdgBox3DTime.min[iAxis] = Q3DTime.bdgBox.min[iAxis];

      }

      if (bdgBox3DTime.max[iAxis] > Q3DTime.bdgBox.max[iAxis]) {

        bdgBox3DTime.max[iAxis] = Q3DTime.bdgBox.max[iAxis];

      }

    }

    AABB3DTimePrint(&bdgBox3DTime);
    printf("\n");

  // Else, the two objects are not intersecting
  } else {

    printf("No intersection.\n");

  }

  return 0;

}
```

6 Unit tests

In this section I introduce the code I've used to test the algorithm and its implementation. The test consists of running the algorithm on a set of cases for which the solution has been computed by hand. The code of the implementation of the SAT algorithm is given in annex (p.279)

6.1 Code

6.1.1 2D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2d.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {
```

```
  FrameType type;
  double orig[2];
  double comp[2][2];

} Param2D;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection algorihtm on the Frames
// and check against the correct results
void UnitTest2D(
  const Param2D paramP,
  const Param2D paramQ,
  const bool correctAnswer,
  const AABB2D* const correctBdgBox) {

  // Create the two Frames
  Frame2D P =
    Frame2DCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.comp);

  Frame2D Q =
    Frame2DCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.comp);

  // Declare a variable to memorize the resulting bounding box
  AABB2D bdgBoxLocal;

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame2D* that = &P;
  Frame2D* tho = &Q;

  // Loop on pairs of Frames
  for (
    int iPair = 2;
    iPair--;) {

    // Display the tested frames
    Frame2DPrint(that);
    printf("\nagainst\n");
    Frame2DPrint(tho);
    printf("\n");

    // Run the FMB intersection test
    bool isIntersecting =
      FMBTestIntersection2D(
        that,
        tho,
        &bdgBoxLocal);

    // If the test hasn't given the expected answer about intersection
    if (isIntersecting != correctAnswer) {

      // Display information about the failure
      printf(" Failed\n");
      printf("Expected : ");
      if (correctAnswer == false) printf("no ");
```

```
  printf("intersection\n");
  printf("Got : ");
  if (isIntersecting == false) printf("no ");
  printf("intersection\n");
  exit(0);

// Else, the test has given the expected answer about intersection
} else {

  // If the Frames were intersecting
  if (isIntersecting == true) {

    AABB2D bdgBox;
    Frame2DExportBdgBox(
      tho,
      &bdgBoxLocal,
      &bdgBox);

    for (
      int iAxis = 2;
      iAxis--;) {

      if (bdgBox.min[iAxis] < that->bdgBox.min[iAxis]) {

        bdgBox.min[iAxis] = that->bdgBox.min[iAxis];

      }

      if (bdgBox.max[iAxis] > that->bdgBox.max[iAxis]) {

        bdgBox.max[iAxis] = that->bdgBox.max[iAxis];

      }

      if (bdgBox.min[iAxis] < tho->bdgBox.min[iAxis]) {

        bdgBox.min[iAxis] = tho->bdgBox.min[iAxis];

      }

      if (bdgBox.max[iAxis] > tho->bdgBox.max[iAxis]) {

        bdgBox.max[iAxis] = tho->bdgBox.max[iAxis];

      }

    }

    // Check the bounding box
    bool flag = true;
    for (
      int i = 2;
      i--;) {

      if (
        bdgBox.min[i] > correctBdgBox->min[i] + EPSILON ||
        bdgBox.max[i] < correctBdgBox->max[i] - EPSILON) {

        flag = false;

      }
```

```
        }

        // If the bounding box is the expected one
        if (flag == true) {

          // Display information
          printf("Succeed\n");
          AABB2DPrint(&bdgBox);
          printf("\n");

        // Else, the bounding box wasn't the expected one
        } else {

          // Display information
          printf("Failed\n");
          printf("Expected : ");
          AABB2DPrint(correctBdgBox);
          printf("\n");
          printf("     Got : ");
          AABB2DPrint(&bdgBox);

          // Terminate the unit tests
          exit(0);

        }

      // Else the Frames were not intersected,
      // no need to check the bounding box
      } else {

        // Display information
        printf(" Succeed (no inter)\n");

      }

    }

    printf("\n");

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Test2D(void) {

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param2D paramP;
  Param2D paramQ;

  // Declare a variable to memorize the correct bounding box
  AABB2D correctBdgBox;

  // Execute the unit test on various cases
  // ----------------------------
  paramP = (Param2D) {

    .type = FrameCuboid,
```

```
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.5, 0.5},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.5},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {-0.5, -0.5},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.5, 0.5},
```

```
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
UnitTest2D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.25, -0.25},
  .comp = {{0.5, 0.0}, {0.0, 2.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.25, 0.0},
  .max = {0.75, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {-0.25, 0.25},
  .comp = {{2.0, 0.0}, {0.0, 0.5}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.25},
  .max = {1.0, 0.75}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);
```

```
// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 1.0}, {-1.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {-0.5, -0.5},
  .comp = {{1.0, 1.0}, {-1.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {0.5, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.5, 1.5},
  .comp = {{1.0, -1.0}, {-1.0, -1.0}}
```

```
};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.0, 0.0},
  .comp = {{-1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.0},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.0, 0.5},
  .comp = {{-0.5, 0.5}, {-0.5, -0.5}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 1.0},
  .comp = {{1.0, 0.0}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {1.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {2.0, -1.0},
  .comp = {{0.0, 1.0}, {-0.5, 1.0}}

};
```

```
correctBdgBox = (AABB2D) {

  .min = {1.5, 0.5},
  .max = {1.5 + 0.5 / 3.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.0, 1.0},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.25},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.0, 2.0},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.75},
  .max = {1.0, 1.25}

};
UnitTest2D(
  paramP,
```

```
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameCuboid,
  .orig = {1.0, 2.0},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.5},
  .max = {0.75, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {1.0, 2.0},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5 + 1.0 / 3.0, 1.0},
  .max = {1.0, 1.0 + 1.0 / 3.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
```

```
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, -0.5},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {0.5, 0.5}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.5, 0.5},
  .comp = {{-0.5, 0.0}, {0.0, -0.5}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, -0.5},
```

```
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.0, 0.0},
  .max = {0.5, 0.5}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.5, 0.5},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
UnitTest2D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param2D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {1.5, 1.5},
  .comp = {{-1.5, 0.0}, {0.0, -1.5}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.5},
  .max = {1.0, 1.0}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);
```

```
// ----------------------------
paramP = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {1.01, 1.01},
  .comp = {{-1.0, 0.0}, {0.0, -1.0}}

};
UnitTest2D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {1.0, 1.0},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

  .min = {0.5, 0.5 - 1.0 / 6.0},
  .max = {1.0, 0.75}

};
UnitTest2D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

  .type = FrameTetrahedron,
  .orig = {1.01, 1.5},
  .comp = {{-0.5, -0.5}, {0.0, -1.0}}
```

```
  };
  UnitTest2D(
    paramP,
    paramQ,
    false,
    NULL);

  // If we reached here, it means all the unit tests succeed
  printf("All unit tests 2D have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

  Test2D();

  return 0;

}
```

6.1.2 3D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb3d.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];

} Param3D;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection alogirhtm on the Frames
// and check against the correct results
void UnitTest3D(
  const Param3D paramP,
  const Param3D paramQ,
  const bool correctAnswer,
  const AABB3D* const correctBdgBox) {

  // Create the two Frames
  Frame3D P =
    Frame3DCreateStatic(
      paramP.type,
      paramP.orig,
```

```
    paramP.comp);

Frame3D Q =
  Frame3DCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.comp);

// Declare a variable to memorize the resulting bounding box
AABB3D bdgBoxLocal;

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame3D* that = &P;
Frame3D* tho = &Q;

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Display the tested frames
  Frame3DPrint(that);
  printf("\nagainst\n");
  Frame3DPrint(tho);
  printf("\n");

  // Run the FMB intersection test
  bool isIntersecting =
    FMBTestIntersection3D(
      that,
      tho,
      &bdgBoxLocal);

  // If the test hasn't given the expected answer about intersection
  if (isIntersecting != correctAnswer) {

    // Display information about the failure
    printf(" Failed\n");
    printf("Expected : ");
    if (correctAnswer == false) printf("no ");
    printf("intersection\n");
    printf("Got : ");
    if (isIntersecting == false) printf("no ");
    printf("intersection\n");
    exit(0);

  // Else, the test has given the expected answer about intersection
  } else {

    // If the Frames were intersecting
    if (isIntersecting == true) {

      AABB3D bdgBox;
      Frame3DExportBdgBox(
        tho,
        &bdgBoxLocal,
        &bdgBox);

      for (
        int iAxis = 3;
        iAxis--;) {
```

```
  if (bdgBox.min[iAxis] < that->bdgBox.min[iAxis]) {

    bdgBox.min[iAxis] = that->bdgBox.min[iAxis];

  }

  if (bdgBox.max[iAxis] > that->bdgBox.max[iAxis]) {

    bdgBox.max[iAxis] = that->bdgBox.max[iAxis];

  }

  if (bdgBox.min[iAxis] < tho->bdgBox.min[iAxis]) {

    bdgBox.min[iAxis] = tho->bdgBox.min[iAxis];

  }

  if (bdgBox.max[iAxis] > tho->bdgBox.max[iAxis]) {

    bdgBox.max[iAxis] = tho->bdgBox.max[iAxis];

  }

}

// Check the bounding box
bool flag = true;
for (
  int i = 3;
  i--;) {

  if (
    bdgBox.min[i] > correctBdgBox->min[i] + EPSILON ||
    bdgBox.max[i] < correctBdgBox->max[i] - EPSILON) {

    flag = false;

  }

}

// If the bounding box is the expected one
if (flag == true) {

  // Display information
  printf("Succeed\n");
  AABB3DPrint(&bdgBox);
  printf("\n");

// Else, the bounding box wasn't the expected one
} else {

  // Display information
  printf("Failed\n");
  printf("Expected : ");
  AABB3DPrint(correctBdgBox);
  printf("\n");
  printf("     Got : ");
  AABB3DPrint(&bdgBox);
  printf("\n");
```

```
          // Terminate the unit tests
          exit(0);

        }

      // Else the Frames were not intersected,
      // no need to check the bounding box
      } else {

        // Display information
        printf(" Succeed (no inter)\n");

      }

    }

    printf("\n");

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Test3D(void) {

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param3D paramP;
  Param3D paramQ;

  // Declare a variable to memorize the correct bounding box
  AABB3D correctBdgBox;

  // Execute the unit test on various cases
  // ----------------------------
  paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

  };
  paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

  };
  correctBdgBox = (AABB3D) {

    .min = {0.0, 0.0, 0.0},
    .max = {1.0, 1.0, 1.0}

  };
  UnitTest3D(
    paramP,
    paramQ,
    true,
```

```
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.5, 0.5, 0.5},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.5, 0.5, 0.5},
  .max = {1.0, 1.0, 1.0}

};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {-0.5, -0.5, -0.5},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.0, 0.0, 0.0},
  .max = {0.5, 0.5, 0.5}

};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}
```

```
};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {1.5, 1.5, 1.5},
  .comp = {{-1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, -1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.5, 0.5, 0.5},
  .max = {1.0, 1.0, 1.0}

};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.5, 1.5, -1.5},
  .comp = {{1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, -1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.5, 1.5, -1.5},
  .comp = {{1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.5, 0.5, -1.0},
  .max = {1.0, 1.0, -0.5}
```

```
};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {-1.01, -1.01, -1.01},
  .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {-1.0, -1.0, -1.0},
  .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, -0.5, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.0, -0.5, 0.0},
  .max = {1.0, 0.0, 1.0}

};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameTetrahedron,
  .orig = {-1.0, -1.0, -1.0},
  .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}
```

```
};
paramQ = (Param3D) {

  .type = FrameCuboid,
  .orig = {0.0, -0.5, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param3D) {

  .type = FrameCuboid,
  .orig = {-1.0, -1.0, -1.0},
  .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameTetrahedron,
  .orig = {0.0, -0.5, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

  .min = {0.0, -0.5, 0.0},
  .max = {0.75, 0.0, 0.75}

};
UnitTest3D(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3D) {

  .type = FrameTetrahedron,
  .orig = {-1.0, -1.0, -1.0},
  .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

  .type = FrameTetrahedron,
  .orig = {0.0, -0.5, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
  paramP,
  paramQ,
  false,
  NULL);
```

```
  // ----------------------------
  paramP = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {-0.5, -1.0, -0.5},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

  };
  paramQ = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

  };
  correctBdgBox = (AABB3D) {

    .min = {0.0, -0.5, 0.0},
    .max = {0.5, -1.0 / 3.0, 0.5}

  };
  UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

  // If we reached here, it means all the unit tests succeed
  printf("All unit tests 3D have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

  Test3D();

  return 0;

}
```

6.1.3 2D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2dt.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];
```

```
  double speed[2];

} Param2DTime;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection algorihtm on the Frames
// and check against the correct results
void UnitTest2DTime(
  const Param2DTime paramP,
  const Param2DTime paramQ,
  const bool correctAnswer,
  const AABB2DTime* const correctBdgBox) {

  // Create the two Frames
  Frame2DTime P =
    Frame2DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);

  Frame2DTime Q =
    Frame2DTimeCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.speed,
      paramQ.comp);

  // Declare a variable to memorize the resulting bounding box
  AABB2DTime bdgBoxLocal;

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame2DTime* that = &P;
  Frame2DTime* tho = &Q;

  // Loop on pairs of Frames
  for (
    int iPair = 2;
    iPair--;) {

    // Display the tested frames
    Frame2DTimePrint(that);
    printf("\nagainst\n");
    Frame2DTimePrint(tho);
    printf("\n");

    // Run the FMB intersection test
    bool isIntersecting =
      FMBTestIntersection2DTime(
        that,
        tho,
        &bdgBoxLocal);

    // If the test hasn't given the expected answer about intersection
    if (isIntersecting != correctAnswer) {

      // Display information about the failure
      printf(" Failed\n");
      printf("Expected : ");
      if (correctAnswer == false) printf("no ");
```

```
  printf("intersection\n");
  printf("Got : ");
  if (isIntersecting == false) printf("no ");
  printf("intersection\n");
  exit(0);

// Else, the test has given the expected answer about intersection
} else {

  // If the Frames were intersecting
  if (isIntersecting == true) {

    AABB2DTime bdgBox;
    Frame2DTimeExportBdgBox(
      tho,
      &bdgBoxLocal,
      &bdgBox);

    // Check the bounding box
    bool flag = true;
    for (
      int i = 3;
      i--;) {

      if (
        bdgBox.min[i] > correctBdgBox->min[i] + EPSILON ||
        bdgBox.max[i] < correctBdgBox->max[i] - EPSILON) {

        flag = false;

      }

    }

    // If the bounding box is the expected one
    if (flag == true) {

      // Display information
      printf("Succeed\n");
      AABB2DTimePrint(&bdgBox);
      printf("\n");

    // Else, the bounding box wasn't the expected one
    } else {

      // Display information
      printf("Failed\n");
      printf("Expected : ");
      AABB2DTimePrint(correctBdgBox);
      printf("\n");
      printf("     Got : ");
      AABB2DTimePrint(&bdgBox);
      printf("\n");

      // Terminate the unit tests
      exit(0);

    }

  // Else the Frames were not intersected,
  // no need to check the bounding box
  } else {
```

```
        // Display information
        printf(" Succeed (no inter)\n");

      }

    }

    printf("\n");

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Test2DTime(void) {

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param2DTime paramP;
  Param2DTime paramQ;

  // Declare a variable to memorize the correct bounding box
  AABB2DTime correctBdgBox;

  // Execute the unit test on various cases
  // ----------------------------
  paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

  };
  paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {-1.0, 0.0}

  };
  UnitTest2DTime(
    paramP,
    paramQ,
    false,
    NULL);

  // ----------------------------
  paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

  };
  paramQ = (Param2DTime) {
```

```
  .type = FrameCuboid,
  .orig = {-1.01, -1.01},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {1.0, 0.0}

};
UnitTest2DTime(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {0.0, 0.0}

};
paramQ = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {-1.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {1.0, 0.0}

};
correctBdgBox = (AABB2DTime) {

  .min = {0.0, 0.0, 0.0},
  .max = {1.0, 1.0, 1.0}

};
UnitTest2DTime(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {0.0, 0.0}

};
paramQ = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {-1.0, 0.25},
  .comp = {{0.5, 0.0}, {0.0, 0.5}},
  .speed = {4.0, 0.0}

};
correctBdgBox = (AABB2DTime) {

  .min = {0.0, 0.25, 0.125},
```

```
  .max = {1.0, 0.75, 0.5}

};
UnitTest2DTime(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {0.0, 0.0}

};
paramQ = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.25, -1.0},
  .comp = {{0.5, 0.0}, {0.0, 0.5}},
  .speed = {0.0, 4.0}

};
correctBdgBox = (AABB2DTime) {

  .min = {0.25, 0.0, 0.125},
  .max = {0.75, 1.0, 0.5}

};
UnitTest2DTime(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0},
  .comp = {{1.0, 0.0}, {0.0, 1.0}},
  .speed = {0.0, 0.0}

};
paramQ = (Param2DTime) {

  .type = FrameCuboid,
  .orig = {0.9, -1.0},
  .comp = {{0.5, 0.0}, {0.0, 0.5}},
  .speed = {0.0, 4.0}

};
correctBdgBox = (AABB2DTime) {

  .min = {0.9, 0.0, 0.125},
  .max = {1.0, 1.0, 0.5}

};
UnitTest2DTime(
```

```
    paramP,
    paramQ,
    true,
    &correctBdgBox);

  // If we reached here, it means all the unit tests succeed
  printf("All unit tests 2DTime have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

  Test2DTime();

  return 0;

}
```

6.1.4 3D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb3dt.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];
  double speed[3];

} Param3DTime;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection algorihtm on the Frames
// and check against the correct results
void UnitTest3DTime(
  const Param3DTime paramP,
  const Param3DTime paramQ,
  const bool correctAnswer,
  const AABB3DTime* const correctBdgBox) {

  // Create the two Frames
  Frame3DTime P =
    Frame3DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);
```

```
Frame3DTime Q =
  Frame3DTimeCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.speed,
    paramQ.comp);

// Declare a variable to memorize the resulting bounding box
AABB3DTime bdgBoxLocal;

// Helper variables to loop on the pairs (that, tho) and (tho, that)
Frame3DTime* that = &P;
Frame3DTime* tho = &Q;

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Display the tested frames
  Frame3DTimePrint(that);
  printf("\nagainst\n");
  Frame3DTimePrint(tho);
  printf("\n");

  // Run the FMB intersection test
  bool isIntersecting =
    FMBTestIntersection3DTime(
      that,
      tho,
      &bdgBoxLocal);

  // If the test hasn't given the expected answer about intersection
  if (isIntersecting != correctAnswer) {

    // Display information about the failure
    printf(" Failed\n");
    printf("Expected : ");
    if (correctAnswer == false) printf("no ");
    printf("intersection\n");
    printf("Got : ");
    if (isIntersecting == false) printf("no ");
    printf("intersection\n");
    exit(0);

  // Else, the test has given the expected answer about intersection
  } else {

    // If the Frames were intersecting
    if (isIntersecting == true) {

      AABB3DTime bdgBox;
      Frame3DTimeExportBdgBox(
        tho,
        &bdgBoxLocal,
        &bdgBox);

      // Check the bounding box
      bool flag = true;
      for (
        int i = 4;
```

```
          i--;) {

          if (
            bdgBox.min[i] > correctBdgBox->min[i] + EPSILON ||
            bdgBox.max[i] < correctBdgBox->max[i] - EPSILON) {

            flag = false;

          }

        }

        // If the bounding box is the expected one
        if (flag == true) {

          // Display information
          printf("Succeed\n");
          AABB3DTimePrint(&bdgBox);
          printf("\n");

        // Else, the bounding box wasn't the expected one
        } else {

          // Display information
          printf("Failed\n");
          printf("Expected : ");
          AABB3DTimePrint(correctBdgBox);
          printf("\n");
          printf("     Got : ");
          AABB3DTimePrint(&bdgBox);
          printf("\n");

          // Terminate the unit tests
          exit(0);

        }

      // Else the Frames were not intersected,
      // no need to check the bounding box
      } else {

        // Display information
        printf(" Succeed (no inter)\n");

      }

    }

    printf("\n");

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Test3DTime(void) {

  // Declare two variables to memorize the arguments to the
  // Validation function
```

```
Param3DTime paramP;
Param3DTime paramQ;

// Declare a variable to memorize the correct bounding box
AABB3DTime correctBdgBox;

// Execute the unit test on various cases
// ----------------------------
paramP = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {-1.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {-1.0, 0.0, 0.0}

};
UnitTest3DTime(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {-1.01, -1.01, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {1.0, 0.0, 0.0}

};
UnitTest3DTime(
  paramP,
  paramQ,
  false,
  NULL);

// ----------------------------
paramP = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {0.0, 0.0, 0.0}

};
```

```
paramQ = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {-1.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {1.0, 0.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

  .min = {0.0, 0.0, 0.0, 0.0},
  .max = {1.0, 1.0, 1.0, 1.0}

};
UnitTest3DTime(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {-1.0, 0.25, 0.0},
  .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {4.0, 0.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

  .min = {0.0, 0.25, 0.0, 0.125},
  .max = {1.0, 0.75, 1.0, 0.5}

};
UnitTest3DTime(
  paramP,
  paramQ,
  true,
  &correctBdgBox);

// ----------------------------
paramP = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.0, 0.0, 0.0},
  .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
  .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

  .type = FrameCuboid,
  .orig = {0.25, -1.0, 0.0},
```

```
    .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 4.0, 0.0}

  };
  correctBdgBox = (AABB3DTime) {

    .min = {0.25, 0.0, 0.0, 0.125},
    .max = {0.75, 1.0, 1.0, 0.5}

  };
  UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

  // ----------------------------
  paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

  };
  paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.9, -1.0, 0.0},
    .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 4.0, 0.0}

  };
  correctBdgBox = (AABB3DTime) {

    .min = {0.9, 0.0, 0.0, 0.125},
    .max = {1.0, 1.0, 1.0, 0.5}

  };
  UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

  // If we reached here, it means all the unit tests succeed
  printf("All unit tests 3DTime have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

  Test3DTime();

  return 0;

}
```

6.2 Results

6.2.1 2D static

```
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)

Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)
```

```
Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
against
Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)
against
Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
against
Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
against
Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
against
Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
Succeed
minXY(1.500000,0.000000)-maxXY(1.666667,1.000000)

Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
Succeed
minXY(1.500000,0.500000)-maxXY(2.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
```

```
against
Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
```

```
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
 Succeed (no inter)

To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
```

```
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
 Succeed (no inter)

To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
 Succeed (no inter)

All unit tests 2D have succeed.
```

6.2.2 3D static

```
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)

Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
```

```
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
```

```
 Succeed (no inter)

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,0.750000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)
```

```
To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

All unit tests 3D have succeed.
```

6.2.3 2D dynamic

```
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(2.000000,1.000000,1.000000)

Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
```

```
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(1.000000,1.000000,1.000000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
Succeed
minXYT(-1.500000,0.000000,0.125000)-maxXYT(2.500000,1.000000,0.500000)

Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(-0.500000,0.000000,0.125000)-maxXYT(1.500000,1.000000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.000000,2.500000,0.500000)

Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.000000,1.500000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.400000,2.500000,0.500000)

Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.400000,1.500000,0.500000)

All unit tests 2DTime have succeed.
```

6.2.4 3D dynamic

```
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
```

```
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
    (2.000000,1.000000,1.000000,1.000000)

Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
    (1.000000,1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.500000,0.000000,0.000000,0.125000)-maxXYZT
    (2.500000,1.000000,1.000000,0.500000)
```

```
Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-0.500000,0.000000,0.000000,0.125000)-maxXYZT
    (1.500000,1.000000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
    (1.000000,2.500000,1.000000,0.500000)

Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT
    (1.000000,1.500000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
    (1.400000,2.500000,1.000000,0.500000)

Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT
    (1.400000,1.500000,1.000000,0.500000)

All unit tests 3DTime have succeed.
```

7 Validation against SAT

In this section I introduce the code I've used to validate the algorithm and its implementation. The validation consists of running the FMB algorithm on randomly generated pairs of Frame and check that its result is equal to the one of running the SAT algorithm on the same pair of Frames. The code of the implementation of the SAT algorithm is given in annex (p.279)

7.1 Code

7.1.1 2D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include FMB and SAT algorithm library
#include "fmb2d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];

} Param2D;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair2D(
  const Param2D paramP,
  const Param2D paramQ) {
```

```
// Create the two Frames
Frame2D P =
  Frame2DCreateStatic(
    paramP.type,
    paramP.orig,
    paramP.comp);

Frame2D Q =
  Frame2DCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.comp);

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame2D* that = &P;
Frame2D* tho = &Q;

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Test intersection with FMB
  bool isIntersectingFMB =
    FMBTestIntersection2D(
      that,
      tho,
      NULL);

  // Test intersection with SAT
  bool isIntersectingSAT =
    SATTestIntersection2D(
      that,
      tho);

  // If the results are different
  if (isIntersectingFMB != isIntersectingSAT) {

    // Print the disagreement
    printf("Validation2D has failed\n");
    Frame2DPrint(that);
    printf(" against ");
    Frame2DPrint(tho);
    printf("\n");
    printf("FMB : ");
    if (isIntersectingFMB == false) printf("no ");
    printf("intersection\n");
    printf("SAT : ");
    if (isIntersectingSAT == false) printf("no ");
    printf("intersection\n");

    // Stop the validation
    exit(0);

  }

  // If the Frames are in intersection
  if (isIntersectingFMB == true) {

    // Update the number of intersection
    nbInter++;
```

```
    // If the Frames are not in intersection
    } else {

      // Update the number of no intersection
      nbNoInter++;

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Validate2D(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param2D paramP;
  Param2D paramQ;

  // Initialize the number of intersection and no intersection
  nbInter = 0;
  nbNoInter = 0;

  // Loop on the tests
  for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param2D* param = &paramP;
    for (
      int iParam = 2;
      iParam--;) {

      // 50% chance of being a Cuboid or a Tetrahedron
      if (rnd() < 0.5) {

        param->type = FrameCuboid;

      } else {

        param->type = FrameTetrahedron;

      }

      for (
        int iAxis = 2;
        iAxis--;) {

        param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        for (
          int iComp = 2;
          iComp--;) {
```

```
          param->comp[iComp][iAxis] =
            -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        }

      }

      param = &paramQ;

    }

    // Calculate the determinant of the Frames' components matrix
    double detP =
      paramP.comp[0][0] * paramP.comp[1][1] -
      paramP.comp[1][0] * paramP.comp[0][1];

    double detQ =
      paramQ.comp[0][0] * paramQ.comp[1][1] -
      paramQ.comp[1][0] * paramQ.comp[0][1];

    // If the determinants are not null, ie the Frame are not degenerate
    if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

      // Run the validation on the two Frames
      ValidationOnePair2D(
        paramP,
        paramQ);

    }

  }

  // If we reached here it means the validation was successfull
  // Print results
  printf("Validation2D has succeed.\n");
  printf("Tested %lu intersections ", nbInter);
  printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

  printf("===== 2D static ======\n");
  Validate2D();

  return 0;

}
```

7.1.2 3D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include FMB and SAT algorithm library
#include "fmb3d.h"
```

```
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];

} Param3D;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair3D(
  const Param3D paramP,
  const Param3D paramQ) {

  // Create the two Frames
  Frame3D P =
    Frame3DCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.comp);

  Frame3D Q =
    Frame3DCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.comp);

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame3D* that = &P;
  Frame3D* tho = &Q;

  // Loop on pairs of Frames
  for (
    int iPair = 2;
    iPair--;) {

    // Test intersection with FMB
    bool isIntersectingFMB =
      FMBTestIntersection3D(
        that,
        tho,
```

```
        NULL);

    // Test intersection with SAT
    bool isIntersectingSAT =
      SATTestIntersection3D(
        that,
        tho);

    // If the results are different
    if (isIntersectingFMB != isIntersectingSAT) {

      // Print the disagreement
      printf("Validation3D has failed\n");
      Frame3DPrint(that);
      printf(" against ");
      Frame3DPrint(tho);
      printf("\n");
      printf("FMB : ");
      if (isIntersectingFMB == false) printf("no ");
      printf("intersection\n");
      printf("SAT : ");
      if (isIntersectingSAT == false) printf("no ");
      printf("intersection\n");

      // Stop the validation
      exit(0);

    }

    // If the Frames are in intersection
    if (isIntersectingFMB == true) {

      // Update the number of intersection
      nbInter++;

    // If the Frames are not in intersection
    } else {

      // Update the number of no intersection
      nbNoInter++;

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Validate3D(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param3D paramP;
  Param3D paramQ;

  // Initialize the number of intersection and no intersection
```

```
nbInter = 0;
nbNoInter = 0;

// Loop on the tests
for (
  unsigned long iTest = NB_TESTS;
  iTest--;) {

  // Create two random Frame definitions
  Param3D* param = &paramP;
  for (
    int iParam = 2;
    iParam--;) {

    // 50% chance of being a Cuboid or a Tetrahedron
    if (rnd() < 0.5) {

      param->type = FrameCuboid;

    } else {

      param->type = FrameTetrahedron;

    }

    for (
      int iAxis = 3;
      iAxis--;) {

      param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      for (
        int iComp = 3;
        iComp--;) {

        param->comp[iComp][iAxis] =
          -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      }

    }

    param = &paramQ;

  }

  // Calculate the determinant of the Frames' components matrix
  double detP =
    paramP.comp[0][0] * (paramP.comp[1][1] * paramP.comp[2][2] -
    paramP.comp[1][2] * paramP.comp[2][1]) -
    paramP.comp[1][0] * (paramP.comp[0][1] * paramP.comp[2][2] -
    paramP.comp[0][2] * paramP.comp[2][1]) +
    paramP.comp[2][0] * (paramP.comp[0][1] * paramP.comp[1][2] -
    paramP.comp[0][2] * paramP.comp[1][1]);

  double detQ =
    paramQ.comp[0][0] * (paramQ.comp[1][1] * paramQ.comp[2][2] -
    paramQ.comp[1][2] * paramQ.comp[2][1]) -
    paramQ.comp[1][0] * (paramQ.comp[0][1] * paramQ.comp[2][2] -
    paramQ.comp[0][2] * paramQ.comp[2][1]) +
    paramQ.comp[2][0] * (paramQ.comp[0][1] * paramQ.comp[1][2] -
    paramQ.comp[0][2] * paramQ.comp[1][1]);
```

```
    // If the determinants are not null, ie the Frame are not degenerate
    if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

      // Run the validation on the two Frames
      ValidationOnePair3D(
        paramP,
        paramQ);

    }

  }

  // If we reached here it means the validation was successfull
  // Print results
  printf("Validation3D has succeed.\n");
  printf("Tested %lu intersections ", nbInter);
  printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

  printf("===== 3D static ======\n");
  Validate3D();

  return 0;

}
```

7.1.3 2D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include the FMB and SAT algorithm library
#include "fmb2dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
```

```
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];
  double speed[2];

} Param2DTime;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair2DTime(
  const Param2DTime paramP,
  const Param2DTime paramQ) {

  // Create the two Frames
  Frame2DTime P =
    Frame2DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);

  Frame2DTime Q =
    Frame2DTimeCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.speed,
      paramQ.comp);

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame2DTime* that = &P;
  Frame2DTime* tho = &Q;

  // Loop on pairs of Frames
  for (
    int iPair = 2;
    iPair--;) {

    // Test intersection with FMB
    bool isIntersectingFMB =
      FMBTestIntersection2DTime(
        that,
        tho,
        NULL);

    // Test intersection with SAT
    bool isIntersectingSAT =
      SATTestIntersection2DTime(
        that,
        tho);

    // If the results are different
    if (isIntersectingFMB != isIntersectingSAT) {

      // Print the disagreement
      printf("Validation2D has failed\n");
      Frame2DTimePrint(that);
      printf(" against ");
      Frame2DTimePrint(tho);
      printf("\n");
```

```
      printf("FMB : ");
      if (isIntersectingFMB == false) printf("no ");
      printf("intersection\n");
      printf("SAT : ");
      if (isIntersectingSAT == false) printf("no ");
      printf("intersection\n");

      // Stop the validation
      exit(0);

    }

    // If the Frames are in intersection
    if (isIntersectingFMB == true) {

      // Update the number of intersection
      nbInter++;

    // If the Frames are not in intersection
    } else {

      // Update the number of no intersection
      nbNoInter++;

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Validate2DTime(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param2DTime paramP;
  Param2DTime paramQ;

  // Initialize the number of intersection and no intersection
  nbInter = 0;
  nbNoInter = 0;

  // Loop on the tests
  for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param2DTime* param = &paramP;
    for (
      int iParam = 2;
      iParam--;) {

      // 50% chance of being a Cuboid or a Tetrahedron
      if (rnd() < 0.5) {
```

```
        param->type = FrameCuboid;

      } else {

        param->type = FrameTetrahedron;

      }

      for (
        int iAxis = 2;
        iAxis--;) {

        param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;
        param->speed[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        for (
          int iComp = 2;
          iComp--;) {

          param->comp[iComp][iAxis] =
            -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        }

      }

      param = &paramQ;

    }

    // Calculate the determinant of the Frames' components matrix
    double detP =
      paramP.comp[0][0] * paramP.comp[1][1] -
      paramP.comp[1][0] * paramP.comp[0][1];

    double detQ =
      paramQ.comp[0][0] * paramQ.comp[1][1] -
      paramQ.comp[1][0] * paramQ.comp[0][1];

    // If the determinants are not null, ie the Frame are not degenerate
    if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

      // Run the validation on the two Frames
      ValidationOnePair2DTime(
        paramP,
        paramQ);

    }

  }

  // If we reached here it means the validation was successfull
  // Print results
  printf("Validation2DTime has succeed.\n");
  printf("Tested %lu intersections ", nbInter);
  printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

  printf("===== 2D dynamic ======\n");
```

```
  Validate2DTime();

  return 0;

}
```

7.1.4 3D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include the FMB and SAT algorithm library
#include "fmb3dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];
  double speed[3];

} Param3DTime;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair3DTime(
  const Param3DTime paramP,
  const Param3DTime paramQ) {

  // Create the two Frames
  Frame3DTime P =
    Frame3DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);
```

```
Frame3DTime Q =
  Frame3DTimeCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.speed,
    paramQ.comp);

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame3DTime* that = &P;
Frame3DTime* tho = &Q;

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Test intersection with FMB
  bool isIntersectingFMB =
    FMBTestIntersection3DTime(
      that,
      tho,
      NULL);

  // Test intersection with SAT
  bool isIntersectingSAT =
    SATTestIntersection3DTime(
      that,
      tho);

  // If the results are different
  if (isIntersectingFMB != isIntersectingSAT) {

    // Print the disagreement
    printf("Validation3D has failed\n");
    Frame3DTimePrint(that);
    printf(" against ");
    Frame3DTimePrint(tho);
    printf("\n");
    printf("FMB : ");
    if (isIntersectingFMB == false) printf("no ");
    printf("intersection\n");
    printf("SAT : ");
    if (isIntersectingSAT == false) printf("no ");
    printf("intersection\n");

    // Stop the validation
    exit(0);

  }

  // If the Frames are in intersection
  if (isIntersectingFMB == true) {

    // Update the number of intersection
    nbInter++;

  // If the Frames are not in intersection
  } else {

    // Update the number of no intersection
    nbNoInter++;
```

```
    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Validate3DTime(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Declare two variables to memorize the arguments to the
  // Validation function
  Param3DTime paramP;
  Param3DTime paramQ;

  // Initialize the number of intersection and no intersection
  nbInter = 0;
  nbNoInter = 0;

  // Loop on the tests
  for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param3DTime* param = &paramP;
    for (
      int iParam = 2;
      iParam--;) {

      // 50% chance of being a Cuboid or a Tetrahedron
      if (rnd() < 0.5) {

        param->type = FrameCuboid;

      } else {

        param->type = FrameTetrahedron;

      }

      for (
        int iAxis = 3;
        iAxis--;) {

        param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;
        param->speed[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        for (
          int iComp = 3;
          iComp--;) {

          param->comp[iComp][iAxis] =
            -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

        }
```

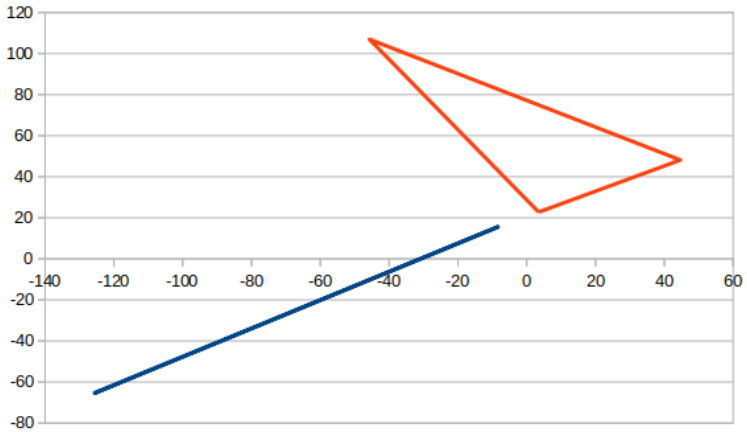
```
      }

      param = &paramQ;

    }

    // Calculate the determinant of the Frames' components matrix
    double detP =
      paramP.comp[0][0] * (paramP.comp[1][1] * paramP.comp[2][2] -
      paramP.comp[1][2] * paramP.comp[2][1]) -
      paramP.comp[1][0] * (paramP.comp[0][1] * paramP.comp[2][2] -
      paramP.comp[0][2] * paramP.comp[2][1]) +
      paramP.comp[2][0] * (paramP.comp[0][1] * paramP.comp[1][2] -
      paramP.comp[0][2] * paramP.comp[1][1]);

    double detQ =
      paramQ.comp[0][0] * (paramQ.comp[1][1] * paramQ.comp[2][2] -
      paramQ.comp[1][2] * paramQ.comp[2][1]) -
      paramQ.comp[1][0] * (paramQ.comp[0][1] * paramQ.comp[2][2] -
      paramQ.comp[0][2] * paramQ.comp[2][1]) +
      paramQ.comp[2][0] * (paramQ.comp[0][1] * paramQ.comp[1][2] -
      paramQ.comp[0][2] * paramQ.comp[1][1]);

    // If the determinants are not null, ie the Frame are not degenerate
    if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

      // Run the validation on the two Frames
      ValidationOnePair3DTime(
        paramP,
        paramQ);

    }

  }

  // If we reached here it means the validation was successfull
  // Print results
  printf("Validation3DTime has succeed.\n");
  printf("Tested %lu intersections ", nbInter);
  printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

  printf("===== 3D dynamic ======\n");
  Validate3DTime();

  return 0;

}
```

7.2 Results

7.2.1 Failures

Validation has failed in one case: when one or both of the frame are degenerated (at least two of there components are colinear). An example is given below for reference:

```
    ===== 2D static ======
Validation2D has failed
Co(-63.571705,-22.581119) x(55.239119,38.152177) y(-62.031537,-42.843548) against To(3.474294,22.751011)
x(-49.195251,84.166201) y(41.179031,-95.350316)
FMB : intersection
SAT : no intersection
```



This case can be detected and avoided prior to the intersection test by checking the determinant of the frame: degenerated frames have a null determinant. In the example above the determinant of the first frame is equal to -0.001667.

7.2.2 2D static

```
===== 2D static ======
Validation2D has succeed.
Tested 470790 intersections and 1529134 no intersections
```

7.2.3 2D dynamic

```
===== 2D dynamic ======
Validation2DTime has succeed.
Tested 743716 intersections and 1256208 no intersections
```

7.2.4 3D static

```
===== 3D static ======
Validation3D has succeed.
Tested 316466 intersections and 1683534 no intersections
```

7.2.5 3D dynamic

```
===== 3D dynamic ======
Validation3DTime has succeed.
Tested 523412 intersections and 1476586 no intersections
```

8 Qualification against SAT

In this section I introduce the code I've used to qualify the algorithm and its implementation. The qualification consists of running the FMB algorithm on randomly generated pairs of Frame, and check its execution time against the one of running the SAT algorithm on the same pair of Frames.

8.1 Code

8.1.1 2D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb2d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_2D 1500

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];
```

```
} Param2D;

// Type of qualification
typedef enum {
  typeQualif_all,
  typeQualif_nearCaseOnly
} TypeQualif;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification2DStatic(
  const Param2D paramP,
  const Param2D paramQ,
```

```
TypeQualif typeQualif) {

// Create the two Frames
Frame2D P =
  Frame2DCreateStatic(
    paramP.type,
    paramP.orig,
    paramP.comp);

Frame2D Q =
  Frame2DCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.comp);

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame2D* that = &P;
Frame2D* tho = &Q;

// If the type of qualifiction is nearCaseOnly
if (typeQualif == typeQualif_nearCaseOnly) {

  // If the AABBs of the two Frame are not in intersection
  bool isIntersectingAABB =
    AABBTestIntersection2D(
      &(that->bdgBox),
      &(tho->bdgBox));
  if (isIntersectingAABB == false) {

    // Skip the test on this pair to simulate pruning of pairs
    // of distant Frame by a prior step in a real collision
    // detection system.
    return;

  }

}

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Declare an array to memorize the results of the repeated
  // test on the same pair,
  // to prevent optimization from the compiler to remove the for loop
  bool isIntersectingFMB[NB_REPEAT_2D] = {false};

  // Start measuring time
  struct timeval start;
  gettimeofday(&start, NULL);

  // Run the FMB intersection test
  for (
    int i = NB_REPEAT_2D;
    i--;) {

    isIntersectingFMB[i] =
      FMBTestIntersection2D(
        that,
        tho,
        NULL);
```

```
}

// Stop measuring time
struct timeval stop;
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausFMB = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
  exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausFMB = stop.tv_sec - start.tv_sec;
  deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausFMB = stop.tv_usec - start.tv_usec;

}

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_2D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
  int i = NB_REPEAT_2D;
  i--;) {

  isIntersectingSAT[i] =
    SATTestIntersection2D(
      that,
      tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
```

```
  exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausSAT = stop.tv_sec - start.tv_sec;
  deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

  // If FMB and SAT disagrees
  if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

    printf("Qualification has failed\n");
    Frame2DPrint(that);
    printf(" against ");
    Frame2DPrint(tho);
    printf("\n");
    printf("FMB : ");
    if (isIntersectingFMB[0] == false) printf("no ");
    printf("intersection\n");
    printf("SAT : ");
    if (isIntersectingSAT[0] == false) printf("no ");
    printf("intersection\n");

    // Stop the qualification test
    exit(0);

  }

  // Get the ratio of execution time
  double ratio = ((double)deltausFMB) / ((double)deltausSAT);

  // If the Frames intersect
  if (isIntersectingSAT[0] == true) {

    // Update the counters
    if (countInter == 0) {

      minInter = ratio;
      maxInter = ratio;

    } else {

      if (minInter > ratio) minInter = ratio;
      if (maxInter < ratio) maxInter = ratio;
```

```
}

sumInter += ratio;
++countInter;

if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameCuboid) {

  if (countInterCC == 0) {

    minInterCC = ratio;
    maxInterCC = ratio;

  } else {

    if (minInterCC > ratio) minInterCC = ratio;
    if (maxInterCC < ratio) maxInterCC = ratio;

  }

  sumInterCC += ratio;
  ++countInterCC;

} else if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameTetrahedron) {

  if (countInterCT == 0) {

    minInterCT = ratio;
    maxInterCT = ratio;

  } else {

    if (minInterCT > ratio) minInterCT = ratio;
    if (maxInterCT < ratio) maxInterCT = ratio;

  }

  sumInterCT += ratio;
  ++countInterCT;

} else if (
  paramP.type == FrameTetrahedron &&
  paramQ.type == FrameCuboid) {

  if (countInterTC == 0) {

    minInterTC = ratio;
    maxInterTC = ratio;

  } else {

    if (minInterTC > ratio) minInterTC = ratio;
    if (maxInterTC < ratio) maxInterTC = ratio;

  }

  sumInterTC += ratio;
  ++countInterTC;
```

```
  } else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countInterTT == 0) {

      minInterTT = ratio;
      maxInterTT = ratio;

    } else {

      if (minInterTT > ratio) minInterTT = ratio;
      if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

  }

// Else, the Frames do not intersect
} else {

  // Update the counters
  if (countNoInter == 0) {

    minNoInter = ratio;
    maxNoInter = ratio;

  } else {

    if (minNoInter > ratio) minNoInter = ratio;
    if (maxNoInter < ratio) maxNoInter = ratio;

  }

  sumNoInter += ratio;
  ++countNoInter;

  if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countNoInterCC == 0) {

      minNoInterCC = ratio;
      maxNoInterCC = ratio;

    } else {

      if (minNoInterCC > ratio) minNoInterCC = ratio;
      if (maxNoInterCC < ratio) maxNoInterCC = ratio;

    }

    sumNoInterCC += ratio;
    ++countNoInterCC;

  } else if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {
```

```
      if (countNoInterCT == 0) {

        minNoInterCT = ratio;
        maxNoInterCT = ratio;

      } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

      }

      sumNoInterCT += ratio;
      ++countNoInterCT;

    } else if (
      paramP.type == FrameTetrahedron &&
      paramQ.type == FrameCuboid) {

      if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

      } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

      }

      sumNoInterTC += ratio;
      ++countNoInterTC;

    } else if (
      paramP.type == FrameTetrahedron &&
      paramQ.type == FrameTetrahedron) {

      if (countNoInterTT == 0) {

        minNoInterTT = ratio;
        maxNoInterTT = ratio;

      } else {

        if (minNoInterTT > ratio) minNoInterTT = ratio;
        if (maxNoInterTT < ratio) maxNoInterTT = ratio;

      }

      sumNoInterTT += ratio;
      ++countNoInterTT;

    }

  }

// Else, if time of execution for FMB was less than 10ms
} else if (deltausFMB < 10) {

  printf("deltausFMB < 10ms, increase NB_REPEAT\n");
```

```
      exit(0);

    // Else, if time of execution for SAT was less than 10ms
    } else if (deltausSAT < 10) {

      printf("deltausSAT < 10ms, increase NB_REPEAT\n");
      exit(0);

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Qualify2DStatic(TypeQualif typeQualif) {

  // Initialise the random generator
  srandom(time(NULL));

  // Open the files to save the results
  FILE* fp = NULL;
  FILE* fpCC = NULL;
  FILE* fpCT = NULL;
  FILE* fpTC = NULL;
  FILE* fpTT = NULL;
  if (typeQualif == typeQualif_all) {

    fp = fopen("../Results/qualification2D.txt", "w");
    fpCC = fopen("../Results/qualification2DCC.txt", "w");
    fpCT = fopen("../Results/qualification2DCT.txt", "w");
    fpTC = fopen("../Results/qualification2DTC.txt", "w");
    fpTT = fopen("../Results/qualification2DTT.txt", "w");

  } else if (typeQualif == typeQualif_nearCaseOnly) {

    fp = fopen("../Results/qualification2Dnearcaseonly.txt", "w");
    fpCC = fopen("../Results/qualification2DCCnearcaseonly.txt", "w");
    fpCT = fopen("../Results/qualification2DCTnearcaseonly.txt", "w");
    fpTC = fopen("../Results/qualification2DTCnearcaseonly.txt", "w");
    fpTT = fopen("../Results/qualification2DTTnearcaseonly.txt", "w");

  } else {

    printf("Unimplemented typeQualif in Qualifiy2DStatic\n");
    exit(1);

  }

  // Loop on runs
  for (
    int iRun = 0;
    iRun < NB_RUNS;
    ++iRun) {

    // Ratio intersection/no intersection for the displayed results
    double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

    // Initialize counters
```

```
minInter = 0.0;
maxInter = 0.0;
sumInter = 0.0;
countInter = 0;
minNoInter = 0.0;
maxNoInter = 0.0;
sumNoInter = 0.0;
countNoInter = 0;

minInterCC = 0.0;
maxInterCC = 0.0;
sumInterCC = 0.0;
countInterCC = 0;
minNoInterCC = 0.0;
maxNoInterCC = 0.0;
sumNoInterCC = 0.0;
countNoInterCC = 0;

minInterCT = 0.0;
maxInterCT = 0.0;
sumInterCT = 0.0;
countInterCT = 0;
minNoInterCT = 0.0;
maxNoInterCT = 0.0;
sumNoInterCT = 0.0;
countNoInterCT = 0;

minInterTC = 0.0;
maxInterTC = 0.0;
sumInterTC = 0.0;
countInterTC = 0;
minNoInterTC = 0.0;
maxNoInterTC = 0.0;
sumNoInterTC = 0.0;
countNoInterTC = 0;

minInterTT = 0.0;
maxInterTT = 0.0;
sumInterTT = 0.0;
countInterTT = 0;
minNoInterTT = 0.0;
maxNoInterTT = 0.0;
sumNoInterTT = 0.0;
countNoInterTT = 0;

// Declare two variables to memorize the arguments to the
// Qualification function
Param2D paramP;
Param2D paramQ;

// Loop on the number of tests
for (
  unsigned long iTest = NB_TESTS;
  iTest--;) {

  // Create two random Frame definitions
  Param2D* param = &paramP;
  for (
    int iParam = 2;
    iParam--;) {

    // 50% chance of being a Cuboid or a Tetrahedron
```

```
    if (rnd() < 0.5) {

      param->type = FrameCuboid;

    } else {

      param->type = FrameTetrahedron;

    }

    for (
      int iAxis = 2;
      iAxis--;) {

      param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      for (
        int iComp = 2;
        iComp--;) {

        param->comp[iComp][iAxis] =
          -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      }

    }

    param = &paramQ;

  }

  // Calculate the determinant of the Frames' components matrix
  double detP =
    paramP.comp[0][0] * paramP.comp[1][1] -
    paramP.comp[1][0] * paramP.comp[0][1];

  double detQ =
    paramQ.comp[0][0] * paramQ.comp[1][1] -
    paramQ.comp[1][0] * paramQ.comp[0][1];

  // If the determinants are not null, ie the Frame are not degenerate
  if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification2DStatic(
      paramP,
      paramQ,
      typeQualif);

  }

}

// Save the results
if (iRun == 0) {

  fprintf(fp, "percPairInter,");
  fprintf(fp, "countInterTo,countNoInterTo,");
  fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
  fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
  fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");
```

```
  fprintf(fpCC, "percPairInter,");
  fprintf(fpCC, "countInterCC,countNoInterCC,");
  fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
  fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
  fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

  fprintf(fpCT, "percPairInter,");
  fprintf(fpCT, "countInterCT,countNoInterCT,");
  fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
  fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
  fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

  fprintf(fpTC, "percPairInter,");
  fprintf(fpTC, "countInterTC,countNoInterTC,");
  fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
  fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
  fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

  fprintf(fpTT, "percPairInter,");
  fprintf(fpTT, "countInterTT,countNoInterTT,");
  fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
  fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
  fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}

fprintf(
  fp,
  "%.1f,",
  ratioInter);
fprintf(
  fp,
  "%lu,%lu,",
  countInter,
  countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minInter,
  avgInter,
  maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minNoInter,
  avgNoInter,
  maxNoInter);
double avg =
  ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
  fp,
  "%f,%f,%f",
  (minNoInter < minInter ? minNoInter : minInter),
  avg,
  (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

  fprintf(fp, "\n");

}
```

```
fprintf(
  fpCC,
  "%.1f,",
  ratioInter);
fprintf(
  fpCC,
  "%lu,%lu,",
  countInterCC,
  countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minInterCC,
  avgInterCC,
  maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minNoInterCC,
  avgNoInterCC,
  maxNoInterCC);
double avgCC =
  ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f",
  (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
  avgCC,
  (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCC, "\n");

}

fprintf(
  fpCT,
  "%.1f,",
  ratioInter);
fprintf(
  fpCT,
  "%lu,%lu,",
  countInterCT,
  countNoInterCT);
double avgInterCT = sumInterCT / (double)countInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minInterCT,
  avgInterCT,
  maxInterCT);
double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minNoInterCT,
  avgNoInterCT,
  maxNoInterCT);
double avgCT =
```

```
  ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f",
  (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
  avgCT,
  (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCT, "\n");

}

fprintf(
  fpTC,
  "%.1f,",
  ratioInter);
fprintf(
  fpTC,
  "%lu,%lu,",
  countInterTC,
  countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
  fpTC,
  "%f,%f,%f,",
  minInterTC,
  avgInterTC,
  maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f,",
  minNoInterTC,
  avgNoInterTC,
  maxNoInterTC);
double avgTC =
  ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f",
  (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
  avgTC,
  (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpTC, "\n");

}

fprintf(
  fpTT,
  "%.1f,",
  ratioInter);
fprintf(
  fpTT,
  "%lu,%lu,",
  countInterTT,
  countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
  fpTT,
```

```
      "%f,%f,%f,",
      minInterTT,
      avgInterTT,
      maxInterTT);
    double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
    fprintf(
      fpTT,
      "%f,%f,%f,",
      minNoInterTT,
      avgNoInterTT,
      maxNoInterTT);
    double avgTT =
      ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
    fprintf(
      fpTT,
      "%f,%f,%f",
      (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
      avgTT,
      (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
    if (iRun < NB_RUNS - 1) {

      fprintf(fpTT, "\n");

    }

  }

  // Close the files
  fclose(fp);
  fclose(fpCC);
  fclose(fpCT);
  fclose(fpTC);
  fclose(fpTT);

}

int main(int argc, char** argv) {

  TypeQualif typeQualif = typeQualif_all;

  for (
    int iArg = 0;
    iArg < argc;
    ++iArg) {

    if (strcmp(argv[iArg], "-nearCaseOnly") == 0) {

      typeQualif = typeQualif_nearCaseOnly;

    }

  }

  Qualify2DStatic(typeQualif);

  return 0;

}
```

8.1.2 3D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb3d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_3D 800

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];

} Param3D;

// Type of qualification
typedef enum {
  typeQualif_all,
  typeQualif_nearCaseOnly
} TypeQualif;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
```

```
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification3DStatic(
  const Param3D paramP,
  const Param3D paramQ,
  TypeQualif typeQualif) {

  // Create the two Frames
  Frame3D P =
    Frame3DCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.comp);

  Frame3D Q =
    Frame3DCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.comp);

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame3D* that = &P;
  Frame3D* tho = &Q;

  // If the type of qualification is nearCaseOnly
  if (typeQualif == typeQualif_nearCaseOnly) {

    // If the AABBs of the two Frame are not in intersection
    bool isIntersectingAABB =
      AABBTestIntersection3D(
```

```
      &(that->bdgBox),
      &(tho->bdgBox));
  if (isIntersectingAABB == false) {

    // Skip the test on this pair to simulate pruning of pairs
    // of distant Frame by a prior step in a real collision
    // detection system.
    return;

  }

}

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Declare an array to memorize the results of the repeated
  // test on the same pair,
  // to prevent optimization from the compiler to remove the for loop
  bool isIntersectingFMB[NB_REPEAT_3D] = {false};

  // Start measuring time
  struct timeval start;
  gettimeofday(&start, NULL);

  // Run the FMB intersection test
  for (
    int i = NB_REPEAT_3D;
    i--;) {

    isIntersectingFMB[i] =
      FMBTestIntersection3D(
        that,
        tho,
        NULL);

  }

  // Stop measuring time
  struct timeval stop;
  gettimeofday(&stop, NULL);

  // Calculate the delay of execution
  unsigned long deltausFMB = 0;
  if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

  }

  if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
    exit(0);

  }

  if (stop.tv_usec < start.tv_usec) {
```

```
  deltausFMB = stop.tv_sec - start.tv_sec;
  deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausFMB = stop.tv_usec - start.tv_usec;

}

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_3D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
  int i = NB_REPEAT_3D;
  i--;) {

  isIntersectingSAT[i] =
    SATTestIntersection3D(
      that,
      tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
  exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausSAT = stop.tv_sec - start.tv_sec;
  deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {
```

```
// If FMB and SAT disagrees
if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

  printf("Qualification has failed\n");
  Frame3DPrint(that);
  printf(" against ");
  Frame3DPrint(tho);
  printf("\n");
  printf("FMB : ");
  if (isIntersectingFMB[0] == false) printf("no ");
  printf("intersection\n");
  printf("SAT : ");
  if (isIntersectingSAT[0] == false) printf("no ");
  printf("intersection\n");

  // Stop the qualification test
  exit(0);

}

// Get the ratio of execution time
double ratio = ((double)deltausFMB) / ((double)deltausSAT);

// If the Frames intersect
if (isIntersectingSAT[0] == true) {

  // Update the counters
  if (countInter == 0) {

    minInter = ratio;
    maxInter = ratio;

  } else {

    if (minInter > ratio) minInter = ratio;
    if (maxInter < ratio) maxInter = ratio;

  }

  sumInter += ratio;
  ++countInter;

  if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countInterCC == 0) {

      minInterCC = ratio;
      maxInterCC = ratio;

    } else {

      if (minInterCC > ratio) minInterCC = ratio;
      if (maxInterCC < ratio) maxInterCC = ratio;

    }

    sumInterCC += ratio;
    ++countInterCC;

  } else if (
```

```
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countInterCT == 0) {

      minInterCT = ratio;
      maxInterCT = ratio;

    } else {

      if (minInterCT > ratio) minInterCT = ratio;
      if (maxInterCT < ratio) maxInterCT = ratio;

    }

    sumInterCT += ratio;
    ++countInterCT;

  } else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countInterTC == 0) {

      minInterTC = ratio;
      maxInterTC = ratio;

    } else {

      if (minInterTC > ratio) minInterTC = ratio;
      if (maxInterTC < ratio) maxInterTC = ratio;

    }

    sumInterTC += ratio;
    ++countInterTC;

  } else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countInterTT == 0) {

      minInterTT = ratio;
      maxInterTT = ratio;

    } else {

      if (minInterTT > ratio) minInterTT = ratio;
      if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

  }

// Else, the Frames do not intersect
} else {

  // Update the counters
```

```
if (countNoInter == 0) {

  minNoInter = ratio;
  maxNoInter = ratio;

} else {

  if (minNoInter > ratio) minNoInter = ratio;
  if (maxNoInter < ratio) maxNoInter = ratio;

}

sumNoInter += ratio;
++countNoInter;

if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameCuboid) {

  if (countNoInterCC == 0) {

    minNoInterCC = ratio;
    maxNoInterCC = ratio;

  } else {

    if (minNoInterCC > ratio) minNoInterCC = ratio;
    if (maxNoInterCC < ratio) maxNoInterCC = ratio;

  }

  sumNoInterCC += ratio;
  ++countNoInterCC;

} else if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameTetrahedron) {

  if (countNoInterCT == 0) {

    minNoInterCT = ratio;
    maxNoInterCT = ratio;

  } else {

    if (minNoInterCT > ratio) minNoInterCT = ratio;
    if (maxNoInterCT < ratio) maxNoInterCT = ratio;

  }

  sumNoInterCT += ratio;
  ++countNoInterCT;

} else if (
  paramP.type == FrameTetrahedron &&
  paramQ.type == FrameCuboid) {

  if (countNoInterTC == 0) {

    minNoInterTC = ratio;
    maxNoInterTC = ratio;
```

```
          } else {

            if (minNoInterTC > ratio) minNoInterTC = ratio;
            if (maxNoInterTC < ratio) maxNoInterTC = ratio;

          }

          sumNoInterTC += ratio;
          ++countNoInterTC;

        } else if (
          paramP.type == FrameTetrahedron &&
          paramQ.type == FrameTetrahedron) {

          if (countNoInterTT == 0) {

            minNoInterTT = ratio;
            maxNoInterTT = ratio;

          } else {

            if (minNoInterTT > ratio) minNoInterTT = ratio;
            if (maxNoInterTT < ratio) maxNoInterTT = ratio;

          }

          sumNoInterTT += ratio;
          ++countNoInterTT;

        }

      }

    // Else, if time of execution for FMB was less than 10ms
    } else if (deltausFMB < 10) {

      printf("deltausFMB < 10ms, increase NB_REPEAT\n");
      exit(0);

    // Else, if time of execution for SAT was less than 10ms
    } else if (deltausSAT < 10) {

      printf("deltausSAT < 10ms, increase NB_REPEAT\n");
      exit(0);

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Qualify3DStatic(TypeQualif typeQualif) {

  // Initialise the random generator
  srandom(time(NULL));

  // Open the files to save the results
  FILE* fp = NULL;
```

```
FILE* fpCC = NULL;
FILE* fpCT = NULL;
FILE* fpTC = NULL;
FILE* fpTT = NULL;
if (typeQualif == typeQualif_all) {

  fp = fopen("../Results/qualification3D.txt", "w");
  fpCC = fopen("../Results/qualification3DCC.txt", "w");
  fpCT = fopen("../Results/qualification3DCT.txt", "w");
  fpTC = fopen("../Results/qualification3DTC.txt", "w");
  fpTT = fopen("../Results/qualification3DTT.txt", "w");

} else if (typeQualif == typeQualif_nearCaseOnly) {

  fp = fopen("../Results/qualification3Dnearcaseonly.txt", "w");
  fpCC = fopen("../Results/qualification3DCCnearcaseonly.txt", "w");
  fpCT = fopen("../Results/qualification3DCTnearcaseonly.txt", "w");
  fpTC = fopen("../Results/qualification3DTCnearcaseonly.txt", "w");
  fpTT = fopen("../Results/qualification3DTTnearcaseonly.txt", "w");

} else {

  printf("Unimplemented typeQualif in Qualifiy3DStatic\n");
  exit(1);

}

// Loop on runs
for (
  int iRun = 0;
  iRun < NB_RUNS;
  ++iRun) {

  // Ratio intersection/no intersection for the displayed results
  double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

  // Initialize counters
  minInter = 0.0;
  maxInter = 0.0;
  sumInter = 0.0;
  countInter = 0;
  minNoInter = 0.0;
  maxNoInter = 0.0;
  sumNoInter = 0.0;
  countNoInter = 0;

  minInterCC = 0.0;
  maxInterCC = 0.0;
  sumInterCC = 0.0;
  countInterCC = 0;
  minNoInterCC = 0.0;
  maxNoInterCC = 0.0;
  sumNoInterCC = 0.0;
  countNoInterCC = 0;

  minInterCT = 0.0;
  maxInterCT = 0.0;
  sumInterCT = 0.0;
  countInterCT = 0;
  minNoInterCT = 0.0;
  maxNoInterCT = 0.0;
  sumNoInterCT = 0.0;
```

```
countNoInterCT = 0;

minInterTC = 0.0;
maxInterTC = 0.0;
sumInterTC = 0.0;
countInterTC = 0;
minNoInterTC = 0.0;
maxNoInterTC = 0.0;
sumNoInterTC = 0.0;
countNoInterTC = 0;

minInterTT = 0.0;
maxInterTT = 0.0;
sumInterTT = 0.0;
countInterTT = 0;
minNoInterTT = 0.0;
maxNoInterTT = 0.0;
sumNoInterTT = 0.0;
countNoInterTT = 0;

// Declare two variables to memorize the arguments to the
// Qualification function
Param3D paramP;
Param3D paramQ;

// Loop on the number of tests
for (
  unsigned long iTest = NB_TESTS;
  iTest--;) {

  // Create two random Frame definitions
  Param3D* param = &paramP;
  for (
    int iParam = 2;
    iParam--;) {

    // 50% chance of being a Cuboid or a Tetrahedron
    if (rnd() < 0.5) {

      param->type = FrameCuboid;

    } else {

      param->type = FrameTetrahedron;

    }

    for (
      int iAxis = 3;
      iAxis--;) {

      param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      for (
        int iComp = 3;
        iComp--;) {

        param->comp[iComp][iAxis] =
          -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      }
```

```
    }

    param = &paramQ;

  }

  // Calculate the determinant of the Frames' components matrix
  double detP =
    paramP.comp[0][0] * (paramP.comp[1][1] * paramP.comp[2][2] -
    paramP.comp[1][2] * paramP.comp[2][1]) -
    paramP.comp[1][0] * (paramP.comp[0][1] * paramP.comp[2][2] -
    paramP.comp[0][2] * paramP.comp[2][1]) +
    paramP.comp[2][0] * (paramP.comp[0][1] * paramP.comp[1][2] -
    paramP.comp[0][2] * paramP.comp[1][1]);

  double detQ =
    paramQ.comp[0][0] * (paramQ.comp[1][1] * paramQ.comp[2][2] -
    paramQ.comp[1][2] * paramQ.comp[2][1]) -
    paramQ.comp[1][0] * (paramQ.comp[0][1] * paramQ.comp[2][2] -
    paramQ.comp[0][2] * paramQ.comp[2][1]) +
    paramQ.comp[2][0] * (paramQ.comp[0][1] * paramQ.comp[1][2] -
    paramQ.comp[0][2] * paramQ.comp[1][1]);

  // If the determinants are not null, ie the Frame are not degenerate
  if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification3DStatic(
      paramP,
      paramQ,
      typeQualif);

  }

}

// Save the results
if (iRun == 0) {

  fprintf(fp, "percPairInter,");
  fprintf(fp, "countInterTo,countNoInterTo,");
  fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
  fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
  fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

  fprintf(fpCC, "percPairInter,");
  fprintf(fpCC, "countInterCC,countNoInterCC,");
  fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
  fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
  fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

  fprintf(fpCT, "percPairInter,");
  fprintf(fpCT, "countInterCT,countNoInterCT,");
  fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
  fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
  fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

  fprintf(fpTC, "percPairInter,");
  fprintf(fpTC, "countInterTC,countNoInterTC,");
  fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
  fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
  fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");
```

```
  fprintf(fpTT, "percPairInter,");
  fprintf(fpTT, "countInterTT,countNoInterTT,");
  fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
  fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
  fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}

fprintf(
  fp,
  "%.1f,",
  ratioInter);
fprintf(
  fp,
  "%lu,%lu,",
  countInter,
  countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minInter,
  avgInter,
  maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minNoInter,
  avgNoInter,
  maxNoInter);
double avg =
  ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
  fp,
  "%f,%f,%f",
  (minNoInter < minInter ? minNoInter : minInter),
  avg,
  (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

  fprintf(fp, "\n");

}

fprintf(
  fpCC,
  "%.1f,",
  ratioInter);
fprintf(
  fpCC,
  "%lu,%lu,",
  countInterCC,
  countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minInterCC,
  avgInterCC,
  maxInterCC);
```

```
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minNoInterCC,
  avgNoInterCC,
  maxNoInterCC);
double avgCC =
  ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f",
  (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
  avgCC,
  (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCC, "\n");

}

fprintf(
  fpCT,
  "%.1f,",
  ratioInter);
fprintf(
  fpCT,
  "%lu,%lu,",
  countInterCT,
  countNoInterCT);
double avgInterCT = sumInterCT / (double)countInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minInterCT,
  avgInterCT,
  maxInterCT);
double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minNoInterCT,
  avgNoInterCT,
  maxNoInterCT);
double avgCT =
  ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f",
  (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
  avgCT,
  (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCT, "\n");

}

fprintf(
  fpTC,
  "%.1f,",
  ratioInter);
```

```
fprintf(
  fpTC,
  "%lu,%lu,",
  countInterTC,
  countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
  fpTC,
  "%f,%f,%f,",
  minInterTC,
  avgInterTC,
  maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f,",
  minNoInterTC,
  avgNoInterTC,
  maxNoInterTC);
double avgTC =
  ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f",
  (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
  avgTC,
  (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpTC, "\n");

}

fprintf(
  fpTT,
  "%.1f,",
  ratioInter);
fprintf(
  fpTT,
  "%lu,%lu,",
  countInterTT,
  countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
  fpTT,
  "%f,%f,%f,",
  minInterTT,
  avgInterTT,
  maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
  fpTT,
  "%f,%f,%f,",
  minNoInterTT,
  avgNoInterTT,
  maxNoInterTT);
double avgTT =
  ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
  fpTT,
  "%f,%f,%f",
  (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
```

```
      avgTT,
      (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
    if (iRun < NB_RUNS - 1) {

      fprintf(fpTT, "\n");

    }

  }

  // Close the files
  fclose(fp);
  fclose(fpCC);
  fclose(fpCT);
  fclose(fpTC);
  fclose(fpTT);

}

int main(int argc, char** argv) {

  TypeQualif typeQualif = typeQualif_all;

  for (
    int iArg = 0;
    iArg < argc;
    ++iArg) {

    if (strcmp(argv[iArg], "-nearCaseOnly") == 0) {

      typeQualif = typeQualif_nearCaseOnly;

    }

  }

  Qualify3DStatic(typeQualif);

  return 0;

}
```

8.1.3 2D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb2dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0
```

```
// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_2D 1500

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

  FrameType type;
  double orig[2];
  double comp[2][2];
  double speed[2];

} Param2DTime;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;
```

```
double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification2DDynamic(
  const Param2DTime paramP,
  const Param2DTime paramQ) {

  // Create the two Frames
  Frame2DTime P =
    Frame2DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);

  Frame2DTime Q =
    Frame2DTimeCreateStatic(
      paramQ.type,
      paramQ.orig,
      paramQ.speed,
      paramQ.comp);

  // Helper variables to loop on the pair (that, tho) and (tho, that)
  Frame2DTime* that = &P;
  Frame2DTime* tho = &Q;

  // Loop on pairs of Frames
  for (
    int iPair = 2;
    iPair--;) {

    // Declare an array to memorize the results of the repeated
    // test on the same pair,
    // to prevent optimization from the compiler to remove the for loop
    bool isIntersectingFMB[NB_REPEAT_2D] = {false};

    // Start measuring time
    struct timeval start;
    gettimeofday(&start, NULL);

    // Run the FMB intersection test
    for (
      int i = NB_REPEAT_2D;
      i--;) {

      isIntersectingFMB[i] =
        FMBTestIntersection2DTime(
          that,
          tho,
          NULL);

    }
```

```
// Stop measuring time
struct timeval stop;
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausFMB = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
  exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausFMB = stop.tv_sec - start.tv_sec;
  deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausFMB = stop.tv_usec - start.tv_usec;

}

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_2D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
  int i = NB_REPEAT_2D;
  i--;) {

  isIntersectingSAT[i] =
    SATTestIntersection2DTime(
      that,
      tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
  exit(0);

}
```

```
if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausSAT = stop.tv_sec - start.tv_sec;
  deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

  // If FMB and SAT disagrees
  if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

    printf("Qualification has failed\n");
    Frame2DTimePrint(that);
    printf(" against ");
    Frame2DTimePrint(tho);
    printf("\n");
    printf("FMB : ");
    if (isIntersectingFMB[0] == false) printf("no ");
    printf("intersection\n");
    printf("SAT : ");
    if (isIntersectingSAT[0] == false) printf("no ");
    printf("intersection\n");

    // Stop the qualification test
    exit(0);

  }

  // Get the ratio of execution time
  double ratio = ((double)deltausFMB) / ((double)deltausSAT);

  // If the Frames intersect
  if (isIntersectingSAT[0] == true) {

    // Update the counters
    if (countInter == 0) {

      minInter = ratio;
      maxInter = ratio;

    } else {

      if (minInter > ratio) minInter = ratio;
      if (maxInter < ratio) maxInter = ratio;

    }

    sumInter += ratio;
```

```
++countInter;

if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameCuboid) {

  if (countInterCC == 0) {

    minInterCC = ratio;
    maxInterCC = ratio;

  } else {

    if (minInterCC > ratio) minInterCC = ratio;
    if (maxInterCC < ratio) maxInterCC = ratio;

  }

  sumInterCC += ratio;
  ++countInterCC;

} else if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameTetrahedron) {

  if (countInterCT == 0) {

    minInterCT = ratio;
    maxInterCT = ratio;

  } else {

    if (minInterCT > ratio) minInterCT = ratio;
    if (maxInterCT < ratio) maxInterCT = ratio;

  }

  sumInterCT += ratio;
  ++countInterCT;

} else if (
  paramP.type == FrameTetrahedron &&
  paramQ.type == FrameCuboid) {

  if (countInterTC == 0) {

    minInterTC = ratio;
    maxInterTC = ratio;

  } else {

    if (minInterTC > ratio) minInterTC = ratio;
    if (maxInterTC < ratio) maxInterTC = ratio;

  }

  sumInterTC += ratio;
  ++countInterTC;

} else if (
  paramP.type == FrameTetrahedron &&
  paramQ.type == FrameTetrahedron) {
```

```
    if (countInterTT == 0) {

      minInterTT = ratio;
      maxInterTT = ratio;

    } else {

      if (minInterTT > ratio) minInterTT = ratio;
      if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

  }

// Else, the Frames do not intersect
} else {

  // Update the counters
  if (countNoInter == 0) {

    minNoInter = ratio;
    maxNoInter = ratio;

  } else {

    if (minNoInter > ratio) minNoInter = ratio;
    if (maxNoInter < ratio) maxNoInter = ratio;

  }

  sumNoInter += ratio;
  ++countNoInter;

  if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countNoInterCC == 0) {

      minNoInterCC = ratio;
      maxNoInterCC = ratio;

    } else {

      if (minNoInterCC > ratio) minNoInterCC = ratio;
      if (maxNoInterCC < ratio) maxNoInterCC = ratio;

    }

    sumNoInterCC += ratio;
    ++countNoInterCC;

  } else if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterCT == 0) {
```

```
        minNoInterCT = ratio;
        maxNoInterCT = ratio;

      } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

      }

      sumNoInterCT += ratio;
      ++countNoInterCT;

    } else if (
      paramP.type == FrameTetrahedron &&
      paramQ.type == FrameCuboid) {

      if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

      } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

      }

      sumNoInterTC += ratio;
      ++countNoInterTC;

    } else if (
      paramP.type == FrameTetrahedron &&
      paramQ.type == FrameTetrahedron) {

      if (countNoInterTT == 0) {

        minNoInterTT = ratio;
        maxNoInterTT = ratio;

      } else {

        if (minNoInterTT > ratio) minNoInterTT = ratio;
        if (maxNoInterTT < ratio) maxNoInterTT = ratio;

      }

      sumNoInterTT += ratio;
      ++countNoInterTT;

    }

  }

// Else, if time of execution for FMB was less than 10ms
} else if (deltausFMB < 10) {

  printf("deltausFMB < 10ms, increase NB_REPEAT\n");
  exit(0);

// Else, if time of execution for SAT was less than 10ms
```

```
    } else if (deltausSAT < 10) {

      printf("deltausSAT < 10ms, increase NB_REPEAT\n");
      exit(0);

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Qualify2DDynamic(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Open the files to save the results
  FILE* fp = fopen("../Results/qualification2DTime.txt", "w");
  FILE* fpCC = fopen("../Results/qualification2DTimeCC.txt", "w");
  FILE* fpCT = fopen("../Results/qualification2DTimeCT.txt", "w");
  FILE* fpTC = fopen("../Results/qualification2DTimeTC.txt", "w");
  FILE* fpTT = fopen("../Results/qualification2DTimeTT.txt", "w");

  // Loop on runs
  for (
    int iRun = 0;
    iRun < NB_RUNS;
    ++iRun) {

    // Ratio intersection/no intersection for the displayed results
    double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

    // Initialize counters
    minInter = 0.0;
    maxInter = 0.0;
    sumInter = 0.0;
    countInter = 0;
    minNoInter = 0.0;
    maxNoInter = 0.0;
    sumNoInter = 0.0;
    countNoInter = 0;

    minInterCC = 0.0;
    maxInterCC = 0.0;
    sumInterCC = 0.0;
    countInterCC = 0;
    minNoInterCC = 0.0;
    maxNoInterCC = 0.0;
    sumNoInterCC = 0.0;
    countNoInterCC = 0;

    minInterCT = 0.0;
    maxInterCT = 0.0;
    sumInterCT = 0.0;
    countInterCT = 0;
    minNoInterCT = 0.0;
    maxNoInterCT = 0.0;
    sumNoInterCT = 0.0;
```

```
countNoInterCT = 0;

minInterTC = 0.0;
maxInterTC = 0.0;
sumInterTC = 0.0;
countInterTC = 0;
minNoInterTC = 0.0;
maxNoInterTC = 0.0;
sumNoInterTC = 0.0;
countNoInterTC = 0;

minInterTT = 0.0;
maxInterTT = 0.0;
sumInterTT = 0.0;
countInterTT = 0;
minNoInterTT = 0.0;
maxNoInterTT = 0.0;
sumNoInterTT = 0.0;
countNoInterTT = 0;

// Declare two variables to memorize the arguments to the
// Qualification function
Param2DTime paramP;
Param2DTime paramQ;

// Loop on the number of tests
for (
  unsigned long iTest = NB_TESTS;
  iTest--;) {

  // Create two random Frame definitions
  Param2DTime* param = &paramP;
  for (
    int iParam = 2;
    iParam--;) {

    // 50% chance of being a Cuboid or a Tetrahedron
    if (rnd() < 0.5) {

      param->type = FrameCuboid;

    } else {

      param->type = FrameTetrahedron;

    }

    for (
      int iAxis = 2;
      iAxis--;) {

      param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;
      param->speed[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      for (
        int iComp = 2;
        iComp--;) {

        param->comp[iComp][iAxis] =
          -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      }
```

```
    }

    param = &paramQ;

  }

  // Calculate the determinant of the Frames' components matrix
  double detP =
    paramP.comp[0][0] * paramP.comp[1][1] -
    paramP.comp[1][0] * paramP.comp[0][1];

  double detQ =
    paramQ.comp[0][0] * paramQ.comp[1][1] -
    paramQ.comp[1][0] * paramQ.comp[0][1];

  // If the determinants are not null, ie the Frame are not degenerate
  if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification2DDynamic(
      paramP,
      paramQ);

  }

}

// Save the results
if (iRun == 0) {

  fprintf(fp, "percPairInter,");
  fprintf(fp, "countInterTo,countNoInterTo,");
  fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
  fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
  fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

  fprintf(fpCC, "percPairInter,");
  fprintf(fpCC, "countInterCC,countNoInterCC,");
  fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
  fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
  fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

  fprintf(fpCT, "percPairInter,");
  fprintf(fpCT, "countInterCT,countNoInterCT,");
  fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
  fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
  fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

  fprintf(fpTC, "percPairInter,");
  fprintf(fpTC, "countInterTC,countNoInterTC,");
  fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
  fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
  fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

  fprintf(fpTT, "percPairInter,");
  fprintf(fpTT, "countInterTT,countNoInterTT,");
  fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
  fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
  fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}
```

```
fprintf(
  fp,
  "%.1f,",
  ratioInter);
fprintf(
  fp,
  "%lu,%lu,",
  countInter,
  countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minInter,
  avgInter,
  maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minNoInter,
  avgNoInter,
  maxNoInter);
double avg =
  ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
  fp,
  "%f,%f,%f",
  (minNoInter < minInter ? minNoInter : minInter),
  avg,
  (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

  fprintf(fp, "\n");

}

fprintf(
  fpCC,
  "%.1f,",
  ratioInter);
fprintf(
  fpCC,
  "%lu,%lu,",
  countInterCC,
  countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minInterCC,
  avgInterCC,
  maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minNoInterCC,
  avgNoInterCC,
  maxNoInterCC);
double avgCC =
```

```
  ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f",
  (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
  avgCC,
  (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCC, "\n");

}

fprintf(
  fpCT,
  "%.1f,",
  ratioInter);
fprintf(
  fpCT,
  "%lu,%lu,",
  countInterCT,
  countNoInterCT);
double avgInterCT = sumInterCT / (double)countInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minInterCT,
  avgInterCT,
  maxInterCT);
double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f,",
  minNoInterCT,
  avgNoInterCT,
  maxNoInterCT);
double avgCT =
  ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
  fpCT,
  "%f,%f,%f",
  (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
  avgCT,
  (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCT, "\n");

}

fprintf(
  fpTC,
  "%.1f,",
  ratioInter);
fprintf(
  fpTC,
  "%lu,%lu,",
  countInterTC,
  countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
  fpTC,
```

```
  "%f,%f,%f,",
  minInterTC,
  avgInterTC,
  maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f,",
  minNoInterTC,
  avgNoInterTC,
  maxNoInterTC);
double avgTC =
  ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
  fpTC,
  "%f,%f,%f",
  (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
  avgTC,
  (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpTC, "\n");

}

fprintf(
  fpTT,
  "%.1f,",
  ratioInter);
fprintf(
  fpTT,
  "%lu,%lu,",
  countInterTT,
  countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
  fpTT,
  "%f,%f,%f,",
  minInterTT,
  avgInterTT,
  maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
  fpTT,
  "%f,%f,%f,",
  minNoInterTT,
  avgNoInterTT,
  maxNoInterTT);
double avgTT =
  ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
  fpTT,
  "%f,%f,%f",
  (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
  avgTT,
  (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
if (iRun < NB_RUNS - 1) {

  fprintf(fpTT, "\n");

}
```

```
  }

  // Close the files
  fclose(fp);
  fclose(fpCC);
  fclose(fpCT);
  fclose(fpTC);
  fclose(fpTT);

}

int main(int argc, char** argv) {

  Qualify2DDynamic();

  return 0;

}
```

8.1.4 3D dynamic

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb3dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_3D 800

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

  FrameType type;
  double orig[3];
  double comp[3][3];
  double speed[3];

} Param3DTime;
```

```
// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification3DDynamic(
  const Param3DTime paramP,
  const Param3DTime paramQ) {

  // Create the two Frames
  Frame3DTime P =
    Frame3DTimeCreateStatic(
      paramP.type,
      paramP.orig,
      paramP.speed,
      paramP.comp);
```

```
Frame3DTime Q =
  Frame3DTimeCreateStatic(
    paramQ.type,
    paramQ.orig,
    paramQ.speed,
    paramQ.comp);

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame3DTime* that = &P;
Frame3DTime* tho = &Q;

// Loop on pairs of Frames
for (
  int iPair = 2;
  iPair--;) {

  // Declare an array to memorize the results of the repeated
  // test on the same pair,
  // to prevent optimization from the compiler to remove the for loop
  bool isIntersectingFMB[NB_REPEAT_3D] = {false};

  // Start measuring time
  struct timeval start;
  gettimeofday(&start, NULL);

  // Run the FMB intersection test
  for (
    int i = NB_REPEAT_3D;
    i--;) {

    isIntersectingFMB[i] =
      FMBTestIntersection3DTime(
        that,
        tho,
        NULL);

  }

  // Stop measuring time
  struct timeval stop;
  gettimeofday(&stop, NULL);

  // Calculate the delay of execution
  unsigned long deltausFMB = 0;
  if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

  }

  if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
    exit(0);

  }

  if (stop.tv_usec < start.tv_usec) {

    deltausFMB = stop.tv_sec - start.tv_sec;
```

```
  deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausFMB = stop.tv_usec - start.tv_usec;

}

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_3D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
  int i = NB_REPEAT_3D;
  i--;) {

  isIntersectingSAT[i] =
    SATTestIntersection3DTime(
      that,
      tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

  printf("time warps, try again\n");
  exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

  printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
  exit(0);

}

if (stop.tv_usec < start.tv_usec) {

  deltausSAT = stop.tv_sec - start.tv_sec;
  deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

  deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

  // If FMB and SAT disagrees
```

```
if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

  printf("Qualification has failed\n");
  Frame3DTimePrint(that);
  printf(" against ");
  Frame3DTimePrint(tho);
  printf("\n");
  printf("FMB : ");
  if (isIntersectingFMB[0] == false) printf("no ");
  printf("intersection\n");
  printf("SAT : ");
  if (isIntersectingSAT[0] == false) printf("no ");
  printf("intersection\n");

  // Stop the qualification test
  exit(0);

}

// Get the ratio of execution time
double ratio = ((double)deltausFMB) / ((double)deltausSAT);

// If the Frames intersect
if (isIntersectingSAT[0] == true) {

  // Update the counters
  if (countInter == 0) {

    minInter = ratio;
    maxInter = ratio;

  } else {

    if (minInter > ratio) minInter = ratio;
    if (maxInter < ratio) maxInter = ratio;

  }

  sumInter += ratio;
  ++countInter;

  if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countInterCC == 0) {

      minInterCC = ratio;
      maxInterCC = ratio;

    } else {

      if (minInterCC > ratio) minInterCC = ratio;
      if (maxInterCC < ratio) maxInterCC = ratio;

    }

    sumInterCC += ratio;
    ++countInterCC;

  } else if (
    paramP.type == FrameCuboid &&
```

```
    paramQ.type == FrameTetrahedron) {

    if (countInterCT == 0) {

      minInterCT = ratio;
      maxInterCT = ratio;

    } else {

      if (minInterCT > ratio) minInterCT = ratio;
      if (maxInterCT < ratio) maxInterCT = ratio;

    }

    sumInterCT += ratio;
    ++countInterCT;

  } else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countInterTC == 0) {

      minInterTC = ratio;
      maxInterTC = ratio;

    } else {

      if (minInterTC > ratio) minInterTC = ratio;
      if (maxInterTC < ratio) maxInterTC = ratio;

    }

    sumInterTC += ratio;
    ++countInterTC;

  } else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countInterTT == 0) {

      minInterTT = ratio;
      maxInterTT = ratio;

    } else {

      if (minInterTT > ratio) minInterTT = ratio;
      if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

  }

// Else, the Frames do not intersect
} else {

  // Update the counters
  if (countNoInter == 0) {
```

```
  minNoInter = ratio;
  maxNoInter = ratio;

} else {

  if (minNoInter > ratio) minNoInter = ratio;
  if (maxNoInter < ratio) maxNoInter = ratio;

}

sumNoInter += ratio;
++countNoInter;

if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameCuboid) {

  if (countNoInterCC == 0) {

    minNoInterCC = ratio;
    maxNoInterCC = ratio;

  } else {

    if (minNoInterCC > ratio) minNoInterCC = ratio;
    if (maxNoInterCC < ratio) maxNoInterCC = ratio;

  }

  sumNoInterCC += ratio;
  ++countNoInterCC;

} else if (
  paramP.type == FrameCuboid &&
  paramQ.type == FrameTetrahedron) {

  if (countNoInterCT == 0) {

    minNoInterCT = ratio;
    maxNoInterCT = ratio;

  } else {

    if (minNoInterCT > ratio) minNoInterCT = ratio;
    if (maxNoInterCT < ratio) maxNoInterCT = ratio;

  }

  sumNoInterCT += ratio;
  ++countNoInterCT;

} else if (
  paramP.type == FrameTetrahedron &&
  paramQ.type == FrameCuboid) {

  if (countNoInterTC == 0) {

    minNoInterTC = ratio;
    maxNoInterTC = ratio;

  } else {
```

```
            if (minNoInterTC > ratio) minNoInterTC = ratio;
            if (maxNoInterTC < ratio) maxNoInterTC = ratio;

          }

          sumNoInterTC += ratio;
          ++countNoInterTC;

        } else if (
          paramP.type == FrameTetrahedron &&
          paramQ.type == FrameTetrahedron) {

          if (countNoInterTT == 0) {

            minNoInterTT = ratio;
            maxNoInterTT = ratio;

          } else {

            if (minNoInterTT > ratio) minNoInterTT = ratio;
            if (maxNoInterTT < ratio) maxNoInterTT = ratio;

          }

          sumNoInterTT += ratio;
          ++countNoInterTT;

        }

      }

    // Else, if time of execution for FMB was less than 10ms
    } else if (deltausFMB < 10) {

      printf("deltausFMB < 10ms, increase NB_REPEAT\n");
      exit(0);

    // Else, if time of execution for SAT was less than 10ms
    } else if (deltausSAT < 10) {

      printf("deltausSAT < 10ms, increase NB_REPEAT\n");
      exit(0);

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

  }

}

void Qualify3DDynamic(void) {

  // Initialise the random generator
  srandom(time(NULL));

  // Open the files to save the results
  FILE* fp = fopen("../Results/qualification3DTime.txt", "w");
  FILE* fpCC = fopen("../Results/qualification3DTimeCC.txt", "w");
```

```
FILE* fpCT = fopen("../Results/qualification3DTimeCT.txt", "w");
FILE* fpTC = fopen("../Results/qualification3DTimeTC.txt", "w");
FILE* fpTT = fopen("../Results/qualification3DTimeTT.txt", "w");

// Loop on runs
for (
  int iRun = 0;
  iRun < NB_RUNS;
  ++iRun) {

  // Ratio intersection/no intersection for the displayed results
  double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

  // Initialize counters
  minInter = 0.0;
  maxInter = 0.0;
  sumInter = 0.0;
  countInter = 0;
  minNoInter = 0.0;
  maxNoInter = 0.0;
  sumNoInter = 0.0;
  countNoInter = 0;

  minInterCC = 0.0;
  maxInterCC = 0.0;
  sumInterCC = 0.0;
  countInterCC = 0;
  minNoInterCC = 0.0;
  maxNoInterCC = 0.0;
  sumNoInterCC = 0.0;
  countNoInterCC = 0;

  minInterCT = 0.0;
  maxInterCT = 0.0;
  sumInterCT = 0.0;
  countInterCT = 0;
  minNoInterCT = 0.0;
  maxNoInterCT = 0.0;
  sumNoInterCT = 0.0;
  countNoInterCT = 0;

  minInterTC = 0.0;
  maxInterTC = 0.0;
  sumInterTC = 0.0;
  countInterTC = 0;
  minNoInterTC = 0.0;
  maxNoInterTC = 0.0;
  sumNoInterTC = 0.0;
  countNoInterTC = 0;

  minInterTT = 0.0;
  maxInterTT = 0.0;
  sumInterTT = 0.0;
  countInterTT = 0;
  minNoInterTT = 0.0;
  maxNoInterTT = 0.0;
  sumNoInterTT = 0.0;
  countNoInterTT = 0;

  // Declare two variables to memorize the arguments to the
  // Qualification function
  Param3DTime paramP;
```

```
Param3DTime paramQ;

// Loop on the number of tests
for (
  unsigned long iTest = NB_TESTS;
  iTest--;) {

  // Create two random Frame definitions
  Param3DTime* param = &paramP;
  for (
    int iParam = 2;
    iParam--;) {

    // 50% chance of being a Cuboid or a Tetrahedron
    if (rnd() < 0.5) {

      param->type = FrameCuboid;

    } else {

      param->type = FrameTetrahedron;

    }

    for (
      int iAxis = 3;
      iAxis--;) {

      param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;
      param->speed[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      for (
        int iComp = 3;
        iComp--;) {

        param->comp[iComp][iAxis] =
          -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

      }

    }

    param = &paramQ;

  }

  // Calculate the determinant of the Frames' components matrix
  double detP =
    paramP.comp[0][0] * (paramP.comp[1][1] * paramP.comp[2][2] -
    paramP.comp[1][2] * paramP.comp[2][1]) -
    paramP.comp[1][0] * (paramP.comp[0][1] * paramP.comp[2][2] -
    paramP.comp[0][2] * paramP.comp[2][1]) +
    paramP.comp[2][0] * (paramP.comp[0][1] * paramP.comp[1][2] -
    paramP.comp[0][2] * paramP.comp[1][1]);

  double detQ =
    paramQ.comp[0][0] * (paramQ.comp[1][1] * paramQ.comp[2][2] -
    paramQ.comp[1][2] * paramQ.comp[2][1]) -
    paramQ.comp[1][0] * (paramQ.comp[0][1] * paramQ.comp[2][2] -
    paramQ.comp[0][2] * paramQ.comp[2][1]) +
    paramQ.comp[2][0] * (paramQ.comp[0][1] * paramQ.comp[1][2] -
    paramQ.comp[0][2] * paramQ.comp[1][1]);
```

```
  // If the determinants are not null, ie the Frame are not degenerate
  if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification3DDynamic(
      paramP,
      paramQ);

  }

}

// Save the results
if (iRun == 0) {

  fprintf(fp, "percPairInter,");
  fprintf(fp, "countInterTo,countNoInterTo,");
  fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
  fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
  fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

  fprintf(fpCC, "percPairInter,");
  fprintf(fpCC, "countInterCC,countNoInterCC,");
  fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
  fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
  fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

  fprintf(fpCT, "percPairInter,");
  fprintf(fpCT, "countInterCT,countNoInterCT,");
  fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
  fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
  fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

  fprintf(fpTC, "percPairInter,");
  fprintf(fpTC, "countInterTC,countNoInterTC,");
  fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
  fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
  fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

  fprintf(fpTT, "percPairInter,");
  fprintf(fpTT, "countInterTT,countNoInterTT,");
  fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
  fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
  fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}

fprintf(
  fp,
  "%.1f,",
  ratioInter);
fprintf(
  fp,
  "%lu,%lu,",
  countInter,
  countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minInter,
```

```
  avgInter,
  maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
  fp,
  "%f,%f,%f,",
  minNoInter,
  avgNoInter,
  maxNoInter);
double avg =
  ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
  fp,
  "%f,%f,%f",
  (minNoInter < minInter ? minNoInter : minInter),
  avg,
  (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

  fprintf(fp, "\n");

}

fprintf(
  fpCC,
  "%.1f,",
  ratioInter);
fprintf(
  fpCC,
  "%lu,%lu,",
  countInterCC,
  countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minInterCC,
  avgInterCC,
  maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f,",
  minNoInterCC,
  avgNoInterCC,
  maxNoInterCC);
double avgCC =
  ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
  fpCC,
  "%f,%f,%f",
  (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
  avgCC,
  (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

  fprintf(fpCC, "\n");

}

fprintf(
  fpCT,
```

```
    "%.1f,",
    ratioInter);
  fprintf(
    fpCT,
    "%lu,%lu,",
    countInterCT,
    countNoInterCT);
  double avgInterCT = sumInterCT / (double)countInterCT;
  fprintf(
    fpCT,
    "%f,%f,%f,",
    minInterCT,
    avgInterCT,
    maxInterCT);
  double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
  fprintf(
    fpCT,
    "%f,%f,%f,",
    minNoInterCT,
    avgNoInterCT,
    maxNoInterCT);
  double avgCT =
    ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
  fprintf(
    fpCT,
    "%f,%f,%f",
    (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
    avgCT,
    (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
  if (iRun < NB_RUNS - 1) {

    fprintf(fpCT, "\n");

  }

  fprintf(
    fpTC,
    "%.1f,",
    ratioInter);
  fprintf(
    fpTC,
    "%lu,%lu,",
    countInterTC,
    countNoInterTC);
  double avgInterTC = sumInterTC / (double)countInterTC;
  fprintf(
    fpTC,
    "%f,%f,%f,",
    minInterTC,
    avgInterTC,
    maxInterTC);
  double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
  fprintf(
    fpTC,
    "%f,%f,%f,",
    minNoInterTC,
    avgNoInterTC,
    maxNoInterTC);
  double avgTC =
    ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
  fprintf(
    fpTC,
```

```
      "%f,%f,%f",
      (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
      avgTC,
      (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
    if (iRun < NB_RUNS - 1) {

      fprintf(fpTC, "\n");

    }

    fprintf(
      fpTT,
      "%.1f,",
      ratioInter);
    fprintf(
      fpTT,
      "%lu,%lu,",
      countInterTT,
      countNoInterTT);
    double avgInterTT = sumInterTT / (double)countInterTT;
    fprintf(
      fpTT,
      "%f,%f,%f,",
      minInterTT,
      avgInterTT,
      maxInterTT);
    double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
    fprintf(
      fpTT,
      "%f,%f,%f,",
      minNoInterTT,
      avgNoInterTT,
      maxNoInterTT);
    double avgTT =
      ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
    fprintf(
      fpTT,
      "%f,%f,%f",
      (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
      avgTT,
      (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
    if (iRun < NB_RUNS - 1) {

      fprintf(fpTT, "\n");

    }

  }

  // Close the files
  fclose(fp);
  fclose(fpCC);
  fclose(fpCT);
  fclose(fpTC);
  fclose(fpTT);

}

int main(int argc, char** argv) {

  Qualify3DDynamic();
```

```
  return 0;
}
```

8.2 Results

In this subsection I give the results of the qualification for each case. These results are commented in the next section.

8.2.1 2D static

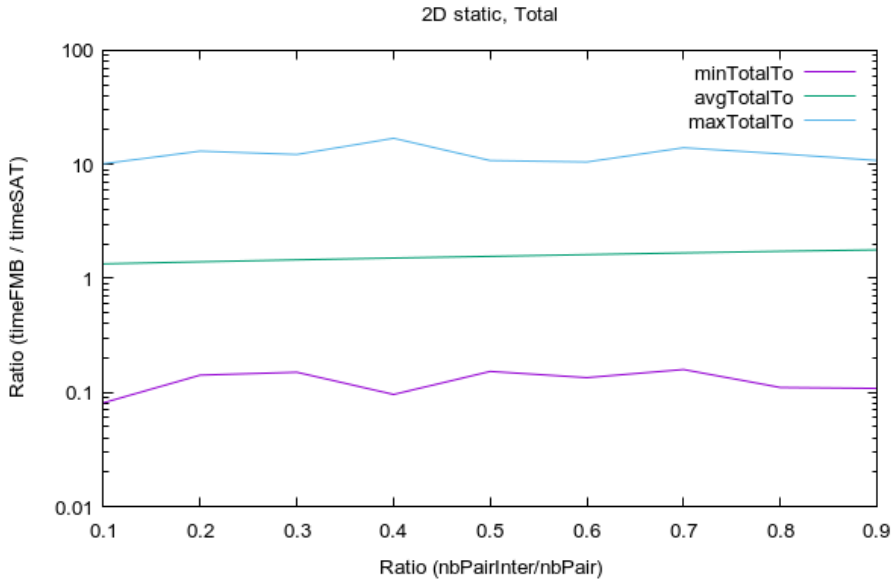
percPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTo
0.1	235414	764542	0.419355	1.834163	5.132075	0.081197	1.286869	10.133333	0.081197	1.341599	10.13
0.2	235376	764592	0.508197	1.837053	5.847458	0.141791	1.287046	13.058824	0.141791	1.397047	13.05
0.3	234904	765048	0.508021	1.840274	6.454545	0.150376	1.289959	12.200000	0.150376	1.455053	12.20
0.4	234232	765724	0.483051	1.837272	5.407407	0.096000	1.289169	16.937500	0.096000	1.508411	16.93
0.5	235370	764584	0.495763	1.835481	5.111111	0.152778	1.287128	10.800000	0.152778	1.561305	10.80
0.6	234270	765686	0.562130	1.836805	6.727273	0.134752	1.287519	10.500000	0.134752	1.617090	10.50
0.7	235118	764822	0.362179	1.839157	5.650794	0.158798	1.289564	14.000000	0.158798	1.674280	14.00
0.8	235226	764732	0.470588	1.837950	5.350000	0.110465	1.290277	12.375000	0.110465	1.728415	12.37
0.9	233676	766280	0.449761	1.833253	5.472727	0.108491	1.287450	10.823529	0.108491	1.778673	10.82

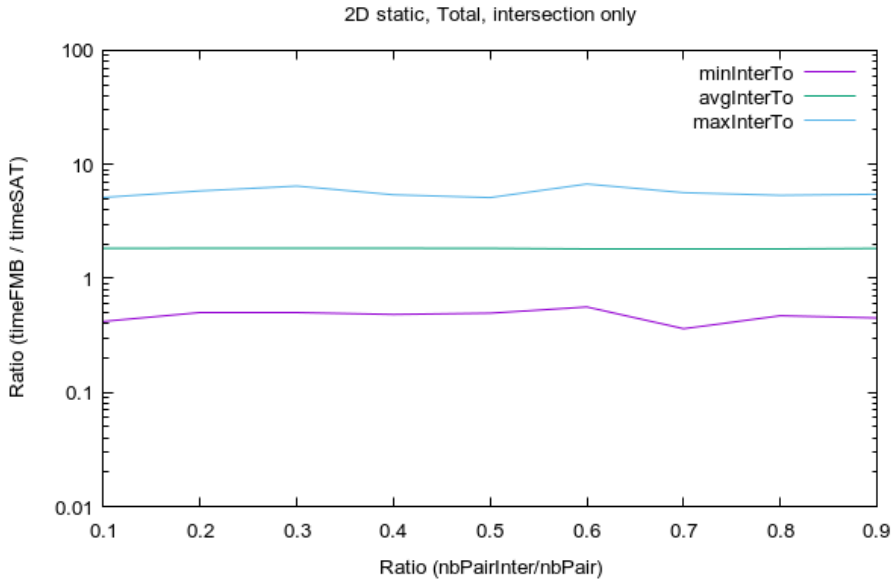
percPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTo
0.1	66182	184516	0.589623	2.316501	5.132075	0.127072	1.203173	5.619048	0.127072	1.314506	5.619
0.2	66206	184070	0.688073	2.319404	5.847458	0.160584	1.201796	6.047619	0.160584	1.425318	6.047
0.3	66162	184448	0.796954	2.322794	6.454545	0.170370	1.202030	5.595238	0.170370	1.538259	6.454
0.4	66046	184118	0.896552	2.318329	5.407407	0.096000	1.204408	10.055556	0.096000	1.649977	10.05
0.5	65972	183904	0.935714	2.317597	5.111111	0.153226	1.204276	7.882353	0.153226	1.760937	7.882
0.6	66392	184102	0.863636	2.317476	6.727273	0.149660	1.202281	5.309524	0.149660	1.871398	6.727
0.7	66224	184532	0.761658	2.321013	5.650794	0.158798	1.202900	6.294118	0.158798	1.985579	6.294
0.8	65988	184164	0.808642	2.321233	5.350000	0.225490	1.203982	6.830508	0.225490	2.097782	6.830
0.9	65062	184638	0.580357	2.316124	5.472727	0.108491	1.201667	6.000000	0.108491	2.204678	6.000

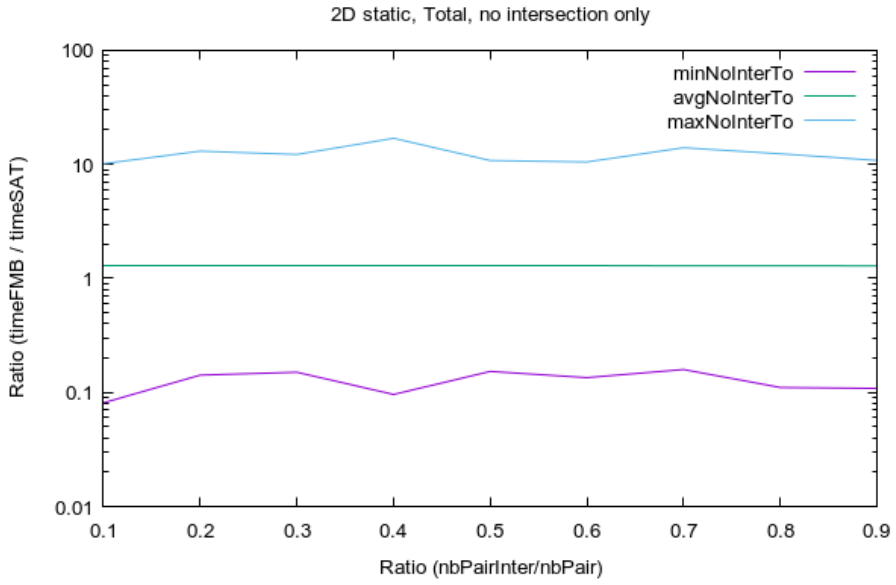
percPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTo
0.1	59068	191048	0.604790	1.791475	4.606557	0.182540	1.294399	9.312500	0.182540	1.344107	9.312
0.2	58830	191642	0.642857	1.792634	4.575000	0.225490	1.294568	13.058824	0.225490	1.394181	13.05
0.3	58564	191682	0.593407	1.795248	6.000000	0.150376	1.300128	10.200000	0.150376	1.448664	10.20
0.4	58994	191452	0.483051	1.792770	4.437500	0.159420	1.296849	16.937500	0.159420	1.495217	16.93
0.5	58982	191314	0.495763	1.792559	3.950000	0.152778	1.292546	10.800000	0.152778	1.542553	10.80
0.6	58158	191396	0.733813	1.791550	3.790323	0.169643	1.295253	10.500000	0.169643	1.593031	10.50
0.7	59040	190376	0.408360	1.795251	4.859155	0.200000	1.296900	12.000000	0.200000	1.645745	12.00
0.8	58718	190748	0.672956	1.794798	3.859649	0.177143	1.298503	11.800000	0.177143	1.695539	11.80
0.9	58850	190874	0.462555	1.792252	4.593220	0.143885	1.296277	10.823529	0.143885	1.742655	10.82

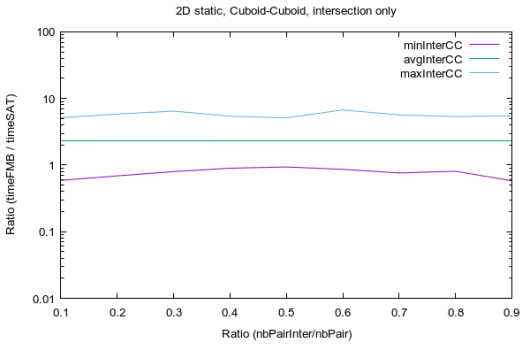
percPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTo
0.1	58686	191926	0.659341	1.788651	4.468750	0.181102	1.292639	10.000000	0.181102	1.342240	10.00
0.2	58788	191252	0.603865	1.791241	5.129032	0.149606	1.291838	10.666667	0.149606	1.391719	10.66
0.3	59054	190672	0.600000	1.794170	4.611940	0.171642	1.295455	9.625000	0.171642	1.445069	9.625
0.4	58284	191688	0.607143	1.791435	4.603175	0.181818	1.293388	10.235294	0.181818	1.492607	10.23
0.5	59046	191718	0.568306	1.791177	3.803279	0.181102	1.291722	10.066667	0.181102	1.541449	10.06
0.6	58670	191068	0.621951	1.790887	4.046875	0.142857	1.292165	8.555556	0.142857	1.591398	8.555
0.7	58518	190956	0.697674	1.793707	4.466667	0.159664	1.296101	12.000000	0.159664	1.644425	12.00
0.8	58738	190564	0.500000	1.793943	4.285714	0.144681	1.293657	12.375000	0.144681	1.693886	12.37
0.9	58152	191340	0.583333	1.790942	3.774194	0.115000	1.291572	10.333333	0.115000	1.741005	10.33

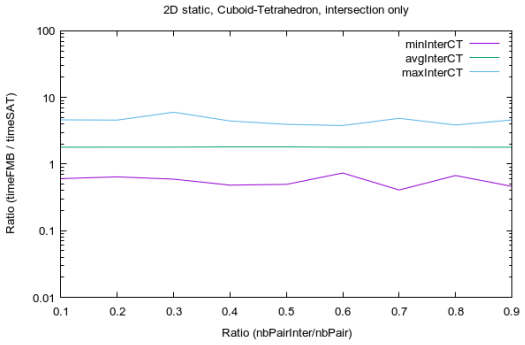
percPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTo
0.1	51478	197052	0.419355	1.314917	3.200000	0.081197	1.352321	10.133333	0.081197	1.348580	10.13
0.2	51552	197628	0.508197	1.320523	3.666667	0.141791	1.354515	10.000000	0.141791	1.347716	10.00
0.3	51124	198246	0.508021	1.320657	3.375000	0.153846	1.356649	12.200000	0.153846	1.346851	12.20
0.4	50908	198466	0.500000	1.317218	3.000000	0.201754	1.356320	13.357143	0.201754	1.340679	13.35
0.5	51370	198648	0.514124	1.316531	3.250000	0.178862	1.354206	10.714286	0.178862	1.335369	10.71
0.6	51050	199120	0.562130	1.316004	3.485714	0.134752	1.354435	10.066667	0.134752	1.331377	10.06
0.7	51336	198958	0.362179	1.319864	3.657143	0.200000	1.356652	14.000000	0.200000	1.330901	14.00
0.8	51782	199256	0.470588	1.320933	3.369231	0.110465	1.358929	12.266667	0.110465	1.328532	12.26
0.9	51612	199428	0.449761	1.318970	2.753623	0.138686	1.354466	9.500000	0.138686	1.322519	9.500

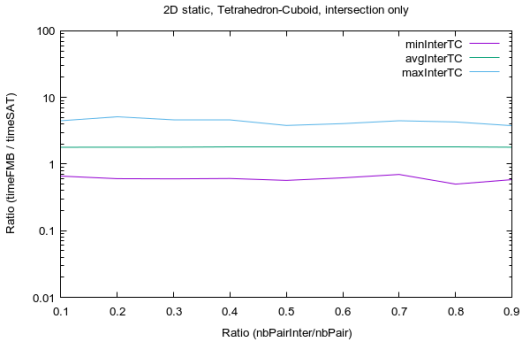

2D static, Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Ratio (nbPairInter/nbPair)

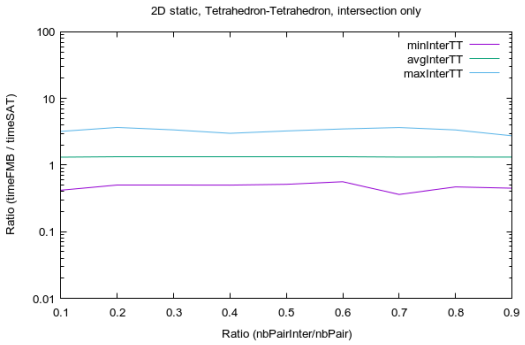

2D static, Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

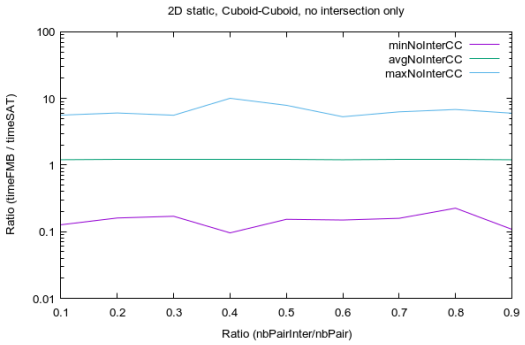

2D static, Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

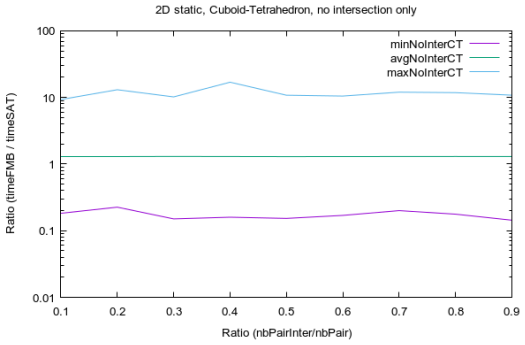

2D static, Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

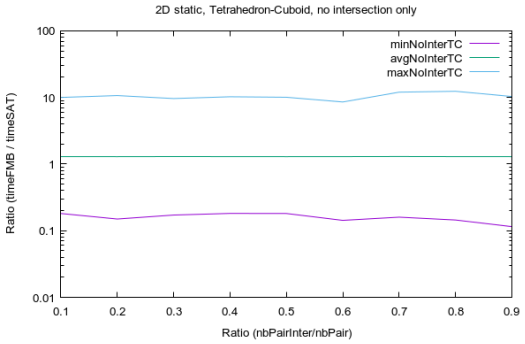

2D static, Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

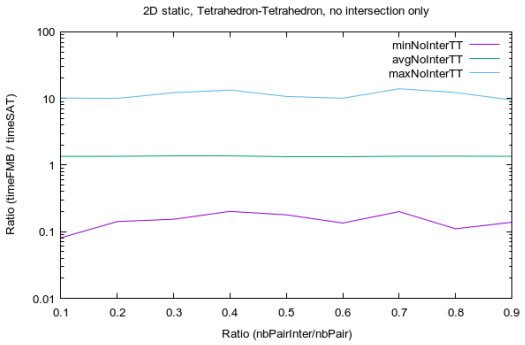

2D static, Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D static, Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D static, Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static, Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static, Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static, Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)

8.2.2 2D static (near case only)

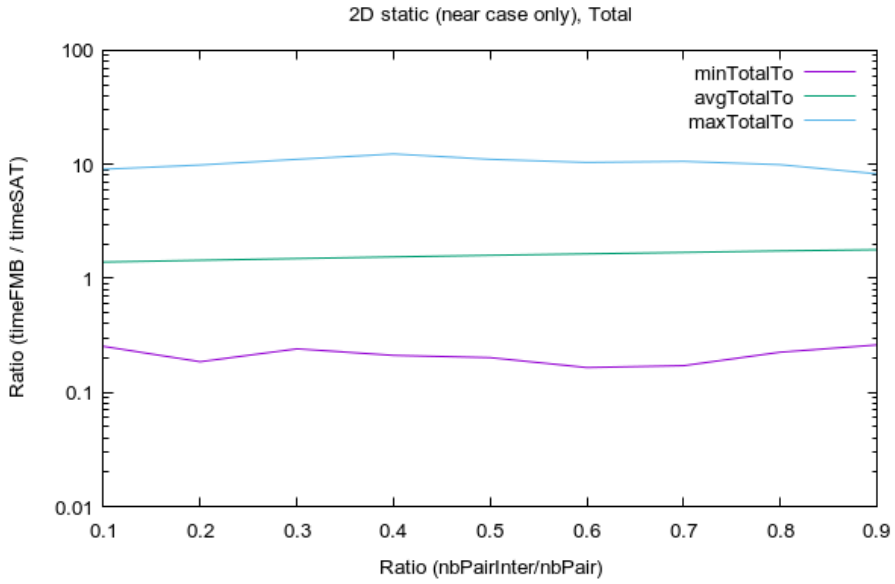
percPairInter	countInterTo	countNoInterTo	miniInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	233406	156388	0.501538	1.838710	5.481481	0.253623	1.341232	9.062500	0.253623	1.390980	9.062500
0.2	234282	157004	0.521739	1.836572	5.049180	0.186047	1.341993	9.866667	0.186047	1.440909	9.866667
0.3	234394	157584	0.546875	1.840049	5.094340	0.241135	1.342700	11.062500	0.241135	1.491905	11.062500
0.4	234076	157760	0.479798	1.843239	5.666667	0.211538	1.344771	12.333333	0.211538	1.544158	12.333333
0.5	234526	158170	0.510870	1.842832	5.396226	0.201835	1.344610	11.071429	0.201835	1.593721	11.071429
0.6	234724	157390	0.000000	1.839467	5.000000	0.165289	1.345216	10.400000	0.165289	1.641766	10.400000
0.7	235002	156462	0.562500	1.838083	5.509434	0.171598	1.346376	10.625000	0.171598	1.690571	10.625000
0.8	236088	156346	0.486111	1.842831	7.232143	0.225490	1.342519	9.937500	0.225490	1.742768	9.937500
0.9	234580	157480	0.564417	1.834748	4.750000	0.261194	1.341711	8.250000	0.261194	1.785444	8.250000

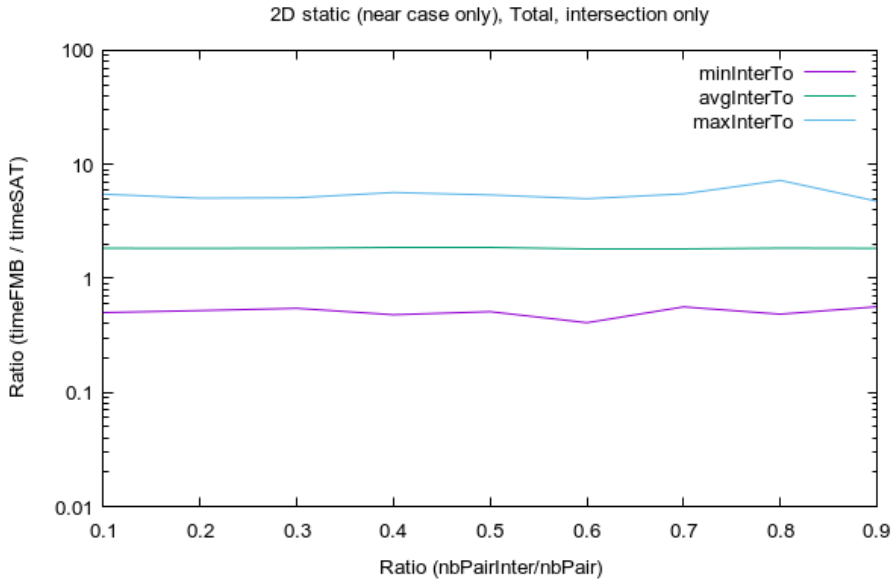
percPairInter	countInterCC	countNoInterCC	miniInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	65948	39230	0.501538	2.319728	5.481481	0.264368	1.260796	5.400000	0.264368	1.366889	5.481481
0.2	65490	39802	0.838509	2.321332	4.603774	0.262411	1.266041	5.175000	0.262411	1.477099	5.175000
0.3	66290	40098	0.765060	2.322963	5.094340	0.250000	1.265120	5.450000	0.250000	1.582473	5.450000
0.4	65528	39298	0.783951	2.329466	5.666667	0.251748	1.264703	5.761905	0.251748	1.690608	5.761905
0.5	66208	39836	0.819209	2.326058	5.396226	0.239583	1.266221	5.463415	0.239583	1.796139	5.463415
0.6	66016	39500	0.889655	2.322306	4.568627	0.218310	1.261099	5.488889	0.218310	1.897824	5.488889
0.7	65652	39252	0.881944	2.322292	5.509434	0.198198	1.259802	5.441860	0.198198	2.003545	5.509434
0.8	66448	39042	0.880000	2.327427	7.232143	0.239583	1.262489	6.235294	0.239583	2.114439	7.232143
0.9	66052	40030	0.984733	2.319563	4.750000	0.298701	1.259939	4.707317	0.298701	2.213600	4.750000

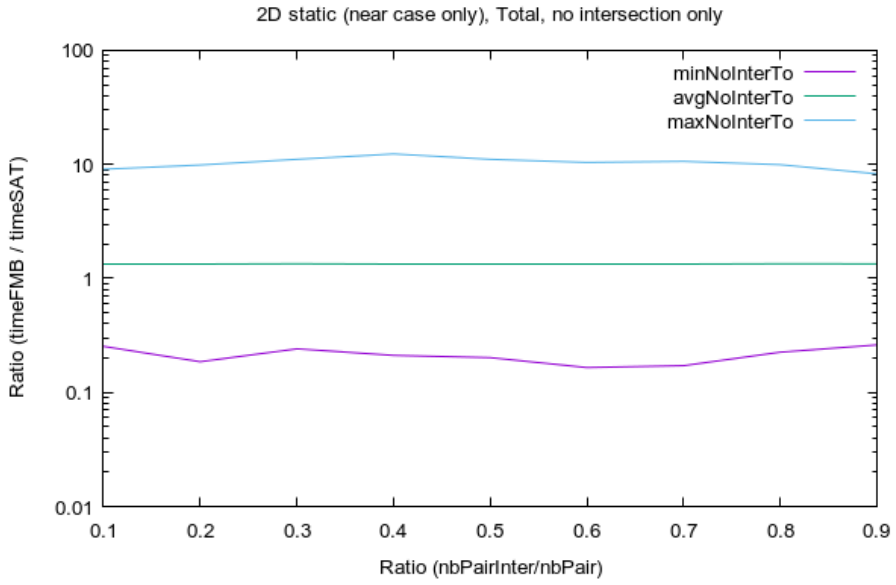
percPairInter	countInterCT	countNoInterCT	miniInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	58084	39462	0.642857	1.793510	4.136364	0.270588	1.356451	9.062500	0.270588	1.400157	9.062500
0.2	58434	38972	0.739130	1.794517	5.049180	0.246914	1.357386	8.764706	0.246914	1.444813	8.764706
0.3	58518	39742	0.693333	1.794414	4.603448	0.267606	1.353885	11.062500	0.267606	1.486044	11.062500
0.4	58730	39876	0.655844	1.797298	4.483333	0.224490	1.355501	12.333333	0.224490	1.532220	12.333333
0.5	58998	39534	0.558685	1.797219	4.156250	0.201835	1.359091	10.750000	0.201835	1.578155	10.750000
0.6	58750	39452	0.634921	1.796283	5.000000	0.165289	1.365561	9.600000	0.165289	1.623994	9.600000
0.7	58916	39398	0.660714	1.795196	4.596774	0.261905	1.369840	8.647059	0.261905	1.667589	8.647059
0.8	59306	39196	0.486111	1.797458	4.500000	0.228916	1.352794	9.937500	0.228916	1.708525	9.937500
0.9	58438	38974	0.682081	1.790702	3.365079	0.312500	1.358857	7.500000	0.312500	1.747518	7.500000

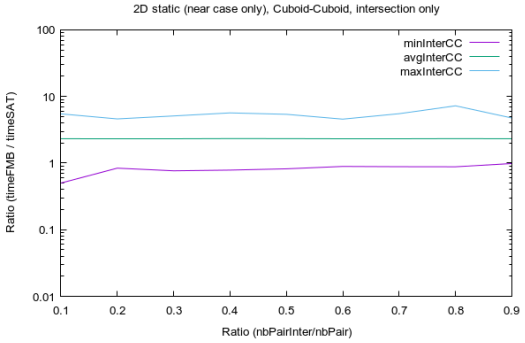
percPairInter	countInterTC	countNoInterTC	miniInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	58688	39050	0.739726	1.791650	4.213115	0.277108	1.348319	7.066667	0.277108	1.392652	7.066667
0.2	58606	39010	0.686275	1.792611	4.786885	0.186047	1.348956	7.687500	0.186047	1.437687	7.687500
0.3	58296	38898	0.594286	1.793437	4.770492	0.283582	1.352914	10.500000	0.283582	1.485071	10.500000
0.4	58578	39678	0.670968	1.797879	4.245902	0.265060	1.355278	9.687500	0.265060	1.532319	9.687500
0.5	58172	39696	0.704819	1.795388	4.693548	0.258427	1.353734	8.687500	0.258427	1.574561	8.687500
0.6	58304	39348	0.772059	1.794255	4.306452	0.243590	1.347491	9.125000	0.243590	1.615550	9.125000
0.7	58898	39290	0.588889	1.792286	4.612903	0.258824	1.351735	10.625000	0.258824	1.660121	10.625000
0.8	58632	39036	0.632530	1.795986	4.741935	0.225490	1.348322	9.937500	0.225490	1.706453	9.937500
0.9	58466	39202	0.801325	1.789436	3.645161	0.261194	1.354912	8.250000	0.261194	1.745983	8.250000

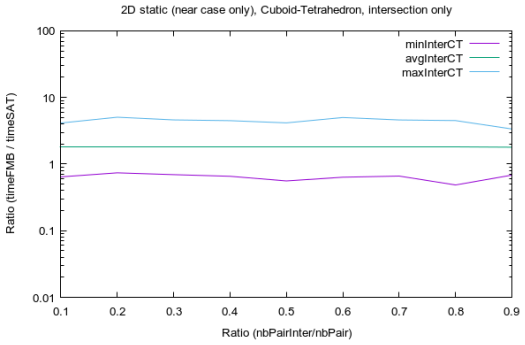
percPairInter	countInterTT	countNoInterTT	miniInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	50686	38646	0.511236	1.319139	3.029851	0.253623	1.400182	8.857143	0.253623	1.392078	8.857143
0.2	51752	39220	0.521739	1.320397	2.736842	0.219512	1.396849	9.866667	0.219512	1.381558	9.866667
0.3	51290	38846	0.546875	1.320949	3.253731	0.241135	1.401111	9.866667	0.241135	1.377062	9.866667
0.4	51240	38908	0.479798	1.325944	3.705882	0.211538	1.403929	10.066667	0.211538	1.372735	10.066667
0.5	51148	39104	0.510870	1.323898	3.301370	0.230769	1.400563	11.071429	0.230769	1.362231	11.071429
0.6	51664	39090	0.408511	1.322525	3.126761	0.231707	1.407389	10.400000	0.231707	1.356471	10.400000
0.7	51536	38522	0.562500	1.322615	3.239437	0.171598	1.405128	9.066667	0.171598	1.347369	9.066667
0.8	51702	39072	0.513661	1.325192	3.527778	0.230000	1.406382	9.875000	0.230000	1.341430	9.875000
0.9	51624	39274	0.564417	1.315612	2.734177	0.286885	1.394864	8.066667	0.286885	1.323538	8.066667

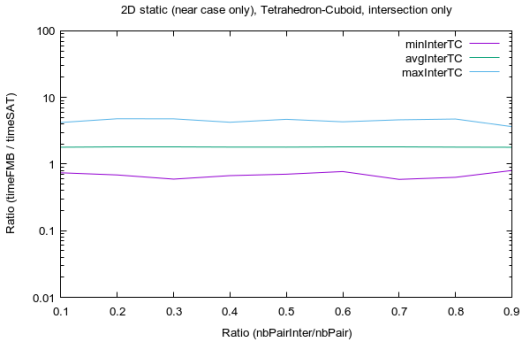

2D static (near case only), Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Ratio (nbPairInter/nbPair)

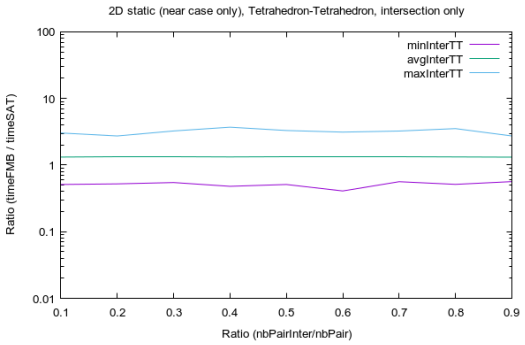

2D static (near case only), Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

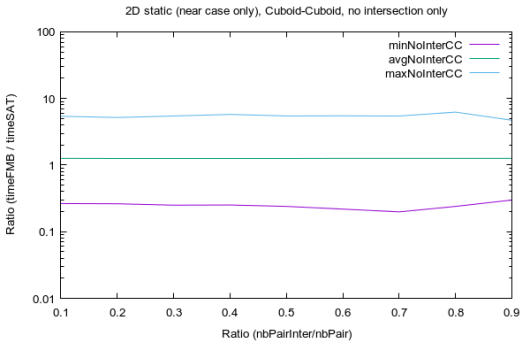

2D static (near case only), Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

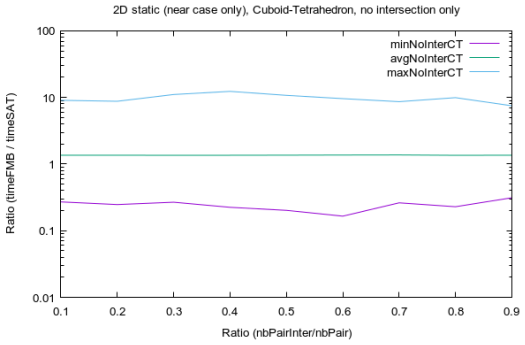

2D static (near case only), Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

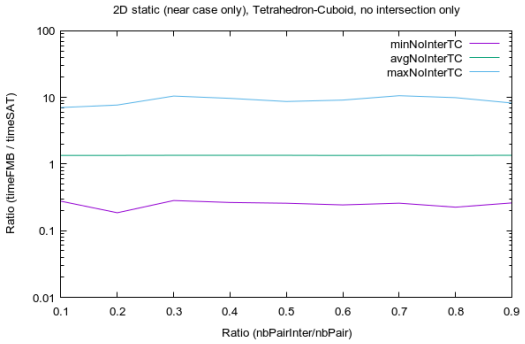

2D static (near case only), Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

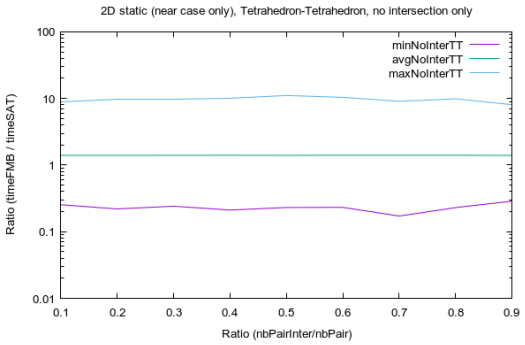

2D static (near case only), Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D static (near case only), Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D static (near case only), Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static (near case only), Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static (near case only), Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


2D static (near case only), Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)

8.2.3 3D static

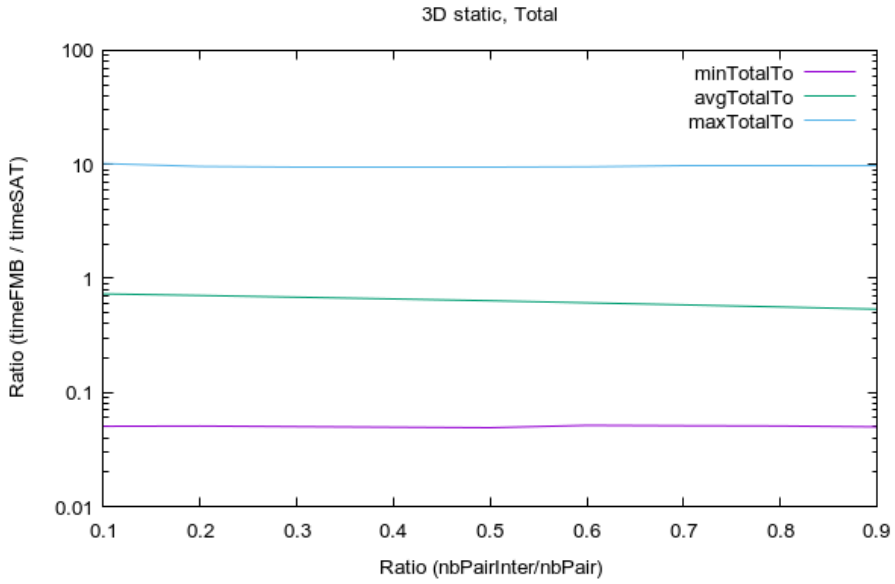
percPairInter	countInterTo	countNoInterTo	miniInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	157578	842422	0.142403	0.513119	1.255121	0.050477	0.756143	10.161290	0.050477	0.731841	10.161290
0.2	157438	842562	0.161316	0.513784	1.033473	0.050776	0.758255	9.593750	0.050776	0.709361	9.593750
0.3	157558	842442	0.185297	0.513251	1.473783	0.050136	0.758252	9.548387	0.050136	0.684751	9.548387
0.4	157876	842124	0.184122	0.513337	1.135569	0.049598	0.758575	9.548387	0.049598	0.660480	9.548387
0.5	158230	841770	0.164116	0.513445	1.022609	0.049046	0.759348	9.548387	0.049046	0.636397	9.548387
0.6	157870	842130	0.143765	0.513607	1.277358	0.051421	0.757680	9.516129	0.051421	0.611236	9.516129
0.7	158074	841924	0.194906	0.512607	1.372323	0.051064	0.758158	9.838710	0.051064	0.586273	9.838710
0.8	157886	842114	0.143987	0.512800	1.116547	0.050824	0.757999	9.833333	0.050824	0.561840	9.833333
0.9	156874	843126	0.156880	0.512751	1.297345	0.049865	0.757428	9.718750	0.049865	0.537218	9.718750

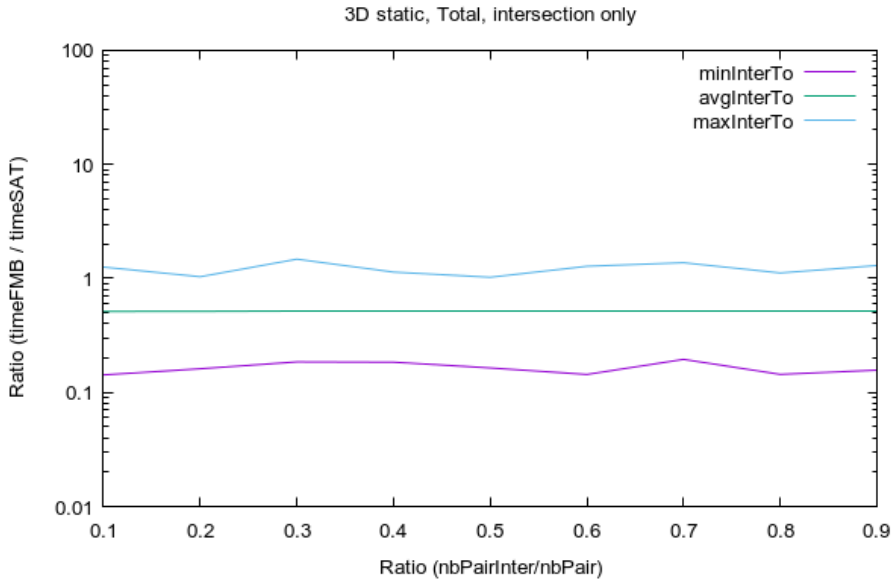
percPairInter	countInterCC	countNoInterCC	miniInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	52862	196782	0.484778	0.769794	1.255121	0.070922	0.560327	3.034965	0.070922	0.581273	3.034965
0.2	52840	196838	0.489893	0.769861	1.033473	0.069767	0.559778	3.284722	0.069767	0.601795	3.284722
0.3	53054	197912	0.616374	0.769991	1.473783	0.084034	0.559998	2.993103	0.084034	0.622996	2.993103
0.4	52842	196776	0.505981	0.770055	1.135569	0.085837	0.558723	3.107914	0.085837	0.643256	3.107914
0.5	53164	196646	0.447284	0.770111	1.022609	0.085653	0.559917	3.124138	0.085653	0.665014	3.124138
0.6	53046	197666	0.442308	0.770038	1.277358	0.074312	0.560469	3.117241	0.074312	0.686211	3.117241
0.7	52648	196448	0.433508	0.770017	1.372323	0.075581	0.558862	3.209790	0.075581	0.706670	3.209790
0.8	52702	196758	0.461538	0.770053	1.114286	0.084926	0.558488	3.731034	0.084926	0.727740	3.731034
0.9	52458	197960	0.464804	0.769882	1.297345	0.089286	0.557596	3.055944	0.089286	0.748653	3.055944

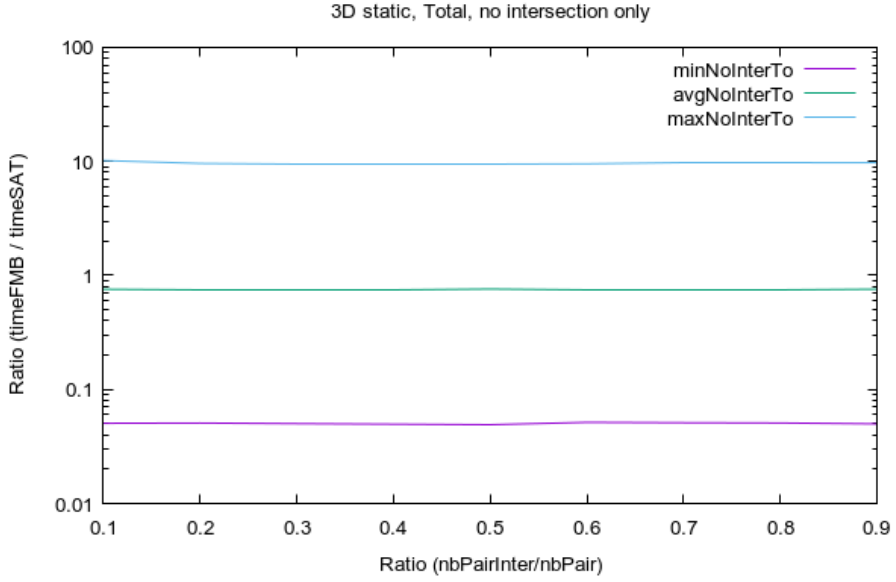
percPairInter	countInterCT	countNoInterCT	miniInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	39324	211112	0.271091	0.431784	0.738247	0.059655	0.732541	9.612903	0.059655	0.702465	9.612903
0.2	39662	210714	0.225365	0.431819	0.709989	0.060127	0.733401	9.483871	0.060127	0.673084	9.483871
0.3	38814	209532	0.258560	0.431967	0.639175	0.060317	0.734073	9.419355	0.060317	0.643441	9.419355
0.4	39432	210132	0.229087	0.431892	1.025899	0.060222	0.733353	9.548387	0.060222	0.612769	9.548387
0.5	39098	210678	0.267094	0.432096	0.753575	0.059282	0.736033	9.483871	0.059282	0.584065	9.483871
0.6	39218	209642	0.206994	0.431919	0.666040	0.060032	0.731587	9.483871	0.060032	0.551786	9.483871
0.7	39438	210726	0.246050	0.431875	0.732946	0.059843	0.734928	9.838710	0.059843	0.522791	9.838710
0.8	39232	211846	0.241517	0.431975	0.799186	0.059748	0.735100	9.833333	0.059748	0.492600	9.833333
0.9	39186	211080	0.275418	0.431890	1.020588	0.059295	0.732005	9.451613	0.059295	0.461901	9.451613

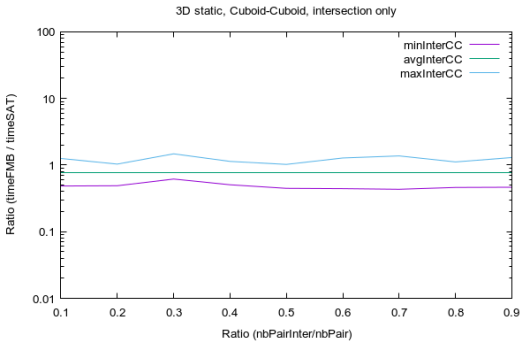
percPairInter	countInterTC	countNoInterTC	miniInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	39004	212080	0.248136	0.431872	0.756018	0.060317	0.730966	10.161290	0.060317	0.701056	10.161290
0.2	39048	210820	0.241266	0.431842	0.666181	0.061093	0.733408	9.593750	0.061093	0.673095	9.593750
0.3	39028	211192	0.198165	0.431786	0.735582	0.059098	0.736357	9.548387	0.059098	0.644986	9.548387
0.4	39480	211166	0.184122	0.431829	1.027586	0.061889	0.735502	9.531250	0.061889	0.614033	9.531250
0.5	39446	210336	0.260404	0.431987	0.705710	0.060317	0.735010	9.548387	0.060317	0.583499	9.548387
0.6	39318	211980	0.226415	0.431921	1.017143	0.058917	0.734739	9.516129	0.058917	0.553048	9.516129
0.7	39706	211016	0.281414	0.431768	0.736614	0.060032	0.734709	9.483871	0.060032	0.522650	9.483871
0.8	39590	209928	0.214981	0.431942	1.116547	0.060413	0.733061	9.580645	0.060413	0.492166	9.580645
0.9	38916	210408	0.251020	0.431803	0.761518	0.060317	0.736310	9.718750	0.060317	0.462254	9.718750

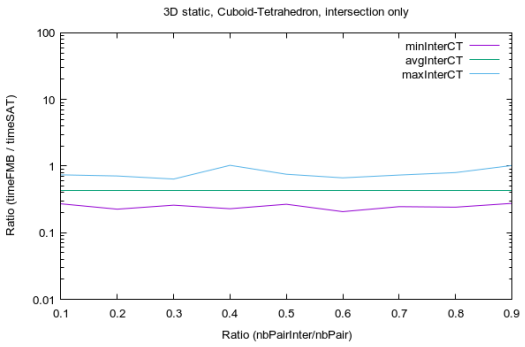
percPairInter	countInterTT	countNoInterTT	miniInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	26388	223448	0.142403	0.240230	0.461350	0.050477	0.973910	9.041667	0.050477	0.900542	9.041667
0.2	25888	224190	0.161316	0.240279	0.428402	0.050776	0.979241	9.120000	0.050776	0.831448	9.120000
0.3	26662	223806	0.185297	0.239952	0.358944	0.050136	0.976865	8.958333	0.050136	0.755791	8.958333
0.4	26122	224050	0.184753	0.240156	0.329146	0.049598	0.979499	9.043478	0.049598	0.683762	9.043478
0.5	26522	224110	0.164116	0.240028	0.304819	0.049046	0.979100	9.173913	0.049046	0.609564	9.173913
0.6	26288	222842	0.143765	0.240198	0.380074	0.051421	0.978981	9.173913	0.051421	0.535711	9.173913
0.7	26282	223734	0.194906	0.240241	0.411125	0.051064	0.977143	9.260870	0.051064	0.461312	9.260870
0.8	26362	223582	0.143987	0.240226	0.372571	0.050824	0.978684	9.217391	0.050824	0.387918	9.217391
0.9	26314	223678	0.156880	0.240279	0.416149	0.049865	0.978140	8.913043	0.049865	0.314065	8.913043

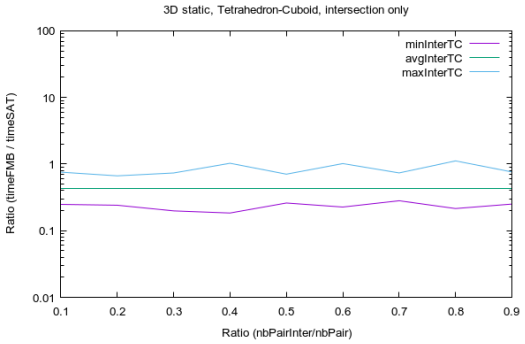

3D static, Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

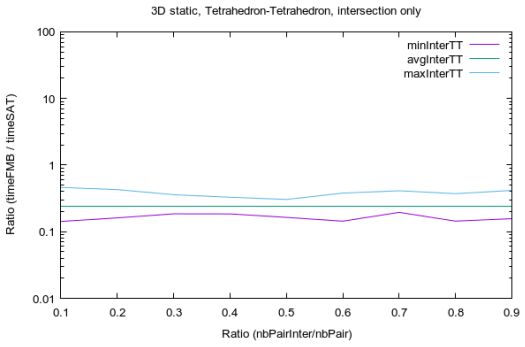

3D static, Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

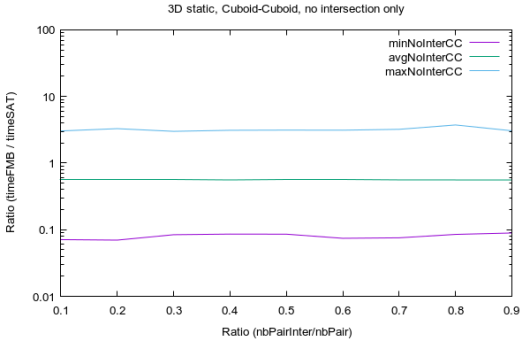

3D static, Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

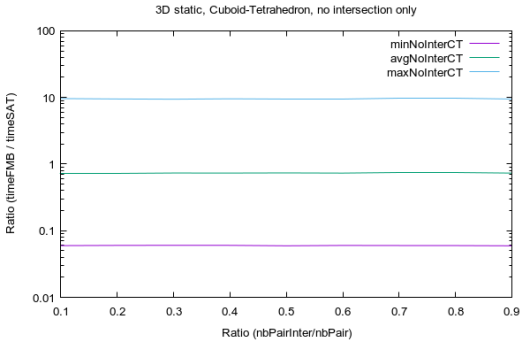

3D static, Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

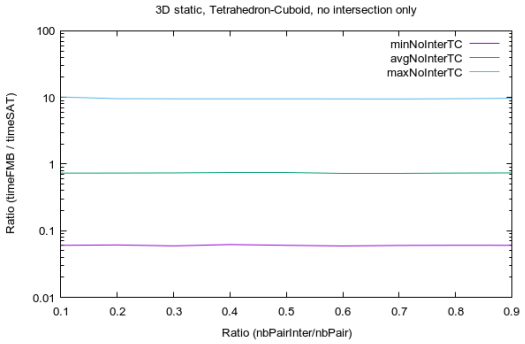

3D static, Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

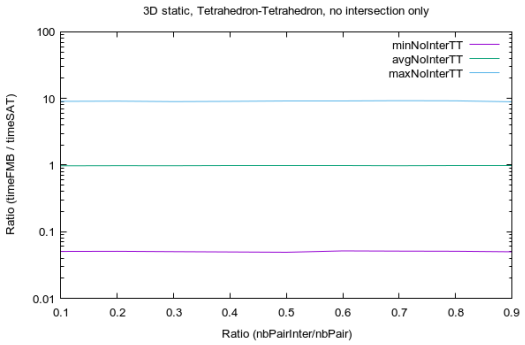

3D static, Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static, Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static, Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static, Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static, Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static, Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

8.2.4 3D static (near case only)

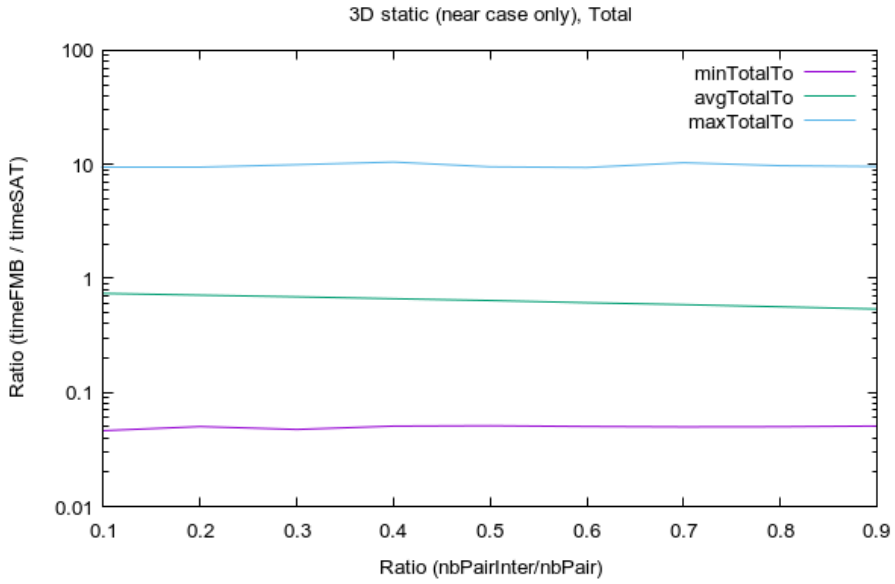
percPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	156960	305452	0.170732	0.513525	1.420959	0.046250	0.764167	9.548387	0.046250	0.739103	9.548387
0.2	158164	304326	0.170330	0.513051	1.005310	0.050204	0.763715	9.548387	0.050204	0.713582	9.548387
0.3	157536	303920	0.185567	0.513070	1.010435	0.047368	0.764565	9.935484	0.047368	0.689117	9.935484
0.4	157736	305200	0.179076	0.514548	1.374126	0.050633	0.763230	10.480000	0.050633	0.663757	10.480000
0.5	158102	304776	0.183847	0.513781	1.234973	0.051064	0.764886	9.516129	0.051064	0.639334	9.516129
0.6	158036	304538	0.167877	0.512539	1.188618	0.050279	0.765151	9.391304	0.050279	0.612128	9.391304
0.7	157382	303356	0.154966	0.513690	1.059369	0.049865	0.765131	10.333333	0.049865	0.589122	10.333333
0.8	158476	304692	0.175455	0.513666	1.067518	0.050070	0.764577	9.741935	0.050070	0.563848	9.741935
0.9	156816	305552	0.194292	0.514038	1.138376	0.050754	0.763847	9.580645	0.050754	0.539019	9.580645

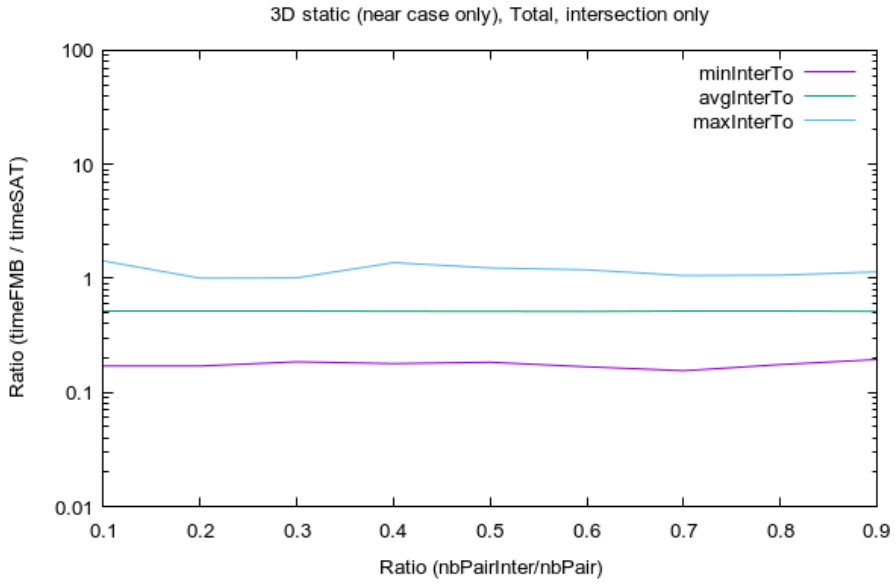
percPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	52760	81254	0.500597	0.769837	1.420959	0.087912	0.587337	3.139860	0.087912	0.605587	3.139860
0.2	53322	80994	0.514851	0.769636	1.005310	0.086768	0.588537	3.324138	0.086768	0.624757	3.324138
0.3	52666	81072	0.454646	0.769466	1.010435	0.071533	0.587764	3.111888	0.071533	0.642274	3.111888
0.4	53290	81784	0.471418	0.769627	1.374126	0.089888	0.591501	3.126374	0.089888	0.662751	3.126374
0.5	53294	81084	0.472579	0.769648	1.234973	0.086768	0.587957	3.095890	0.086768	0.678803	3.095890
0.6	52892	81128	0.469989	0.769651	1.188618	0.086207	0.589422	3.294521	0.086207	0.697559	3.294521
0.7	53056	80734	0.585331	0.770033	1.059369	0.088889	0.587234	3.241379	0.088889	0.715194	3.241379
0.8	53268	80868	0.587904	0.770274	1.067518	0.086694	0.586710	3.187500	0.086694	0.733561	3.187500
0.9	53026	81610	0.474596	0.770136	1.138376	0.085653	0.587901	3.416667	0.085653	0.751913	3.416667

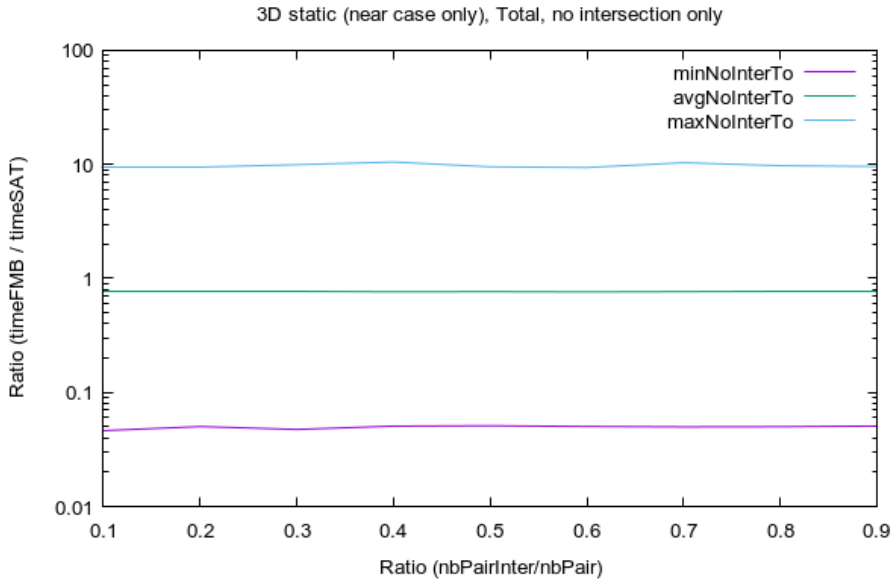
percPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	39340	76988	0.215278	0.431518	0.874674	0.058824	0.766887	9.548387	0.058824	0.733350	9.548387
0.2	38516	76690	0.281501	0.431507	0.618209	0.058642	0.765989	9.548387	0.058642	0.699092	9.548387
0.3	39798	76660	0.273196	0.431402	0.660844	0.059190	0.768381	9.935484	0.059190	0.667287	9.935484
0.4	39318	76964	0.272841	0.431518	0.727273	0.061192	0.766371	9.548387	0.061192	0.632430	9.548387
0.5	38986	77034	0.279528	0.431513	0.632117	0.061093	0.769232	9.516129	0.061093	0.600373	9.516129
0.6	39236	76712	0.282322	0.431617	0.653901	0.060995	0.764808	9.281250	0.060995	0.564893	9.281250
0.7	39202	76632	0.209144	0.431668	0.837004	0.059655	0.767024	9.322581	0.059655	0.532275	9.322581
0.8	39164	76828	0.219561	0.431912	0.741888	0.058282	0.767685	9.483871	0.058282	0.499067	9.483871
0.9	39084	77286	0.251012	0.431595	0.681351	0.057453	0.768507	9.451613	0.057453	0.465286	9.451613

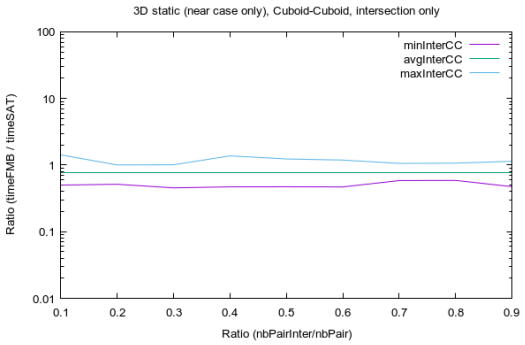
percPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	38898	76942	0.243434	0.431617	0.727151	0.059968	0.767207	9.322581	0.059968	0.733648	9.322581
0.2	39586	76834	0.251876	0.431416	0.852555	0.059561	0.765627	9.451613	0.059561	0.698785	9.451613
0.3	39352	76744	0.278726	0.431372	0.749640	0.060032	0.769871	9.387097	0.060032	0.668321	9.387097
0.4	39498	77170	0.219067	0.431584	1.030612	0.058915	0.764164	9.419355	0.058915	0.631132	9.419355
0.5	39696	76432	0.220508	0.431570	0.622318	0.060413	0.767253	9.218750	0.060413	0.599411	9.218750
0.6	39466	76902	0.258683	0.431351	0.641834	0.060656	0.759138	9.322581	0.060656	0.562466	9.322581
0.7	38870	75978	0.224192	0.431612	0.671470	0.057315	0.768948	9.451613	0.057315	0.532813	9.451613
0.8	39714	77058	0.260173	0.431769	0.717579	0.060510	0.768384	9.741935	0.060510	0.499092	9.741935
0.9	38500	76702	0.280105	0.431759	0.953103	0.060163	0.764277	9.580645	0.060163	0.465011	9.580645

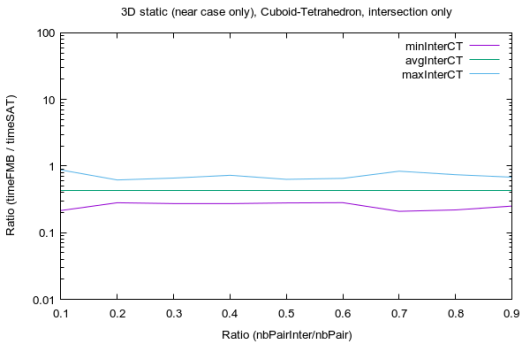
percPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	25962	70268	0.170732	0.239631	0.505857	0.046250	0.962337	8.791667	0.046250	0.890066	8.791667
0.2	26740	69808	0.170330	0.239701	0.337838	0.050204	0.962363	8.791667	0.050204	0.817831	8.791667
0.3	25720	69444	0.185567	0.239427	0.495283	0.047368	0.960895	8.791667	0.047368	0.744454	8.791667
0.4	25630	69282	0.179076	0.239415	0.447520	0.050633	0.961420	10.480000	0.050633	0.672618	10.480000
0.5	26126	70226	0.183847	0.239516	0.336970	0.051064	0.961828	8.833333	0.051064	0.600672	8.833333
0.6	26442	69796	0.167877	0.239493	0.442557	0.050279	0.960532	9.391304	0.050279	0.527909	9.391304
0.7	26254	70012	0.154966	0.239646	0.467259	0.049865	0.964057	10.333333	0.049865	0.456969	10.333333
0.8	26330	69938	0.175455	0.239653	0.389024	0.050070	0.962632	9.043478	0.050070	0.384249	9.043478
0.9	26206	69954	0.194292	0.239678	0.330498	0.050754	0.963489	9.173913	0.050754	0.312059	9.173913

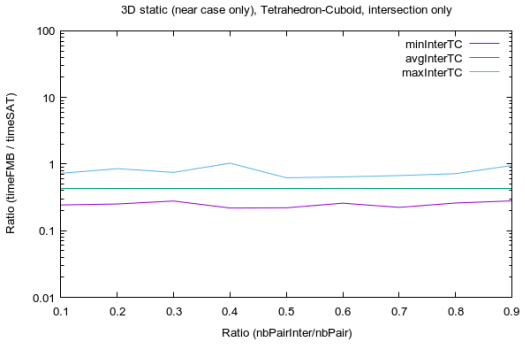

3D static (near case only), Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Ratio (nbPairInter/nbPair)

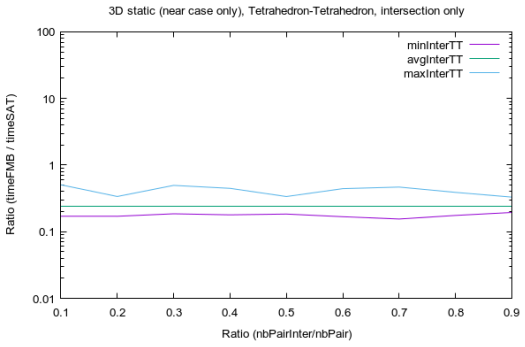

3D static (near case only), Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

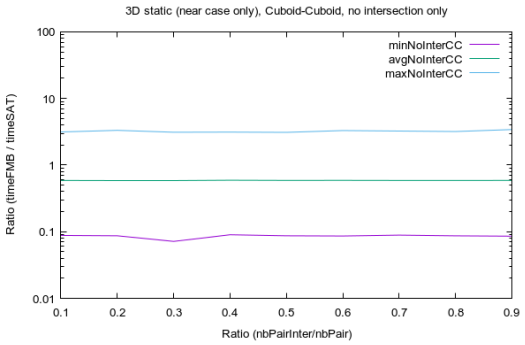

3D static (near case only), Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

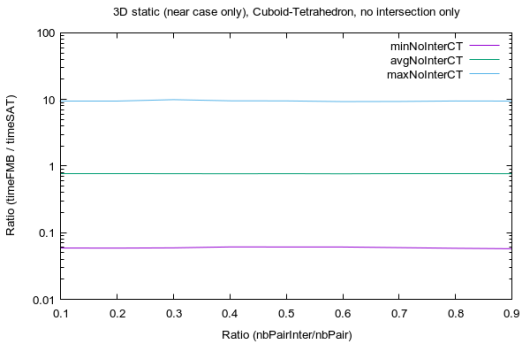

3D static (near case only), Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

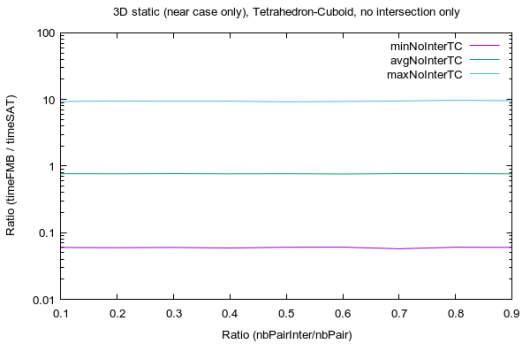

3D static (near case only), Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

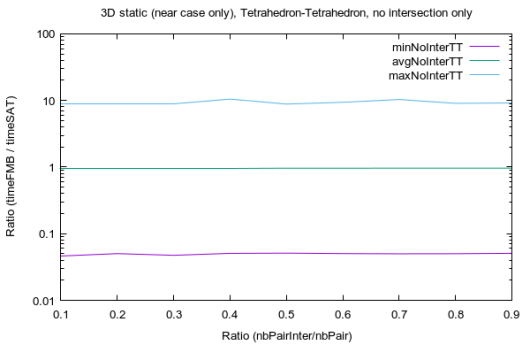

3D static (near case only), Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static (near case only), Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static (near case only), Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static (near case only), Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static (near case only), Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D static (near case only), Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

8.2.5 2D dynamic

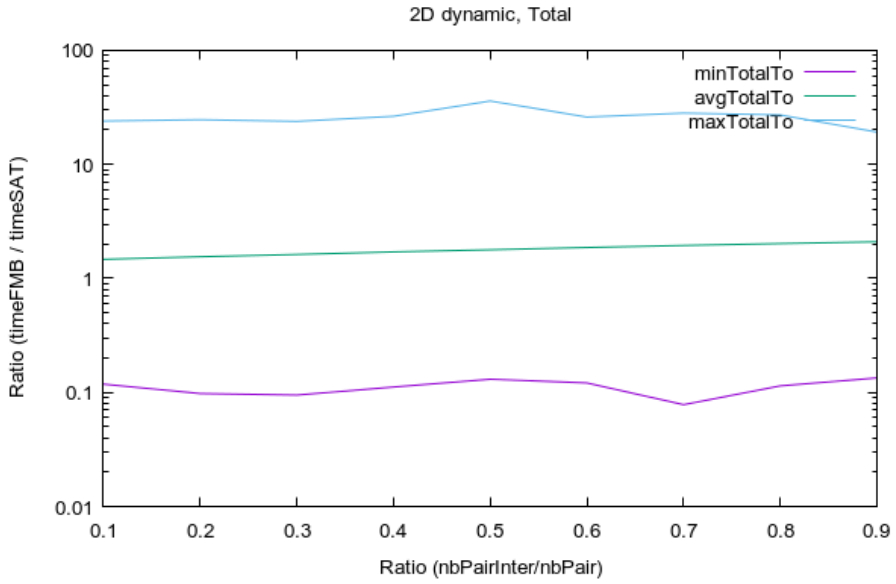
percPairInter	countInterTo	countNoInterTo	miniInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	371892	628060	0.778761	2.175164	5.396739	0.118227	1.390808	23.961538	0.118227	1.469244	23.961538
0.2	371430	628520	0.562874	2.174903	5.381679	0.097872	1.393616	24.640000	0.097872	1.549873	24.640000
0.3	371230	628738	0.497207	2.172810	5.838235	0.094915	1.386621	23.807692	0.094915	1.622478	23.807692
0.4	371408	628558	0.762463	2.175611	5.559211	0.111628	1.394623	26.407407	0.111628	1.707018	26.407407
0.5	373114	626834	0.685039	2.172567	5.978417	0.130612	1.392596	35.931034	0.130612	1.782581	35.931034
0.6	372788	627156	0.678481	2.181973	6.779412	0.121212	1.395769	26.000000	0.121212	1.867491	26.000000
0.7	371566	628404	0.699519	2.178499	7.315789	0.078078	1.397811	28.142857	0.078078	1.944293	28.142857
0.8	371772	628186	0.689873	2.177925	6.263889	0.114155	1.392820	27.137931	0.114155	2.020904	27.137931
0.9	371560	628406	0.701058	2.170090	5.176471	0.134021	1.391954	19.200000	0.134021	2.092277	19.200000

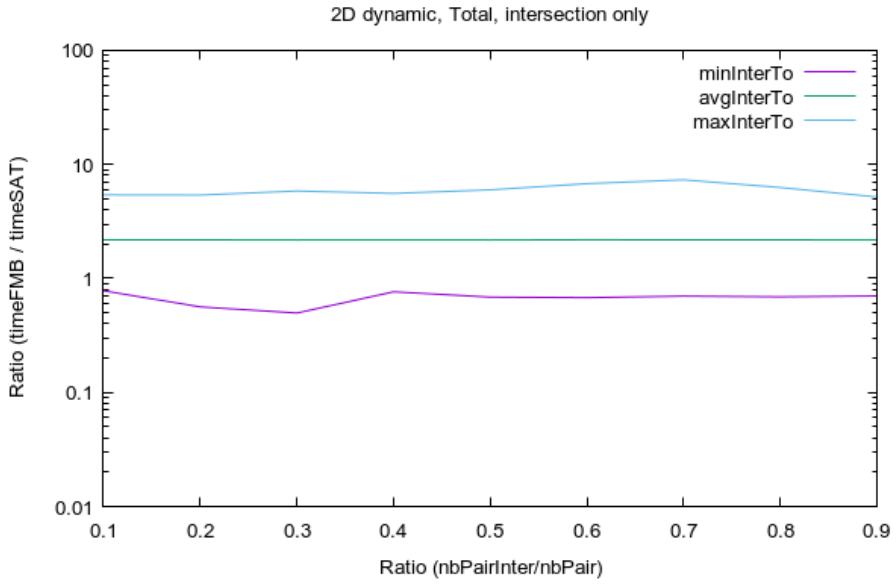
percPairInter	countInterCC	countNoInterCC	miniInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	99722	150420	1.132353	2.759415	5.396739	0.166667	1.342687	20.428571	0.166667	1.484360	20.428571
0.2	99578	149786	1.165138	2.759695	5.381679	0.097872	1.365786	23.806452	0.097872	1.644568	23.806452
0.3	98958	150646	1.079365	2.759582	5.838235	0.165644	1.342536	20.896552	0.165644	1.767650	20.896552
0.4	99184	150358	1.181818	2.760641	5.559211	0.150327	1.346299	21.862069	0.150327	1.912036	21.862069
0.5	99324	150514	1.153846	2.758382	5.843284	0.130612	1.347610	23.310345	0.130612	2.052996	23.310345
0.6	100050	150404	1.106509	2.768083	6.779412	0.136667	1.355624	23.900000	0.136667	2.203100	23.900000
0.7	99126	149514	1.124629	2.764860	6.568182	0.139037	1.347958	22.321429	0.139037	2.339789	22.321429
0.8	99804	150522	1.127726	2.763051	6.263889	0.137566	1.361993	27.137931	0.137566	2.482840	27.137931
0.9	99578	150850	1.162088	2.755616	5.176471	0.162921	1.345540	18.551724	0.162921	2.614608	18.551724

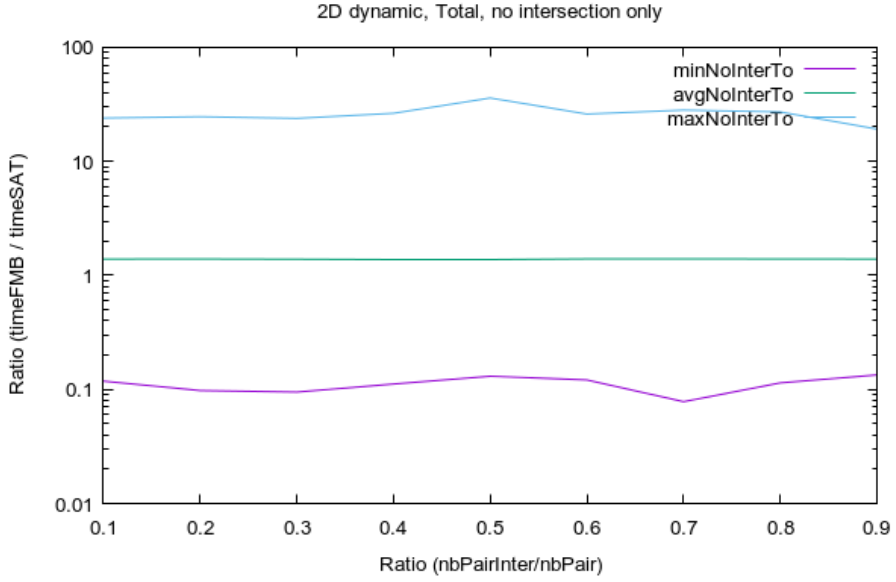
percPairInter	countInterCT	countNoInterCT	miniInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	93284	156876	0.905028	2.119622	4.755396	0.128713	1.394380	19.400000	0.128713	1.466904	19.400000
0.2	92932	157356	0.778393	2.120209	5.000000	0.131148	1.384859	19.769231	0.131148	1.531929	19.769231
0.3	92660	156854	0.765306	2.119680	4.870504	0.140244	1.388088	23.807692	0.140244	1.607565	23.807692
0.4	93112	157370	0.794872	2.120695	4.748201	0.118812	1.403397	26.407407	0.118812	1.690316	26.407407
0.5	93540	156988	0.833760	2.118487	5.978417	0.130682	1.396997	35.931034	0.130682	1.757742	35.931034
0.6	92912	155924	0.757979	2.126465	6.086331	0.121212	1.398272	26.000000	0.121212	1.835188	26.000000
0.7	93280	157288	0.699519	2.124176	7.315789	0.118557	1.414027	28.142857	0.118557	1.911131	28.142857
0.8	92796	157570	0.757212	2.122765	5.226277	0.114155	1.397737	23.083333	0.114155	1.977760	23.083333
0.9	93116	157050	0.789604	2.114516	4.834532	0.134021	1.400024	19.200000	0.134021	2.043067	19.200000

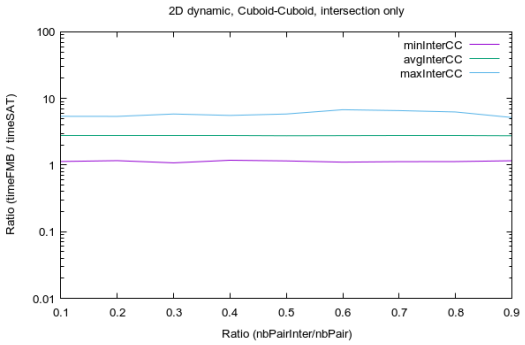
percPairInter	countInterTC	countNoInterTC	miniInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	92660	156858	0.863768	2.114987	4.966667	0.118227	1.389391	21.370370	0.118227	1.461951	21.370370
0.2	92640	157662	0.562874	2.114838	4.901408	0.129213	1.394852	24.640000	0.129213	1.538849	24.640000
0.3	92742	156606	0.763723	2.114069	4.950704	0.094915	1.384176	21.692308	0.094915	1.603144	21.692308
0.4	93202	156600	0.877676	2.115652	4.719424	0.111628	1.396910	24.346154	0.111628	1.684407	24.346154
0.5	93380	156644	0.838120	2.115217	5.473333	0.142857	1.399256	24.461538	0.142857	1.757237	24.461538
0.6	93074	157388	0.822275	2.121025	6.108844	0.125926	1.387319	25.320000	0.125926	1.827543	25.320000
0.7	92594	158122	0.864023	2.119804	5.553957	0.078078	1.399456	21.884615	0.078078	1.903700	21.884615
0.8	92586	156204	0.770992	2.117338	5.205714	0.125000	1.382531	26.400000	0.125000	1.970376	26.400000
0.9	92622	156858	0.772152	2.110620	4.583893	0.150327	1.396690	17.040000	0.150327	2.039227	17.040000

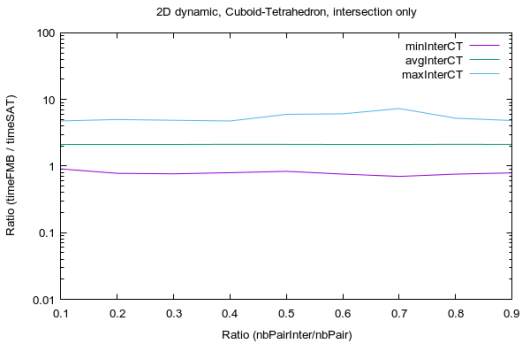
percPairInter	countInterTT	countNoInterTT	miniInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	86226	163906	0.777761	1.624224	3.628743	0.153846	1.432907	23.961538	0.153846	1.452039	23.961538
0.2	86280	163716	0.762319	1.623381	3.962025	0.106299	1.426304	19.400000	0.106299	1.465719	19.400000
0.3	86870	164632	0.497207	1.623770	3.475000	0.130233	1.427891	17.800000	0.130233	1.486654	17.800000
0.4	85910	164230	0.762463	1.624757	3.650602	0.129353	1.428277	19.520000	0.129353	1.506869	19.520000
0.5	86870	162688	0.685039	1.622648	3.478788	0.147727	1.423555	21.888889	0.147727	1.523101	21.888889
0.6	86752	163440	0.678481	1.630857	4.712575	0.126984	1.438461	20.560000	0.126984	1.553898	20.560000
0.7	86566	163480	0.774011	1.628380	4.408537	0.129353	1.426212	20.680000	0.129353	1.567730	20.680000
0.8	86586	163890	0.689873	1.627377	3.648485	0.138728	1.426210	26.928571	0.138728	1.587144	26.928571
0.9	86244	163648	0.701058	1.617908	3.943396	0.151515	1.422456	16.222222	0.151515	1.598363	16.222222

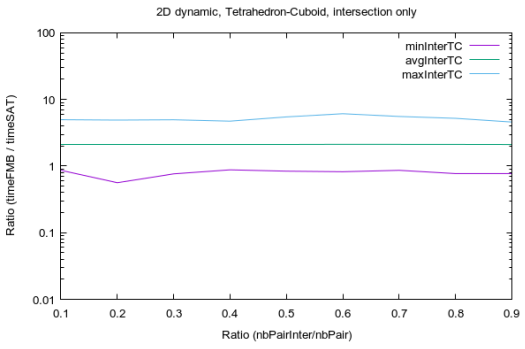

2D dynamic, Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

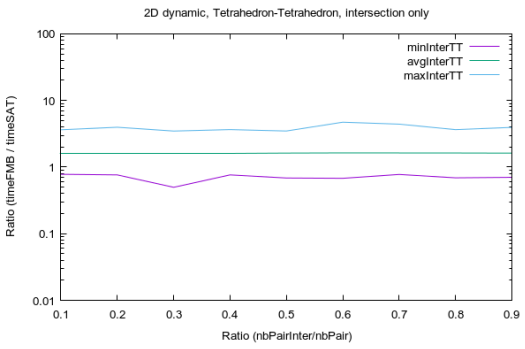

2D dynamic, Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

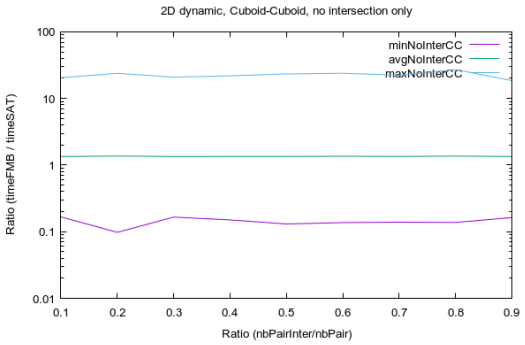

2D dynamic, Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

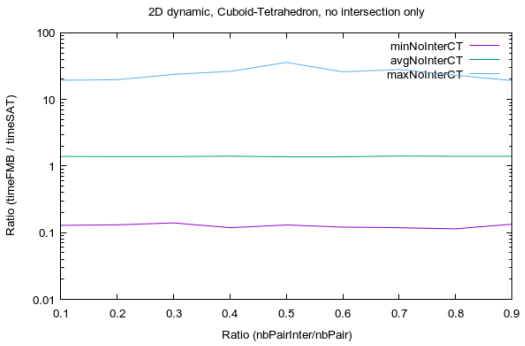

2D dynamic, Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

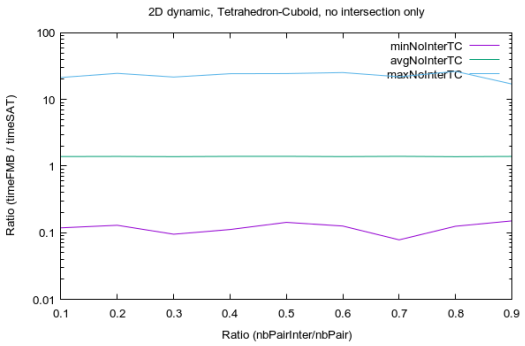

2D dynamic, Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

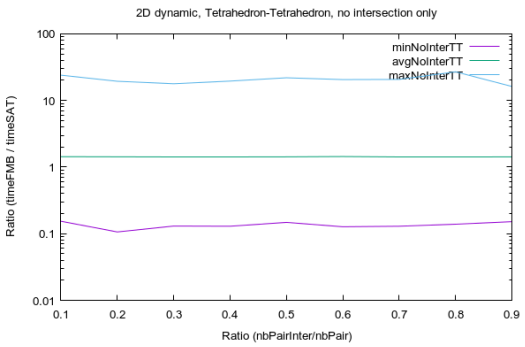

2D dynamic, Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D dynamic, Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D dynamic, Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D dynamic, Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D dynamic, Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


2D dynamic, Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

8.2.6 3D dynamic

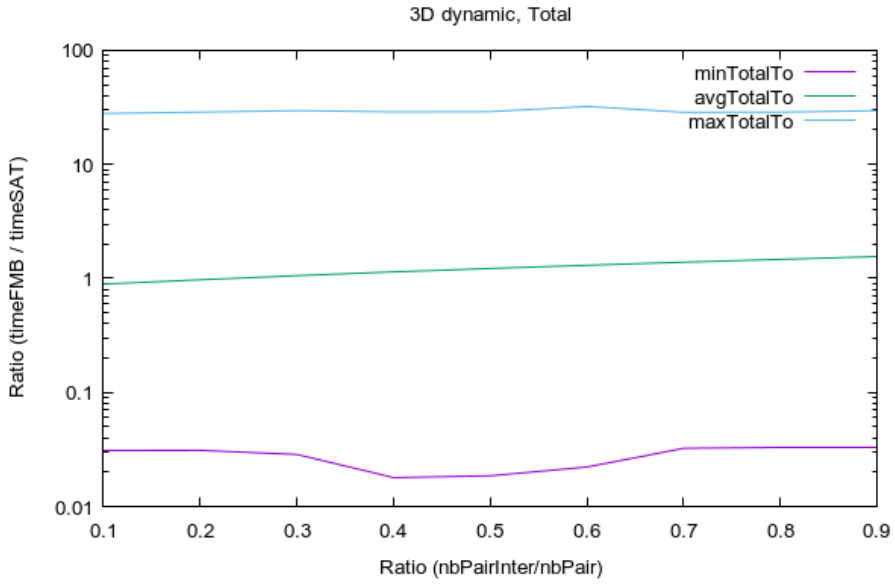
percPairInter	countInterTo	countNoInterTo	miniInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	263194	736806	0.175549	1.639173	4.248101	0.030902	0.808404	27.916667	0.030902	0.891481	27.916667
0.2	260930	739070	0.180339	1.636240	4.259292	0.031052	0.806947	28.694444	0.031052	0.972805	28.694444
0.3	262892	737108	0.155391	1.635564	4.250213	0.028604	0.809016	29.589744	0.028604	1.056980	29.589744
0.4	261756	738244	0.124575	1.637993	4.834813	0.017921	0.808752	28.891892	0.017921	1.140448	28.891892
0.5	261870	738130	0.132956	1.633757	4.557876	0.018583	0.810421	29.000000	0.018583	1.222089	29.000000
0.6	262326	737674	0.181921	1.634627	4.575652	0.022170	0.806904	32.210526	0.022170	1.303538	32.210526
0.7	261348	738652	0.292173	1.635224	3.520737	0.032382	0.806485	28.638889	0.032382	1.386602	28.638889
0.8	262002	737998	0.292089	1.632599	3.457875	0.032941	0.808056	28.694444	0.032941	1.467690	28.694444
0.9	261110	738888	0.294479	1.636754	3.275362	0.032915	0.807629	29.571429	0.032915	1.553842	29.571429

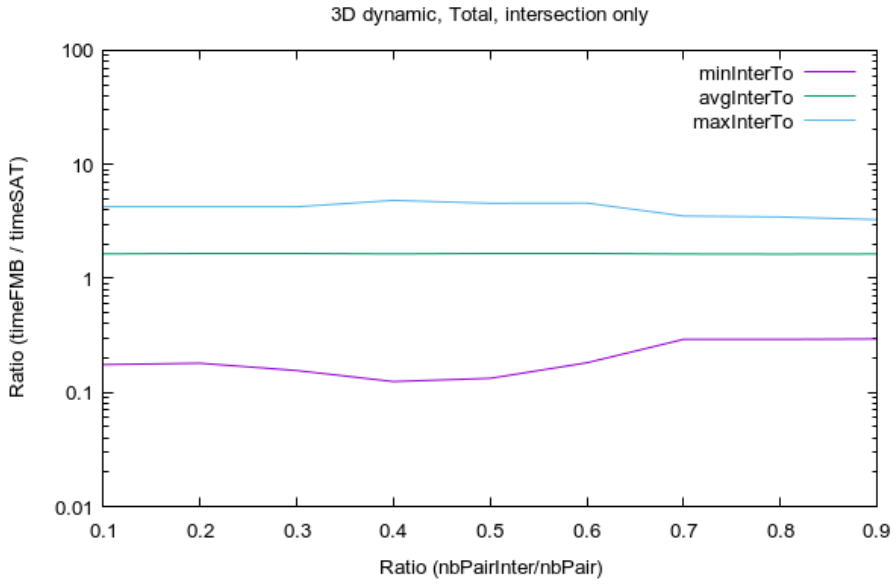
percPairInter	countInterCC	countNoInterCC	miniInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	80640	169390	0.881746	2.641749	4.248101	0.041852	0.699632	15.767932	0.041852	0.893844	15.767932
0.2	79784	170862	0.915978	2.641303	4.259292	0.045100	0.698866	14.404255	0.045100	1.087353	14.404255
0.3	79954	169380	0.905803	2.642452	4.250213	0.047667	0.700021	16.177489	0.047667	1.282750	16.177489
0.4	80092	170958	0.785884	2.647210	4.834813	0.030259	0.693000	15.141593	0.030259	1.474684	15.141593
0.5	79542	169634	0.704525	2.644882	4.557876	0.030539	0.700971	23.033333	0.030539	1.672927	23.033333
0.6	79996	168230	0.775031	2.644611	4.575652	0.033869	0.689986	20.543933	0.033869	1.862761	20.543933
0.7	79862	171062	1.870947	2.640875	3.520737	0.048574	0.690431	13.177778	0.048574	2.055742	13.177778
0.8	80046	169354	1.876884	2.638368	3.457875	0.049180	0.695166	13.213974	0.049180	2.249727	13.213974
0.9	79698	169972	1.719172	2.640427	3.275362	0.049303	0.697159	13.248908	0.049303	2.446101	13.248908

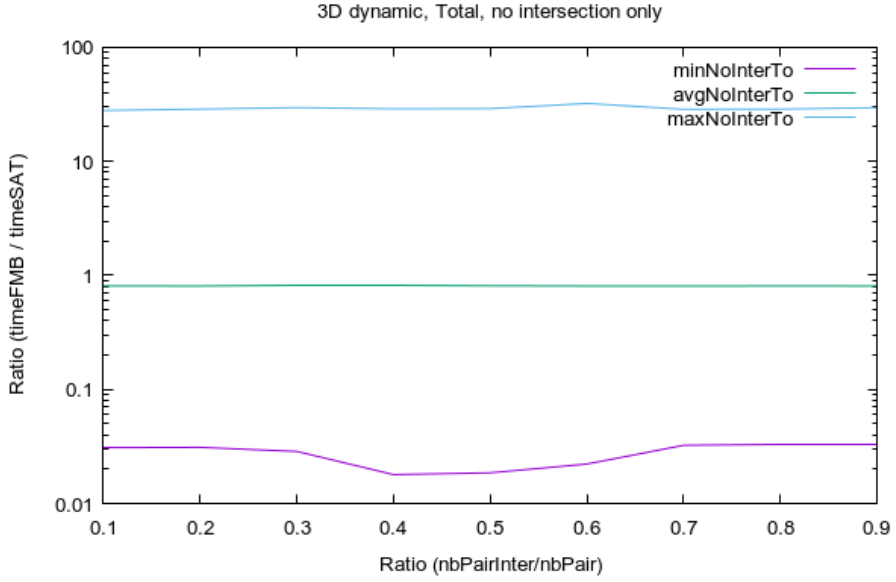
percPairInter	countInterCT	countNoInterCT	miniInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	65846	183950	0.233984	1.435141	3.999372	0.033254	0.799167	22.300000	0.033254	0.862765	22.300000
0.2	65112	184734	0.280967	1.433443	3.498914	0.031052	0.801635	23.940000	0.031052	0.927996	23.940000
0.3	66150	184072	0.285161	1.433925	3.888715	0.028604	0.797905	27.795918	0.028604	0.988711	27.795918
0.4	65432	184814	0.230411	1.433711	4.099338	0.017921	0.806479	26.777778	0.017921	1.057372	26.777778
0.5	65640	184526	0.220530	1.433468	3.627119	0.024416	0.803610	24.264151	0.024416	1.118539	24.264151
0.6	65152	185366	0.181921	1.432126	3.535494	0.022170	0.801024	24.166667	0.022170	1.179685	24.166667
0.7	65438	184192	0.467785	1.434412	2.901884	0.033153	0.798271	22.979592	0.033153	1.243570	22.979592
0.8	65090	183506	0.417153	1.434354	2.742879	0.033541	0.798776	22.571429	0.033541	1.307238	22.571429
0.9	65466	184964	0.410076	1.433482	2.691388	0.033568	0.803939	22.775510	0.033568	1.370528	22.775510

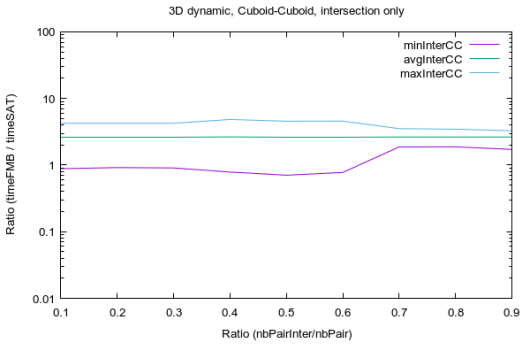
percPairInter	countInterTC	countNoInterTC	miniInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	66278	185384	0.175549	1.433192	3.026667	0.032428	0.802924	21.607843	0.032428	0.865951	21.607843
0.2	65568	184706	0.180339	1.433615	3.367135	0.033489	0.795388	26.908108	0.033489	0.923033	26.908108
0.3	66144	184800	0.247301	1.434365	3.408661	0.029912	0.805509	23.600000	0.029912	0.994166	23.600000
0.4	65238	183698	0.184826	1.436486	3.853957	0.031711	0.798870	23.632653	0.031711	1.053916	23.632653
0.5	65816	184380	0.238968	1.432823	4.318557	0.030627	0.800580	22.980392	0.030627	1.116702	22.980392
0.6	66206	183932	0.182671	1.431859	3.744275	0.028994	0.804502	23.888889	0.028994	1.180916	23.888889
0.7	65188	184540	0.423190	1.433584	3.104151	0.033844	0.805940	21.803922	0.033844	1.245291	21.803922
0.8	65476	185318	0.437811	1.432745	2.839526	0.032941	0.801852	22.854167	0.032941	1.306566	22.854167
0.9	65892	184622	0.460561	1.433279	2.633858	0.033205	0.801208	23.040816	0.033205	1.370072	23.040816

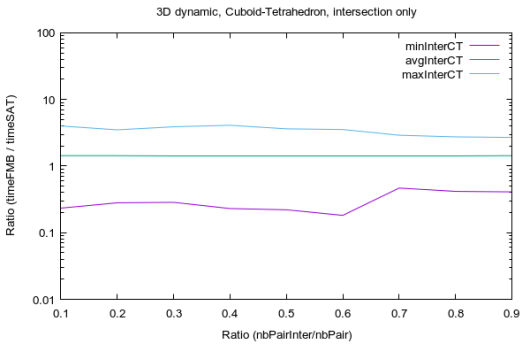
percPairInter	countInterTT	countNoInterTT	miniInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	50430	198082	0.219393	0.573120	1.048798	0.030902	0.915127	27.916667	0.030902	0.880926	27.916667
0.2	50466	198768	0.254104	0.572201	1.017988	0.032508	0.915531	28.694444	0.032508	0.846865	28.694444
0.3	50644	198856	0.155391	0.572095	1.062241	0.030585	0.915399	29.589744	0.030585	0.812408	29.589744
0.4	50994	198774	0.124575	0.572813	1.257481	0.019876	0.919551	28.891892	0.019876	0.780856	28.891892
0.5	50872	199590	0.132956	0.571182	1.205326	0.018583	0.918830	29.000000	0.018583	0.745006	29.000000
0.6	50972	200146	0.190711	0.571752	1.339779	0.026212	0.912830	32.210526	0.026212	0.708183	32.210526
0.7	50860	198858	0.292173	0.572935	0.777547	0.032382	0.914432	28.638889	0.032382	0.675384	28.638889
0.8	51390	199820	0.292089	0.571724	0.947181	0.033281	0.918008	28.694444	0.033281	0.640981	28.694444
0.9	50054	199330	0.294479	0.572387	0.985356	0.032915	0.911198	29.571429	0.032915	0.606268	29.571429

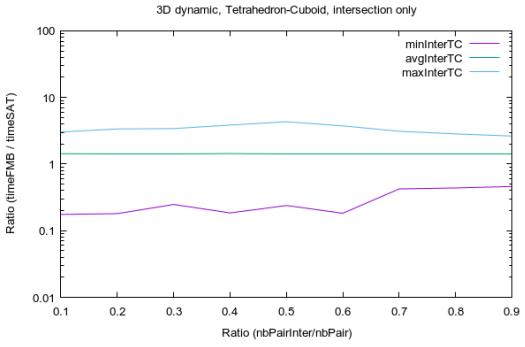

3D dynamic, Total
minTotalTo
avgTotalTo
maxTotalTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

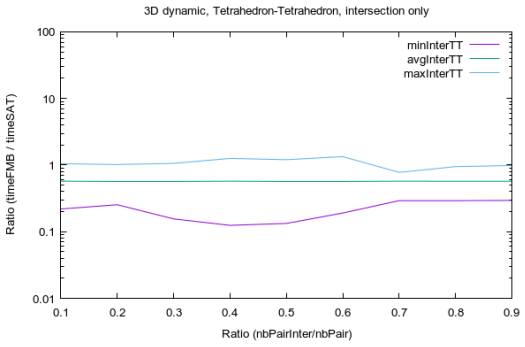

3D dynamic, Total, intersection only
minInterTo
avgInterTo
maxInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

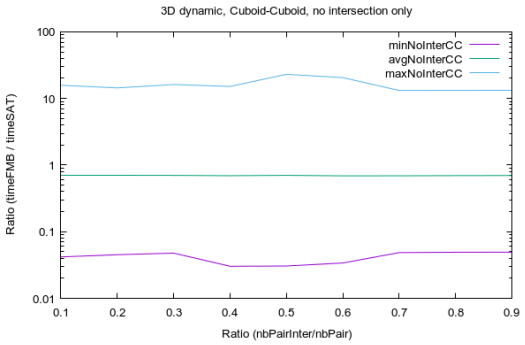

3D dynamic, Total, no intersection only
minNoInterTo
avgNoInterTo
maxNoInterTo
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

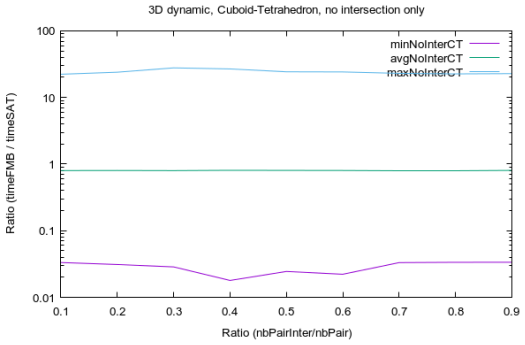

3D dynamic, Cuboid-Cuboid, intersection only
minInterCC
avgInterCC
maxInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)

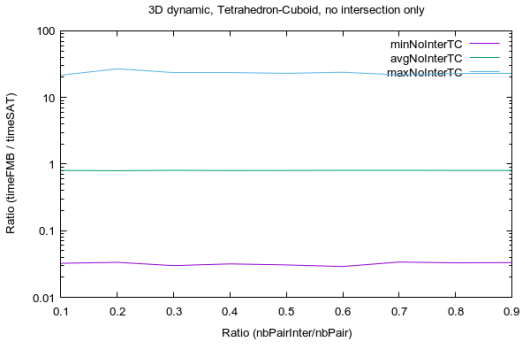

3D dynamic, Cuboid-Tetrahedron, intersection only
minInterCT
avgInterCT
maxInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)

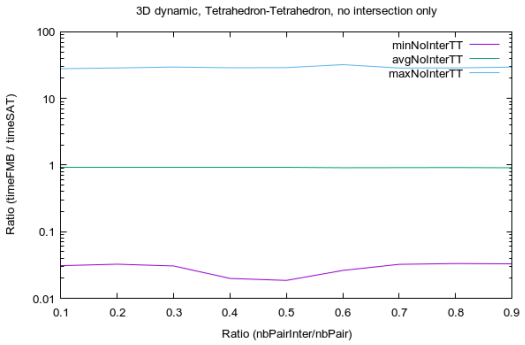

3D dynamic, Tetrahedron-Cuboid, intersection only
minInterTC
avgInterTC
maxInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


3D dynamic, Tetrahedron-Tetrahedron, intersection only
minInterTT
avgInterTT
maxInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)


3D dynamic, Cuboid-Cuboid, no intersection only
minNoInterCC
avgNoInterCC
maxNoInterCC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D dynamic, Cuboid-Tetrahedron, no intersection only
minNoInterCT
avgNoInterCT
maxNoInterCT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D dynamic, Tetrahedron-Cuboid, no intersection only
minNoInterTC
avgNoInterTC
maxNoInterTC
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


3D dynamic, Tetrahedron-Tetrahedron, no intersection only
minNoInterTT
avgNoInterTT
maxNoInterTT
Ratio (timeFMB / timeSAT)
Ratio (nbPairInter/nbPair)
100
10
1
0.1
0.01
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

9 Comments about the qualification results

For the 2D static case:

- FMB is in average 1.2 times slower than SAT to detect intersection between Tetrahedrons, and 1.2 times faster to detect non intersection.

- FMB is in average 1.7 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.2 times faster to detect non intersection.

- FMB is in average 2.3 times slower than SAT to detect intersection between Cuboids, and 1.3 times faster to detect non intersection.

FMB is then in average faster than SAT for a set of Tetrahedron containing less than around 45% of Frames in intersection, and less than around 20% for combinaisons of Tetrahedrons and Cuboids.

For the 3D static case:

- FMB is in average 4.8 times faster than SAT to detect intersection between Tetrahedrons, and 1.7 times faster to detect non intersection.

- FMB is in average 2.6 times faster than SAT to detect intersection between a Tetrahedron and a Cuboid, and 2.3 times faster to detect non intersection.

- FMB is in average 1.5 times faster than SAT to detect intersection between Cuboids, and 3.0 times faster to detect non intersection.

FMB is then in average always faster (from 4.8 times to 1.5 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

For the 2D dynamic case:

- FMB is in average 1.6 times slower than SAT to detect intersection between Tetrahedrons, and 1.2 times slower to detect non intersection.

- FMB is in average 2.0 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.1 times slower to detect non intersection.

- FMB is in average 2.7 times slower than SAT to detect intersection between Cuboids, and 1.1 times slower to detect non intersection.

FMB is then in average always slower (from 2.7 times to 1.1 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

For the 3D dynamic case:

- FMB is in average 1.8 times faster than SAT to detect intersection between Tetrahedrons, and 1.4 times faster to detect non intersection.

- FMB is in average 1.4 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.5 times faster to detect non intersection.

- FMB is in average 2.6 times slower than SAT to detect intersection between Cuboids, and 1.7 times faster to detect non intersection.

FMB is then in average always faster than SAT for a set of Tetrahedron, and faster than SAT for a combinaison of Tetrahedrons and Cuboids containining less than around 35% of intersection.

Overall, FMB is faster than SAT, at least if the percentage of intersecting Frames is low, for all cases but the 2D dynamic one. In practice, for example in applications where the Frames represents real world objects supposedly normally not in intersection, FMB would be a better choice than SAT.

In a real world collision detection system the pair of Frames would be pruned by first applying a rough but fast detection algorithm. The slower but accurate SAT and FMB algorithms would be then only applied to pairs of Frames closed together. To ensure the results stay valid on this subset of possible cases, I've also run the qualification for 2D and 3D static cases in 'near case only' mode. In this mode, only pairs of Frames whose AABBs are intersecting are used. The results are as follow.

For the 2D static near only case:

- FMB is in average 1.2 times slower than SAT to detect intersection between Tetrahedrons, and as fast to detect non intersection.

- FMB is in average 1.6 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.1 times faster to detect non intersection.

- FMB is in average 2.1 times slower than SAT to detect intersection between Cuboids, and 1.2 times faster to detect non intersection.

FMB is then in average 1.2 faster to 2.1 slower than SAT.

For the 3D static near only case:

- FMB is in average 4.4 times faster than SAT to detect intersection between Tetrahedrons, and 1.4 times faster to detect non intersection.

- FMB is in average 2.5 times faster than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.7 times faster to detect non intersection.

- FMB is in average 1.3 times faster than SAT to detect intersection between Cuboids, and 2.3 times faster to detect non intersection.

FMB is then in average always faster (from 4.4 times to 1.3 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

In comparison to the general case, in 'near only mode' the FMB algorithm stays faster than the SAT algorithm for the 3D static case, but is same or slower in the 2D static case.

SAT and FMB follows the same strategy: assume that the pair of Frames is in intersection and try to prove it is false by checking a list of conditions. These conditions are the difference between the two algorithms. The results of the qualification show that in average the conditions used by FMB allows to detect a non intersection faster than those of SAT.

For one given pair in intersection, all the conditions must be checked before the algorithms give their answer. The algorithm with the smallest execution time of all these conditions is then the fastest, and the results shows that this is in general SAT (the exceptions are the 3D static case and 3D dynamic case for Tetrahedrons pairs). This is shown in the results by the low variability of the ratio timeFMB/timeSAT for intersecting pairs.

For one given pair not in intersection, the algorithms reply as soon as one condition is verified. This may be the first one, as it may be the last one depending on the geometry of the pair of Frames. Then, the variability of the ratio timeFMB/timeSAT varies widely as shown in the results, from 50 times faster to 29 times slower, but the resuts shows that in general the advantage goes to the FMB algorithm (the exception is the 2D dynamic case).

In the SAT algorithm, one must perform the projection of all vertices on one axis and then check the result which is the intersection condition. Every axis comes from the geometry of the Frames and one cannot preview which one will be lead to the checked condition for a non intersecting pair. In the FMB algorithm, the conditions depends on the way the system of linear inequation is built. Then, for best performances, it must be done in such a way that inequality (41) is encountered as soon as possible. With the FMB representation, contrary to the SAT one, it is possible to do so independantly of the geometry of the pair of Frames by reordering the inequalities of the system. For example, the $X_i \leq 1.0$ inequalities must be moved down to the end of the system of the linear inequalities for better performance, as they will never lead to '(41) is true' at the first step of the Fourier-Motzkin algorithm.

Looking for other rearrangement of the inequations, I've come to the conclusion that the best possible case (in term of speed) is, when checking Frame A against Frame B, to have:

- B's origin is the nearest vertex of B relative to A's origin
- the projection of B's origin in A's coordinate system is such as components of $\overrightarrow{AB_A}$ are all positive

This the best possible case because it minimised the a_i in (41) in the initial system or during Fourier-Motzkin algorithm, which leads quickly to '(41) is true' if the Frames are not in intersection. The Frame representation is invariant of the vertex choosen as origin, so it's possible to rearrange them to try to fit the conditions above (however it's not always possible to fit both). I've checked that it effectively leads to slightly better performances by first modifying the qualification program to generate only these cases, and then by adding a rearrangement of the origins at the beginning of the FMB algorithm. Unfortunately, the cost of the origin rearrangement is heavier than its benefit. Still, I believe one may find some clever rearrangement which would lead to even better performance for the FMB algorithm.

10 Conclusion

In this paper I've introduced the FMB algorithm which solve efficiently the intersection detection problem of 2D/3D static/dynamic cuboid/tetrahedron by using the Fourier-Motzkin elimination method. All information necessary to implement and use the FMB algorithm, or reproduce the results introduced in this paper are included in this paper, and available on the GitHub repository https://github.com/BayashiPascal/FMB/ .

Validation and qualification against the SAT algorithm prove the correctness of the results from the FMB algorithm and prove it's a valid alternative in term of performance to the SAT algorithm, especially when applied to tetrahedrons and/or in the 3D static case. It is also important to note its simplicity to implement, and the fact that the FMB algorithm returns a bounding box of the intersection, if any, while the SAT algorithm only returns a boolean answer.

Idea on direction to explore with the view to improve the FMB algorithm is given. Steps of the Fourier-Motzkin could also be easily parallelized on an appropriate architecture to improve performance. Tests of implementation with others programming languages, or on other runtime environments, or against other algorithms (such as CJK) would also be interesting to perform. Finally, while the algorithm is introduced here in 2D and 3D, its extension to upper dimensions is straightforward.

11 Annex

11.1 Runtime environment

Results introduce in this paper have been produced by compiling and running the corresponding algorithms in the following environment:

```
uname -v

#40~18.04.1-Ubuntu SMP Thu Nov 14 12:06:39 UTC 2019

=============

lshw -short

H/W path        Device      Class          Description
=====================================================
                            system         VC65-C1
/0                          bus            VC65-C1
/0/0                        memory         64KiB BIOS
/0/2f                       memory         16GiB System Memory
/0/2f/0                     memory         [empty]
/0/2f/1                     memory         16GiB SODIMM DDR4 Synchronous 2400
     MHz (0.4 ns)
/0/39                       memory         384KiB L1 cache
/0/3a                       memory         1536KiB L2 cache
/0/3b                       memory         12MiB L3 cache
/0/3c                       processor      Intel(R) Core(TM) i7-8700T CPU @
    2.40GHz
/0/100                      bridge         8th Gen Core Processor Host Bridge
    /DRAM Registers
/0/100/2                    display        Intel Corporation
/0/100/12                   generic        Cannon Lake PCH Thermal Controller
/0/100/14                   bus            Cannon Lake PCH USB 3.1 xHCI Host
    Controller
/0/100/14/0     usb1        bus            xHCI Host Controller
/0/100/14/0/5               input          ELECOM Wired Keyboard
/0/100/14/0/6               input          PTZ-630
/0/100/14/0/7               generic        USB2.0-CRW
/0/100/14/0/e               communication  Bluetooth wireless interface
/0/100/14/1     usb2        bus            xHCI Host Controller
/0/100/14.2                 memory         RAM memory
/0/100/14.3     wlo1        network        Wireless-AC 9560 [Jefferson Peak]
/0/100/16                   communication  Cannon Lake PCH HECI Controller
```

```
/0/100/17                   storage        Cannon Lake PCH SATA AHCI
    Controller
/0/100/1f                   bridge         Intel Corporation
/0/100/1f.3                 multimedia     Cannon Lake PCH cAVS
/0/100/1f.4                 bus            Cannon Lake PCH SMBus Controller
/0/100/1f.5                 bus            Cannon Lake PCH SPI Controller
/0/100/1f.6    eno2         network        Ethernet Connection (7) I219-V
/0/1           scsi0        storage
/0/1/0.0.0     /dev/sda     disk           128GB HFS128G39TND-N21
/0/1/0.0.0/1                volume         99MiB Windows FAT volume
/0/1/0.0.0/2   /dev/sda2     volume         15MiB reserved partition
/0/1/0.0.0/3   /dev/sda3     volume         83GiB Windows NTFS volume
/0/1/0.0.0/4   /dev/sda4     volume         499MiB Windows NTFS volume
/0/1/0.0.0/5   /dev/sda5     volume         35GiB EXT4 volume
/0/2           scsi2        storage
/0/2/0.0.0     /dev/sdb     disk           500GB ST500LM034-2GH17
/0/2/0.0.0/1   /dev/sdb1    volume         463GiB EXT4 volume
/0/2/0.0.0/2   /dev/sdb2    volume         499MiB Windows FAT volume
/0/3           scsi5        storage
/0/3/0.0.0     /dev/cdrom   disk           BD-RE BU50N
/1                          power          To Be Filled By O.E.M.

=============

lscpu

Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              12
On-line CPU(s) list: 0-11
Thread(s) per core:  2
Core(s) per socket:  6
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               158
Model name:          Intel(R) Core(TM) i7-8700T CPU @ 2.40GHz
Stepping:            10
CPU MHz:             2216.548
CPU max MHz:         4000.0000
CPU min MHz:         800.0000
BogoMIPS:            4800.00
Virtualization:      VT-x
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            12288K
NUMA node0 CPU(s):   0-11
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
    mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
    syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts
    rep_good nopl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni
    pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
     pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes
    xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb
    invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept
     vpid ept_ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid
    rtm mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1
    xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
    md_clear flush_l1d
```

```
=============

gcc -v

Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.4.0-1
    ubuntu1~18.04.1' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs
    --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/
    usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=
    x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir
    =/usr/lib --without-included-gettext --enable-threads=posix --libdir=/
    usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable-
    libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi
    =new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx
    --enable-plugin --enable-default-pie --with-system-zlib --with-target-
    system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --
    with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --
    enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none
    --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu
     --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)
```

11.2 SAT implementation

In this section I introduce the code of the implementation of the SAT algorithm, used to validate and qualify the FMB algorithm.

11.2.1 Header

```
#ifndef __SAT_H_
#define __SAT_H_

#include <stdbool.h>
#include <string.h>
#include "frame.h"

// ------------- Functions declaration -------------

// Test for intersection between 2D Frame that and 2D Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection2D(
  const Frame2D* const that,
  const Frame2D* const tho);

// Test for intersection between moving 2D Frame that and 2D
// Frame tho
// Return true if the two Frames are intersecting, else false
```

```
bool SATTestIntersection2DTime(
  const Frame2DTime* const that,
  const Frame2DTime* const tho);

// Test for intersection between 3D Frame that and 3D Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3D(
  const Frame3D* const that,
  const Frame3D* const tho);

// Test for intersection between moving 3D Frame that and 3D
// Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3DTime(
  const Frame3DTime* const that,
  const Frame3DTime* const tho);

#endif
```

11.2.2 Body

```
#include "sat.h"

// ------------- Macros -------------

#define EPSILON 0.0000001

// ------------- Functions declaration -------------

// Check the intersection constraint along one axis for 3D Frames
bool CheckAxis3D(
  const Frame3D* const that,
  const Frame3D* const tho,
  const double* const axis);

// Check the intersection constraint along one axis for moving 3D Frames
bool CheckAxis3DTime(
  const Frame3DTime* const that,
  const Frame3DTime* const tho,
  const double* const axis,
  const double* const relSpeed);

// ------------- Functions implementation -------------

// Test for intersection between 2D Frame that and 2D Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection2D(
  const Frame2D* const that,
  const Frame2D* const tho) {

  // Declare a variable to loop on Frames and commonalize code
  const Frame2D* frameEdge = that;

  // Loop to commonalize code when checking SAT based on that's edges
  // and then tho's edges
  for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
```

```
FrameType frameEdgeType = frameEdge->type;
const double* frameEdgeCompA = frameEdge->comp[0];
const double* frameEdgeCompB = frameEdge->comp[1];

// Declare a variable to memorize the number of edges, by default 2
int nbEdges = 2;

// Declare a variable to memorize the third edge in case of
// tetrahedron
double thirdEdge[2];

// If the frame is a tetrahedron
if (frameEdgeType == FrameTetrahedron) {

  // Initialise the third edge
  thirdEdge[0] = frameEdgeCompB[0] - frameEdgeCompA[0];
  thirdEdge[1] = frameEdgeCompB[1] - frameEdgeCompA[1];

  // Correct the number of edges
  nbEdges = 3;

}

// Loop on the frame's edges
for (
  int iEdge = nbEdges;
  iEdge--;) {

  // Get the current edge
  const double* edge =
    (iEdge == 2 ? thirdEdge : frameEdge->comp[iEdge]);

  // Declare variables to memorize the boundaries of projection
  // of the two frames on the current edge
  double bdgBoxA[2];
  double bdgBoxB[2];

  // Declare two variables to loop on Frames and commonalize code
  const Frame2D* frame = that;
  double* bdgBox = bdgBoxA;

  // Loop on Frames
  for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    const double* frameOrig = frame->orig;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    FrameType frameType = frame->type;

    // Get the number of vertices of frame
    int nbVertices = (frameType == FrameTetrahedron ? 3 : 4);

    // Declare a variable to memorize if the current vertex is
    // the first in the loop, used to initialize the boundaries
    bool firstVertex = true;

    // Loop on vertices of the frame
    for (
      int iVertex = nbVertices;
```

```
  iVertex--;) {

  // Get the vertex
  double vertex[2];
  vertex[0] = frameOrig[0];
  vertex[1] = frameOrig[1];
  switch (iVertex) {

    case 3:
      vertex[0] += frameCompA[0] + frameCompB[0];
      vertex[1] += frameCompA[1] + frameCompB[1];
      break;
    case 2:
      vertex[0] += frameCompA[0];
      vertex[1] += frameCompA[1];
      break;
    case 1:
      vertex[0] += frameCompB[0];
      vertex[1] += frameCompB[1];
      break;
    default:
      break;

  }

  // Get the projection of the vertex on the normal of the edge
  // Orientation of the normal doesn't matter, so we
  // use arbitrarily the normal (edge[1], -edge[0])
  double proj = vertex[0] * edge[1] - vertex[1] * edge[0];

  // If it's the first vertex
  if (firstVertex == true) {

    // Initialize the boundaries of the projection of the
    // Frame on the edge
    bdgBox[0] = proj;
    bdgBox[1] = proj;

    // Update the flag to memorize we did the first vertex
    firstVertex = false;

  // Else, it's not the first vertex
  } else {

    // Update the boundaries of the projection of the Frame on
    // the edge
    if (bdgBox[0] > proj) {

      bdgBox[0] = proj;

    }

    if (bdgBox[1] < proj) {

      bdgBox[1] = proj;

    }

  }

}
```

```
        // Switch the frame to check the vertices of the second Frame
        frame = tho;
        bdgBox = bdgBoxB;

      }

      // If the projections of the two frames on the edge are
      // not intersecting
      if (
        bdgBoxB[1] < bdgBoxA[0] ||
        bdgBoxA[1] < bdgBoxB[0]) {

        // There exists an axis which separates the Frames,
        // thus they are not in intersection
        return false;

      }

    }

    // Switch the frames to test against the second Frame's edges
    frameEdge = tho;

  }

  // If we reaches here, it means the two Frames are intersecting
  return true;

}

// Test for intersection between moving 2D Frame that and 2D
// Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection2DTime(
  const Frame2DTime* const that,
  const Frame2DTime* const tho) {

  // Declare a variable to loop on Frames and commonalize code
  const Frame2DTime* frameEdge = that;

  // Declare a variable to memorize the speed of tho relative to that
  double relSpeed[2];
  relSpeed[0] = tho->speed[0] - that->speed[0];
  relSpeed[1] = tho->speed[1] - that->speed[1];

  // Loop to commonalize code when checking SAT based on that's edges
  // and then tho's edges
  for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    FrameType frameEdgeType = frameEdge->type;
    const double* frameEdgeCompA = frameEdge->comp[0];
    const double* frameEdgeCompB = frameEdge->comp[1];

    // Declare a variable to memorize the number of edges, by default 2
    int nbEdges = 2;

    // Declare a variable to memorize the third edge in case of
    // tetrahedron
    double thirdEdge[2];
```

```
// If the frame is a tetrahedron
if (frameEdgeType == FrameTetrahedron) {

  // Initialise the third edge
  thirdEdge[0] = frameEdgeCompB[0] - frameEdgeCompA[0];
  thirdEdge[1] = frameEdgeCompB[1] - frameEdgeCompA[1];

  // Correct the number of edges
  nbEdges = 3;

}

// If the current frame is the second frame
if (iFrame == 1) {

  // Add one more edge to take into account the movement
  // of tho relative to that
  ++nbEdges;

}

// Loop on the frame's edges
for (
  int iEdge = nbEdges;
  iEdge--;) {

  // Get the current edge
  const double* edge = 0;

  if (iEdge == 3) {

    edge = relSpeed;

  } else if (iEdge == 2) {

    if (frameEdgeType == FrameTetrahedron) {

      edge = thirdEdge;

    } else {

      edge = relSpeed;

    }

  } else {

    edge = frameEdge->comp[iEdge];

  }

  // Declare variables to memorize the boundaries of projection
  // of the two frames on the current edge
  double bdgBoxA[2];
  double bdgBoxB[2];

  // Declare two variables to loop on Frames and commonalize code
  const Frame2DTime* frame = that;
  double* bdgBox = bdgBoxA;

  // Loop on Frames
```

```
for (
  int iFrame = 2;
  iFrame--;) {

  // Shortcuts
  const double* frameOrig = frame->orig;
  const double* frameCompA = frame->comp[0];
  const double* frameCompB = frame->comp[1];
  FrameType frameType = frame->type;

  // Get the number of vertices of frame
  int nbVertices = (frameType == FrameTetrahedron ? 3 : 4);

  // Declare a variable to memorize if the current vertex is
  // the first in the loop, used to initialize the boundaries
  bool firstVertex = true;

  // Loop on vertices of the frame
  for (
    int iVertex = nbVertices;
    iVertex--;) {

    // Get the vertex
    double vertex[2];
    vertex[0] = frameOrig[0];
    vertex[1] = frameOrig[1];
    switch (iVertex) {

      case 3:
        vertex[0] += frameCompA[0] + frameCompB[0];
        vertex[1] += frameCompA[1] + frameCompB[1];
        break;
      case 2:
        vertex[0] += frameCompA[0];
        vertex[1] += frameCompA[1];
        break;
      case 1:
        vertex[0] += frameCompB[0];
        vertex[1] += frameCompB[1];
        break;
      default:
        break;

    }

    // Get the projection of the vertex on the normal of the edge
    // Orientation of the normal doesn't matter, so we
    // use arbitrarily the normal (edge[1], -edge[0])
    double proj = vertex[0] * edge[1] - vertex[1] * edge[0];

    // If it's the first vertex
    if (firstVertex == true) {

      // Initialize the boundaries of the projection of the
      // Frame on the edge
      bdgBox[0] = proj;
      bdgBox[1] = proj;

      // Update the flag to memorize we did the first vertex
      firstVertex = false;

    // Else, it's not the first vertex
```

```
      } else {

        // Update the boundaries of the projection of the Frame on
        // the edge
        if (bdgBox[0] > proj) {

          bdgBox[0] = proj;

        }

        if (bdgBox[1] < proj) {

          bdgBox[1] = proj;

        }

      }

      // If we are checking the second frame's vertices
      if (frame == tho) {

        // Check also the vertices moved by the relative speed
        vertex[0] += relSpeed[0];
        vertex[1] += relSpeed[1];

        proj = vertex[0] * edge[1] - vertex[1] * edge[0];

        if (bdgBox[0] > proj) {

          bdgBox[0] = proj;

        }

        if (bdgBox[1] < proj) {

          bdgBox[1] = proj;

        }

      }

    }

    // Switch the frame to check the vertices of the second Frame
    frame = tho;
    bdgBox = bdgBoxB;

  }

  // If the projections of the two frames on the edge are
  // not intersecting
  if (
    bdgBoxB[1] < bdgBoxA[0] ||
    bdgBoxA[1] < bdgBoxB[0]) {

    // There exists an axis which separates the Frames,
    // thus they are not in intersection
    return false;

  }

}
```

```
    // Switch the frames to test against the second Frame's edges
    frameEdge = tho;

  }

  // If we reaches here, it means the two Frames are intersecting
  return true;

}

// Test for intersection between 3D Frame that and 3D Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3D(
  const Frame3D* const that,
  const Frame3D* const tho) {

  // Declare two variables to memorize the opposite edges in case
  // of tetrahedron
  double oppEdgesThat[3][3];
  double oppEdgesTho[3][3];

  // Declare two variables to memorize the number of edges, by default 3
  int nbEdgesThat = 3;
  int nbEdgesTho = 3;

  // If the first Frame is a tetrahedron
  if (that->type == FrameTetrahedron) {

    // Shortcuts
    const double* frameCompA = that->comp[0];
    const double* frameCompB = that->comp[1];
    const double* frameCompC = that->comp[2];

    // Initialise the opposite edges
    oppEdgesThat[0][0] = frameCompB[0] - frameCompA[0];
    oppEdgesThat[0][1] = frameCompB[1] - frameCompA[1];
    oppEdgesThat[0][2] = frameCompB[2] - frameCompA[2];

    oppEdgesThat[1][0] = frameCompB[0] - frameCompC[0];
    oppEdgesThat[1][1] = frameCompB[1] - frameCompC[1];
    oppEdgesThat[1][2] = frameCompB[2] - frameCompC[2];

    oppEdgesThat[2][0] = frameCompC[0] - frameCompA[0];
    oppEdgesThat[2][1] = frameCompC[1] - frameCompA[1];
    oppEdgesThat[2][2] = frameCompC[2] - frameCompA[2];

    // Correct the number of edges
    nbEdgesThat = 6;

  }

  // If the second Frame is a tetrahedron
  if (tho->type == FrameTetrahedron) {

    // Shortcuts
    const double* frameCompA = tho->comp[0];
    const double* frameCompB = tho->comp[1];
    const double* frameCompC = tho->comp[2];

    // Initialise the opposite edges
    oppEdgesTho[0][0] = frameCompB[0] - frameCompA[0];
```

```
  oppEdgesTho[0][1] = frameCompB[1] - frameCompA[1];
  oppEdgesTho[0][2] = frameCompB[2] - frameCompA[2];

  oppEdgesTho[1][0] = frameCompB[0] - frameCompC[0];
  oppEdgesTho[1][1] = frameCompB[1] - frameCompC[1];
  oppEdgesTho[1][2] = frameCompB[2] - frameCompC[2];

  oppEdgesTho[2][0] = frameCompC[0] - frameCompA[0];
  oppEdgesTho[2][1] = frameCompC[1] - frameCompA[1];
  oppEdgesTho[2][2] = frameCompC[2] - frameCompA[2];

  // Correct the number of edges
  nbEdgesTho = 6;

}

// Declare variables to loop on Frames and commonalize code
const Frame3D* frame = that;
const double (*oppEdgesA)[3] = oppEdgesThat;

// Loop to commonalize code when checking SAT based on that's edges
// and then tho's edges
for (
  int iFrame = 2;
  iFrame--;) {

  // Shortcuts
  FrameType frameType = frame->type;
  const double* frameCompA = frame->comp[0];
  const double* frameCompB = frame->comp[1];
  const double* frameCompC = frame->comp[2];

  // Declare a variable to memorize the number of faces, by default 3
  int nbFaces = 3;

  // Declare a variable to memorize the normal to faces
  // Arrangement is normFaces[iFace][iAxis]
  double normFaces[4][3];

  // Initialise the normal to faces
  normFaces[0][0] =
    frameCompA[1] * frameCompB[2] -
    frameCompA[2] * frameCompB[1];
  normFaces[0][1] =
    frameCompA[2] * frameCompB[0] -
    frameCompA[0] * frameCompB[2];
  normFaces[0][2] =
    frameCompA[0] * frameCompB[1] -
    frameCompA[1] * frameCompB[0];

  normFaces[1][0] =
    frameCompA[1] * frameCompC[2] -
    frameCompA[2] * frameCompC[1];
  normFaces[1][1] =
    frameCompA[2] * frameCompC[0] -
    frameCompA[0] * frameCompC[2];
  normFaces[1][2] =
    frameCompA[0] * frameCompC[1] -
    frameCompA[1] * frameCompC[0];

  normFaces[2][0] =
    frameCompC[1] * frameCompB[2] -
```

```
      frameCompC[2] * frameCompB[1];
    normFaces[2][1] =
      frameCompC[2] * frameCompB[0] -
      frameCompC[0] * frameCompB[2];
    normFaces[2][2] =
      frameCompC[0] * frameCompB[1] -
      frameCompC[1] * frameCompB[0];

    // If the frame is a tetrahedron
    if (frameType == FrameTetrahedron) {

      // Shortcuts
      const double* oppEdgeA = oppEdgesA[0];
      const double* oppEdgeB = oppEdgesA[1];

      // Initialise the normal to the opposite face
      normFaces[3][0] =
        oppEdgeA[1] * oppEdgeB[2] -
        oppEdgeA[2] * oppEdgeB[1];
      normFaces[3][1] =
        oppEdgeA[2] * oppEdgeB[0] -
        oppEdgeA[0] * oppEdgeB[2];
      normFaces[3][2] =
        oppEdgeA[0] * oppEdgeB[1] -
        oppEdgeA[1] * oppEdgeB[0];

      // Correct the number of faces
      nbFaces = 4;

    }

    // Loop on the frame's faces
    for (
      int iFace = nbFaces;
      iFace--;) {

      // Check against the current face's normal
      bool isIntersection =
        CheckAxis3D(
          that,
          tho,
          normFaces[iFace]);

      // If the axis is separating the Frames
      if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;

      }

    }

    // Switch the frame to test against the second Frame
    frame = tho;
    oppEdgesA = oppEdgesTho;

  }

  // Loop on the pair of edges between the two frames
  for (
```

```
    int iEdgeThat = nbEdgesThat;
    iEdgeThat--;) {

    // Get the first edge
    const double* edgeThat = NULL;
    if (iEdgeThat < 3) {

      edgeThat = that->comp[iEdgeThat];

    } else {

      edgeThat = oppEdgesThat[iEdgeThat - 3];

    }

    for (
      int iEdgeTho = nbEdgesTho;
      iEdgeTho--;) {

      // Get the second edge
      const double* edgeTho = NULL;
      if (iEdgeTho < 3) {

        edgeTho = tho->comp[iEdgeTho];

      } else {

        edgeTho = oppEdgesTho[iEdgeTho - 3];

      }

      // Get the cross product of the two edges
      double axis[3];
      axis[0] = edgeThat[1] * edgeTho[2] - edgeThat[2] * edgeTho[1];
      axis[1] = edgeThat[2] * edgeTho[0] - edgeThat[0] * edgeTho[2];
      axis[2] = edgeThat[0] * edgeTho[1] - edgeThat[1] * edgeTho[0];

      // Check against the cross product of the two edges
      bool isIntersection =
        CheckAxis3D(
          that,
          tho,
          axis);

      // If the axis is separating the Frames
      if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;

      }

    }

  }

  // If we reaches here, it means the two Frames are intersecting
  return true;

}
```

```
// Test for intersection between moving 3D Frame that and 3D
// Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3DTime(
  const Frame3DTime* const that,
  const Frame3DTime* const tho) {

  // Declare two variables to memorize the opposite edges in case
  // of tetrahedron
  double oppEdgesThat[3][3];
  double oppEdgesTho[3][3];

  // Declare a variable to memorize the speed of tho relative to that
  double relSpeed[3];
  relSpeed[0] = tho->speed[0] - that->speed[0];
  relSpeed[1] = tho->speed[1] - that->speed[1];
  relSpeed[2] = tho->speed[2] - that->speed[2];

  // Declare two variables to memorize the number of edges, by default 3
  int nbEdgesThat = 3;
  int nbEdgesTho = 3;

  // If the first Frame is a tetrahedron
  if (that->type == FrameTetrahedron) {

    // Shortcuts
    const double* frameCompA = that->comp[0];
    const double* frameCompB = that->comp[1];
    const double* frameCompC = that->comp[2];

    // Initialise the opposite edges
    oppEdgesThat[0][0] = frameCompB[0] - frameCompA[0];
    oppEdgesThat[0][1] = frameCompB[1] - frameCompA[1];
    oppEdgesThat[0][2] = frameCompB[2] - frameCompA[2];

    oppEdgesThat[1][0] = frameCompB[0] - frameCompC[0];
    oppEdgesThat[1][1] = frameCompB[1] - frameCompC[1];
    oppEdgesThat[1][2] = frameCompB[2] - frameCompC[2];

    oppEdgesThat[2][0] = frameCompC[0] - frameCompA[0];
    oppEdgesThat[2][1] = frameCompC[1] - frameCompA[1];
    oppEdgesThat[2][2] = frameCompC[2] - frameCompA[2];

    // Correct the number of edges
    nbEdgesThat = 6;

  }

  // If the second Frame is a tetrahedron
  if (tho->type == FrameTetrahedron) {

    // Shortcuts
    const double* frameCompA = tho->comp[0];
    const double* frameCompB = tho->comp[1];
    const double* frameCompC = tho->comp[2];

    // Initialise the opposite edges
    oppEdgesTho[0][0] = frameCompB[0] - frameCompA[0];
    oppEdgesTho[0][1] = frameCompB[1] - frameCompA[1];
    oppEdgesTho[0][2] = frameCompB[2] - frameCompA[2];

    oppEdgesTho[1][0] = frameCompB[0] - frameCompC[0];
```

```
  oppEdgesTho[1][1] = frameCompB[1] - frameCompC[1];
  oppEdgesTho[1][2] = frameCompB[2] - frameCompC[2];

  oppEdgesTho[2][0] = frameCompC[0] - frameCompA[0];
  oppEdgesTho[2][1] = frameCompC[1] - frameCompA[1];
  oppEdgesTho[2][2] = frameCompC[2] - frameCompA[2];

  // Correct the number of edges
  nbEdgesTho = 6;

}

// Declare variables to loop on Frames and commonalize code
const Frame3DTime* frame = that;
const double (*oppEdgesA)[3] = oppEdgesThat;

// Loop to commonalize code when checking SAT based on that's edges
// and then tho's edges
for (
  int iFrame = 2;
  iFrame--;) {

  // Shortcuts
  FrameType frameType = frame->type;
  const double* frameCompA = frame->comp[0];
  const double* frameCompB = frame->comp[1];
  const double* frameCompC = frame->comp[2];

  // Declare a variable to memorize the number of faces, by default 3
  int nbFaces = 3;

  // Declare a variable to memorize the normal to faces
  // Arrangement is normFaces[iFace][iAxis]
  double normFaces[10][3];

  // Initialise the normal to faces
  normFaces[0][0] =
    frameCompA[1] * frameCompB[2] -
    frameCompA[2] * frameCompB[1];
  normFaces[0][1] =
    frameCompA[2] * frameCompB[0] -
    frameCompA[0] * frameCompB[2];
  normFaces[0][2] =
    frameCompA[0] * frameCompB[1] -
    frameCompA[1] * frameCompB[0];

  normFaces[1][0] =
    frameCompA[1] * frameCompC[2] -
    frameCompA[2] * frameCompC[1];
  normFaces[1][1] =
    frameCompA[2] * frameCompC[0] -
    frameCompA[0] * frameCompC[2];
  normFaces[1][2] =
    frameCompA[0] * frameCompC[1] -
    frameCompA[1] * frameCompC[0];

  normFaces[2][0] =
    frameCompC[1] * frameCompB[2] -
    frameCompC[2] * frameCompB[1];
  normFaces[2][1] =
    frameCompC[2] * frameCompB[0] -
    frameCompC[0] * frameCompB[2];
```

```
normFaces[2][2] =
  frameCompC[0] * frameCompB[1] -
  frameCompC[1] * frameCompB[0];

// If the frame is a tetrahedron
if (frameType == FrameTetrahedron) {

  // Shortcuts
  const double* oppEdgeA = oppEdgesA[0];
  const double* oppEdgeB = oppEdgesA[1];

  // Initialise the normal to the opposite face
  normFaces[3][0] =
    oppEdgeA[1] * oppEdgeB[2] -
    oppEdgeA[2] * oppEdgeB[1];
  normFaces[3][1] =
    oppEdgeA[2] * oppEdgeB[0] -
    oppEdgeA[0] * oppEdgeB[2];
  normFaces[3][2] =
    oppEdgeA[0] * oppEdgeB[1] -
    oppEdgeA[1] * oppEdgeB[0];

  // Correct the number of faces
  nbFaces = 4;

}

// If we are checking the frame 'tho'
if (frame == tho) {

  // Add the normal to the virtual faces created by the speed
  // of tho relative to that
  normFaces[nbFaces][0] =
    relSpeed[1] * frameCompA[2] -
    relSpeed[2] * frameCompA[1];
  normFaces[nbFaces][1] =
    relSpeed[2] * frameCompA[0] -
    relSpeed[0] * frameCompA[2];
  normFaces[nbFaces][2] =
    relSpeed[0] * frameCompA[1] -
    relSpeed[1] * frameCompA[0];
  if (
    fabs(normFaces[nbFaces][0]) > EPSILON ||
    fabs(normFaces[nbFaces][1]) > EPSILON ||
    fabs(normFaces[nbFaces][2]) > EPSILON) {

    ++nbFaces;

  }

  normFaces[nbFaces][0] =
    relSpeed[1] * frameCompB[2] -
    relSpeed[2] * frameCompB[1];
  normFaces[nbFaces][1] =
    relSpeed[2] * frameCompB[0] -
    relSpeed[0] * frameCompB[2];
  normFaces[nbFaces][2] =
    relSpeed[0] * frameCompB[1] -
    relSpeed[1] * frameCompB[0];
  if (
    fabs(normFaces[nbFaces][0]) > EPSILON ||
    fabs(normFaces[nbFaces][1]) > EPSILON ||
```

```
  fabs(normFaces[nbFaces][2]) > EPSILON) {

  ++nbFaces;

}

normFaces[nbFaces][0] =
  relSpeed[1] * frameCompC[2] -
  relSpeed[2] * frameCompC[1];
normFaces[nbFaces][1] =
  relSpeed[2] * frameCompC[0] -
  relSpeed[0] * frameCompC[2];
normFaces[nbFaces][2] =
  relSpeed[0] * frameCompC[1] -
  relSpeed[1] * frameCompC[0];
if (
  fabs(normFaces[nbFaces][0]) > EPSILON ||
  fabs(normFaces[nbFaces][1]) > EPSILON ||
  fabs(normFaces[nbFaces][2]) > EPSILON) {

  ++nbFaces;

}

if (frameType == FrameTetrahedron) {

  const double* oppEdgeA = oppEdgesA[0];
  const double* oppEdgeB = oppEdgesA[1];
  const double* oppEdgeC = oppEdgesA[2];

  normFaces[nbFaces][0] =
    relSpeed[1] * oppEdgeA[2] -
    relSpeed[2] * oppEdgeA[1];
  normFaces[nbFaces][1] =
    relSpeed[2] * oppEdgeA[0] -
    relSpeed[0] * oppEdgeA[2];
  normFaces[nbFaces][2] =
    relSpeed[0] * oppEdgeA[1] -
    relSpeed[1] * oppEdgeA[0];
  if (
    fabs(normFaces[nbFaces][0]) > EPSILON ||
    fabs(normFaces[nbFaces][1]) > EPSILON ||
    fabs(normFaces[nbFaces][2]) > EPSILON) {

    ++nbFaces;

  }

  normFaces[nbFaces][0] =
    relSpeed[1] * oppEdgeB[2] -
    relSpeed[2] * oppEdgeB[1];
  normFaces[nbFaces][1] =
    relSpeed[2] * oppEdgeB[0] -
    relSpeed[0] * oppEdgeB[2];
  normFaces[nbFaces][2] =
    relSpeed[0] * oppEdgeB[1] -
    relSpeed[1] * oppEdgeB[0];
  if (
    fabs(normFaces[nbFaces][0]) > EPSILON ||
    fabs(normFaces[nbFaces][1]) > EPSILON ||
    fabs(normFaces[nbFaces][2]) > EPSILON) {
```

```
        ++nbFaces;

      }

      normFaces[nbFaces][0] =
        relSpeed[1] * oppEdgeC[2] -
        relSpeed[2] * oppEdgeC[1];
      normFaces[nbFaces][1] =
        relSpeed[2] * oppEdgeC[0] -
        relSpeed[0] * oppEdgeC[2];
      normFaces[nbFaces][2] =
        relSpeed[0] * oppEdgeC[1] -
        relSpeed[1] * oppEdgeC[0];
      if (
        fabs(normFaces[nbFaces][0]) > EPSILON ||
        fabs(normFaces[nbFaces][1]) > EPSILON ||
        fabs(normFaces[nbFaces][2]) > EPSILON) {

        ++nbFaces;

      }

    }

  }

  // Loop on the frame's faces
  for (
    int iFace = nbFaces;
    iFace--;) {

    // Check against the current face's normal
    bool isIntersection =
      CheckAxis3DTime(
        that,
        tho,
        normFaces[iFace],
        relSpeed);

    // If the axis is separating the Frames
    if (isIntersection == false) {

      // The Frames are not in intersection,
      // terminate the test
      return false;

    }

  }

  // Switch the frame to test against the second Frame
  frame = tho;
  oppEdgesA = oppEdgesTho;

}

// Loop on the pair of edges between the two frames
for (
  int iEdgeThat = nbEdgesThat;
  iEdgeThat--;) {

  // Get the first edge
```

```
    const double* edgeThat = NULL;
    if (iEdgeThat < 3) {

      edgeThat = that->comp[iEdgeThat];

    } else {

      edgeThat = oppEdgesThat[iEdgeThat - 3];

    }

    for (
      int iEdgeTho = nbEdgesTho + 1;
      iEdgeTho--;) {

      // Get the second edge
      const double* edgeTho = NULL;
      if (iEdgeTho == nbEdgesTho) {

        edgeTho = relSpeed;

      } else if (iEdgeTho < 3) {

        edgeTho = tho->comp[iEdgeTho];

      } else {

        edgeTho = oppEdgesTho[iEdgeTho - 3];

      }

      // Get the cross product of the two edges
      double axis[3];
      axis[0] = edgeThat[1] * edgeTho[2] - edgeThat[2] * edgeTho[1];
      axis[1] = edgeThat[2] * edgeTho[0] - edgeThat[0] * edgeTho[2];
      axis[2] = edgeThat[0] * edgeTho[1] - edgeThat[1] * edgeTho[0];

      // Check against the cross product of the two edges
      bool isIntersection =
        CheckAxis3DTime(
          that,
          tho,
          axis,
          relSpeed);

      // If the axis is separating the Frames
      if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;

      }

    }

  }

  // If we reaches here, it means the two Frames are intersecting
  return true;

}
```

```
// Check the intersection constraint for Frames that and tho
// relatively to axis
bool CheckAxis3D(
  const Frame3D* const that,
  const Frame3D* const tho,
  const double* const axis) {

  // Declare variables to memorize the boundaries of projection
  // of the two frames on the current edge
  double bdgBoxA[2];
  double bdgBoxB[2];

  // Declare two variables to loop on Frames and commonalize code
  const Frame3D* frame = that;
  double* bdgBox = bdgBoxA;

  // Loop on Frames
  for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    const double* frameOrig = frame->orig;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    const double* frameCompC = frame->comp[2];
    FrameType frameType = frame->type;

    // Get the number of vertices of frame
    int nbVertices = (frameType == FrameTetrahedron ? 4 : 8);

    // Declare a variable to memorize if the current vertex is
    // the first in the loop, used to initialize the boundaries
    bool firstVertex = true;

    // Loop on vertices of the frame
    for (
      int iVertex = nbVertices;
      iVertex--;) {

      // Get the vertex
      double vertex[3];
      vertex[0] = frameOrig[0];
      vertex[1] = frameOrig[1];
      vertex[2] = frameOrig[2];
      switch (iVertex) {

        case 7:
          vertex[0] +=
            frameCompA[0] +  frameCompB[0] + frameCompC[0];
          vertex[1] +=
            frameCompA[1] +  frameCompB[1] + frameCompC[1];
          vertex[2] +=
            frameCompA[2] +  frameCompB[2] + frameCompC[2];
          break;
        case 6:
          vertex[0] += frameCompB[0] + frameCompC[0];
          vertex[1] += frameCompB[1] + frameCompC[1];
          vertex[2] += frameCompB[2] + frameCompC[2];
          break;
        case 5:
```

```
    vertex[0] += frameCompA[0] + frameCompC[0];
    vertex[1] += frameCompA[1] + frameCompC[1];
    vertex[2] += frameCompA[2] + frameCompC[2];
    break;
  case 4:
    vertex[0] += frameCompA[0] + frameCompB[0];
    vertex[1] += frameCompA[1] + frameCompB[1];
    vertex[2] += frameCompA[2] + frameCompB[2];
    break;
  case 3:
    vertex[0] += frameCompC[0];
    vertex[1] += frameCompC[1];
    vertex[2] += frameCompC[2];
    break;
  case 2:
    vertex[0] += frameCompB[0];
    vertex[1] += frameCompB[1];
    vertex[2] += frameCompB[2];
    break;
  case 1:
    vertex[0] += frameCompA[0];
    vertex[1] += frameCompA[1];
    vertex[2] += frameCompA[2];
    break;
  default:
    break;

}

// Get the projection of the vertex on the axis
double proj =
  vertex[0] * axis[0] +
  vertex[1] * axis[1] +
  vertex[2] * axis[2];

// If it's the first vertex
if (firstVertex == true) {

  // Initialize the boundaries of the projection of the
  // Frame on the edge
  bdgBox[0] = proj;
  bdgBox[1] = proj;

  // Update the flag to memorize we did the first vertex
  firstVertex = false;

// Else, it's not the first vertex
} else {

  // Update the boundaries of the projection of the Frame on
  // the edge
  if (bdgBox[0] > proj) {

    bdgBox[0] = proj;

  }

  if (bdgBox[1] < proj) {

    bdgBox[1] = proj;

  }
```

```
      }

    }

    // Switch the frame to check the vertices of the second Frame
    frame = tho;
    bdgBox = bdgBoxB;

  }

  // If the projections of the two frames on the edge are
  // not intersecting
  if (
    bdgBoxB[1] < bdgBoxA[0] ||
    bdgBoxA[1] < bdgBoxB[0]) {

    // There exists an axis which separates the Frames,
    // thus they are not in intersection
    return false;

  }

  // If we reaches here the two Frames are in intersection
  return true;

}

// Check the intersection constraint for Frames that and tho
// relatively to axis
bool CheckAxis3DTime(
  const Frame3DTime* const that,
  const Frame3DTime* const tho,
  const double* const axis,
  const double* const relSpeed) {

  // Declare variables to memorize the boundaries of projection
  // of the two frames on the current edge
  double bdgBoxA[2];
  double bdgBoxB[2];

  // Declare two variables to loop on Frames and commonalize code
  const Frame3DTime* frame = that;
  double* bdgBox = bdgBoxA;

  // Loop on Frames
  for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    const double* frameOrig = frame->orig;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    const double* frameCompC = frame->comp[2];
    FrameType frameType = frame->type;

    // Get the number of vertices of frame
    int nbVertices = (frameType == FrameTetrahedron ? 4 : 8);

    // Declare a variable to memorize if the current vertex is
    // the first in the loop, used to initialize the boundaries
```

```
bool firstVertex = true;

// Loop on vertices of the frame
for (
  int iVertex = nbVertices;
  iVertex--;) {

  // Get the vertex
  double vertex[3];
  vertex[0] = frameOrig[0];
  vertex[1] = frameOrig[1];
  vertex[2] = frameOrig[2];
  switch (iVertex) {

    case 7:
      vertex[0] +=
        frameCompA[0] +  frameCompB[0] + frameCompC[0];
      vertex[1] +=
        frameCompA[1] +  frameCompB[1] + frameCompC[1];
      vertex[2] +=
        frameCompA[2] +  frameCompB[2] + frameCompC[2];
      break;
    case 6:
      vertex[0] += frameCompB[0] + frameCompC[0];
      vertex[1] += frameCompB[1] + frameCompC[1];
      vertex[2] += frameCompB[2] + frameCompC[2];
      break;
    case 5:
      vertex[0] += frameCompA[0] + frameCompC[0];
      vertex[1] += frameCompA[1] + frameCompC[1];
      vertex[2] += frameCompA[2] + frameCompC[2];
      break;
    case 4:
      vertex[0] += frameCompA[0] + frameCompB[0];
      vertex[1] += frameCompA[1] + frameCompB[1];
      vertex[2] += frameCompA[2] + frameCompB[2];
      break;
    case 3:
      vertex[0] += frameCompC[0];
      vertex[1] += frameCompC[1];
      vertex[2] += frameCompC[2];
      break;
    case 2:
      vertex[0] += frameCompB[0];
      vertex[1] += frameCompB[1];
      vertex[2] += frameCompB[2];
      break;
    case 1:
      vertex[0] += frameCompA[0];
      vertex[1] += frameCompA[1];
      vertex[2] += frameCompA[2];
      break;
    default:
      break;

  }

  // Get the projection of the vertex on the axis
  double proj =
    vertex[0] * axis[0] +
    vertex[1] * axis[1] +
    vertex[2] * axis[2];
```

```
  // If it's the first vertex
  if (firstVertex == true) {

    // Initialize the boundaries of the projection of the
    // Frame on the edge
    bdgBox[0] = proj;
    bdgBox[1] = proj;

    // Update the flag to memorize we did the first vertex
    firstVertex = false;

  // Else, it's not the first vertex
  } else {

    // Update the boundaries of the projection of the Frame on
    // the edge
    if (bdgBox[0] > proj) {

      bdgBox[0] = proj;

    }

    if (bdgBox[1] < proj) {

      bdgBox[1] = proj;

    }

  }

  // If we are checking the second frame's vertices
  if (frame == tho) {

    // Check also the vertices moved by the relative speed
    vertex[0] += relSpeed[0];
    vertex[1] += relSpeed[1];
    vertex[2] += relSpeed[2];

    proj =
      vertex[0] * axis[0] +
      vertex[1] * axis[1] +
      vertex[2] * axis[2];

    if (bdgBox[0] > proj) {

      bdgBox[0] = proj;

    }

    if (bdgBox[1] < proj) {

      bdgBox[1] = proj;

    }

  }

}

// Switch the frame to check the vertices of the second Frame
frame = tho;
```

```
    bdgBox = bdgBoxB;

  }

  // If the projections of the two frames on the edge are
  // not intersecting
  if (
    bdgBoxB[1] < bdgBoxA[0] ||
    bdgBoxA[1] < bdgBoxB[0]) {

    // There exists an axis which separates the Frames,
    // thus they are not in intersection
    return false;

  }

  // If we reaches here the two Frames are in intersection
  return true;

}
```

11.3 Makefile

In this section I introduce the Makefile used to compile the code given in the previous sections. It also includes command used to run the unit tests, validation and qualification, and to generate the documentation.

```
COMPILER=gcc
OPTIMIZATION=-O3

all : compile run plot dynamicAnalysis doc

install :
	sudo apt-get install gnuplot

compile : main unitTests validation qualification

main : main2D main2DTime main3D main3DTime

main2D:
	cd 2D; make main OPTIMIZATION=$(OPTIMIZATION); cd -

main2DTime:
	cd 2DTime; make main OPTIMIZATION=$(OPTIMIZATION); cd -

main3D:
	cd 3D; make main OPTIMIZATION=$(OPTIMIZATION); cd -

main3DTime:
	cd 3DTime; make main OPTIMIZATION=$(OPTIMIZATION); cd -

unitTests : unitTests2D unitTests2DTime unitTests3D unitTests3DTime

unitTests2D:
	cd 2D; make unitTests OPTIMIZATION=$(OPTIMIZATION); cd -

unitTests2DTime:
	cd 2DTime; make unitTests OPTIMIZATION=$(OPTIMIZATION); cd -
```

```
unitTests3D:
	cd 3D; make unitTests OPTIMIZATION=$(OPTIMIZATION); cd -

unitTests3DTime:
	cd 3DTime; make unitTests OPTIMIZATION=$(OPTIMIZATION); cd -

validation : validation2D validation2DTime validation3D validation3DTime

validation2D:
	cd 2D; make validation OPTIMIZATION=$(OPTIMIZATION); cd -

validation2DTime:
	cd 2DTime; make validation OPTIMIZATION=$(OPTIMIZATION); cd -

validation3D:
	cd 3D; make validation OPTIMIZATION=$(OPTIMIZATION); cd -

validation3DTime:
	cd 3DTime; make validation OPTIMIZATION=$(OPTIMIZATION); cd -

qualification : qualification2D qualification2DTime qualification3D
    qualification3DTime

qualification2D:
	cd 2D; make qualification OPTIMIZATION=$(OPTIMIZATION); cd -

qualification2DTime:
	cd 2DTime; make qualification OPTIMIZATION=$(OPTIMIZATION); cd -

qualification3D:
	cd 3D; make qualification OPTIMIZATION=$(OPTIMIZATION); cd -

qualification3DTime:
	cd 3DTime; make qualification OPTIMIZATION=$(OPTIMIZATION); cd -

clean : clean2D clean2DTime clean3D clean3DTime

clean2D:
	cd 2D; make clean; cd -

clean2DTime:
	cd 2DTime; make clean; cd -

clean3D:
	cd 3D; make clean; cd -

clean3DTime:
	cd 3DTime; make clean; cd -

valgrind : valgrind2D valgrind2DTime valgrind3D valgrind3DTime

valgrind2D:
	cd 2D; make valgrind; cd -

valgrind2DTime:
	cd 2DTime; make valgrind; cd -

valgrind3D:
	cd 3D; make valgrind; cd -

valgrind3DTime:
	cd 3DTime; make valgrind; cd -
```

```
cppcheck : cppcheck2D cppcheck2DTime cppcheck3D cppcheck3DTime

cppcheck2D:
	cd 2D; make cppcheck; cd -

cppcheck2DTime:
	cd 2DTime; make cppcheck; cd -

cppcheck3D:
	cd 3D; make cppcheck; cd -

cppcheck3DTime:
	cd 3DTime; make cppcheck; cd -

run : run2D run2DNearCaseOnly run2DTime run3D run3DNearCaseOnly run3DTime

run2D:
	cd 2D; ./main > ../Results/main2D.txt; ./unitTests > ../Results/
	    unitTests2D.txt; ./validation > ../Results/validation2D.txt; ./
	    qualification; cd -

run3D:
	cd 3D; ./main > ../Results/main3D.txt; ./unitTests > ../Results/
	    unitTests3D.txt; ./validation > ../Results/validation3D.txt; ./
	    qualification; cd -

run2DNearCaseOnly:
	cd 2D; ./qualification -nearCaseOnly; cd -

run3DNearCaseOnly:
	cd 3D; ./qualification -nearCaseOnly; cd -

run2DTime:
	cd 2DTime; ./main > ../Results/main2DTime.txt; ./unitTests > ../
	    Results/unitTests2DTime.txt; ./validation > ../Results/
	    validation2DTime.txt; ./qualification; cd -

run3DTime:
	cd 3DTime; ./main > ../Results/main3DTime.txt; ./unitTests > ../
	    Results/unitTests3DTime.txt; ./validation > ../Results/
	    validation3DTime.txt; ./qualification; cd -

plot: cleanPlot plot2D plot2DNearCaseOnly plot2DTime plot3D
    plot3DNearCaseOnly plot3DTime

cleanPlot:
	rm -f Results/*.png

plot2D:
	cd Results; gnuplot qualification2D.gnu; cd -

plot2DNearCaseOnly:
	cd Results; gnuplot qualification2DNearCaseOnly.gnu; cd -

plot2DTime:
	cd Results; gnuplot qualification2DTime.gnu; cd -

plot3D:
	cd Results; gnuplot qualification3D.gnu; cd -

plot3DNearCaseOnly:
```

```
	cd Results; gnuplot qualification3DNearCaseOnly.gnu; cd -

plot3DTime:
	cd Results; gnuplot qualification3DTime.gnu; cd -

doc:
	cd Doc; make latex; cd -

getRuntimeEnvironment:
	echo "uname -v\n" > runtimeEnv.txt; uname -v >> runtimeEnv.txt; echo
	     "\n=============\n" >> runtimeEnv.txt; echo "lshw -short\n" >>
	    runtimeEnv.txt; sudo lshw -short >> runtimeEnv.txt; echo "\n
	    =============\n" >> runtimeEnv.txt; echo "lscpu\n" >> runtimeEnv
	    .txt; lscpu >> runtimeEnv.txt; echo "\n=============\n" >>
	    runtimeEnv.txt; echo "$(COMPILER) -v\n" >> runtimeEnv.txt; $(
	    COMPILER) -v 1>> runtimeEnv.txt 2>> runtimeEnv.txt

dynamicAnalysis:
	make valgrind 1> dynamicAnalysis.txt 2> dynamicAnalysis.txt

staticAnalysis:
	make cppcheck 1> staticAnalysis.txt 2> staticAnalysis.txt
```

11.3.1 2D static

```
all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=$(OPTIMIZATION) -I../SAT -I../Frame

main : main.o fmb2d.o frame.o Makefile
	$(COMPILER) -o main main.o fmb2d.o frame.o

main.o : main.c fmb2d.h ../Frame/frame.h Makefile
	$(COMPILER) -c main.c $(BUILD_ARG)

unitTests : unitTests.o fmb2d.o frame.o Makefile
	$(COMPILER) -o unitTests unitTests.o fmb2d.o frame.o $(LINK_ARG)

unitTests.o : unitTests.c fmb2d.h ../Frame/frame.h Makefile
	$(COMPILER) -c unitTests.c $(BUILD_ARG)

validation : validation.o fmb2d.o sat.o frame.o Makefile
	$(COMPILER) -o validation validation.o fmb2d.o sat.o frame.o

validation.o : validation.c fmb2d.h ../SAT/sat.h ../Frame/frame.h Makefile
	$(COMPILER) -c validation.c $(BUILD_ARG)

qualification : qualification.o fmb2d.o sat.o frame.o Makefile
	$(COMPILER) -o qualification qualification.o fmb2d.o sat.o frame.o $
	    (LINK_ARG)

qualification.o : qualification.c fmb2d.h ../SAT/sat.h ../Frame/frame.h
    Makefile
	$(COMPILER) -c qualification.c $(BUILD_ARG)

fmb2d.o : fmb2d.c fmb2d.h ../Frame/frame.h Makefile
	$(COMPILER) -c fmb2d.c $(BUILD_ARG)
```

```
sat.o : ../SAT/sat.c ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c ../SAT/sat.c $(BUILD_ARG)

frame.o : ../Frame/frame.c ../Frame/frame.h Makefile
        $(COMPILER) -c ../Frame/frame.c $(BUILD_ARG)

clean :
        rm -f *.o main unitTests validation qualification

valgrind :
        valgrind -v --track-origins=yes --leak-check=full \
        --gen-suppressions=yes --show-leak-kinds=all ./unitTests

cppcheck :
        cppcheck --enable=all ./
```

11.3.2 3D static

```
all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=$(OPTIMIZATION) -I../SAT -I../Frame

main : main.o fmb3d.o frame.o Makefile
        $(COMPILER) -o main main.o fmb3d.o frame.o

main.o : main.c fmb3d.h ../Frame/frame.h Makefile
        $(COMPILER) -c main.c $(BUILD_ARG)

unitTests : unitTests.o fmb3d.o frame.o Makefile
        $(COMPILER) -o unitTests unitTests.o fmb3d.o frame.o $(LINK_ARG)

unitTests.o : unitTests.c fmb3d.h ../Frame/frame.h Makefile
        $(COMPILER) -c unitTests.c $(BUILD_ARG)

validation : validation.o fmb3d.o sat.o frame.o Makefile
        $(COMPILER) -o validation validation.o fmb3d.o sat.o frame.o

validation.o : validation.c fmb3d.h ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c validation.c $(BUILD_ARG)

qualification : qualification.o fmb3d.o sat.o frame.o Makefile
        $(COMPILER) -o qualification qualification.o fmb3d.o sat.o frame.o $
            (LINK_ARG)

qualification.o : qualification.c fmb3d.h ../SAT/sat.h ../Frame/frame.h
    Makefile
        $(COMPILER) -c qualification.c $(BUILD_ARG)

fmb3d.o : fmb3d.c fmb3d.h ../Frame/frame.h Makefile
        $(COMPILER) -c fmb3d.c $(BUILD_ARG)

sat.o : ../SAT/sat.c ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c ../SAT/sat.c $(BUILD_ARG)

frame.o : ../Frame/frame.c ../Frame/frame.h Makefile
        $(COMPILER) -c ../Frame/frame.c $(BUILD_ARG)

clean :
```

```
        rm -f *.o main unitTests validation qualification

valgrind :
        valgrind -v --track-origins=yes --leak-check=full \
        --gen-suppressions=yes --show-leak-kinds=all ./unitTests

cppcheck :
        cppcheck --enable=all ./
```

11.3.3 2D dynamic

```
all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=$(OPTIMIZATION) -I../SAT -I../Frame

main : main.o fmb2dt.o frame.o Makefile
        $(COMPILER) -o main main.o fmb2dt.o frame.o

main.o : main.c fmb2dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c main.c $(BUILD_ARG)

unitTests : unitTests.o fmb2dt.o frame.o Makefile
        $(COMPILER) -o unitTests unitTests.o fmb2dt.o frame.o $(LINK_ARG)

unitTests.o : unitTests.c fmb2dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c unitTests.c $(BUILD_ARG)

validation : validation.o fmb2dt.o sat.o frame.o Makefile
        $(COMPILER) -o validation validation.o fmb2dt.o sat.o frame.o

validation.o : validation.c fmb2dt.h ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c validation.c $(BUILD_ARG)

qualification : qualification.o fmb2dt.o sat.o frame.o Makefile
        $(COMPILER) -o qualification qualification.o fmb2dt.o sat.o frame.o
            $(LINK_ARG)

qualification.o : qualification.c fmb2dt.h ../SAT/sat.h ../Frame/frame.h
    Makefile
        $(COMPILER) -c qualification.c $(BUILD_ARG)

fmb2dt.o : fmb2dt.c fmb2dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c fmb2dt.c $(BUILD_ARG)

sat.o : ../SAT/sat.c ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c ../SAT/sat.c $(BUILD_ARG)

frame.o : ../Frame/frame.c ../Frame/frame.h Makefile
        $(COMPILER) -c ../Frame/frame.c $(BUILD_ARG)

clean :
        rm -f *.o main unitTests validation qualification

valgrind :
        valgrind -v --track-origins=yes --leak-check=full \
        --gen-suppressions=yes --show-leak-kinds=all ./unitTests

cppcheck :
        cppcheck --enable=all ./
```

11.3.4 3D dynamic

```
all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=$(OPTIMIZATION) -I../SAT -I../Frame

main : main.o fmb3dt.o frame.o Makefile
        $(COMPILER) -o main main.o fmb3dt.o frame.o

main.o : main.c fmb3dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c main.c $(BUILD_ARG)

unitTests : unitTests.o fmb3dt.o frame.o Makefile
        $(COMPILER) -o unitTests unitTests.o fmb3dt.o frame.o $(LINK_ARG)

unitTests.o : unitTests.c fmb3dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c unitTests.c $(BUILD_ARG)

validation : validation.o fmb3dt.o sat.o frame.o Makefile
        $(COMPILER) -o validation validation.o fmb3dt.o sat.o frame.o

validation.o : validation.c fmb3dt.h ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c validation.c $(BUILD_ARG)

qualification : qualification.o fmb3dt.o sat.o frame.o Makefile
        $(COMPILER) -o qualification qualification.o fmb3dt.o sat.o frame.o
            $(LINK_ARG)

qualification.o : qualification.c fmb3dt.h ../SAT/sat.h ../Frame/frame.h
    Makefile
        $(COMPILER) -c qualification.c $(BUILD_ARG)

fmb3dt.o : fmb3dt.c fmb3dt.h ../Frame/frame.h Makefile
        $(COMPILER) -c fmb3dt.c $(BUILD_ARG)

sat.o : ../SAT/sat.c ../SAT/sat.h ../Frame/frame.h Makefile
        $(COMPILER) -c ../SAT/sat.c $(BUILD_ARG)

frame.o : ../Frame/frame.c ../Frame/frame.h Makefile
        $(COMPILER) -c ../Frame/frame.c $(BUILD_ARG)

clean :
        rm -f *.o main unitTests validation qualification

valgrind :
        valgrind -v --track-origins=yes --leak-check=full \
        --gen-suppressions=yes --show-leak-kinds=all ./unitTests

cppcheck :
        cppcheck --enable=all ./
```

11.3.5 Doc

```
latex:
        pdflatex -synctex=1 -interaction=nonstopmode -shell-escape  fmb.tex
```

11.4 Dynamic analysis

```
make[1]: Entering directory '/home/bayashi/GitHub/FMB'
cd 2D; make valgrind; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2D'
valgrind -v --track-origins=yes --leak-check=full \
--gen-suppressions=yes --show-leak-kinds=all ./unitTests
==3946== Memcheck, a memory error detector
==3946== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3946== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3946== Command: ./unitTests
==3946==
--3946-- Valgrind options:
--3946--    -v
--3946--    --track-origins=yes
--3946--    --leak-check=full
--3946--    --gen-suppressions=yes
--3946--    --show-leak-kinds=all
--3946-- Contents of /proc/version:
--3946--   Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
    version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
    Mar 31 04:17:56 UTC 2020
--3946--
--3946-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
    avx-avx2-bmi
--3946-- Page sizes: currently 4096, max supported 4096
--3946-- Valgrind library directory: /usr/lib/valgrind
--3946-- Reading syms from /home/bayashi/GitHub/FMB/2D/unitTests
--3946-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--3946--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--3946--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--3946--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--3946--   .. CRC is valid
--3946-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--3946--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--3946--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--3946--    object doesn't have a symbol table
--3946--    object doesn't have a dynamic symbol table
--3946-- Scheduler: using generic scheduler lock implementation.
--3946-- Reading suppressions file: /usr/lib/valgrind/default.supp
==3946== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-3946-
    by-bayashi-on-???
==3946== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-3946-
    by-bayashi-on-???
==3946== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
    -3946-by-bayashi-on-???
==3946==
==3946== TO CONTROL THIS PROCESS USING vgdb (which you probably
==3946== don't want to do, unless you know exactly what you're doing,
==3946== or are doing some strange experiment):
==3946==   /usr/lib/valgrind/../../bin/vgdb --pid=3946 ...command...
==3946==
==3946== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3946==   /path/to/gdb ./unitTests
==3946== and then give GDB the following command
==3946==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=3946
==3946== --pid is optional if only one valgrind process is running
==3946==
--3946-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
    x580608c1 (???)
--3946-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
    x580608db (???)
--3946-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--3946--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
```

```
--3946--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--3946--    object doesn't have a symbol table
--3946-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
    so
--3946--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
    ..
--3946--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--3946--    object doesn't have a symbol table
==3946== WARNING: new redirection conflicts with existing -- ignoring it
--3946--     old: 0x0401f2f0 (strlen              ) R-> (0000.0) 0x580608c1
    ???
--3946--     new: 0x0401f2f0 (strlen              ) R-> (2007.0) 0x04c32db0
    strlen
--3946-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
    x4c33ee0 (strcmp)
--3946-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
    x4c374f0 (mempcpy)
--3946-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--3946--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--3946--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--3946--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--3946--   .. CRC is valid
--3946-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
    x4a2a6e0 (_vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
```

```
--3946-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edaef0 (libc.so.6:stpcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
    (_vgnU_ifunc_wrapper)
--3946-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
     (rindex)
--3946-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
    )
--3946-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
    x4c37020 (strchrnul)
--3946-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
    redirected to 0x4c37130 (mempcpy)
--3946-- REDIR: 0x4fca590 (libc.so.6:__strlen_avx2) redirected to 0x4c32cf0
    (strlen)
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)
```

```
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)

Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)

Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
against
Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)
against
Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
against
```

```
Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
against
Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
against
Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
Succeed
minXY(1.500000,0.000000)-maxXY(1.666667,1.000000)

Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
Succeed
minXY(1.500000,0.500000)-maxXY(2.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
```

```
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
against
```

```
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
 Succeed (no inter)

To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
 Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
 Succeed (no inter)

To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
 Succeed (no inter)

All unit tests 2D have succeed.
--3946-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==3946==
==3946== HEAP SUMMARY:
==3946==     in use at exit: 0 bytes in 0 blocks
==3946==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==3946==
==3946== All heap blocks were freed -- no leaks are possible
==3946==
==3946== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==3946== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2D'
/home/bayashi/GitHub/FMB
cd 2DTime; make valgrind; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2DTime'
valgrind -v --track-origins=yes --leak-check=full \
--gen-suppressions=yes --show-leak-kinds=all ./unitTests
==3949== Memcheck, a memory error detector
==3949== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3949== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3949== Command: ./unitTests
==3949==
--3949-- Valgrind options:
--3949--    -v
--3949--    --track-origins=yes
--3949--    --leak-check=full
```

```
--3949--    --gen-suppressions=yes
--3949--    --show-leak-kinds=all
--3949-- Contents of /proc/version:
--3949--   Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
    version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
    Mar 31 04:17:56 UTC 2020
--3949--
--3949-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
    avx-avx2-bmi
--3949-- Page sizes: currently 4096, max supported 4096
--3949-- Valgrind library directory: /usr/lib/valgrind
--3949-- Reading syms from /home/bayashi/GitHub/FMB/2DTime/unitTests
--3949-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--3949--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--3949--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--3949--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--3949--   .. CRC is valid
--3949-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--3949--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--3949--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--3949--    object doesn't have a symbol table
--3949--    object doesn't have a dynamic symbol table
--3949-- Scheduler: using generic scheduler lock implementation.
--3949-- Reading suppressions file: /usr/lib/valgrind/default.supp
==3949== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-3949-
    by-bayashi-on-???
==3949== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-3949-
    by-bayashi-on-???
==3949== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
    -3949-by-bayashi-on-???
==3949==
==3949== TO CONTROL THIS PROCESS USING vgdb (which you probably
==3949== don't want to do, unless you know exactly what you're doing,
==3949== or are doing some strange experiment):
==3949==   /usr/lib/valgrind/../../bin/vgdb --pid=3949 ...command...
==3949==
==3949== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3949==   /path/to/gdb ./unitTests
==3949== and then give GDB the following command
==3949==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=3949
==3949== --pid is optional if only one valgrind process is running
==3949==
--3949-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
    x580608c1 (???)
--3949-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
    x580608db (???)
--3949-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--3949--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--3949--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--3949--    object doesn't have a symbol table
--3949-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
    so
--3949--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
    ..
--3949--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--3949--    object doesn't have a symbol table
==3949== WARNING: new redirection conflicts with existing -- ignoring it
--3949--     old: 0x0401f2f0 (strlen              ) R-> (0000.0) 0x580608c1
    ???
--3949--     new: 0x0401f2f0 (strlen              ) R-> (2007.0) 0x04c32db0
    strlen
--3949-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
```

```
    x4c33ee0 (strcmp)
--3949-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
    x4c374f0 (mempcpy)
--3949-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--3949--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--3949--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--3949--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--3949--   .. CRC is valid
--3949-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
    x4a2a6e0 (_vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edaef0 (libc.so.6:stpcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
```

```
--3949-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
    (_vgnU_ifunc_wrapper)
--3949-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
     (rindex)
--3949-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
    )
--3949-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
    x4c37020 (strchrnul)
--3949-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
    redirected to 0x4c37130 (mempcpy)
--3949-- REDIR: 0x4fca590 (libc.so.6:__strlen_avx2) redirected to 0x4c32cf0
    (strlen)
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(2.000000,1.000000,1.000000)

Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(1.000000,1.000000,1.000000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
```

```
    (0.000000,0.500000)
Succeed
minXYT(-1.500000,0.000000,0.125000)-maxXYT(2.500000,1.000000,0.500000)

Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(-0.500000,0.000000,0.125000)-maxXYT(1.500000,1.000000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.000000,2.500000,0.500000)

Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.000000,1.500000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
against
Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.400000,2.500000,0.500000)

Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
    (0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
    (0.000000,1.000000)
Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.400000,1.500000,0.500000)

All unit tests 2DTime have succeed.
--3949-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==3949==
==3949== HEAP SUMMARY:
==3949==     in use at exit: 0 bytes in 0 blocks
==3949==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==3949==
==3949== All heap blocks were freed -- no leaks are possible
==3949==
==3949== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==3949== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2DTime'
/home/bayashi/GitHub/FMB
cd 3D; make valgrind; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3D'
valgrind -v --track-origins=yes --leak-check=full \
--gen-suppressions=yes --show-leak-kinds=all ./unitTests
==3952== Memcheck, a memory error detector
```

```
==3952== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3952== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3952== Command: ./unitTests
==3952==
--3952-- Valgrind options:
--3952--    -v
--3952--    --track-origins=yes
--3952--    --leak-check=full
--3952--    --gen-suppressions=yes
--3952--    --show-leak-kinds=all
--3952-- Contents of /proc/version:
--3952--   Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
    version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
    Mar 31 04:17:56 UTC 2020
--3952--
--3952-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
    avx-avx2-bmi
--3952-- Page sizes: currently 4096, max supported 4096
--3952-- Valgrind library directory: /usr/lib/valgrind
--3952-- Reading syms from /home/bayashi/GitHub/FMB/3D/unitTests
--3952-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--3952--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--3952--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--3952--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--3952--   .. CRC is valid
--3952-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--3952--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--3952--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--3952--    object doesn't have a symbol table
--3952--    object doesn't have a dynamic symbol table
--3952-- Scheduler: using generic scheduler lock implementation.
--3952-- Reading suppressions file: /usr/lib/valgrind/default.supp
==3952== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-3952-
    by-bayashi-on-???
==3952== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-3952-
    by-bayashi-on-???
==3952== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
    -3952-by-bayashi-on-???
==3952==
==3952== TO CONTROL THIS PROCESS USING vgdb (which you probably
==3952== don't want to do, unless you know exactly what you're doing,
==3952== or are doing some strange experiment):
==3952==   /usr/lib/valgrind/../../bin/vgdb --pid=3952 ...command...
==3952==
==3952== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3952==   /path/to/gdb ./unitTests
==3952== and then give GDB the following command
==3952==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=3952
==3952== --pid is optional if only one valgrind process is running
==3952==
--3952-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
    x580608c1 (???)
--3952-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
    x580608db (???)
--3952-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--3952--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--3952--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--3952--    object doesn't have a symbol table
--3952-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
    so
--3952--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
    ..
```

```
--3952--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--3952--    object doesn't have a symbol table
==3952== WARNING: new redirection conflicts with existing -- ignoring it
--3952--     old: 0x0401f2f0 (strlen              ) R-> (0000.0) 0x580608c1
    ???
--3952--     new: 0x0401f2f0 (strlen              ) R-> (2007.0) 0x04c32db0
    strlen
--3952-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
    x4c33ee0 (strcmp)
--3952-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
    x4c374f0 (mempcpy)
--3952-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--3952--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--3952--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--3952--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--3952--   .. CRC is valid
--3952-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
    x4a2a6e0 (_vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
```

```
--3952-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edaef0 (libc.so.6:stpcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
    (_vgnU_ifunc_wrapper)
--3952-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
     (rindex)
--3952-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
    )
--3952-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
    x4c37020 (strchrnul)
--3952-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
    redirected to 0x4c37130 (mempcpy)
--3952-- REDIR: 0x4fca590 (libc.so.6:__strlen_avx2) redirected to 0x4c32cf0
    (strlen)
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)
```

```
Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
    (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
```

```
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,0.750000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
```

```
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
 Succeed (no inter)

To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
    (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
    (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

All unit tests 3D have succeed.
--3952-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==3952==
==3952== HEAP SUMMARY:
==3952==     in use at exit: 0 bytes in 0 blocks
==3952==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==3952==
==3952== All heap blocks were freed -- no leaks are possible
==3952==
==3952== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==3952== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3D'
/home/bayashi/GitHub/FMB
cd 3DTime; make valgrind; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3DTime'
valgrind -v --track-origins=yes --leak-check=full \
--gen-suppressions=yes --show-leak-kinds=all ./unitTests
==3955== Memcheck, a memory error detector
==3955== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3955== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3955== Command: ./unitTests
==3955==
--3955-- Valgrind options:
--3955--    -v
--3955--    --track-origins=yes
--3955--    --leak-check=full
--3955--    --gen-suppressions=yes
--3955--    --show-leak-kinds=all
--3955-- Contents of /proc/version:
--3955--   Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
    version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
    Mar 31 04:17:56 UTC 2020
--3955--
--3955-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
    avx-avx2-bmi
--3955-- Page sizes: currently 4096, max supported 4096
--3955-- Valgrind library directory: /usr/lib/valgrind
--3955-- Reading syms from /home/bayashi/GitHub/FMB/3DTime/unitTests
--3955-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--3955--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--3955--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--3955--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
```

```
--3955--   .. CRC is valid
--3955-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--3955--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--3955--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--3955--    object doesn't have a symbol table
--3955--    object doesn't have a dynamic symbol table
--3955-- Scheduler: using generic scheduler lock implementation.
--3955-- Reading suppressions file: /usr/lib/valgrind/default.supp
==3955== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-3955-
    by-bayashi-on-???
==3955== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-3955-
    by-bayashi-on-???
==3955== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
    -3955-by-bayashi-on-???
==3955==
==3955== TO CONTROL THIS PROCESS USING vgdb (which you probably
==3955== don't want to do, unless you know exactly what you're doing,
==3955== or are doing some strange experiment):
==3955==   /usr/lib/valgrind/../../bin/vgdb --pid=3955 ...command...
==3955==
==3955== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==3955==   /path/to/gdb ./unitTests
==3955== and then give GDB the following command
==3955==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=3955
==3955== --pid is optional if only one valgrind process is running
==3955==
--3955-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
    x580608c1 (???)
--3955-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
    x580608db (???)
--3955-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--3955--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--3955--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--3955--    object doesn't have a symbol table
--3955-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
    so
--3955--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
    ..
--3955--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--3955--    object doesn't have a symbol table
==3955== WARNING: new redirection conflicts with existing -- ignoring it
--3955--     old: 0x0401f2f0 (strlen              ) R-> (0000.0) 0x580608c1
    ???
--3955--     new: 0x0401f2f0 (strlen              ) R-> (2007.0) 0x04c32db0
    strlen
--3955-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
    x4c33ee0 (strcmp)
--3955-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
    x4c374f0 (mempcpy)
--3955-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--3955--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--3955--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--3955--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--3955--   .. CRC is valid
--3955-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
```

```
--3955-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
    x4a2a6e0 (_vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edaef0 (libc.so.6:stpcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
    (_vgnU_ifunc_wrapper)
--3955-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
     (rindex)
--3955-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
    )
--3955-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
    x4c37020 (strchrnul)
--3955-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
    redirected to 0x4c37130 (mempcpy)
--3955-- REDIR: 0x4fca590 (libc.so.6:__strlen_avx2) redirected to 0x4c32cf0
    (strlen)
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
```

```
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
 Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
    (2.000000,1.000000,1.000000,1.000000)

Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
    (1.000000,1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
```

```
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.500000,0.000000,0.000000,0.125000)-maxXYZT
    (2.500000,1.000000,1.000000,0.500000)

Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(-0.500000,0.000000,0.000000,0.125000)-maxXYZT
    (1.500000,1.000000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
    (1.000000,2.500000,1.000000,0.500000)

Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT
    (1.000000,1.500000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
    (1.400000,2.500000,1.000000,0.500000)

Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
    (0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
    (0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
    (1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
    (0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT
    (1.400000,1.500000,1.000000,0.500000)

All unit tests 3DTime have succeed.
--3955-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
```

```
==3955==
==3955== HEAP SUMMARY:
==3955==     in use at exit: 0 bytes in 0 blocks
==3955==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==3955==
==3955== All heap blocks were freed -- no leaks are possible
==3955==
==3955== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==3955== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3DTime'
/home/bayashi/GitHub/FMB
make[1]: Leaving directory '/home/bayashi/GitHub/FMB'
```

11.5 Static analysis

```
make[1]: Entering directory '/home/bayashi/GitHub/FMB'
cd 2D; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2D'
cppcheck --enable=all ./
Checking fmb2d.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
     for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2D'
/home/bayashi/GitHub/FMB
cd 2DTime; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2DTime'
cppcheck --enable=all ./
Checking fmb2dt.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
     for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2DTime'
/home/bayashi/GitHub/FMB
cd 3D; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3D'
cppcheck --enable=all ./
Checking fmb3d.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
```

```
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
     for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3D'
/home/bayashi/GitHub/FMB
cd 3DTime; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3DTime'
cppcheck --enable=all ./
Checking fmb3dt.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
     for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3DTime'
/home/bayashi/GitHub/FMB
make[1]: Leaving directory '/home/bayashi/GitHub/FMB'
```

References

[1] J.J.-B. Fourier. Oeuvres II. Paris, 1890

[2] T.S. Motzkin. *Beiträge zur Theorie der linearen Ungleichungen*. Thesis, 1936. Reprinted in: *Theodore S. Motzkin: selected papers* (D.Cantor et al., eds,), Birkhäuser, Boston, 1983.