

The FMB Algorithm

P. Baillehache

January 21, 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.

Contents

1	The problem as a system of linear inequations	4
1.1	Notations and definitions	4
1.2	Static case	5
1.3	Dynamic case	8
2	Resolution of the problem by Fourier-Motzkin method	11
2.1	The Fourier-Motzkin elimination method	11
2.2	Application of the Fourier-Motzkin method to the intersection problem	13
2.3	About the size of the system of linear inequations	13
3	Algorithms of the solution	15
3.1	2D static	16
3.2	3D static	22
3.3	2D dynamic	29
3.4	3D dynamic	36
4	Implementation of the algorithms in C	45
4.1	Frames	45
4.1.1	Header	45
4.1.2	Body	48
4.2	FMB	69
4.2.1	2D static	69
4.2.2	3D static	78
4.2.3	2D dynamic	89
4.2.4	3D dynamic	99
5	Minimal example of use	110
5.1	2D static	111
5.2	3D static	112
5.3	2D dynamic	114
5.4	3D dynamic	115
6	Unit tests	117
6.1	Code	117
6.1.1	2D static	117
6.1.2	3D static	129
6.1.3	2D dynamic	137
6.1.4	3D dynamic	143
6.2	Results	148

6.2.1	2D static	148
6.2.2	3D static	153
6.2.3	2D dynamic	156
6.2.4	3D dynamic	157
7	Validation against SAT	159
7.1	Code	160
7.1.1	2D static	160
7.1.2	3D static	163
7.1.3	2D dynamic	166
7.1.4	3D dynamic	170
7.2	Results	173
7.2.1	Failures	173
7.2.2	2D static	174
7.2.3	2D dynamic	174
7.2.4	3D static	174
7.2.5	3D dynamic	174
8	Qualification against SAT	175
8.1	Code	175
8.1.1	2D static	175
8.1.2	3D static	186
8.1.3	2D dynamic	197
8.1.4	3D dynamic	209
8.2	Results	220
8.2.1	2D static	221
8.2.2	3D static	226
8.2.3	2D dynamic	231
8.2.4	3D dynamic	236
9	Comments about the qualification results	241
10	Conclusion	241
11	Annex	241
11.1	Runtime environment	241
11.2	SAT implementation	243
11.2.1	Header	244
11.2.2	Body	244
11.3	Makefile	264
11.3.1	2D static	267

11.3.2	3D static	267
11.3.3	2D dynamic	268
11.3.4	3D dynamic	269
11.3.5	Doc	270
11.4	Dynamic analysis	270

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 last three sections introduce the validation and qualification in term of relative performance of the FMB algorithm against the SAT algorithm.

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 \vec{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:

$$\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \vec{O}_{\mathbb{A}} + C_{\mathbb{A}} \cdot \vec{X} \end{array} \right\} \quad (1)$$

$$\mathbb{B} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{X} \end{array} \right\} \quad (2)$$

where $\vec{O}_{\mathbb{A}}$ 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 $\vec{O}_{\mathbb{B}}$ and $C_{\mathbb{B}}$.

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

$$\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \vec{O}_{\mathbb{A}} + C_{\mathbb{A}} \cdot \vec{X} \end{array} \right\} \quad (3)$$

$$\mathbb{B} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{X} \end{array} \right\} \quad (4)$$

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:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \end{array} \right\} \quad (5)$$

If \mathbb{B} is a tetrahedron:

$$\mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \end{array} \right\} \quad (6)$$

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

$$\mathbb{A}_{\mathbb{A}} = \left\{ \vec{X} \in [0.0, 1.0]^D \right\} \quad (7)$$

and for a tetrahedron:

$$\mathbb{A}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \end{array} \right\} \quad (8)$$

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:

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \cap [0.0, 1.0]^D \end{array} \right\} \quad (9)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \cap [0.0, 1.0]^D \end{array} \right\} \quad (10)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X})]_i \leq 1.0 \end{array} \right\} \quad (11)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X}) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X})]_i \leq 1.0 \end{array} \right\} \quad (12)$$

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

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

$$\left\{ \begin{array}{l} [X]_0 \leq 1.0 \\ \dots \\ [X]_{D-1} \leq 1.0 \\ -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-1} \leq 0.0 \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_A}]_0 \\ \dots \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_A}]_{D-1} \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \end{array} \right. \quad (13)$$

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

$$\left\{ \begin{array}{l} -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-1} \leq 0.0 \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_A}]_0 \\ \dots \\ \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_A}]_{D-1} \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \end{array} \right. \quad (14)$$

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

$$\left\{ \begin{array}{l} [X]_0 \leq 1.0 \\ \dots \\ [X]_{D-1} \leq 1.0 \\ -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-1} \leq 0.0 \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{j=0}^{D-1} \left(\left(\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{j,i} \right) \cdot [X]_i \right) \leq 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{B}_A}]_j \end{array} \right. \quad (15)$$

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

$$\left\{ \begin{array}{l} -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-1} \leq 0.0 \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{0,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{D-1,i} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \sum_{j=0}^{D-1} \left(\left(\sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{j,i} \right) \cdot [X]_i \right) \leq 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{B}_A}]_j \end{array} \right. \quad (16)$$

1.3 Dynamic case

If the frames \mathbb{A} and \mathbb{B} are moving linearly along the vectors $\vec{V}_{\mathbb{A}}$ and $\vec{V}_{\mathbb{B}}$ 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:

$$\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \vec{O}_{\mathbb{A}} + C_{\mathbb{A}} \cdot \vec{X} + \vec{V}_{\mathbb{A}} \cdot t \end{array} \right\} \quad (17)$$

$$\mathbb{B} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{X} + \vec{V}_{\mathbb{B}} \cdot t \end{array} \right\} \quad (18)$$

where $\vec{O}_{\mathbb{A}}$ 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 $\vec{O}_{\mathbb{B}}$ and $C_{\mathbb{B}}$.

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

$$\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \vec{O}_{\mathbb{A}} + C_{\mathbb{A}} \cdot \vec{X} + \vec{V}_{\mathbb{A}} \cdot t \end{array} \right\} \quad (19)$$

$$\mathbb{B} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{X} + \vec{V}_{\mathbb{B}} \cdot t \end{array} \right\} \quad (20)$$

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

$$\mathbb{B}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \end{array} \right\} \quad (21)$$

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

$$\mathbb{B}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \end{array} \right\} \quad (22)$$

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

$$\mathbb{A}_\mathbb{A} = \left\{ \vec{X} \in [0.0, 1.0]^D \right\} \quad (23)$$

and for a tetrahedron:

$$\mathbb{A}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \end{array} \right\} \quad (24)$$

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:

$$\mathbb{A}_\mathbb{A} \cap \mathbb{B}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \cap [0.0, 1.0]^D \end{array} \right\} \quad (25)$$

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

$$\mathbb{A}_\mathbb{A} \cap \mathbb{B}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \cap [0.0, 1.0]^D \end{array} \right\} \quad (26)$$

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

$$\mathbb{A}_\mathbb{A} \cap \mathbb{B}_\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_\mathbb{A}^{-1} \cdot (\vec{O}_\mathbb{B} - \vec{O}_\mathbb{A} + C_\mathbb{B} \cdot \vec{X} + (\vec{V}_\mathbb{B} - \vec{V}_\mathbb{A}) \cdot t) \right]_i \leq 1.0 \end{array} \right\} \quad (27)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \right]_i \leq 1.0 \end{array} \right\} \quad (28)$$

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

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

$$\left\{ \begin{array}{ll} t & \leq 1.0 \\ -t & \leq 0.0 \\ [X]_0 & \leq 1.0 \\ \dots & \\ [X]_{D-1} & \leq 1.0 \\ -[X]_0 & \leq 0.0 \\ \dots & \\ -[X]_{D-1} & \leq 0.0 \\ [V_{\mathbb{B}_{\mathbb{A}}}]_0 \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} [X]_i & \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots & \\ [V_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} [X]_i & \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -[V_{\mathbb{B}_{\mathbb{A}}}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} [X]_i & \leq [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots & \\ -[V_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} [X]_i & \leq [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \end{array} \right. \quad (29)$$

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

$$\left\{ \begin{array}{ll} t & \leq 1.0 \\ -t & \leq 0.0 \\ -[X]_0 & \leq 0.0 \\ \dots & \\ -[X]_{D-1} & \leq 0.0 \\ [V_{\mathbb{B}_{\mathbb{A}}}]_0 \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} [X]_i & \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots & \\ [V_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} [X]_i & \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -[V_{\mathbb{B}_{\mathbb{A}}}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{0,i} [X]_i & \leq [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots & \\ -[V_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_{\mathbb{A}}}]_{D-1,i} [X]_i & \leq [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq 1.0 \end{array} \right. \quad (30)$$

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

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

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

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

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

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

with

$$\begin{aligned} i &\in 1, 2, \dots, m \\ j &\in 1, 2, \dots, n \\ x_i &\in \mathbb{R} \\ a_{ij} &\in \mathbb{R} \\ b_j &\in \mathbb{R} \end{aligned} \quad (34)$$

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

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

where

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

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

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

and

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

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

$$\max_{l \in \mathcal{I}_-'''} (-b_l''') \leq x_n \leq \min_{k \in \mathcal{I}_+'''} (b_k''') \quad (38)$$

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 \vec{S} , or (\vec{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 \vec{S}' , or (\vec{S}', t) , in the real world's coordinates system as follow:

$$\vec{S}' = \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{S} \quad (39)$$

$$(\vec{S}', t) = \left(\vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{S} + \vec{V}_{\mathbb{B}} \cdot t, t \right) \quad (40)$$

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 inconsistency 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 $\forall i, X_i \in [0.0, 1.0]$:

$$Y < \sum_{i \in I^-} a_i \quad (41)$$

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

2.3 About the size of the system of linear inequations

During implementation in languages where the developer 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:

$$n_- + n_+ + n_0 = N \quad (42)$$

and

$$n_-.n_+ + n_0 = N' \quad (43)$$

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

$$n_- + n_+ = K \quad (44)$$

then,

$$n_-.n_+ = n_-.(K - n_-) \quad (45)$$

then,

$$n_-.n_+ = K.n_- - n_-^2 \quad (46)$$

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

$$n_-.n_+ \leq K^2/2 - K^2/4 \quad (47)$$

or,

$$n_-.n_+ \leq K^2/4 \quad (48)$$

and putting back the definition of K

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

which is also

$$n_-.n_+ \leq N^2/4 \quad (50)$$

From (43) we get,

$$N' \leq N^2/4 + n_0 \quad (51)$$

and finally,

$$N' \leq N^2/4 + N \quad (52)$$

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 corresponding 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 practice, 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 developer responsibility.

3.1 2D static

```
ENUM FrameType
    FrameCuboid,
    FrameTetrahedron
END ENUM

STRUCT AAB2D
    // x,y
    real min[2]
    real max[2]
END STRUCT

STRUCT Frame2D
    FrameType type
    real orig[2]
    // comp[iComp][iAxis]
    real comp[2][2]
    AAB2D 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]
    END FOR
  END FOR
END FUNCTION

```

```

        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

```

```

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 AAB3DPrint(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
            END IF
        END FOR
    END FOR
END FUNCTION

```



```

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
        END IF
    END FOR
END FUNCTION

```

```

        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]
    END IF
END FUNCTION

```

```

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)
    AAB3DPrint(bdgBox3D)
ELSE
    PRINT "No intersection."
END IF

```

3.3 2D dynamic

```

ENUM FrameType
    FrameCuboid,
    FrameTetrahedron
END ENUM

STRUCT AAB2DTime
    // 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]
    AAB2DTime 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

```

```

        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
    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
                END IF
            END FOR
        END IF
    END FOR
END FUNCTION

```



```

        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

```

```

        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)
    AAB3DTimePrint(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>

// ----- 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(
    const Frame2D* const P,
    const Frame2D* const Q,
    Frame2D* const Qp);
void Frame3DImportFrame(
    const Frame3D* const P,
    const Frame3D* const Q,
    Frame3D* const Qp);
void Frame2DTimeImportFrame(
    const Frame2DTime* const P,
    const Frame2DTime* const Q,
    Frame2DTime* const Qp);
void Frame3DTimeImportFrame(
    const 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);

// 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][0] * 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(
    const Frame2D* const P,
    const Frame2D* const Q,
    Frame2D* const Qp) {

    // 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(
    const Frame3D* const P,
    const Frame3D* const Q,
    Frame3D* const Qp) {

// 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(
    const Frame2DTime* const P,
    const Frame2DTime* const Q,
    Frame2DTime* const Qp) {

    // 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(
    const Frame3DTime* const P,
    const Frame3DTime* const Q,
    Frame3DTime* const Qp) {

    // 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 (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];

    }
    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* bbpma = bdgBoxProj->min;
    double* bbpma = bdgBoxProj->max;
    const double (*tc)[3] = that->comp;

    // The time component is not affected
    bbpma[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];


```

```

    }

    bbpma[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 (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];

        }

        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 AABBB2DTimePrint(const AABBB2DTime* 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 AABBB3DTimePrint(const AABBB3DTime* 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(")");

}

// 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(
    const 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,
    AABBB2D* 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 AABBB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2D(
    const int iVar,
    const double (*M)[2],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABBB2D* 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 AABBB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2D(
    const int iVar,
    const double (*M)[2],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABBB2D* 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(
    const 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 <= 0_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 memorise the nb of rows in the system
    int nbRows = 2;

    if (that->type == FrameCuboid) {

        // sum_iC_j, iX_i <= 1.0-0_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_i0_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_i X_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(
    const 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  $M.X \leq Y$  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  $M.X \leq Y$ 
// 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  $M.X \leq Y$  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  $M.X \leq Y$  where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABD bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3D(
    const int iVar,
    const double (*M)[3],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABD3D* 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* MRow = M[iRow];
    double fabsMRowIVar = fabs(MRow[0]);

    // If the coefficient of the first variable on this row is not null
    if (fabsMRowIVar > 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 (MRow[iCol] > EPSILON) {
                min += MRow[iCol] * bdgBoxMin[iCol + iVar];
                max -= MRow[iCol] * bdgBoxMin[iCol + iVar];
            } else if (MRow[iCol] < EPSILON) {
                min += MRow[iCol] * bdgBoxMax[iCol + iVar];
                max -= MRow[iCol] * bdgBoxMax[iCol + iVar];
            }
        }

        min /= -1.0 * MRow[0];
        max /= MRow[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(
    const 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 <= 0_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 memorise the nb of rows in the system
    int nbRows = 3;

    if (that->type == FrameCuboid) {

        // 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.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_i0_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 the
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(
    const 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  $M.X \leq Y$ 
// 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  $M.X \leq Y$  which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABBB 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,
    AABBB2DTime* const bdgBox);

// Get the bounds of the iVar-th variable in the nbRows rows
// system  $M.X \leq Y$  where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the AABBB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2DTime(
    const int iVar,
    const double (*M)[3],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABBB2DTime* const bdgBox);

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

// Eliminate the first variable in the system  $M.X \leq Y$ 
// 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  $M.X \leq Y$  which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABBB 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,
    AABBB2DTime* 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 AAB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar2DTime(
    const int iVar,
    const double (*M)[3],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AAB2DTime* 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* MRow = M[iRow];
        double fabsMRowIVar = fabs(MRow[0]);

        // If the coefficient of the first variable on this row is not null
        if (fabsMRowIVar > 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 (MRow[iCol] > EPSILON) {
                    min += MRow[iCol] * bdgBoxMin[iCol + iVar];
                    max -= MRow[iCol] * bdgBoxMin[iCol + iVar];
                } else if (MRow[iCol] < EPSILON) {
                    min += MRow[iCol] * bdgBoxMax[iCol + iVar];
                    max -= MRow[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(
    const 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 memorise 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_i0_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 the
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(
    const 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
// AABBB 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,
    AABBB3DTime* 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 AABBB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3DTime(
    const int iVar,
    const double (*M)[4],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABBB3DTime* 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
// AABBB 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,
    AABBB3DTime* 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* MRow = M[iRow];
        double fabsMRowiVar = fabs(MRow[0]);

        // If the coefficient of the first variable on this row is not null
        if (fabsMRowiVar > 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 (MRow[iCol] > EPSILON) {
                    min += MRow[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(
    const 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<=0_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<=1.0-0_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_i0_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_i X_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];
        }
    }

    AABBox2DPrint(&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.243)

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 algorithm 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 memoize 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 algorithm 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 memoize 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 algorithm 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 memoize 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 algorithm 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 pair (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 memoize 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)

```



```

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.243)

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++;

    } else {

        // Update the number of no intersection
        nbNoInter++;

    }

    // Flip the pair of Frames
    that = &Q;
    tho = &P;

}

```

```

}

void Validate2D(void) {

    // Initialise the random generator
    srand(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 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
    srand(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 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
    srand(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 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
    srand(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 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

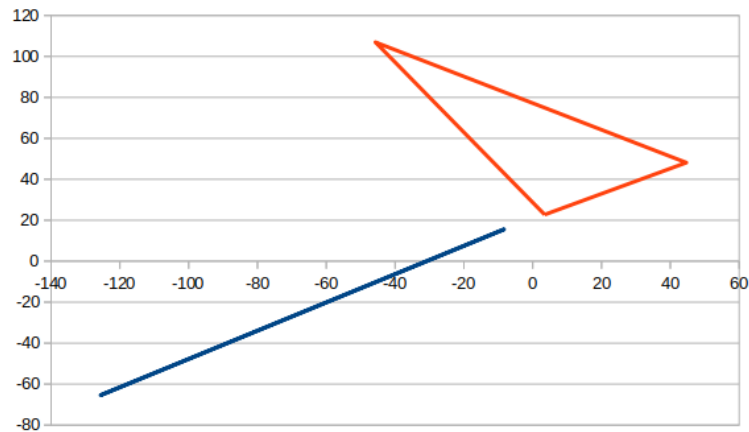
Validation has failed in one case: when one or both of the frame are degenerated (at least two of there components are colinear). An example is given below for reference:

```

===== 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 469212 intersections and 1530724 no intersections
```

7.2.3 2D dynamic

```
===== 2D dynamic =====
Validation2DTime has succeed.
Tested 743642 intersections and 1256258 no intersections
```

7.2.4 3D static

```
===== 3D static =====
Validation3D has succeed.
Tested 315722 intersections and 1684278 no intersections
```

7.2.5 3D dynamic

```

===== 3D dynamic =====
Validation3DTime has succeed.
Tested 523914 intersections and 1476084 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 100000
// 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;

// 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) {

    // 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--;) {

    // 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 a 10ms
} else if (deltausFMB < 10) {

    printf("deltausFMB < 10ms, increase NB_REPEAT\n");
    exit(0);

// Else, if time of execution for SAT was less than a 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(void) {

    // Initialise the random generator
    srand(time(NULL));

    // Open the files to save the results
    FILE* fp = fopen("../Results/qualification2D.txt", "w");
    FILE* fpCC = fopen("../Results/qualification2DCC.txt", "w");
    FILE* fpCT = fopen("../Results/qualification2DCT.txt", "w");
    FILE* fpTC = fopen("../Results/qualification2DTC.txt", "w");
    FILE* fpTT = fopen("../Results/qualification2DTT.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
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);

    }

}

// Display 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) {

    Qualify2DStatic();

    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 100000
// 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;

// 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) {

```

```

// 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--;) {

    // 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 a 10ms
} else if (deltausFMB < 10) {

    printf("deltausFMB < 10ms, increase NB_REPEAT\n");
    exit(0);

// Else, if time of execution for SAT was less than a 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(void) {

    // Initialise the random generator
    srand(time(NULL));

    // Open the files to save the results
    FILE* fp = fopen("../Results/qualification3D.txt", "w");
    FILE* fpCC = fopen("../Results/qualification3DCC.txt", "w");
    FILE* fpCT = fopen("../Results/qualification3DCT.txt", "w");
    FILE* fpTC = fopen("../Results/qualification3DTC.txt", "w");
    FILE* fpTT = fopen("../Results/qualification3DTT.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
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);

}

}

// Display 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) {

    Qualify3DStatic();

    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 10000
// 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 a 10ms
} else if (deltausFMB < 10) {

    printf("deltausFMB < 10ms, increase NB_REPEAT\n");
    exit(0);

// Else, if time of execution for SAT was less than a 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
    srand(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);

    }

}

// Display 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 100000
// 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 a 10ms
} else if (deltausFMB < 10) {

    printf("deltausFMB < 10ms, increase NB_REPEAT\n");
    exit(0);

// Else, if time of execution for SAT was less than a 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
    srand(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);

    }

}

// Display 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;
    printf(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;
    printf(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;
    printf(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;
}

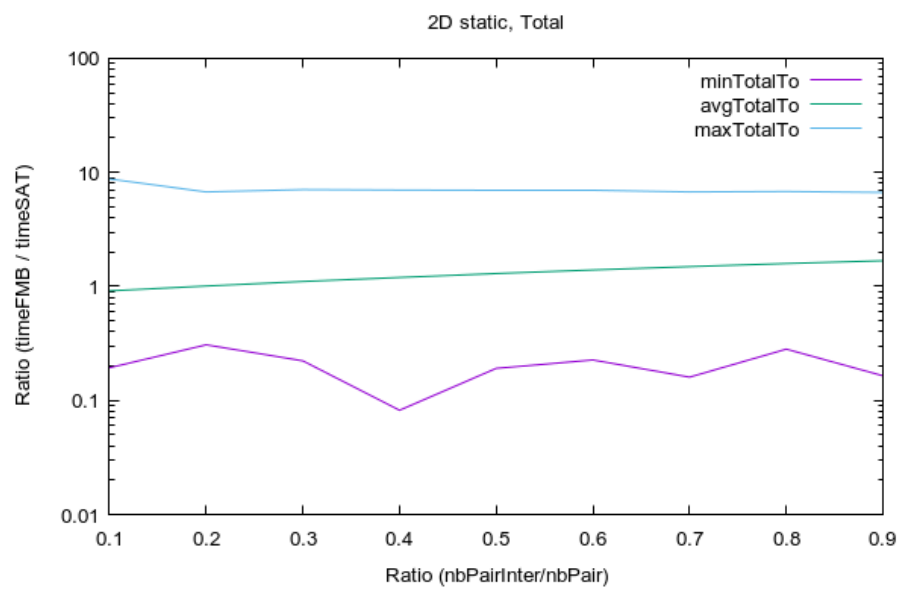
```

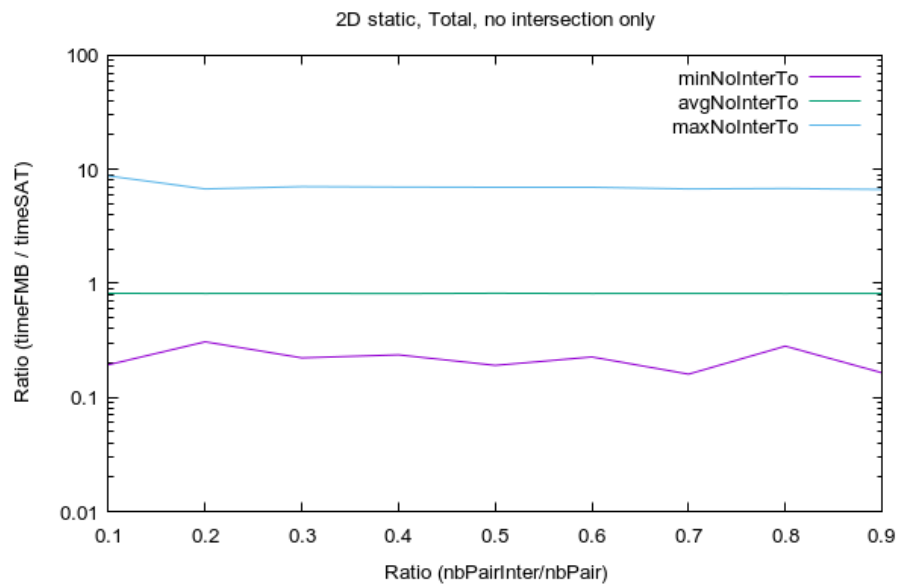
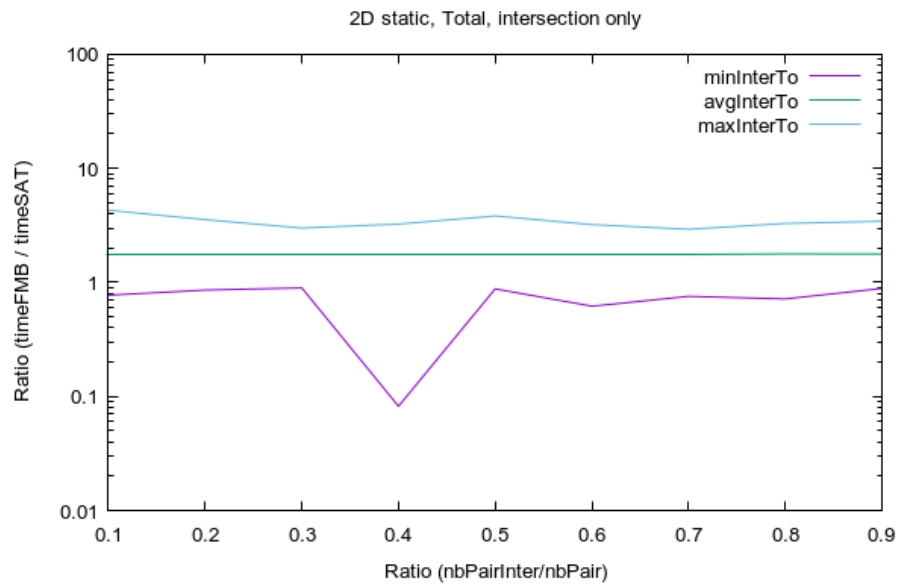
8.2 Results

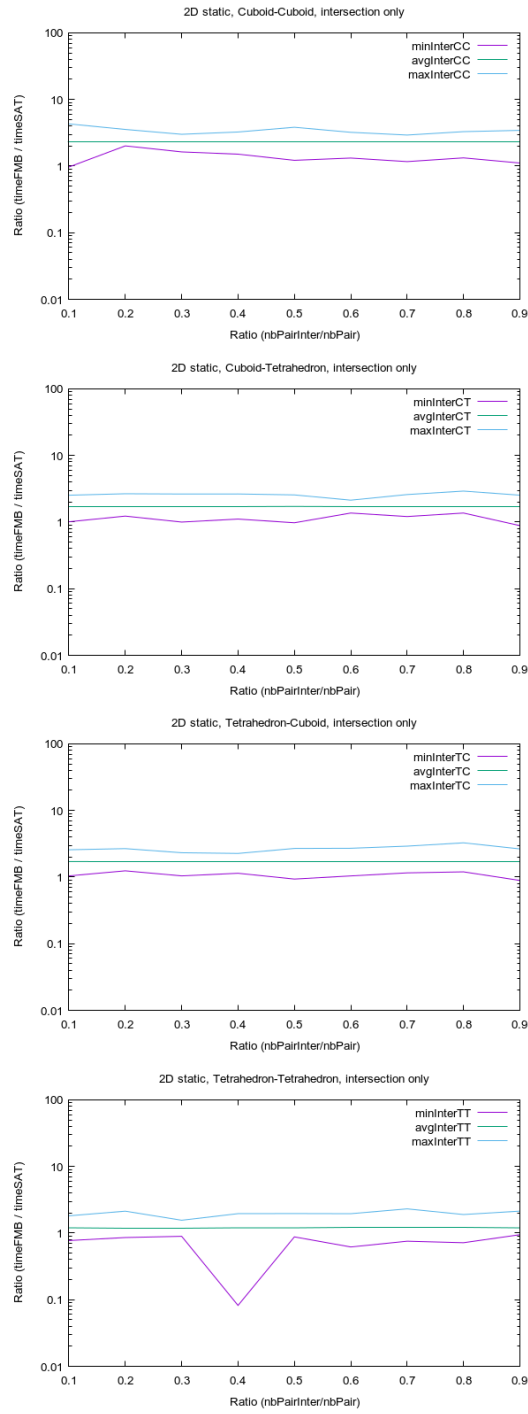
In this subsection I give the results of the qualification for each case. These results are commented in the next section.

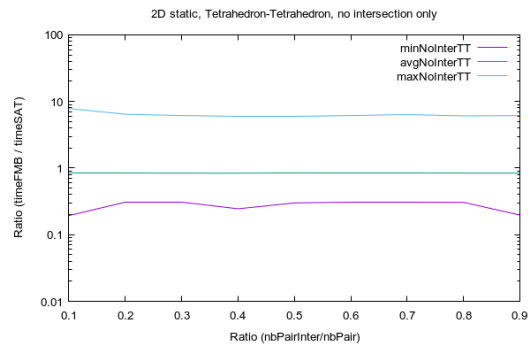
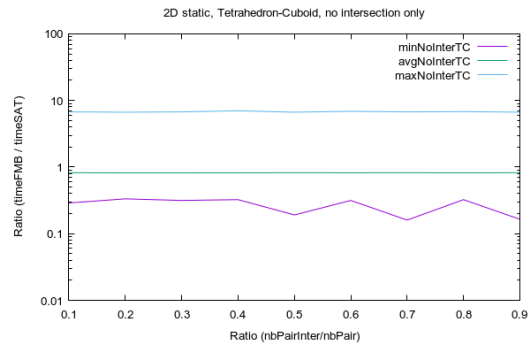
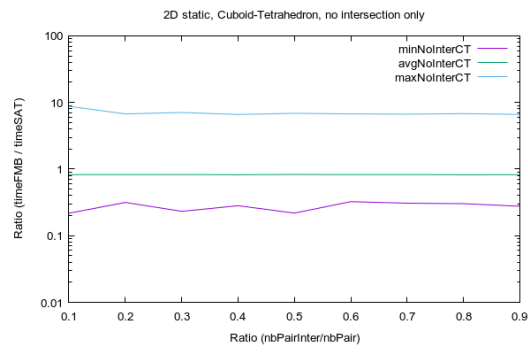
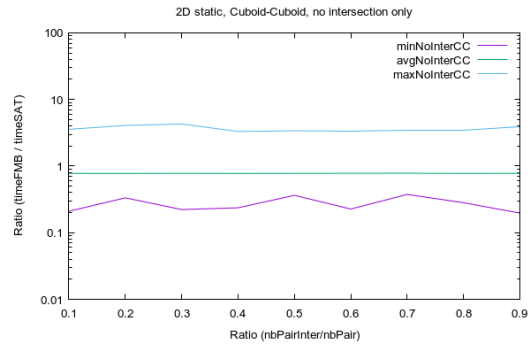
8.2.1 2D static

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	47406	152588	0.772727	1.774545	4.296296	0.193548	0.818249	8.750000	0.193548	0.913879	8.750000
0.2	46896	153094	0.857143	1.774937	3.549020	0.307692	0.815669	6.733333	0.307692	1.075127	6.733333
0.3	47218	152778	0.894737	1.773439	3.000000	0.222222	0.817024	7.066667	0.222222	1.103949	7.066667
0.4	46856	153132	0.082042	1.774700	3.245283	0.236364	0.814509	7.000000	0.082042	1.198585	7.000000
0.5	47680	152316	0.877358	1.776000	3.826923	0.191176	0.819332	6.866667	0.191176	1.297666	6.866667
0.6	46944	153048	0.618705	1.777251	3.211538	0.226415	0.816493	6.866667	0.226415	1.392948	6.866667
0.7	47366	152624	0.754386	1.777663	2.923077	0.160000	0.817333	6.733333	0.160000	1.489665	6.733333
0.8	46808	153188	0.716667	1.779339	3.288462	0.282051	0.815331	6.800000	0.282051	1.586537	6.800000
0.9	47176	152820	0.884615	1.774765	3.433962	0.154557	0.817121	6.666667	0.154557	1.679001	6.666667
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	13262	36800	0.968254	2.343391	4.296296	0.209677	0.775211	3.562500	0.209677	0.932029	4.296296
0.2	12974	37340	2.010101	2.343833	3.549020	0.333333	0.774786	4.073171	0.333333	1.088595	4.073171
0.3	13066	36724	1.631579	2.344679	3.000000	0.222222	0.777742	4.275000	0.222222	1.247823	4.275000
0.4	13018	36756	1.509804	2.343691	3.245283	0.236364	0.774182	3.307692	0.236364	1.401986	3.307692
0.5	13388	36556	1.220000	2.343697	3.826923	0.368336	0.776539	3.384615	0.368336	1.580118	3.826923
0.6	13306	36924	1.319149	2.343599	3.211538	0.226415	0.779392	3.342105	0.226415	1.717916	3.342105
0.7	13336	36754	1.166667	2.343364	2.923077	0.375000	0.780052	3.410256	0.375000	1.874371	3.410256
0.8	13254	37156	1.326316	2.343609	3.288462	0.282051	0.775222	3.410256	0.282051	2.029332	3.410256
0.9	13082	36688	1.111111	2.344977	3.433962	0.196970	0.778204	3.902439	0.196970	2.189299	3.902439
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	12002	38444	1.010417	1.713067	2.532258	0.217391	0.825422	8.750000	0.217391	0.914187	8.750000
0.2	11956	38176	1.229885	1.713030	2.661290	0.315789	0.824548	6.733333	0.315789	1.002244	6.733333
0.3	11746	38022	1.000000	1.713535	2.636364	0.232143	0.827049	7.066667	0.232143	1.092995	7.066667
0.4	11746	38216	1.109890	1.713291	2.639344	0.282051	0.821866	6.600000	0.282051	1.178436	6.600000
0.5	11956	38182	0.975155	1.714598	2.557377	0.218182	0.829538	6.866667	0.218182	1.272068	6.866667
0.6	11472	38530	1.367089	1.712784	2.131148	0.324324	0.824166	6.733333	0.324324	1.357337	6.733333
0.7	11706	37916	1.211111	1.713712	2.590164	0.307692	0.826515	6.666667	0.307692	1.447553	6.666667
0.8	11890	38480	1.366197	1.713645	2.919355	0.302326	0.822881	6.800000	0.302326	1.535452	6.800000
0.9	11956	38500	0.884615	1.713273	2.534483	0.276596	0.823944	6.600000	0.276596	1.624340	6.600000
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	11678	37808	1.042553	1.707585	2.568966	0.289474	0.824774	6.733333	0.289474	0.913055	6.733333
0.2	11868	38498	1.240506	1.705976	2.672131	0.333333	0.819566	6.666667	0.333333	0.996840	6.666667
0.3	12070	37892	1.042553	1.706505	2.318182	0.315789	0.820123	6.733333	0.315789	1.086038	6.733333
0.4	11906	38670	1.141176	1.705560	2.262285	0.324324	0.821448	7.000000	0.324324	1.175092	7.000000
0.5	11856	37864	0.933333	1.707431	2.689655	0.191176	0.823707	6.666667	0.191176	1.265569	6.666667
0.6	11840	38138	1.040816	1.707618	2.704918	0.315789	0.822087	6.866667	0.315789	1.353406	6.866667
0.7	12056	38046	1.156250	1.707664	2.913793	0.160000	0.823394	6.733333	0.160000	1.442383	6.733333
0.8	11568	38352	1.197531	1.707071	3.267606	0.324324	0.820798	6.800000	0.324324	1.529816	6.800000
0.9	11854	38354	0.890909	1.707463	2.650000	0.154557	0.823948	6.666667	0.154557	1.619102	6.666667
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	10464	39536	0.772727	1.198335	1.819901	0.193548	0.845095	7.800000	0.193548	0.880469	7.800000
0.2	10098	39080	0.857143	1.198453	2.130435	0.307692	0.840231	6.428571	0.307692	0.913476	6.428571
0.3	10346	40140	0.894737	1.198666	1.558823	0.307692	0.840542	6.142857	0.307692	0.947979	6.142857
0.4	10186	39490	0.082042	1.199142	1.958333	0.244898	0.838128	6.071429	0.082042	0.982534	6.071429
0.5	10450	39714	0.877358	1.198399	1.964286	0.300000	0.844742	6.066667	0.300000	1.021568	6.066667
0.6	10326	39456	0.618705	1.198925	1.954545	0.307692	0.838316	6.142857	0.307692	1.054631	6.142857
0.7	10288	39908	0.754386	1.199020	2.308824	0.307692	0.840785	6.357143	0.307692	1.090949	6.357143
0.8	10096	39200	0.716667	1.197737	1.897059	0.305556	0.834450	6.071429	0.305556	1.127147	6.071429
0.9	10284	39298	0.945055	1.198480	2.136364	0.196970	0.840186	6.133333	0.196970	1.162651	6.133333



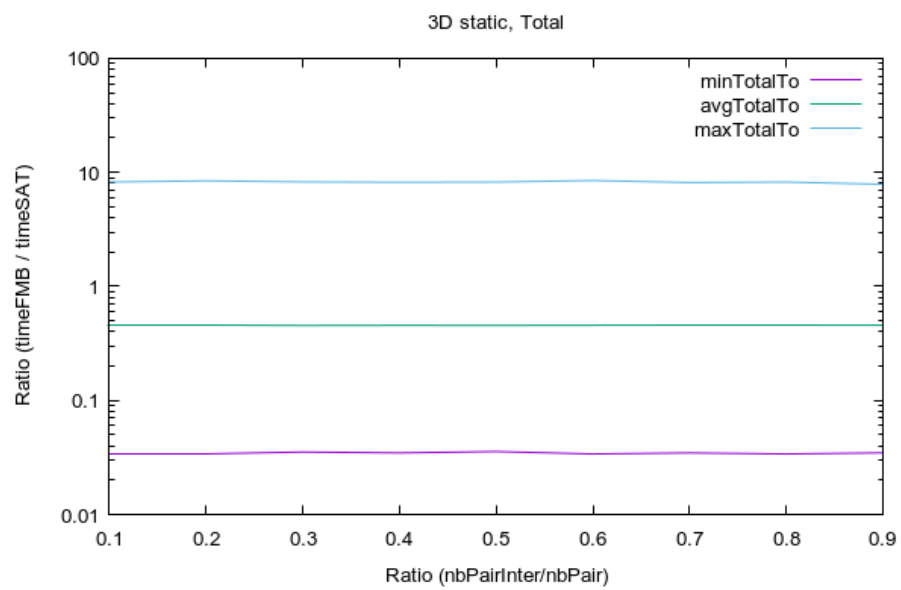


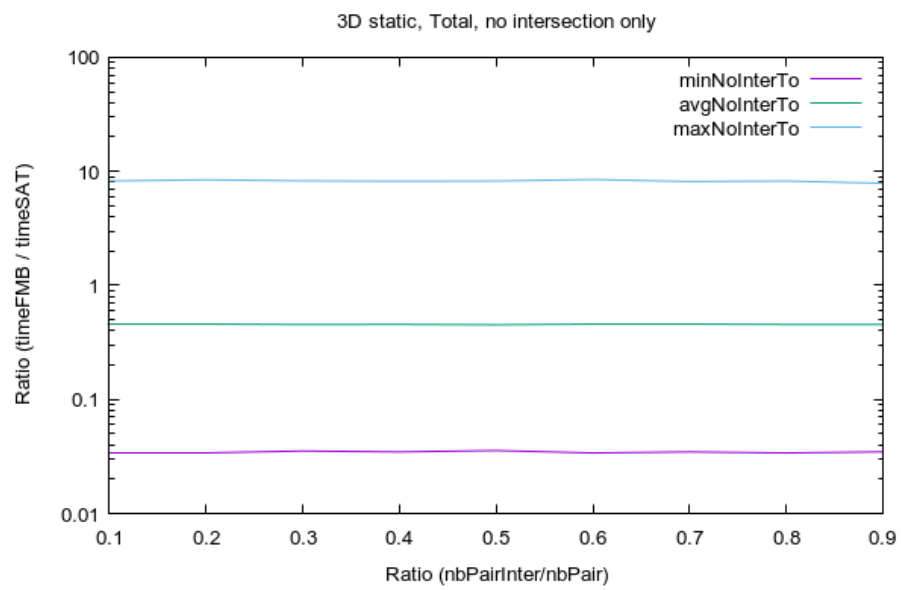
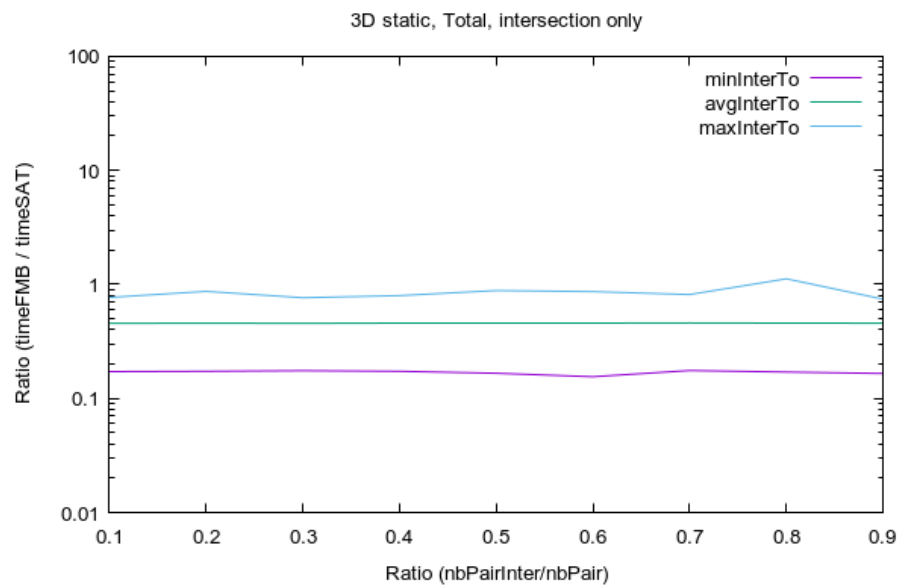


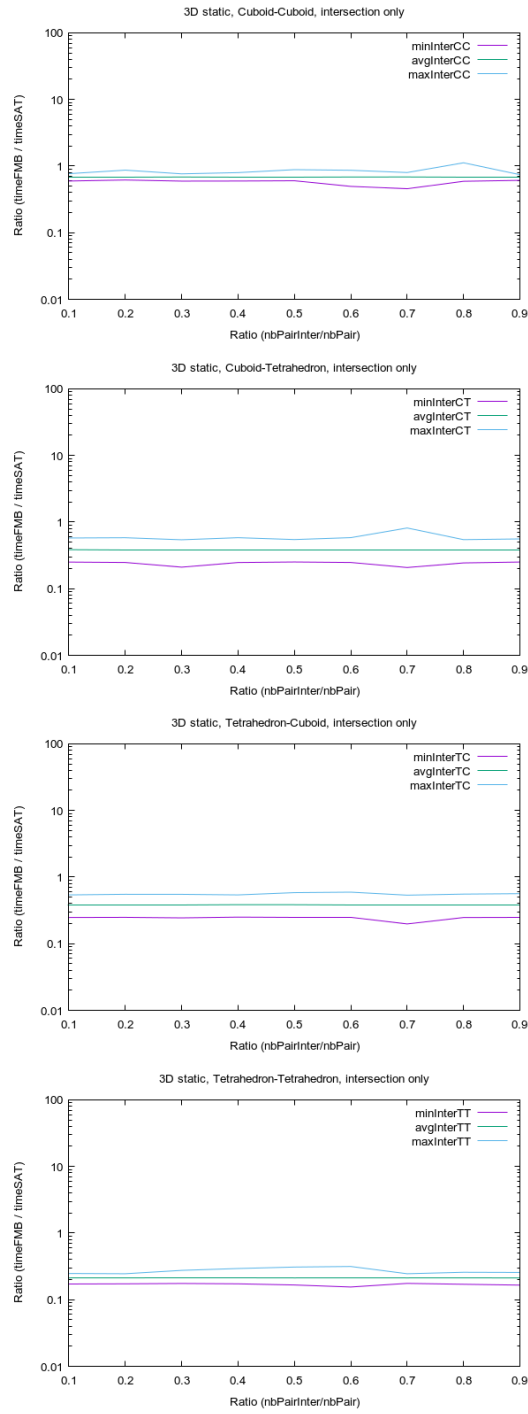


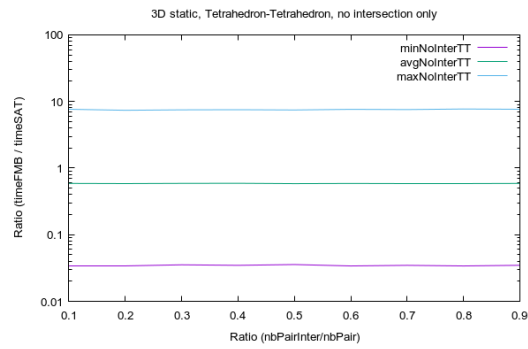
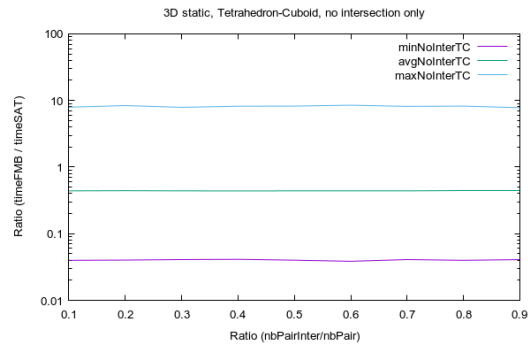
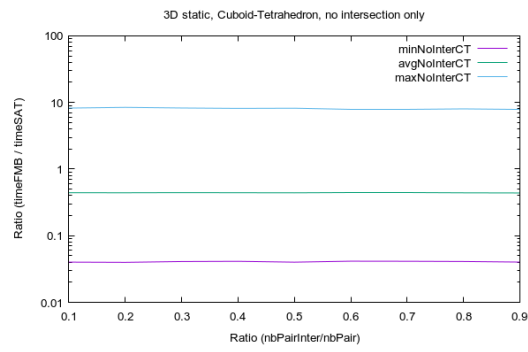
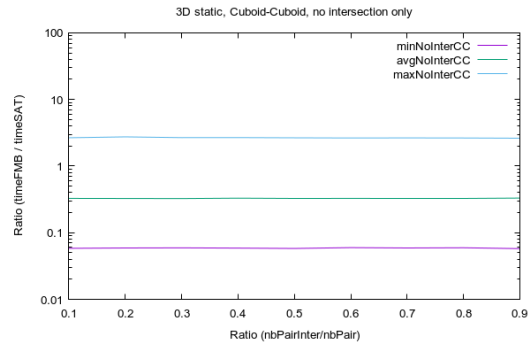
8.2.2 3D static

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	31348	16650	0.172088	0.455944	0.768836	0.034005	0.453132	8.222222	0.034005	0.453413	8.222222
0.2	31664	166336	0.457077	0.457077	0.868118	0.034005	0.453264	8.432432	0.034005	0.453562	8.432432
0.3	31766	168214	0.175214	0.455768	0.764103	0.035294	0.453289	8.250000	0.034660	0.454033	8.250000
0.4	31456	168544	0.172996	0.457370	0.797639	0.034660	0.454649	8.194444	0.034660	0.455737	8.194444
0.5	31440	168560	0.163993	0.458186	0.833178	0.035623	0.451684	8.222222	0.035623	0.454935	8.222222
0.6	32016	167984	0.154879	0.458421	0.864139	0.033920	0.452731	8.485714	0.033920	0.456145	8.485714
0.7	31318	166682	0.175661	0.459256	0.814961	0.034660	0.457285	8.138889	0.034660	0.457314	8.138889
0.8	31354	166646	0.170443	0.457473	1.120135	0.033962	0.454221	8.222222	0.033962	0.456823	8.222222
0.9	31728	168272	0.165548	0.456980	0.742857	0.034704	0.453263	7.861111	0.034704	0.456500	7.861111
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	10390	39592	0.595930	0.685998	0.768836	0.058233	0.327497	2.660377	0.058233	0.363347	2.660377
0.2	10538	38946	0.617512	0.686027	0.868118	0.058943	0.326211	2.743750	0.058943	0.398174	2.743750
0.3	10600	39464	0.594958	0.685942	0.764103	0.059184	0.325275	2.672956	0.059184	0.433445	2.672956
0.4	10558	39282	0.596131	0.686025	0.797639	0.058704	0.329687	2.679245	0.058704	0.472222	2.679245
0.5	10618	39386	0.601732	0.686029	0.833178	0.058000	0.326951	2.658250	0.058000	0.506490	2.658250
0.6	10842	39114	0.495434	0.685883	0.864139	0.059671	0.327616	2.639241	0.059671	0.542576	2.639241
0.7	10642	39810	0.456790	0.685913	0.799342	0.058943	0.328168	2.647436	0.058943	0.578559	2.647436
0.8	10538	39444	0.591033	0.685890	1.120135	0.059426	0.327747	2.641509	0.059426	0.614282	2.641509
0.9	10692	39566	0.610044	0.685710	0.742857	0.057613	0.330888	2.621118	0.057613	0.650227	2.621118
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	7716	42006	0.260900	0.384947	0.576975	0.040119	0.441014	8.222222	0.040119	0.436388	8.222222
0.2	8144	42450	0.247608	0.384309	0.581646	0.039764	0.440253	8.432432	0.039764	0.429065	8.432432
0.3	7750	41638	0.210473	0.384639	0.542625	0.040876	0.442772	8.250000	0.040876	0.425332	8.250000
0.4	7990	42376	0.247273	0.384651	0.583222	0.041221	0.440996	8.138889	0.041221	0.418458	8.138889
0.5	7984	42000	0.261526	0.384428	0.546419	0.040057	0.439683	8.194444	0.040057	0.412065	8.194444
0.6	7840	42218	0.247642	0.384456	0.583558	0.041411	0.438302	7.861111	0.041411	0.405994	7.861111
0.7	7788	42248	0.207767	0.384438	0.814961	0.041298	0.438669	7.864865	0.041298	0.400707	7.864865
0.8	7860	42110	0.243697	0.384246	0.544607	0.040971	0.439741	8.000000	0.040971	0.395345	8.000000
0.9	7806	42076	0.251225	0.384882	0.558680	0.040179	0.437544	7.861111	0.040179	0.389968	7.861111
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	8036	42172	0.247653	0.384632	0.539894	0.039823	0.439030	7.918919	0.039823	0.433590	7.918919
0.2	7872	42212	0.248492	0.384524	0.551724	0.040179	0.441964	8.371429	0.040179	0.430476	8.371429
0.3	8034	42198	0.244048	0.384604	0.550667	0.040847	0.439259	7.888889	0.040847	0.422862	7.888889
0.4	7714	41686	0.250608	0.384895	0.540214	0.041221	0.438555	8.194444	0.041221	0.417091	8.194444
0.5	7698	42190	0.247273	0.384614	0.583777	0.040059	0.439146	8.222222	0.040059	0.411880	8.222222
0.6	8076	41716	0.246988	0.384788	0.593454	0.038516	0.439599	8.485714	0.038516	0.406713	8.485714
0.7	7836	41976	0.198095	0.385020	0.534574	0.040847	0.439474	8.138889	0.040847	0.401366	8.138889
0.8	7790	42028	0.247031	0.384677	0.554813	0.039829	0.442127	8.222222	0.039829	0.396167	8.222222
0.9	7858	42172	0.248193	0.384477	0.563492	0.040785	0.441553	7.777778	0.040785	0.390185	7.777778
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	5206	44880	0.172088	0.212261	0.247434	0.034005	0.588555	7.629630	0.034005	0.550926	7.629630
0.2	5110	44728	0.173137	0.212670	0.245132	0.034005	0.584770	7.370370	0.034005	0.510310	7.370370
0.3	5402	44914	0.175214	0.212193	0.275281	0.035294	0.588702	7.481481	0.035294	0.475756	7.481481
0.4	5194	45200	0.172996	0.212079	0.293803	0.034660	0.590892	7.518519	0.034660	0.439367	7.518519
0.5	5140	44984	0.163993	0.212271	0.309051	0.035623	0.583858	7.444444	0.035623	0.398065	7.444444
0.6	5258	44936	0.154879	0.212777	0.315145	0.033920	0.587383	7.607143	0.033920	0.362619	7.607143
0.7	5052	44648	0.175661	0.212287	0.245198	0.034660	0.585556	7.555556	0.034660	0.325531	7.555556
0.8	5166	45064	0.170443	0.212716	0.257993	0.033962	0.589738	7.703704	0.033962	0.288119	7.703704
0.9	5372	44458	0.165548	0.212138	0.256236	0.034704	0.588159	7.629630	0.034704	0.249740	7.629630



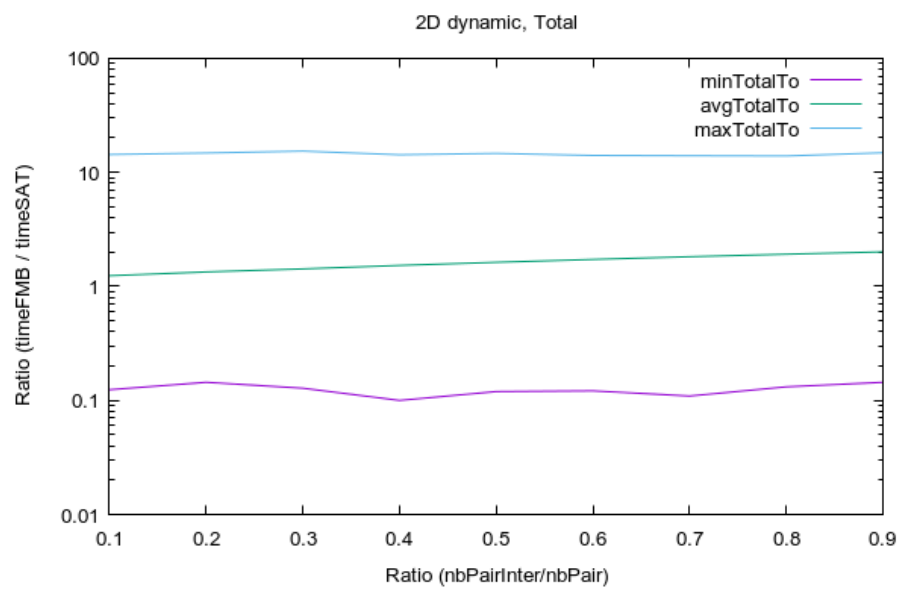


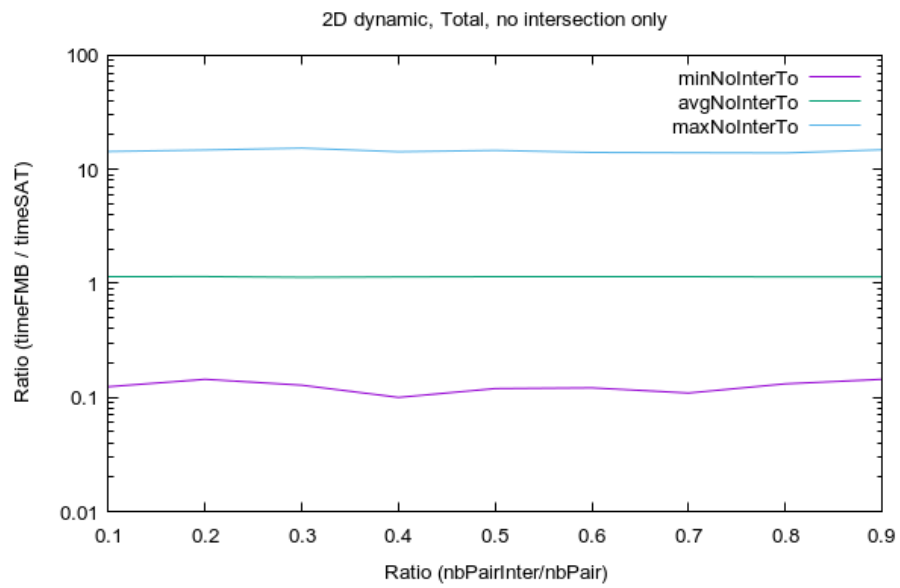
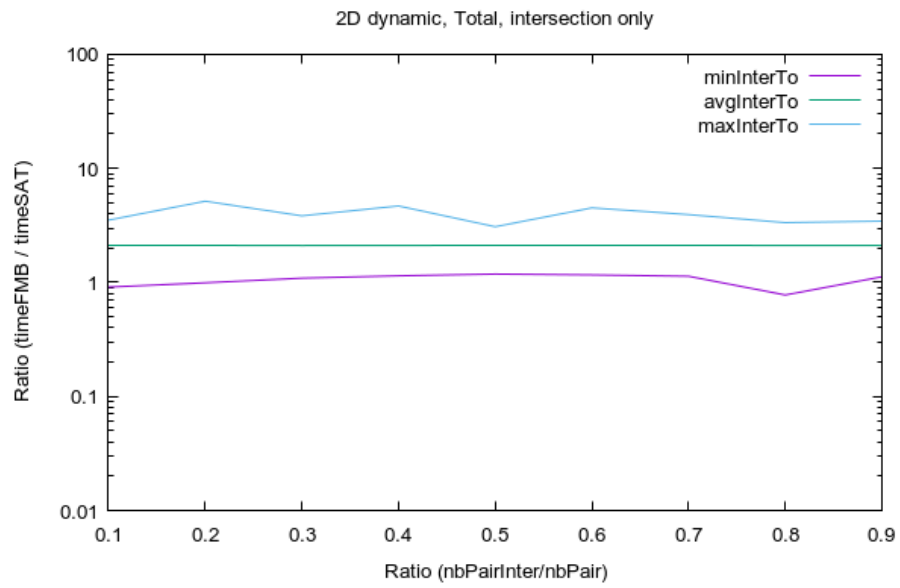


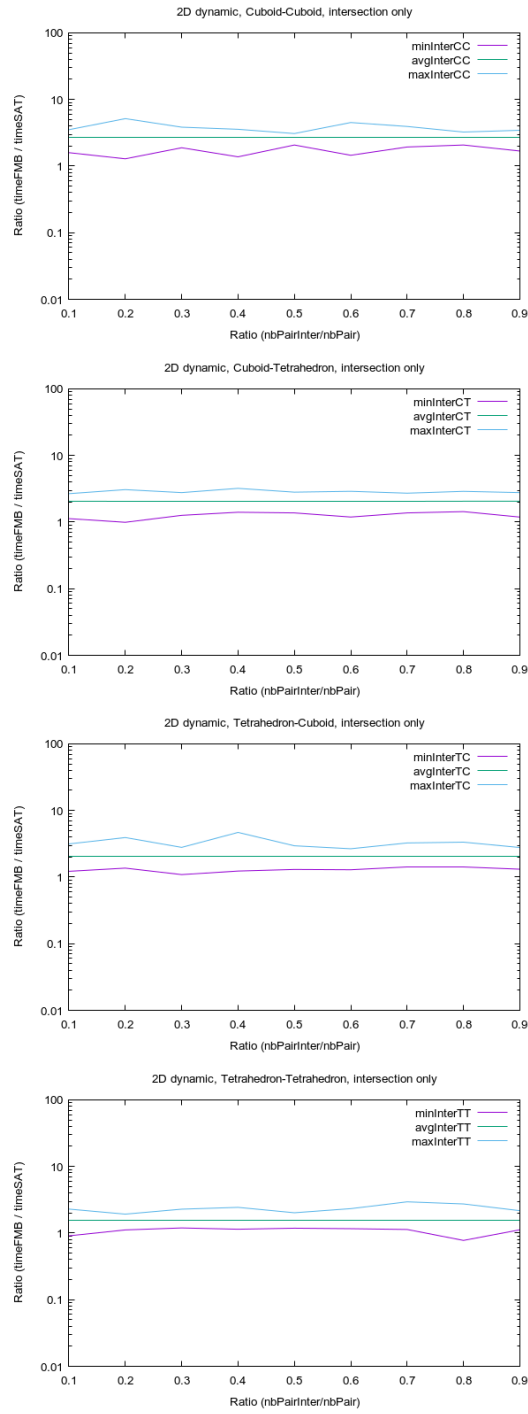


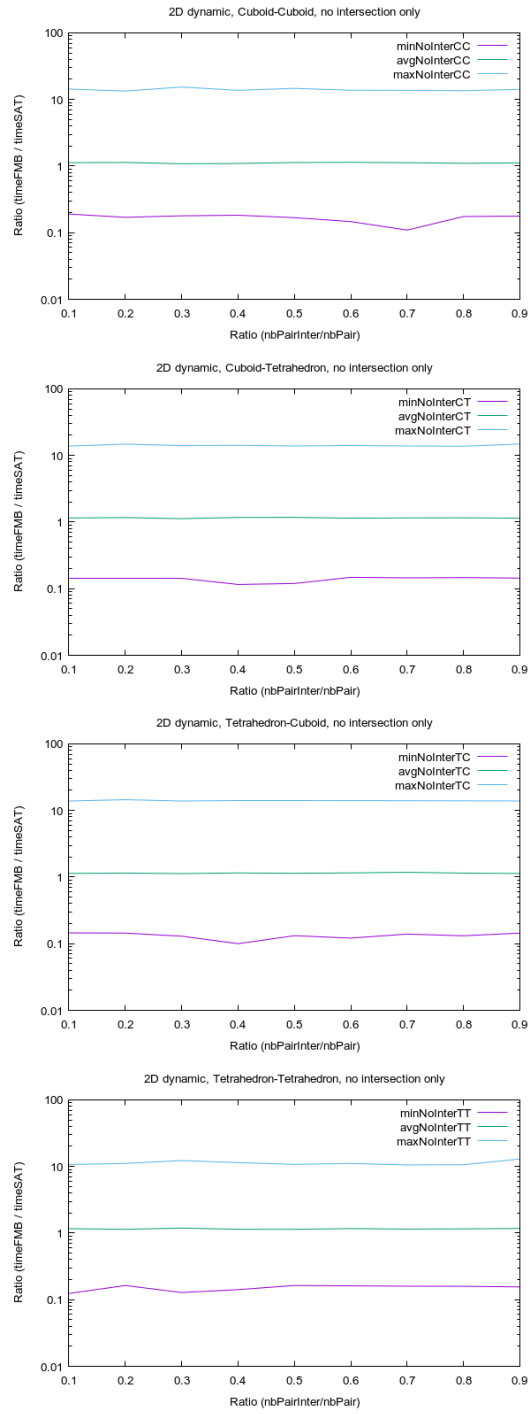
8.2.3 2D dynamic

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	74634	126360	0.910072	2.110064	3.514925	0.124088	1.146043	14.344828	0.124088	1.241545	14.344828
0.2	73860	126132	0.992727	2.110075	5.152672	0.144144	1.147273	14.791667	0.144144	1.339833	14.791667
0.3	74130	126538	1.089844	2.106538	3.837037	0.132652	1.132652	15.357143	0.128000	1.424818	15.357143
0.4	74030	125960	1.141553	2.108509	4.676259	0.100000	1.140739	14.269231	0.100000	1.527847	14.269231
0.5	74402	125588	1.183099	2.106226	3.076923	0.119658	1.146142	14.678571	0.119658	1.626184	14.678571
0.6	74214	126778	1.162879	2.105662	4.507463	0.121429	1.151892	14.080000	0.121429	1.724154	14.080000
0.7	74054	126934	1.131222	2.106917	3.932331	0.109290	1.151658	14.000000	0.109290	1.820339	14.000000
0.8	74348	126644	0.776074	2.107760	3.343195	0.131579	1.141260	13.920000	0.131579	1.914462	13.920000
0.9	74448	125544	1.120370	2.106101	3.446970	0.144144	1.142434	14.840000	0.144144	2.009734	14.840000
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	20156	30228	1.589744	2.686982	3.514925	0.190476	1.125729	14.344828	0.190476	1.281854	14.344828
0.2	19996	30140	1.284810	2.686885	5.152672	0.170000	1.134145	13.370370	0.170000	1.444693	13.370370
0.3	19770	29672	1.883598	2.686887	3.837037	0.179245	1.082581	15.357143	0.179245	1.563873	15.357143
0.4	19890	29634	1.377289	2.667154	3.561728	0.182927	1.093540	13.724138	0.182927	1.730985	13.724138
0.5	19750	29988	2.069767	2.667485	3.076923	0.168142	1.129351	14.678571	0.168142	1.908418	14.678571
0.6	19608	30138	1.463061	2.667214	4.507463	0.146789	1.139496	13.740741	0.146789	2.068127	13.740741
0.7	19514	30048	1.933333	2.667709	3.932331	0.109290	1.123099	13.689655	0.109290	2.218326	13.689655
0.8	19808	29816	2.069767	2.667400	3.240602	0.175000	1.100581	13.555556	0.175000	2.370036	13.555556
0.9	19838	30248	1.681614	2.667086	3.446970	0.177215	1.115490	14.142857	0.177215	2.529277	14.142857
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	18500	31146	1.127049	2.056518	2.669118	0.145455	1.151714	13.791667	0.145455	1.242194	13.791667
0.2	18232	31416	0.992727	2.057213	3.069182	0.145455	1.168256	14.791667	0.145455	1.346047	14.791667
0.3	18768	31700	1.262172	2.056980	2.769784	0.142857	1.125788	14.083333	0.142857	1.405146	14.083333
0.4	18260	32166	1.404145	2.057648	3.207143	0.115646	1.173489	14.208333	0.115646	1.527152	14.208333
0.5	18742	31236	1.373134	2.066833	2.811594	0.119658	1.175631	13.875000	0.119658	1.616232	13.875000
0.6	18730	31408	1.189655	2.057401	2.895425	0.148148	1.144032	14.080000	0.148148	1.692053	14.080000
0.7	18730	31546	1.373134	2.057163	2.715556	0.145455	1.155447	13.875000	0.145455	1.786648	13.875000
0.8	18674	31252	1.434555	2.056744	2.892086	0.146789	1.160466	13.750000	0.146789	1.877489	13.750000
0.9	18536	30958	1.188841	2.056909	2.771429	0.144144	1.144618	14.840000	0.144144	1.965680	14.840000
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	18668	30972	1.220264	2.052872	3.141844	0.145455	1.135274	13.833333	0.145455	1.227034	13.833333
0.2	18396	31548	1.361386	2.053088	3.921788	0.144144	1.147674	14.521739	0.144144	1.328757	14.521739
0.3	18176	31374	1.089844	2.052387	2.791367	0.129630	1.128056	13.833333	0.129630	1.405355	13.833333
0.4	18558	31474	1.231111	2.053298	4.676259	0.100000	1.152509	14.269231	0.100000	1.512824	14.269231
0.5	18500	31534	1.301897	2.052393	2.946905	0.131783	1.140832	14.280000	0.131783	1.596612	14.280000
0.6	18524	31374	1.286232	2.052597	2.661871	0.121429	1.153910	14.041667	0.121429	1.693122	14.041667
0.7	18702	31736	1.424870	2.052920	3.253623	0.139344	1.179029	14.000000	0.139344	1.790753	14.000000
0.8	18608	31586	1.424870	2.052751	3.343195	0.131579	1.145446	13.920000	0.131579	1.871290	13.920000
0.9	18562	31392	1.312796	2.052651	2.777778	0.144144	1.133875	13.875000	0.144144	1.960693	13.875000
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	17310	33014	0.910072	1.557200	2.291925	0.124088	1.165599	10.708333	0.124088	1.204759	10.708333
0.2	17236	33028	1.114679	1.557638	1.925000	0.163265	1.138910	11.086957	0.163265	1.222566	11.086957
0.3	17416	33118	1.193396	1.557668	2.284810	0.128000	1.188439	12.269231	0.128000	1.299207	12.269231
0.4	17322	32786	1.141553	1.556945	2.430380	0.141667	1.139829	11.416667	0.141667	1.306635	11.416667
0.5	17410	32830	1.183099	1.557219	2.018750	0.163265	1.138520	10.750000	0.163265	1.347870	10.750000
0.6	17352	32858	1.162679	1.557244	2.325153	0.161616	1.168947	11.050000	0.161616	1.401885	11.050000
0.7	17208	32604	1.131222	1.557946	2.944785	0.159674	1.147668	10.541667	0.159674	1.434882	10.541667
0.8	17188	32990	0.776074	1.556988	2.735849	0.158879	1.155859	10.625000	0.158879	1.476782	10.625000
0.9	17512	32946	1.120370	1.556765	2.169581	0.155556	1.173277	12.916667	0.155556	1.518416	12.916667



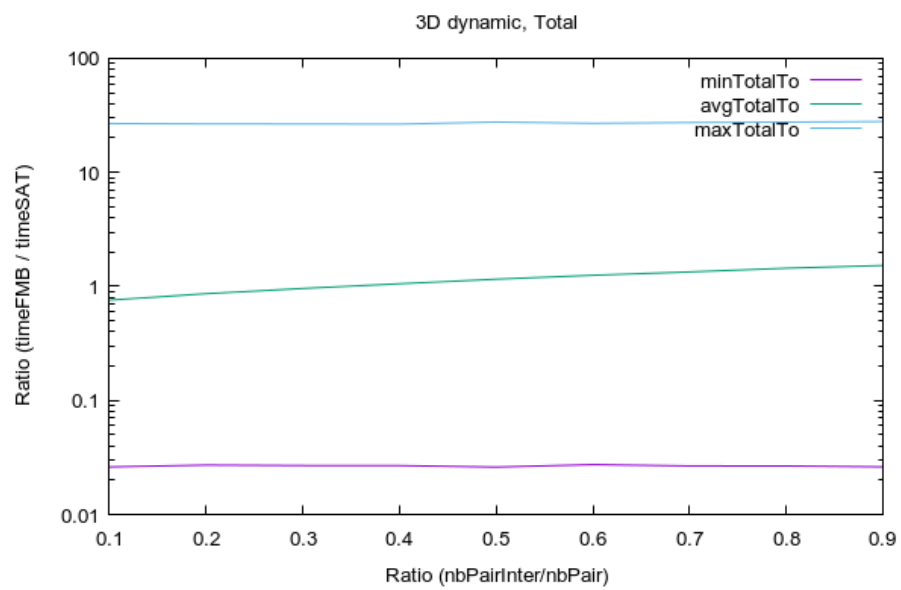


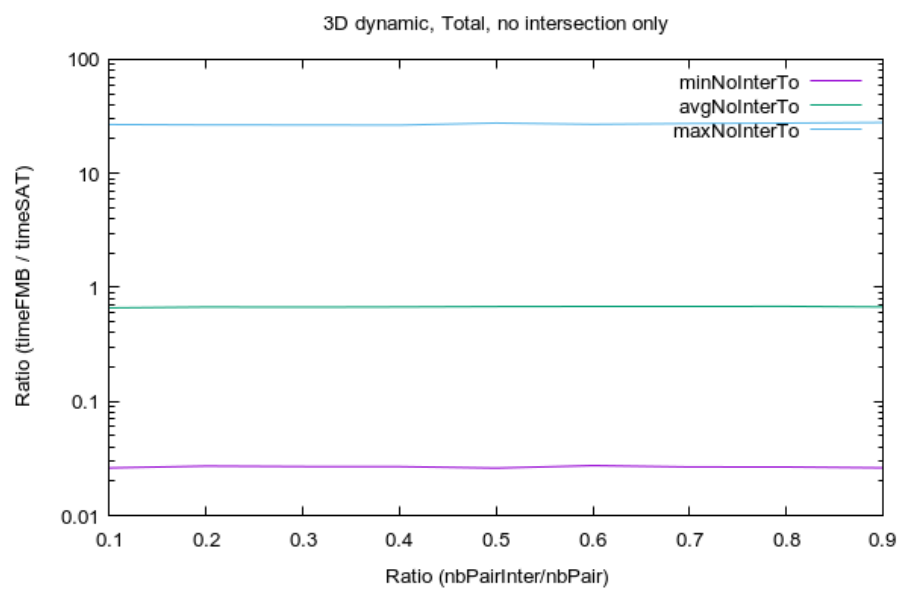
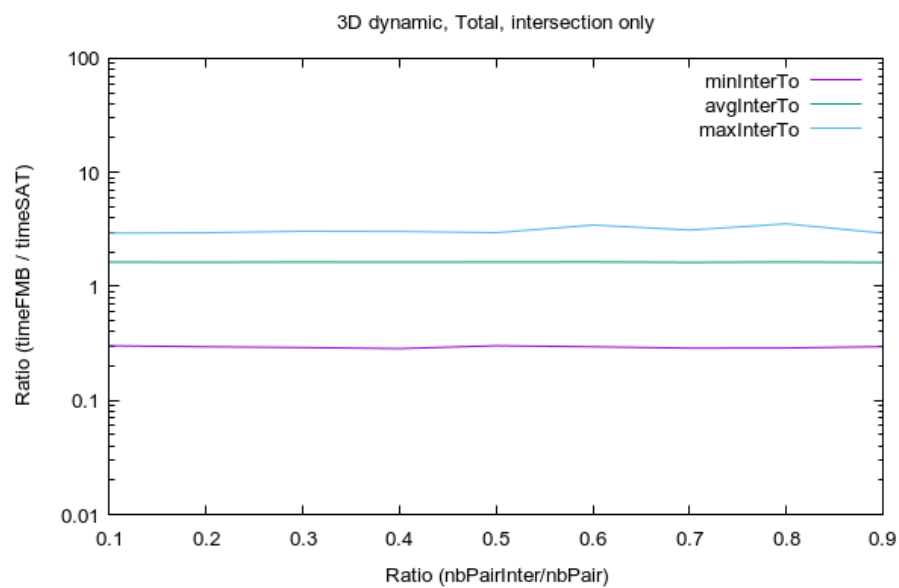


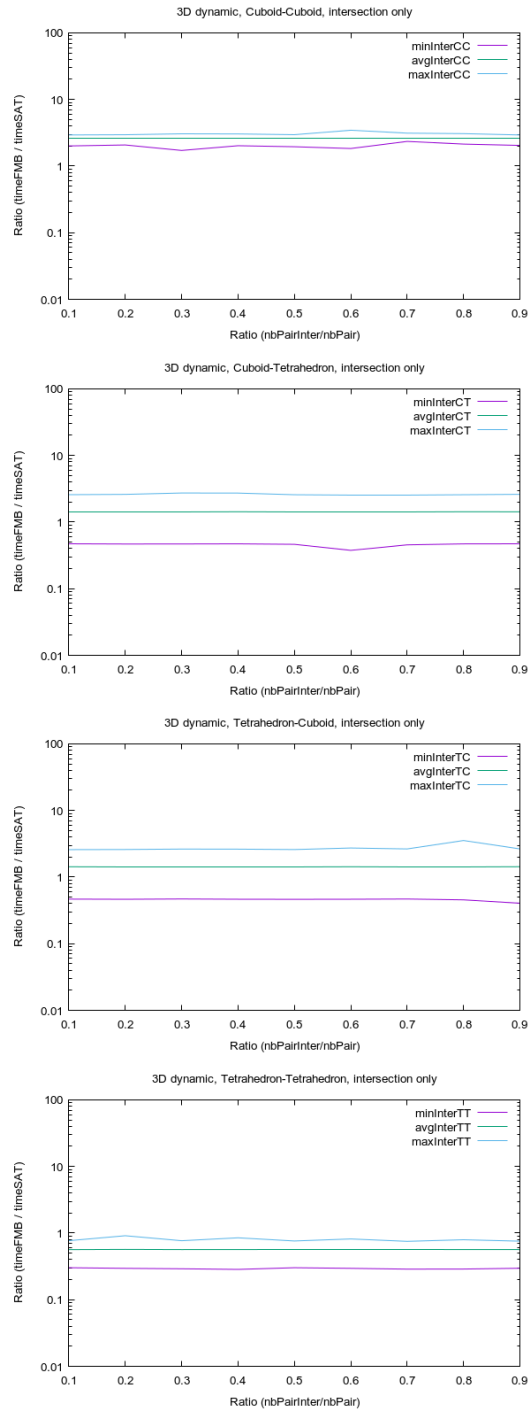


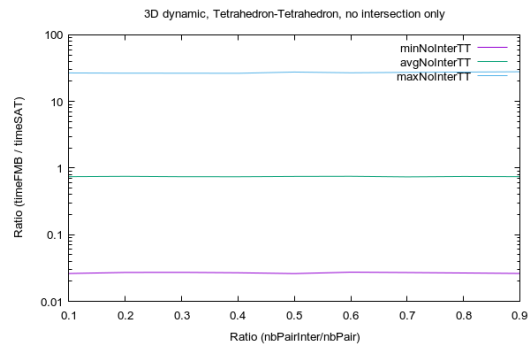
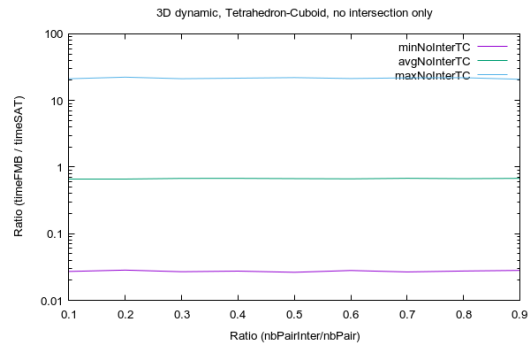
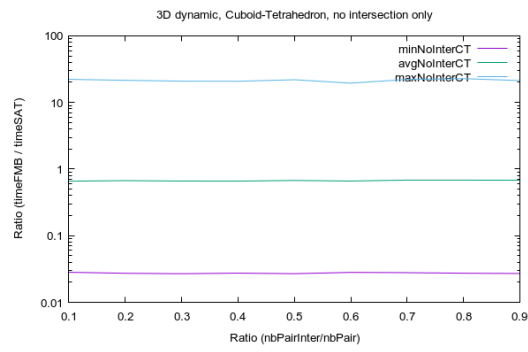
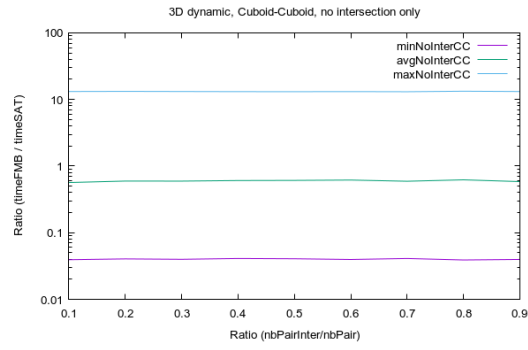
8.2.4 3D dynamic

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	52432	147568	0.302134	1.630643	2.930846	0.026139	0.661285	26.810811	0.026139	0.758221	26.810811
0.2	52180	147820	0.295630	1.626781	2.954587	0.027092	0.673350	26.611111	0.027092	0.864037	26.611111
0.3	52248	147752	0.291235	1.633945	3.050224	0.026877	0.670821	26.567568	0.026877	0.959758	26.567568
0.4	52180	147820	0.284750	1.628638	3.037367	0.026814	0.673728	26.486486	0.026814	1.055692	26.486486
0.5	52680	147318	0.301759	1.634563	2.958521	0.026054	0.678659	27.648649	0.026054	1.156611	27.648649
0.6	52366	147634	0.295597	1.635157	3.451040	0.027353	0.676351	26.921053	0.027353	1.251635	26.921053
0.7	52046	147954	0.287386	1.624435	3.121185	0.026657	0.675858	27.342857	0.026657	1.339882	27.342857
0.8	52548	147452	0.288344	1.632646	3.537377	0.026604	0.682564	27.600000	0.026604	1.442631	27.600000
0.9	52282	147718	0.296322	1.620332	2.929745	0.026154	0.674348	27.916667	0.026154	1.525733	27.916667
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	15938	33790	2.006906	2.637067	2.930846	0.039320	0.565685	13.157205	0.039320	0.773824	13.157205
0.2	15858	34214	2.074652	2.637926	2.954587	0.040393	0.595183	13.229075	0.040393	1.003731	13.229075
0.3	16060	34238	1.716621	2.636809	3.050224	0.039832	0.594179	13.169511	0.039832	1.206968	13.169511
0.4	15856	34026	2.022145	2.636592	3.037367	0.040975	0.606213	13.098655	0.040975	1.418364	13.098655
0.5	16376	33828	1.952131	2.637730	2.958521	0.040615	0.609411	13.084444	0.040615	1.623570	13.084444
0.6	16224	34008	1.835840	2.637114	3.451040	0.039615	0.616276	13.110619	0.039615	1.828779	13.110619
0.7	15714	34000	2.343877	2.637677	3.121185	0.040975	0.592244	13.064935	0.040975	2.024047	13.064935
0.8	16266	34058	2.140136	2.637085	3.066116	0.038947	0.621098	13.294372	0.038947	2.233888	13.294372
0.9	15644	34282	2.041491	2.637381	2.929745	0.039700	0.584295	13.162896	0.039700	2.432072	13.162896
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	13106	37006	0.472222	1.425827	2.579031	0.028239	0.658123	22.240000	0.028239	0.734893	22.240000
0.2	13032	37008	0.467997	1.425228	2.606891	0.027287	0.670448	21.490196	0.027287	0.821404	21.490196
0.3	13000	36934	0.468938	1.426846	2.731013	0.026877	0.660335	20.900000	0.026877	0.890639	20.900000
0.4	13238	36944	0.471399	1.427499	2.723305	0.027397	0.658867	20.780000	0.027397	0.966320	20.780000
0.5	13250	37232	0.463431	1.425816	2.574822	0.026899	0.674866	21.940000	0.026899	1.050341	21.940000
0.6	13092	36970	0.376242	1.425273	2.556701	0.028135	0.660268	19.500000	0.028135	1.119271	19.500000
0.7	13066	37330	0.455041	1.425779	2.558011	0.027846	0.683470	22.163265	0.027846	1.203086	22.163265
0.8	12872	36648	0.470270	1.427704	2.574265	0.027331	0.681742	22.795918	0.027331	1.278511	22.795918
0.9	13322	36466	0.471841	1.426264	2.610224	0.027049	0.680328	21.320000	0.027049	1.351670	21.320000
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	13410	37072	0.467119	1.427916	2.580315	0.027092	0.664325	21.13673	0.027092	0.740684	21.13673
0.2	13114	36722	0.464115	1.425397	2.591680	0.028409	0.664562	22.387755	0.028409	0.816721	22.387755
0.3	13200	36976	0.469051	1.425700	2.632970	0.026877	0.675806	21.215686	0.026877	0.900774	21.215686
0.4	13044	36636	0.464924	1.425422	2.625100	0.027441	0.677009	21.600000	0.027441	0.976374	21.600000
0.5	12746	36716	0.463398	1.426045	2.590657	0.026316	0.670671	22.040816	0.026316	1.046358	22.040816
0.6	12938	36656	0.464640	1.427113	2.731873	0.028030	0.665690	21.346939	0.028030	1.122544	21.346939
0.7	13136	36960	0.468041	1.426586	2.645929	0.026657	0.678530	21.840000	0.026657	1.202169	21.840000
0.8	13084	36816	0.464662	1.426393	3.537377	0.027508	0.669689	21.959184	0.027508	1.275484	21.959184
0.9	13042	36924	0.404858	1.429366	2.650508	0.028135	0.676746	20.780000	0.028135	1.354347	20.780000
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	9978	39700	0.302134	0.564551	0.769176	0.026139	0.742763	26.810811	0.026139	0.724942	26.810811
0.2	10176	39676	0.295530	0.568691	0.912080	0.027092	0.751215	26.611111	0.027092	0.714710	26.611111
0.3	9988	39604	0.291235	0.565655	0.767725	0.027200	0.747377	26.567568	0.027200	0.688913	26.567568
0.4	10042	40214	0.284750	0.566230	0.849043	0.026814	0.741519	26.486486	0.026814	0.671403	26.486486
0.5	10308	39542	0.301759	0.567024	0.760319	0.026054	0.748849	27.648649	0.026054	0.657957	27.648649
0.6	10112	40000	0.295597	0.565111	0.815406	0.027353	0.752061	26.921053	0.027353	0.640131	26.921053
0.7	10130	39664	0.287386	0.565454	0.751237	0.027006	0.732857	27.342857	0.027006	0.617181	27.342857
0.8	10326	40030	0.288344	0.566538	0.793594	0.026604	0.747468	27.600000	0.026604	0.602774	27.600000
0.9	10274	40046	0.296322	0.565406	0.756254	0.026154	0.743782	27.916667	0.026154	0.583244	27.916667









9 Comments about the qualification results

10 Conclusion

The validation proves that the FMB algorithm correctly identifies intersection of pairs of Frames in accordance with the results of the SAT algorithm.

The qualification shows that the FMB is 1.2 to 1.8 times slower than the SAT algorithm in the 2D dynamic case. However it is around 2 times faster in the 3D static case, and up to 1.25 times faster in 3D dynamic and up to 1.1 times faster in the 2D static case if the percentage of tested pairs in intersection is less than, respectively, around 40% and 25%.

On one given pair of Frame, the relative speed of the FMB algorithm varies widely, from around 20 times slower to 50 times faster. This is explained by the way the 2 algorithms works: they both make the assumption that the Frames are intersecting and run through a series of tests to try to prove it wrong. This leads to best cases and worst cases for both algorithm: a non intersecting detected right from the first test, or one detected by the last test. These best and worst cases are different for the two algorithms as the tests they performed are completely different. But in average, the FMB algorithm has the advantage for all but the 2D dynamic case.

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 the relative to that
    ++nbEdges;

}

// Loop on the frame's edges
for (int iEdge = nbEdges;
     iEdge--;) {

    // Get the current edge
    const double* edge =
        (iEdge == 3 ? relSpeed :
         (iEdge == 2 ?
          (frameEdgeType == FrameTetrahedron ? thirdEdge : relSpeed) :
          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 {

        // 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 =
        (iEdgeThat < 3 ?
         that->comp[iEdgeThat] :
         oppEdgesThat[iEdgeThat - 3]);

    for (int iEdgeTho = nbEdgesTho;
         iEdgeTho--;) {

        // Get the second edge
        const double* edgeTho =
            (iEdgeTho < 3 ?
             tho->comp[iEdgeTho] :
             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 =
        (iEdgeThat < 3 ?
         that->comp[iEdgeThat] :
         oppEdgesThat[iEdgeThat - 3]);

    for (int iEdgeTho = nbEdgesTho + 1;
         iEdgeTho--;) {

        // Get the second edge
        const double* edgeTho =
            (iEdgeTho == nbEdgesTho ?
             relSpeed :

```

```

        (iEdgeTho < 3 ?
         tho->comp[iEdgeTho] :
         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 -

run : run2D run2DTime run3D 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 -

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 plot2DTime plot3D plot3DTime

cleanPlot:
    rm -f Results/*.png

plot2D:
    cd Results; gnuplot qualification2D.gnu; cd -

plot2DTime:
    cd Results; gnuplot qualification2DTime.gnu; cd -

plot3D:
    cd Results; gnuplot qualification3D.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

```

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
```

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

```

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

```

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

```

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
==7017== Memcheck, a memory error detector
==7017== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7017== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7017== Command: ./unitTests
==7017==
--7017-- Valgrind options:
--7017--      -v
--7017--      --track-origins=yes
--7017--      --leak-check=full
--7017--      --gen-suppressions=yes
--7017--      --show-leak-kinds=all
--7017-- Contents of /proc/version:
--7017--      Linux version 5.3.0-26-generic (buildd@lgw01-amd64-039) (gcc
version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #28~18.04.1-Ubuntu SMP
Wed Dec 18 16:40:14 UTC 2019
--7017--
--7017-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
avx-avx2-bmi
--7017-- Page sizes: currently 4096, max supported 4096
--7017-- Valgrind library directory: /usr/lib/valgrind
--7017-- Reading syms from /home/bayashi/GitHub/FMB/2D/unitTests
--7017-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--7017--      Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--7017--      .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--7017--      Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--7017--      .. CRC is valid
--7017-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--7017--      Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--7017--      .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--7017--      object doesn't have a symbol table
--7017--      object doesn't have a dynamic symbol table

```

```

--7017-- Scheduler: using generic scheduler lock implementation.
--7017-- Reading suppressions file: /usr/lib/valgrind/default.supp
==7017== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-7017-
by-bayashi-on-???
==7017== embedded gdbserver: writing to /tmp/vgdb-pipe-to-vgdb-from-7017-
by-bayashi-on-???
==7017== embedded gdbserver: shared mem /tmp/vgdb-pipe-shared-mem-vgdb
-7017-by-bayashi-on-???
==7017==
==7017== TO CONTROL THIS PROCESS USING vgdb (which you probably
==7017== don't want to do, unless you know exactly what you're doing,
==7017== or are doing some strange experiment):
==7017== /usr/lib/valgrind/../../bin/vgdb --pid=7017 ...command...
==7017==
==7017== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==7017== /path/to/gdb ./unitTests
==7017== and then give GDB the following command
==7017== target remote | /usr/lib/valgrind/../../bin/vgdb --pid=7017
==7017== --pid is optional if only one valgrind process is running
==7017==
--7017-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
x580608c1 (???)
--7017-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
x580608db (???)
--7017-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--7017-- Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--7017-- .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--7017-- object doesn't have a symbol table
--7017-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--7017-- Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--7017-- .. CRC mismatch (computed f893b962 wanted 95ee359e)
--7017-- object doesn't have a symbol table
==7017== WARNING: new redirection conflicts with existing -- ignoring it
--7017-- old: 0x0401f2f0 (strlen ) R-> (0000.0) 0x580608c1
???
--7017-- new: 0x0401f2f0 (strlen ) R-> (2007.0) 0x04c32db0
strlen
--7017-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
x4c33ee0 (strcmp)
--7017-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
x4c374f0 (mempcpy)
--7017-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--7017-- Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--7017-- .. CRC mismatch (computed b1c74187 wanted 042cc048)
--7017-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--7017-- .. CRC is valid
--7017-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)

```

```

--7017-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--7017-- REDIR: 0x4fca3c0 (libc.so.6:__strchr_avx2) redirected to 0x4c32730
(rindex)
--7017-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--7017-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--7017-- REDIR: 0x4fcaab0 (libc.so.6:__memcpy_avx_unaligned_erms)
redirected to 0x4c37130 (memcpy)
--7017-- 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.
--7017-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==7017==
==7017== HEAP SUMMARY:
==7017==     in use at exit: 0 bytes in 0 blocks
==7017==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==7017==
==7017== All heap blocks were freed -- no leaks are possible
==7017==
==7017== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==7017== 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
==7020== Memcheck, a memory error detector
==7020== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7020== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7020== Command: ./unitTests
==7020==
--7020-- Valgrind options:
--7020--     -v
--7020--     --track-origins=yes
--7020--     --leak-check=full
--7020--     --gen-suppressions=yes
--7020--     --show-leak-kinds=all
--7020-- Contents of /proc/version:
--7020--   Linux version 5.3.0-26-generic (buildd@lgw01-amd64-039) (gcc
        version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #28~18.04.1-Ubuntu SMP
        Wed Dec 18 16:40:14 UTC 2019
--7020--
--7020-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
        avx-avx2-bmi
--7020-- Page sizes: currently 4096, max supported 4096
--7020-- Valgrind library directory: /usr/lib/valgrind
--7020-- Reading syms from /home/bayashi/GitHub/FMB/2DTime/unitTests
--7020-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--7020--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--7020--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--7020--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--7020--   .. CRC is valid
--7020-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--7020--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--7020--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--7020--   object doesn't have a symbol table
--7020--   object doesn't have a dynamic symbol table
--7020-- Scheduler: using generic scheduler lock implementation.
--7020-- Reading suppressions file: /usr/lib/valgrind/default.supp
==7020== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-7020-
        by-bayashi-on-???
==7020== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-7020-
        by-bayashi-on-???
==7020== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
        -7020-by-bayashi-on-???
==7020==
==7020== TO CONTROL THIS PROCESS USING vgdb (which you probably
==7020== don't want to do, unless you know exactly what you're doing,
==7020== or are doing some strange experiment):
==7020==   /usr/lib/valgrind/../../bin/vgdb --pid=7020 ...command...
==7020==

```

```

==7020== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==7020== /path/to/gdb ./unitTests
==7020== and then give GDB the following command
==7020== target remote | /usr/lib/valgrind/../../bin/vgdb --pid=7020
==7020== --pid is optional if only one valgrind process is running
==7020==
--7020-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
x580608c1 (???)
--7020-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
x580608db (???)
--7020-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--7020-- Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--7020-- .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--7020-- object doesn't have a symbol table
--7020-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--7020-- Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--7020-- .. CRC mismatch (computed f893b962 wanted 95ee359e)
--7020-- object doesn't have a symbol table
==7020== WARNING: new redirection conflicts with existing -- ignoring it
--7020-- old: 0x0401f2f0 (strlen ) R-> (0000.0) 0x580608c1
???
--7020-- new: 0x0401f2f0 (strlen ) R-> (2007.0) 0x04c32db0
strlen
--7020-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
x4c33ee0 (strcmp)
--7020-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
x4c374f0 (mempcpy)
--7020-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--7020-- Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--7020-- .. CRC mismatch (computed b1c74187 wanted 042cc048)
--7020-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--7020-- .. CRC is valid
--7020-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)

```

```

--7020-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--7020-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
(rindex)
--7020-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--7020-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--7020-- REDIR: 0x4fcaab0 (libc.so.6:__memcpy_avx_unaligned_erms)
redirected to 0x4c37130 (memcpy)
--7020-- 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.
--7020-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==7020==
==7020== HEAP SUMMARY:
==7020==       in use at exit: 0 bytes in 0 blocks
==7020==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==7020==
==7020== All heap blocks were freed -- no leaks are possible
==7020==
==7020== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==7020== 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
==7023== Memcheck, a memory error detector
==7023== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7023== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7023== Command: ./unitTests
==7023==
--7023-- Valgrind options:
--7023--   -v
--7023--   --track-origins=yes
--7023--   --leak-check=full
--7023--   --gen-suppressions=yes
--7023--   --show-leak-kinds=all
--7023-- Contents of /proc/version:
--7023--   Linux version 5.3.0-26-generic (buildd@lgw01-amd64-039) (gcc
version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #28~18.04.1-Ubuntu SMP
Wed Dec 18 16:40:14 UTC 2019
--7023--
--7023-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
avx-avx2-bmi
--7023-- Page sizes: currently 4096, max supported 4096
--7023-- Valgrind library directory: /usr/lib/valgrind
--7023-- Reading syms from /home/bayashi/GitHub/FMB/3D/unitTests
--7023-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--7023--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--7023--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--7023--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--7023--   .. CRC is valid
--7023-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--7023--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--7023--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--7023--   object doesn't have a symbol table
--7023--   object doesn't have a dynamic symbol table
--7023-- Scheduler: using generic scheduler lock implementation.
--7023-- Reading suppressions file: /usr/lib/valgrind/default.supp
==7023== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-7023-
by-bayashi-on-???
==7023== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-7023-
by-bayashi-on-???

```

```

==7023== embedded gdbserver: shared mem    /tmp/vgdb-pipe-shared-mem-vgdb
          -7023-by-bayashi-on-???
==7023==
==7023== TO CONTROL THIS PROCESS USING vgdb (which you probably
==7023== don't want to do, unless you know exactly what you're doing,
==7023== or are doing some strange experiment):
==7023==    /usr/lib/valgrind/../../bin/vgdb --pid=7023 ...command...
==7023==
==7023== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==7023==    /path/to/gdb ./unitTests
==7023== and then give GDB the following command
==7023==    target remote | /usr/lib/valgrind/../../bin/vgdb --pid=7023
==7023== --pid is optional if only one valgrind process is running
==7023==
--7023-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
          x580608c1 (???)
--7023-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
          x580608db (???)
--7023-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--7023--    Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--7023--    .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--7023--    object doesn't have a symbol table
--7023-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--7023--    Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--7023--    .. CRC mismatch (computed f893b962 wanted 95ee359e)
--7023--    object doesn't have a symbol table
==7023== WARNING: new redirection conflicts with existing -- ignoring it
--7023--    old: 0x0401f2f0 (strlen
          ???
          ) R-> (0000.0) 0x580608c1
--7023--    new: 0x0401f2f0 (strlen
          strlen
          ) R-> (2007.0) 0x04c32db0
--7023-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
          x4c33ee0 (strcmp)
--7023-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
          x4c374f0 (mempcpy)
--7023-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--7023--    Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--7023--    .. CRC mismatch (computed b1c74187 wanted 042cc048)
--7023--    Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--7023--    .. CRC is valid
--7023-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
          _vgnU_ifunc_wrapper)

```

```

--7023-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--7023-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
(rindex)
--7023-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--7023-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--7023-- REDIR: 0x4fcaab0 (libc.so.6:__memcpy_avx_unaligned_erms)
redirected to 0x4c37130 (memcpy)
--7023-- 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

```



```

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.
--7023-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==7023==
==7023== HEAP SUMMARY:
==7023==       in use at exit: 0 bytes in 0 blocks
==7023==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==7023==
==7023== All heap blocks were freed -- no leaks are possible
==7023==
==7023== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==7023== 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
==7026== Memcheck, a memory error detector
==7026== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7026== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7026== Command: ./unitTests
==7026==
--7026-- Valgrind options:
--7026--     -v
--7026--     --track-origins=yes
--7026--     --leak-check=full
--7026--     --gen-suppressions=yes
--7026--     --show-leak-kinds=all
--7026-- Contents of /proc/version:
--7026--     Linux version 5.3.0-26-generic (build@lgw01-amd64-039) (gcc
        version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #28~18.04.1-Ubuntu SMP
        Wed Dec 18 16:40:14 UTC 2019
--7026--
--7026-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
        avx-avx2-bmi
--7026-- Page sizes: currently 4096, max supported 4096
--7026-- Valgrind library directory: /usr/lib/valgrind
--7026-- Reading syms from /home/bayashi/GitHub/FMB/3DTime/unitTests
--7026-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--7026--     Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--7026--     .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--7026--     Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--7026--     .. CRC is valid
--7026-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--7026--     Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--7026--     .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--7026--     object doesn't have a symbol table
--7026--     object doesn't have a dynamic symbol table
--7026-- Scheduler: using generic scheduler lock implementation.
--7026-- Reading suppressions file: /usr/lib/valgrind/default.supp
==7026== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-7026-
        by-bayashi-on-???
==7026== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-7026-
        by-bayashi-on-???
==7026== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
        -7026-by-bayashi-on-???
==7026==
==7026== TO CONTROL THIS PROCESS USING vgdb (which you probably
==7026== don't want to do, unless you know exactly what you're doing,
==7026== or are doing some strange experiment):
==7026==     /usr/lib/valgrind/../../bin/vgdb --pid=7026 ...command...
==7026==
==7026== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==7026==     /path/to/gdb ./unitTests
==7026== and then give GDB the following command
==7026==     target remote | /usr/lib/valgrind/../../bin/vgdb --pid=7026
==7026== --pid is optional if only one valgrind process is running
==7026==
--7026-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
        x580608c1 (???)
--7026-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
        x580608db (???)
--7026-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--7026--     Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--7026--     .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--7026--     object doesn't have a symbol table
--7026-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
        so

```

```

--7026--    Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--7026--    .. CRC mismatch (computed f893b962 wanted 95ee359e)
--7026--    object doesn't have a symbol table
==7026== WARNING: new redirection conflicts with existing -- ignoring it
--7026--    old: 0x0401f2f0 (strlen                ) R-> (0000.0) 0x580608c1
    ???
--7026--    new: 0x0401f2f0 (strlen                ) R-> (2007.0) 0x04c32db0
    strlen
--7026-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
    x4c33ee0 (strcmp)
--7026-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
    x4c374f0 (mempcpy)
--7026-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--7026--    Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--7026--    .. CRC mismatch (computed b1c74187 wanted 042cc048)
--7026--    Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--7026--    .. CRC is valid
--7026-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
    x4a2a6e0 (_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
    _vgnU_ifunc_wrapper)

```



```

--7026-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--7026-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
(rindex)
--7026-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--7026-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--7026-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
redirected to 0x4c37130 (mempcpy)
--7026-- 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.
--7026-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==7026==
==7026== HEAP SUMMARY:
==7026==      in use at exit: 0 bytes in 0 blocks
==7026==    total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==7026==
==7026== All heap blocks were freed -- no leaks are possible
==7026==
==7026== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==7026== 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'

```

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.