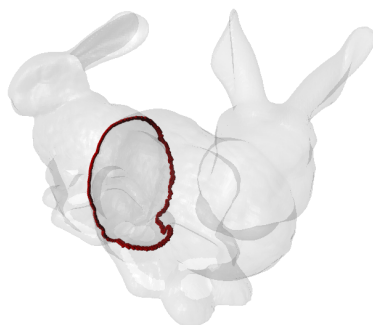


The FMB Algorithm

Pascal Baillehache
bayashipascal@gmail.com

May 13, 2020



Abstract

This paper introduces how to perform intersection detection of pair of static/dynamic cuboid/tetrahedron in 2D/3D by using the Fourier-Motzkin elimination method.

It includes the mathematical definition of the problem, its mathematical resolution with the Fourier-Motzkin elimination method, the resulting algorithm and its implementation in C, and its validation and qualification against the SAT algorithm. Results are commented and show that the FMB algorithm can be in average up to 4 times faster than the SAT algorithm.

Contents

1	The problem as a system of linear inequations	5
1.1	Notations and definitions	5
1.2	Static case	5
1.3	Dynamic case	10
2	Resolution of the problem by Fourier-Motzkin method	14
2.1	The Fourier-Motzkin elimination method	14
2.2	Application of the Fourier-Motzkin method to the intersection problem	16
2.3	About the size of the system of linear inequations	16
3	Algorithms of the solution	18
3.1	2D static	19
3.2	3D static	25
3.3	2D dynamic	32
3.4	3D dynamic	39
3.5	Generic algorithm	48
4	Implementation of the algorithms in C	50
4.1	Frames	50
4.1.1	Header	50
4.1.2	Body	54
4.2	FMB	81
4.2.1	2D static	81
4.2.2	3D static	90
4.2.3	2D dynamic	102
4.2.4	3D dynamic	112
5	Minimal example of use	124
5.1	2D static	124
5.2	3D static	126
5.3	2D dynamic	128
5.4	3D dynamic	130
6	Unit tests	132
6.1	Code	132
6.1.1	2D static	132
6.1.2	3D static	144
6.1.3	2D dynamic	153
6.1.4	3D dynamic	158

6.2	Results	164
6.2.1	2D static	164
6.2.2	3D static	168
6.2.3	2D dynamic	171
6.2.4	3D dynamic	173
7	Validation against SAT	175
7.1	Code	175
7.1.1	2D static	175
7.1.2	3D static	179
7.1.3	2D dynamic	182
7.1.4	3D dynamic	186
7.2	Results	190
7.2.1	Failures	190
7.2.2	2D static	191
7.2.3	2D dynamic	191
7.2.4	3D static	191
7.2.5	3D dynamic	191
8	Qualification against SAT	191
8.1	Code	191
8.1.1	2D static	191
8.1.2	3D static	206
8.1.3	2D dynamic	221
8.1.4	3D dynamic	234
8.2	Results	248
8.2.1	2D static	249
8.2.2	2D static (near case only)	254
8.2.3	3D static	259
8.2.4	3D static (near case only)	264
8.2.5	2D dynamic	269
8.2.6	3D dynamic	274
9	Comments about the qualification results	279
10	Conclusion	283
11	Annex	284
11.1	Runtime environment	284
11.2	SAT implementation	286
11.2.1	Header	286

11.2.2	Body	287
11.3	Makefile	313
11.3.1	2D static	316
11.3.2	3D static	317
11.3.3	2D dynamic	317
11.3.4	3D dynamic	318
11.3.5	Doc	319
11.4	Dynamic analysis	319
11.5	Static analysis	340
11.6	Versions of this paper	341

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, parallelepipeds and triangles in 3D, and cuboids and tetrahedrons in 3D.

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

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

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

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

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

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

<https://github.com/BayashiPascal/FMB>

Please, make sure you have the most recent version of this paper by referring to the master branch of this repository.

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:

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

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

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

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

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

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

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

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:

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

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

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

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

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

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

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

If one wants to consider faces (parallelepipeds and triangles) in 3D, by assuming the third component of cuboid and tetrahedron is the normal of the faces, she can simply modifies the inequations above to express the constraint $X_2 = 0$ and obtain the relevant systems as follow.

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

$$\left\{ \begin{array}{l} [X]_0 \leq 1.0 \\ \dots \\ [X]_{D-2} \leq 1.0 \\ -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-2} \leq 0.0 \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,0} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-2} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-2} \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-1} \cdot [X]_i \leq -[O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,0} \cdot [X]_i \leq [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-1} \cdot [X]_i \leq [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \end{array} \right. \quad (17)$$

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

$$\left\{ \begin{array}{l} -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-2} \leq 0.0 \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,0} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-2} \cdot [X]_i \leq 1.0 - [O_{\mathbb{B}_{\mathbb{A}}}]_{D-2} \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-1} \cdot [X]_i \leq -[O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,0} \cdot [X]_i \leq [O_{\mathbb{B}_{\mathbb{A}}}]_0 \\ \dots \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_{\mathbb{A}}}]_{i,D-1} \cdot [X]_i \leq [O_{\mathbb{B}_{\mathbb{A}}}]_{D-1} \\ \sum_{i=0}^{D-2} [X]_i \leq 1.0 \end{array} \right. \quad (18)$$

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

$$\left\{ \begin{array}{l} [X]_0 \leq 1.0 \\ \dots \\ [X]_{D-2} \leq 1.0 \\ -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-2} \leq 0.0 \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,0} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,D-1} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{j=0}^{D-2} \left(\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,j} \cdot [X]_i \right) \leq 1.0 - \sum_{j=0}^{D-2} [O_{\mathbb{B}_A}]_j \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,D-1} \cdot [X]_i \leq -[O_{\mathbb{B}_A}]_{D-1} \end{array} \right. \quad (19)$$

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

$$\left\{ \begin{array}{l} -[X]_0 \leq 0.0 \\ \dots \\ -[X]_{D-2} \leq 0.0 \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,0} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_0 \\ \dots \\ -\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,D-1} \cdot [X]_i \leq [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{i=0}^{D-2} [X]_i \leq 1.0 \\ \sum_{j=0}^{D-2} \left(\sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,j} \cdot [X]_i \right) \leq 1.0 - \sum_{j=0}^{D-2} [O_{\mathbb{B}_A}]_j \\ \sum_{i=0}^{D-2} [C_{\mathbb{B}_A}]_{i,D-1} \cdot [X]_i \leq -[O_{\mathbb{B}_A}]_{D-1} \end{array} \right. \quad (20)$$

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:

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

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

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:

$$\mathbb{A} = \left\{ \begin{array}{l} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ \vec{O}_\mathbb{A} + C_\mathbb{A} \cdot \vec{X} + \vec{V}_\mathbb{A} \cdot t \end{array} \right\} \quad (23)$$

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

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

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

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

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

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

$$\mathbb{A}_\mathbb{A} = \left\{ \vec{X} \in [0.0, 1.0]^D \right\} \quad (27)$$

and for a tetrahedron:

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

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

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \cap [0.0, 1.0]^D \end{array} \right\} \quad (29)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \cap [0.0, 1.0]^D \end{array} \right\} \quad (30)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \right]_i \leq 1.0 \end{array} \right\} \quad (31)$$

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

$$\mathbb{A}_{\mathbb{A}} \cap \mathbb{B}_{\mathbb{A}} = \left\{ \begin{array}{c} \vec{X} \in [0.0, 1.0]^D \\ t \in [0.0, 1.0] \\ \sum_{i=0}^{D-1} [X]_i \leq 1.0 \\ C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \cap [0.0, 1.0]^D \\ \sum_{i=0}^{D-1} \left[C_{\mathbb{A}}^{-1} \cdot (\vec{O}_{\mathbb{B}} - \vec{O}_{\mathbb{A}} + C_{\mathbb{B}} \cdot \vec{X} + (\vec{V}_{\mathbb{B}} - \vec{V}_{\mathbb{A}}) \cdot t) \right]_i \leq 1.0 \end{array} \right\} \quad (32)$$

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:

$$\left\{ \begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ [X]_0 & \leq & 1.0 \\ \dots & & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ [V_{\mathbb{B}_A}]_0 \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,0} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_A}]_0 \\ \dots & & \\ [V_{\mathbb{B}_A}]_{D-1} \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,D-1} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_A}]_{D-1} \\ -[V_{\mathbb{B}_A}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,0} [X]_i & \leq & [O_{\mathbb{B}_A}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_A}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,D-1} [X]_i & \leq & [O_{\mathbb{B}_A}]_{D-1} \end{array} \right. \quad (33)$$

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

$$\left\{ \begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ [V_{\mathbb{B}_A}]_0 \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,0} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_A}]_0 \\ \dots & & \\ [V_{\mathbb{B}_A}]_{D-1} \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,D-1} [X]_i & \leq & 1.0 - [O_{\mathbb{B}_A}]_{D-1} \\ -[V_{\mathbb{B}_A}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,0} [X]_i & \leq & [O_{\mathbb{B}_A}]_0 \\ \dots & & \\ -[V_{\mathbb{B}_A}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{B}_A}]_{i,D-1} [X]_i & \leq & [O_{\mathbb{B}_A}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq & 1.0 \end{array} \right. \quad (34)$$

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

$$\left\{ \begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ [X]_0 & \leq & 1.0 \\ \dots & & \\ [X]_{D-1} & \leq & 1.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -[V_{\mathbb{A}}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,0} [X]_i & \leq & [O_{\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{A}}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,D-1} [X]_i & \leq & [O_{\mathbb{A}}]_{D-1} \\ \sum_{j=0}^{D-1} \left([V_{\mathbb{A}}]_j \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,j} [X]_i \right) & \leq & 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{A}}]_j \end{array} \right. \quad (35)$$

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

$$\left\{ \begin{array}{rcl} t & \leq & 1.0 \\ -t & \leq & 0.0 \\ -[X]_0 & \leq & 0.0 \\ \dots & & \\ -[X]_{D-1} & \leq & 0.0 \\ -[V_{\mathbb{A}}]_0 \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,0} [X]_i & \leq & [O_{\mathbb{A}}]_0 \\ \dots & & \\ -[V_{\mathbb{A}}]_{D-1} \cdot t - \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,D-1} [X]_i & \leq & [O_{\mathbb{A}}]_{D-1} \\ \sum_{i=0}^{D-1} [X]_i & \leq & 1.0 \\ \sum_{j=0}^{D-1} \left([V_{\mathbb{A}}]_j \cdot t + \sum_{i=0}^{D-1} [C_{\mathbb{A}}]_{i,j} [X]_i \right) & \leq & 1.0 - \sum_{j=0}^{D-1} [O_{\mathbb{A}}]_j \end{array} \right. \quad (36)$$

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

$$\left\{ \begin{array}{cccc} a_{11}.x_1 & +a_{12}.x_2 & +\cdots & +a_{1n}.x_n & \leq b_1 \\ a_{21}.x_1 & +a_{22}.x_2 & +\cdots & +a_{2n}.x_n & \leq b_2 \\ & & \vdots & & \\ a_{m1}.x_1 & +a_{m2}.x_2 & +\cdots & +a_{mn}.x_n & \leq b_m \end{array} \right. \quad (37)$$

with

$$\begin{aligned} i &\in 1, 2, \dots, m \\ j &\in 1, 2, \dots, n \\ x_i &\in \mathbb{R} \\ a_{ij} &\in \mathbb{R} \\ b_j &\in \mathbb{R} \end{aligned} \quad (38)$$

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

$$\left\{ \begin{array}{ll} x_1 & +a'_{i2}.x_2 +\cdots +a'_{in}.x_n \leq b'_i \quad (i \in \mathcal{I}_+) \\ & a_{i2}.x_2 +\cdots +a_{in}.x_n \leq b_i \quad (i \in \mathcal{I}_0) \\ -x_1 & +a'_{i2}.x_2 +\cdots +a'_{in}.x_n \leq b'_i \quad (i \in \mathcal{I}_-) \end{array} \right. \quad (39)$$

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

$$\left\{ \begin{array}{ll} \sum_{j=2}^n ((a'_{kj} + a'_{lj}).x_j) \leq b'_k + b'_l & (k \in \mathcal{I}_+, l \in \mathcal{I}_-) \\ \sum_{j=2}^n (a_{ij}.x_j) \leq b_i & i \in \mathcal{I}_0 \end{array} \right. \quad (40)$$

and

$$\max_{l \in \mathcal{I}_-} \left(\sum_{j=2}^n (a'_{lj}.x_j) - b'_l \right) \leq x_1 \leq \min_{k \in \mathcal{I}_+} (b'_k - \sum_{j=2}^n (a'_{kj}.x_j)) \quad (41)$$

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

$$\max_{l \in \mathcal{I}_-'''} (-b_l''') \leq x_n \leq \min_{k \in \mathcal{I}_+'''} (b_k''') \quad (42)$$

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:

$$\vec{S}' = \vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{S} \quad (43)$$

$$(\vec{S}', t) = \left(\vec{O}_{\mathbb{B}} + C_{\mathbb{B}} \cdot \vec{S} + \vec{V}_{\mathbb{B}} \cdot t, t \right) \quad (44)$$

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

$$Y < \sum_{i \in I^-} a_i \quad (45)$$

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 (39) 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:

$$n_- + n_+ + n_0 = N \quad (46)$$

and

$$n_-.n_+ + n_0 = N' \quad (47)$$

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

$$n_- + n_+ = K \quad (48)$$

then,

$$n_-.n_+ = n_-.(K - n_-) \quad (49)$$

then,

$$n_-.n_+ = K.n_- - n_-^2 \quad (50)$$

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

$$n_-.n_+ \leq K^2/2 - K^2/4 \quad (51)$$

or,

$$n_-.n_+ \leq K^2/4 \quad (52)$$

and putting back the definition of K

$$n_-.n_+ \leq (N - n_0)^2/4 \quad (53)$$

which is also

$$n_-.n_+ \leq N^2/4 \quad (54)$$

From (47) we get,

$$N' \leq N^2/4 + n_0 \quad (55)$$

and finally,

$$N' \leq N^2/4 + N \quad (56)$$

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]

```

```

        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);
    AAB2DPrint(bdgBox2D)
ELSE
    PRINT "No intersection."
END IF

```

3.2 3D static

```

ENUM FrameType
    FrameCuboid,
    FrameTetrahedron
END ENUM

STRUCT AAB3D
    // 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]
    AAB3D 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 AAB2DTimePrint(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

```

3.5 Generic algorithm

In this subsection I give the generic algorithm valid for all cases and dimensions.

DIM is the space dimension

```

STRUCT Frame
  FLOAT orig[DIM]
  FLOAT comp[DIM][DIM] <- comp[i-th component][j-th axis]
  FLOAT invComp[DIM][DIM] <- inverse matrix of comp
  IF DYNAMIC CASE
    FLOAT speed[DIM]

FUNCTION FMB(Frame A, Frame B)
  Frame Ba := PROJ(A, B)
  INT nbVars := DIM
  IF DYNAMIC CASE
    nbVars := DIM + 1
  FLOAT M[][nbVars]
  FLOAT Y[]
  FOR j IN 0..DIM-1
    FOR i IN 0..DIM-1
      M[j][i] := -Ba.comp[i][j]
    IF DYNAMIC CASE
      M[j][DIM] := -Ba.speed[j]
  Y[j] := Ba.orig[j]
  IF Y[j] < SUM OF NEGATIVE VALUES IN M[j][0..nbVars-1]
    A AND B ARE NOT INTERSECTING, STOP
  nbRows := DIM
  IF A IS A CUBOID
    FOR j IN 0..DIM-1
      FOR i IN 0..DIM-1
        M[nbRows][i] := Ba.comp[i][j]
      IF DYNAMIC CASE
        M[nbRows][DIM] := Ba.speed[j]
  Y[nbRows] := 1 - Ba.orig[j]
  IF Y[nbRows] <
    SUM OF NEGATIVE VALUES IN M[nbRows][0..nbVars-1]
    A AND B ARE NOT INTERSECTING, STOP
  nbRows := nbRows + 1
  ELSE
    FOR i IN 0..DIM-1
      M[nbRows][i] := SUM OF Ba.comp[i][0..DIM-1]
    IF DYNAMIC CASE
      M[nbRows][DIM] := SUM OF Ba.speed[0..DIM-1]
  Y[nbRows] = 1 - (SUM OF Ba.orig[0..DIM-1])
  IF Y[nbRows] < SUM OF NEGATIVE VALUES IN M[nbRows][0..nbVars-1]
    A AND B ARE NOT INTERSECTING, STOP
  nbRows := nbRows + 1
  IF B IS A CUBOID
    FOR j IN 0..DIM-1
      FOR i IN 0..nbVars-1
        IF i = j
          M[nbRows][i] := 1
        ELSE

```



```

        M[nbRows][i] := 0
        Y[nbRows] := 1
        nbRows := nbRows + 1
ELSE
    FOR i IN 0..DIM-1
        M[nbRows][0] := 1
        IF DYNAMIC CASE
            M[nbRows][DIM] := 0
            Y[nbRows] := 1
            nbRows := nbRows + 1
    FOR j IN 0..DIM-1
        FOR i IN 0..nbVars-1
            IF i = j
                M[nbRows][i] := -1
            ELSE
                M[nbRows][i] := 0
            Y[nbRows] := 0
            nbRows := nbRows + 1
    IF DYNAMIC CASE
        FOR i IN 0..nbVars-1
            IF i = nbVars - 1
                M[nbRows][i] := 1
                M[nbRows+1][i] := -1
            ELSE
                M[nbRows][i] := 0
                M[nbRows+1][i] := 0
            Y[nbRows] := 1
            Y[nbRows+1] := 0
            nbRows := nbRows + 2
    FOR i IN 0..nbVars-2
        M, Y, nbRows := ELIM_VAR(M, Y, nbVars - i, nbRows)
    FLOAT bounds[2] := [0, 1]
    FOR i IN 0..nbRows-1
        IF M[i][0] > 0
            y := Y[i] / M[i][0]
            IF bounds[1] > y
                bounds[1] := y
        ELSE IF M[i][0] < 0
            y := Y[i] / M[i][0]
            IF bounds[0] < y
                bounds[0] := y
    IF bounds[0] >= bounds[1]
        A AND B ARE NOT INTERSECTING, STOP
    ELSE
        A AND B ARE INTERSECTING, STOP

FUNCTION PROJ(Frame A, Frame B)
    Frame Ba
    FLOAT v[DIM]
    IF NOT ALREADY COMPUTED
        A.invComp := INVERSE MATRIX OF A.comp
    IF DYNAMIC CASE
        FLOAT s[DIM]
        FOR i IN 0..DIM-1
            v[i] := B.orig[i] - A.orig[i]
            IF DYNAMIC CASE
                s[i] := B.speed[i] - A.speed[i]
        FOR i IN 0..DIM-1
            Ba.orig[i] := 0
            IF DYNAMIC CASE
                Ba.speed[i] := 0
        FOR j IN 0..DIM-1

```

```

        Ba.orig[i] := Ba.orig[i] + A.invComp[j][i] * v[j]
    IF DYNAMIC CASE
        Ba.speed[i] := Ba.speed[i] + A.invComp[j][i] * s[j]
    Ba.comp[j][i] = 0
    FOR k IN 0..DIM-1
        Ba.comp[j][i] :=
            Ba.comp[j][i] + A.invComp[k][i] * B.comp[j][k]
    RETURN Ba

FUNCTION ELIM_VAR(FLOAT M[][], FLOAT Y[], INT nbVars, INT nbRows)
    INT nbRowsP := 0
    FLOAT Mp[][nbVars-1]
    FLOAT Yp[]
    FOR iRow IN 0..nbRows-2
        IF M[iRow][0] <> 0
            FOR jRow IN iRow+1..nbRows-1
                IF M[jRow][0] <> 0 AND M[iRow][0] * M[jRow][0] < 0
                    FOR iCol IN 1..nbCols-1
                        Mp[nbRowsP][iCol-1] :=
                            M[iRow][iCol] / ABS(M[iRow][0]) +
                            M[jRow][iCol] / ABS(M[jRow][0])
                    Yp[nbRowsP] :=
                        Y[iRow] / ABS(M[iRow][0]) +
                        Y[jRow] / ABS(M[jRow][0])
                    IF Yp[nbRowsP] <
                        SUM OF NEGATIVE VALUES IN Mp[nbRowsP][0..nbCols-2]
                        A AND B ARE NOT INTERSECTING, STOP
                    nbRowsP := nbRowsP + 1
    FOR iRow IN 0..nbRows-1
        IF M[iRow][0] = 0
            FOR iCol IN 1..nbCols-1
                Mp[nbRowsP][iCol-1] := M[iRow][iCol]
            Yp[nbRowsP] := Y[iRow]
            nbRowsP := nbRowsP + 1
    RETURN Mp, Yp, nbRowsP

```

4 Implementation of the algorithms in C

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

4.1 Frames

4.1.1 Header

```

#ifndef __FRAME_H_
#define __FRAME_H_

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

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

```

```

#include <stdbool.h>

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

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

typedef enum {

    FrameCuboid,
    FrameTetrahedron

} FrameType;

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

// Axis aligned bounding box structure
typedef struct {

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

} AABB2D;

typedef struct {

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

} AABB3D;

typedef struct {

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

} AABB2DTime;

typedef struct {

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

} AABB3DTime;

// Axis unaligned cuboid and tetrahedron structure
typedef struct {

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

    // AABB of the frame
    AABB2D bdgBox;

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

} Frame2D;

```

```

typedef struct {

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

    // AABB of the frame
    AABB3D bdgBox;

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

    // Flag to memorize if it's a face
    bool isFace;
} 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];

    // Flag to memorize if it's a face
    bool isFace;
} Frame3DTime;

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

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

// Print the Frame that on stdout

```

```

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

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

// Project the Frame Q in the Frame P's coordinates system and
// memorize the result in the Frame Qp
void Frame2DImportFrame(
    Frame2D* const P,
    const Frame2D* const Q,
    Frame2D* const Qp);
void Frame3DImportFrame(
    Frame3D* const P,
    const Frame3D* const Q,
    Frame3D* const Qp);
void Frame2DTimeImportFrame(
    Frame2DTime* const P,
    const Frame2DTime* const Q,
    Frame2DTime* const Qp);
void Frame3DTimeImportFrame(
    Frame3DTime* const P,
    const Frame3DTime* const Q,
    Frame3DTime* const Qp);

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

```

```

    const AABB3D* const bdgBox,
    AABB3D* const bdgBoxProj);
void Frame3DFaceExportBdgBox(
    const Frame3D* const that,
    const AABB2D* const bdgBox,
    AABB3D* const bdgBoxProj);
void Frame2DTimeExportBdgBox(
    const Frame2DTime* const that,
    const AABB2DTime* const bdgBox,
    AABB2DTime* const bdgBoxProj);
void Frame3DTimeExportBdgBox(
    const Frame3DTime* const that,
    const AABB3DTime* const bdgBox,
    AABB3DTime* const bdgBoxProj);

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

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

#endif

```

4.1.2 Body

```

#include "frame.h"

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

#define EPSILON 0.0000001

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

// Update the inverse components of the Frame that
static inline void Frame2DUpdateInv(Frame2D* const that);
static inline void Frame3DUpdateInv(Frame3D* const that);
static inline void Frame2DTimeUpdateInv(Frame2DTime* const that);
static inline 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
static inline 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;
}

static inline void Frame3DUpdateInv(Frame3D* const that) {

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

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

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

    }*/

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

// Update the inverse components of the Frame that
static inline 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;

}

static inline void Frame3DTimeUpdateInv(Frame3DTime* const that) {

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

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

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

    }*/

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

}

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

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

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

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

```

```

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

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

}

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

    qpo[i] = 0.0;

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

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

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

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

        }

    }

}

}

}

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

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

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

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

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

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

```

```

    }

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

        qpo[i] = 0.0;

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

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

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

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

            }

        }

    }

}

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

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

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

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

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

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

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

    }
}

```



```

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

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

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

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

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

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

        }

    }

}

}

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

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

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

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

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

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

        v[i] = qo[i] - po[i];
        s[i] = qps[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 Frame3DFaceExportBdgBox(
    const Frame3D* const that,
    const AABB2D* 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 = 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, 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 = 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[3];

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

```

```

        i--;) {

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

            bbpma[i] = w[i];

        }

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

            bbpma[i] = w[i];

        }

    }

}

}

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

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

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

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

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

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

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

        }

        bbpmi[i] = bbpma[i];

    }

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

```

```

int iVertex = nbVertices;
iVertex-- && iVertex;) {

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

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

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

}

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

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

    w[i] = to[i];

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

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

    }

}

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

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

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

    }

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

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

    }

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

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

    }

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

```



```

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

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

    // Shortcuts
    const double* to      = that->orig;
    const double* ts      = that->speed;
    const double* bbmi    = bdgBox->min;
    const double* bbma    = bdgBox->max;
    double* bbpmi = bdgBoxProj->min;
    double* bbpma = bdgBoxProj->max;
    const double (*tc)[3] = that->comp;

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

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

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

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

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

        }

        bbpmi[i] = bbpma[i];
    }

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

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

        // Calculate the coordinates of the vertex in
        // that's coordinates system

```

```

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

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

}

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

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

    w[i] = to[i];

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

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

    }

}

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

    if (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 AABB2DTimePrint(const AABB2DTime* const that) {

```

```

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

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

    }

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

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

    }

    printf(")");
}

void AAB3DTimePrint(const AAB3DTime* 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");

    }

    if (that->isFace == true) {

        printf("f");

    }

}

```

```

}

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");

    }

    if (that->isFace == true) {

        printf("f");

    }

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

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

    }

    printf(") s(");
    for (

```

```

    int i = 0;
    i < 3;
    ++i) {

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

    }

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

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

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

        }

    }

    printf(")");

}

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

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

        return false;

    }

    return true;

}

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

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

```



```

        return false;

    }

    return true;

}

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

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

    return res;

}

```

4.2 FMB

4.2.1 2D static

Header

```

#ifndef __FMB2D_H_
#define __FMB2D_H_

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

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

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

#endif

```

Body

```

#include "fmb2d.h"

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

```

```

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

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

#define FST_VAR 0
#define SND_VAR 1

#define EPSILON 0.0000001

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

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

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

    // 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* 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 (MIRow[iCol] > EPSILON) {

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

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

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

    }

}

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

    bdgBoxMin[iVar] = min;

}

if (bdgBoxMax[iVar] < max) {

    bdgBoxMax[iVar] = max;

}

}

}

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

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

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

```

```

double M[8][2];
double Y[8];

// Create the inequality system
// -sum_iC_j, iX_i <= 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 memorize 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_iX_i <= 1.0
        M[nbRows][0] = 1.0;
        M[nbRows][1] = 1.0;
        Y[nbRows] = 1.0;
        ++nbRows;

    }

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

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

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

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

    };

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

    // Eliminate the first variable
    bool inconsistency =

```

```

    ElimVar2D(
        M,
        Y,
        nbRows,
        2,
        Mp,
        Yp,
        &nbRowsP);

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

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

}

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

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

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

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

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

    // Memorize the result
    *bdgBox = bdgBoxLocal;

}

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

}

```

4.2.2 3D static

Header

```

#ifndef __FMB3D_H_
#define __FMB3D_H_

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

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

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

#endif

Body

#include "fmb3d.h"

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

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

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

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

#define EPSILON 0.0000001

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

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

```

```

// Get the bounds of the iVar-th variable in the nbRows rows
// system  $M.X \leq Y$  which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABBBdgBox
// (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,
    AABBB3D* 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 AABBBdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3D(
    const int iVar,
    const double (*M)[3],
    const double* Y,
    const int nbRows,
    const int nbCols,
    AABBB3D* 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<=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 AABB bdgBox
// (M arrangement is [iRow][iCol])
void GetBoundVar3D(

```

```

const int iVar,
const double (*M)[3],
const double* Y,
const int nbRows,
const int nbCols,
AABB3D* const bdgBox) {

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

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

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

// Shortcuts
const double* 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(
    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 memorize 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_i X_i <= 1.0
    M[nbRows][0] = 1.0;
    M[nbRows][1] = 1.0;
    M[nbRows][2] = 1.0;
    Y[nbRows] = 1.0;
    ++nbRows;

}

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

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

M[nbRows][0] = 0.0;

```

```

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

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

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

};

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

// Eliminate the first variable in the original system
bool inconsistency =
    ElimVar3D(
        M,
        Y,
        nbRows,
        3,
        Mp,
        Yp,
        &nbRowsP);

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

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

}

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

// Eliminate the second variable (which is the first in the new system)
inconsistency =
    ElimVar3D(
        Mp,
        Yp,
        nbRowsP,

```

```

        2,
        Mpp,
        Ypp,
        &nbRowsPP);

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

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

}

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

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

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

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

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

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

    // Memorize the result
    *bdgBox = bdgBoxLocal;

}

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

}

```

4.2.3 2D dynamic

Header

```
#ifndef __FMB2DT_H_
#define __FMB2DT_H_

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

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

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

#endif
```

Body

```
#include "fmb2dt.h"

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

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

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

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

#define EPSILON 0.0000001

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

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

```

double (*Mp)[3],
double* Yp,
int* const nbRemainRows);

// Get the bounds of the iVar-th variable in the nbRows rows
// system  $M.X \leq Y$  which has been reduced to only one variable
// and store them in the iVar-th axis of the
// AABBBdgBox
// (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 AABBBdgBox
// (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<=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,
    AAB2DTime* 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  $M.X \leq Y$  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 * 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 FMBTestIntersection2DTime(
    Frame2DTime* const that,
    const Frame2DTime* const tho,
    AABB2DTime* const bdgBox) {

    // Get the projection of the Frame tho in Frame that coordinates
    // system
    Frame2DTime thoProj;
    Frame2DTimeImportFrame(that, tho, &thoProj);

    // Declare two variables to memorize the system to be solved M.X <= Y
    // (M arrangement is [iRow][iCol])
    double M[10][3];
    double Y[10];

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

        return false;

    }

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

        return false;

    }

}

```

```

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

if (that->type == FrameCuboid) {

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

        return false;

    }

    ++nbRows;

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

        return false;

    }

    ++nbRows;

} else {

    // sum_j(V_jT+sum_iC_j,iX_i)<=1.0-sum_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(

```



```

    Frame3DTime* const that,
    const Frame3DTime* const tho,
    AABBB3DTime* 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,
AABB3DTime* const bdgBox);

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

// Eliminate the first variable in the system M.X<=Y
// using the Fourier-Motzkin method and return
// the resulting system in Mp and Yp, and the number of rows of
// the resulting system in nbRemainRows
// (M arrangement is [iRow][iCol])
// Return true if the system becomes inconsistent during elimination,
// else return false
bool ElimVar3DTime(
const double (*M)[4],
const double* Y,
const int nbRows,
const int nbCols,
double (*Mp)[4],
double* Yp,
int* const nbRemainRows) {

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

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

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

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

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

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

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

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

```

```

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

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

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

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

}

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

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

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

}

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

}

}

}

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

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

    // If the coefficient of the eliminated variable is null on
    // this row

```

```

    if (fabs(MiRow[0]) < EPSILON) {

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

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

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

        }

        Yp[nbResRows] = Y[iRow];

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

    }

}

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

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

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y and store them in the iVar-th axis of the
// AABB bdgBox
// (M arrangement is [iRow][iCol])
// The system is supposed to have been reduced to only one variable
// per row, the one in argument
// May return inconsistent values (max < min), which would
// mean the system has no solution
void GetBoundLastVar3DTime(
    const int iVar,
    const double (*M)[4],
    const double* Y,
    const int nbRows,
    AABB3DTime* const bdgBox) {

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

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

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

```

```

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

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

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

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

        // Update the maximum bound
        *max = y;

    }

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

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

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

        // Update the minimum bound
        *min = y;

    }

}

}

}

// Get the bounds of the iVar-th variable in the nbRows rows
// system M.X<=Y where the iVar-th variable is on the first column
// and store them in the iVar-th axis of the 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) {

    // 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 FMBTestIntersection3DTime(
    Frame3DTime* const that,
    const Frame3DTime* const tho,
    AABB3DTime* const bdgBox) {

    // Get the projection of the Frame tho in Frame that coordinates
    // system
    Frame3DTime thoProj;
    Frame3DTimeImportFrame(that, tho, &thoProj);

    // Declare two variables to memorize the system to be solved M.X <= Y
    // (M arrangement is [iRow][iCol])
    double M[14][4];
    double Y[14];

    // Create the inequality system
    // -V_jT-sum_iC_j,iX_i<=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_iX_i<=1.0
    M[nbRows][0] = 1.0;
    M[nbRows][1] = 1.0;
    M[nbRows][2] = 1.0;
    M[nbRows][3] = 0.0;
    Y[nbRows] = 1.0;
    ++nbRows;

}

// -X_i <= 0.0
M[nbRows][0] = -1.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = -1.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = -1.0;
M[nbRows][3] = 0.0;
Y[nbRows] = 0.0;
++nbRows;

```

```

// 0.0 <= t <= 1.0
M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = 1.0;
Y[nbRows] = 1.0;
++nbRows;

M[nbRows][0] = 0.0;
M[nbRows][1] = 0.0;
M[nbRows][2] = 0.0;
M[nbRows][3] = -1.0;
Y[nbRows] = 0.0;
++nbRows;

// Solve the system
// Declare a AABB to memorize the bounding box of the intersection
// in the coordinates system of that
AABB3DTime bdgBoxLocal = {

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

};

// Declare variables to eliminate the first variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mp[63][4];
//double Yp[63];
double Mp[22][4];
double Yp[22];
int nbRowsP;

// Eliminate the first variable in the original system
bool inconsistency =
    ElimVar3DTime(
        M,
        Y,
        nbRows,
        4,
        Mp,
        Yp,
        &nbRowsP);

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

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

}

// Declare variables to eliminate the second variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mpp[1056][4];
//double Ypp[1056];

```

```

double Mpp[57][4];
double Ypp[57];
int nbRowsPP;

// Eliminate the second variable (which is the first in the new system)
inconsistency =
    ElimVar3DTime(
        Mp,
        Yp,
        nbRowsP,
        3,
        Mpp,
        Ypp,
        &nbRowsPP);

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

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

}

// Declare variables to eliminate the third variable
// The size of the array given in the doc is a majoring value.
// Instead I use a smaller value which has proven to be sufficient
// during tests, validation and qualification, to avoid running
// into the heap limit and to optimize slightly the performance
//double Mppp[279840][4];
//double Yppp[279840];
double Mppp[560][4];
double Yppp[560];
int nbRowsPPP;

// Eliminate the third variable (which is the first in the new system)
inconsistency =
    ElimVar3DTime(
        Mpp,
        Ypp,
        nbRowsPP,
        2,
        Mppp,
        Yppp,
        &nbRowsPPP);

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

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

}

// Get the bounds for the remaining fourth variable
GetBoundLastVar3DTime(
    FOR_VAR,
    Mppp,
    Yppp,
    nbRowsPPP,
    &bdgBoxLocal);

// If the bounds are inconsistent

```

```

    if (bdgBoxLocal.min[FOR_VAR] >= bdgBoxLocal.max[FOR_VAR]) {

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

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

        // Get the bounds of the other variables
        GetBoundVar3DTime(
            THD_VAR,
            Mpp,
            Ypp,
            nbRowsPP,
            2,
            &bdgBoxLocal);

        GetBoundVar3DTime(
            SND_VAR,
            Mp,
            Yp,
            nbRowsP,
            3,
            &bdgBoxLocal);

        GetBoundVar3DTime(
            FST_VAR,
            M,
            Y,
            nbRows,
            4,
            &bdgBoxLocal);

        // Memorize the result
        *bdgBox = bdgBoxLocal;

    }

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

}

```

5 Minimal example of use

In this section I give a minimal example for each case of how to use the code given in the previous section.

5.1 2D static

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>

```

```

#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2d.h"

// Main function
int main(int argc, char** argv) {

    // Create the two objects to be tested for intersection
    double origP2D[2] = {0.0, 0.0};
    double compP2D[2][2] = {

        {1.0, 0.0}, // First component
        {0.0, 1.0} // Second component
    };

    Frame2D P2D =
        Frame2DCreateStatic(
            FrameCuboid,
            origP2D,
            compP2D);

    double origQ2D[2] = {0.0, 0.0};
    double compQ2D[2][2] = {{1.0, 0.0}, {0.0, 1.0}};
    Frame2D Q2D =
        Frame2DCreateStatic(
            FrameCuboid,
            origQ2D,
            compQ2D);

    // Declare a variable to memorize the result of the intersection
    // detection
    AABB2D bdgBox2DLocal;

    // Test for intersection between P and Q
    bool isIntersecting2D =
        FMBTestIntersection2D(
            &P2D,
            &Q2D,
            &bdgBox2DLocal);

    // If the two objects are intersecting
    if (isIntersecting2D) {

        printf("Intersection detected in AABB ");

        // Export the local bounding box toward the real coordinates
        // system
        AABB2D bdgBox2D;
        Frame2DExportBdgBox(
            &Q2D,
            &bdgBox2DLocal,
            &bdgBox2D);

        // Clip with the AABB of 'Q2D' and 'P2D' to improve results
        for (
            int iAxis = 2;
            iAxis--;) {

            if (bdgBox2D.min[iAxis] < P2D.bdgBox.min[iAxis]) {

                bdgBox2D.min[iAxis] = P2D.bdgBox.min[iAxis];
            }
        }
    }
}

```

```

    }

    if (bdgBox2D.max[iAxis] > P2D.bdgBox.max[iAxis]) {
        bdgBox2D.max[iAxis] = P2D.bdgBox.max[iAxis];
    }

    if (bdgBox2D.min[iAxis] < Q2D.bdgBox.min[iAxis]) {
        bdgBox2D.min[iAxis] = Q2D.bdgBox.min[iAxis];
    }

    if (bdgBox2D.max[iAxis] > Q2D.bdgBox.max[iAxis]) {
        bdgBox2D.max[iAxis] = Q2D.bdgBox.max[iAxis];
    }

    }

    AABBB2DPrint(&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
    AAB3DTime 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.286)

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 memorize the arguments to the
    // Validation function
    Param2D paramP;
    Param2D paramQ;

    // Declare a variable to memorize the correct bounding box
    AABB2D correctBdgBox;

    // Execute the unit test on various cases
    // -----
    paramP = (Param2D) {

        .type = FrameCuboid,
        .orig = {0.0, 0.0},
        .comp = {{1.0, 0.0}, {0.0, 1.0}}

    };
    paramQ = (Param2D) {

        .type = FrameCuboid,
        .orig = {0.0, 0.0},
        .comp = {{1.0, 0.0}, {0.0, 1.0}}

    };
    correctBdgBox = (AABB2D) {

        .min = {0.0, 0.0},
        .max = {1.0, 1.0}

    };
    UnitTest2D(
        paramP,
        paramQ,
        true,
        &correctBdgBox);

    // -----
    paramP = (Param2D) {

        .type = FrameCuboid,
        .orig = {0.0, 0.0},
        .comp = {{1.0, 0.0}, {0.0, 1.0}}
    }
}

```

```

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.5, 0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.5},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {-0.5, -0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.5, 0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
UnitTest2D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.25, -0.25},
    .comp = {{0.5, 0.0}, {0.0, 2.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.25, 0.0},
    .max = {0.75, 1.0}

```



```

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {-0.25, 0.25},
    .comp = {{2.0, 0.0}, {0.0, 0.5}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.25},
    .max = {1.0, 0.75}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 1.0}, {-1.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----

```

```

paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {-0.5, -0.5},
    .comp = {{1.0, 1.0}, {-1.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {0.5, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.5, 1.5},
    .comp = {{1.0, -1.0}, {-1.0, -1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.0, 0.0},
    .comp = {{-1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.0},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.0, 0.5},
    .comp = {{-0.5, 0.5}, {-0.5, -0.5}}

};
paramQ = (Param2D) {

```

```

        .type = FrameCuboid,
        .orig = {0.0, 1.0},
        .comp = {{1.0, 0.0}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {1.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {2.0, -1.0},
    .comp = {{0.0, 1.0}, {-0.5, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {1.5, 0.5},
    .max = {1.5 + 0.5 / 3.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.0, 1.0},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

```

```

        .min = {0.5, 0.25},
        .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.0, 2.0},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.75},
    .max = {1.0, 1.25}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameCuboid,
    .orig = {1.0, 2.0},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.5},
    .max = {0.75, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,

```

```

    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {1.0, 2.0},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5 + 1.0 / 3.0, 1.0},
    .max = {1.0, 1.0 + 1.0 / 3.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

```

```

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {0.5, 0.5}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.5, 0.5},
    .comp = {{-0.5, 0.0}, {0.0, -0.5}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.0, 0.0},
    .max = {0.5, 0.5}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.5, 0.5},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

```

```

};
UnitTest2D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param2D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {1.5, 1.5},
    .comp = {{-1.5, 0.0}, {0.0, -1.5}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.5},
    .max = {1.0, 1.0}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {1.01, 1.01},
    .comp = {{-1.0, 0.0}, {0.0, -1.0}}

};
UnitTest2D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

```

```

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {1.0, 1.0},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
correctBdgBox = (AABB2D) {

    .min = {0.5, 0.5 - 1.0 / 6.0},
    .max = {1.0, 0.75}

};
UnitTest2D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.5}, {0.5, 1.0}}

};
paramQ = (Param2D) {

    .type = FrameTetrahedron,
    .orig = {1.01, 1.5},
    .comp = {{-0.5, -0.5}, {0.0, -1.0}}

};
UnitTest2D(
    paramP,
    paramQ,
    false,
    NULL);

// If we reached here, it means all the unit tests succeed
printf("All unit tests 2D have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

    Test2D();

    return 0;

}

```

6.1.2 3D static

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>

```



```

#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb3d.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

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

} Param3D;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection 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");
        AABBE3DPrint(&bdgBox);
        printf("\n");

    // Else, the bounding box wasn't the expected one
    } else {

        // Display information
        printf("Failed\n");
        printf("Expected : ");
        AABBE3DPrint(correctBdgBox);
        printf("\n");
        printf("      Got : ");
        AABBE3DPrint(&bdgBox);
        printf("\n");

        // Terminate the unit tests
        exit(0);

    }

    // Else the Frames were not intersected,
    // no need to check the bounding box
    } else {

        // Display information
        printf(" Succeed (no inter)\n");

    }

}

printf("\n");

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

}

}

void Test3D(void) {

    // Declare two variables to memorize the arguments to the

```

```

// Validation function
Param3D paramP;
Param3D paramQ;

// Declare a variable to memorize the correct bounding box
AABB3D correctBdgBox;

// Execute the unit test on various cases
// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.0, 0.0, 0.0},
    .max = {1.0, 1.0, 1.0}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.5, 0.5, 0.5},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.5, 0.5, 0.5},
    .max = {1.0, 1.0, 1.0}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

```

```

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {-0.5, -0.5, -0.5},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.0, 0.0, 0.0},
    .max = {0.5, 0.5, 0.5}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {1.5, 1.5, 1.5},
    .comp = {{-1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, -1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.5, 0.5, 0.5},
    .max = {1.0, 1.0, 1.0}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};

```

```

paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.5, 1.5, -1.5},
    .comp = {{1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, -1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.5, 1.5, -1.5},
    .comp = {{1.0, 0.0, 0.0}, {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.5, 0.5, -1.0},
    .max = {1.0, 1.0, -0.5}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {-1.01, -1.01, -1.01},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
    paramP,
    paramQ,
    false,
    NULL);

// -----

```

```

paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {-1.0, -1.0, -1.0},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
correctBdgBox = (AABB3D) {

    .min = {0.0, -0.5, 0.0},
    .max = {1.0, 0.0, 1.0}

};
UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {-1.0, -1.0, -1.0},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameCuboid,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};
UnitTest3D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param3D) {

    .type = FrameCuboid,
    .orig = {-1.0, -1.0, -1.0},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};
paramQ = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};

```

```

correctBdgBox = (AABB3D) {

    .min = {0.0, -0.5, 0.0},
    .max = {0.75, 0.0, 0.75}

};

UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {-1.0, -1.0, -1.0},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};

paramQ = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};

UnitTest3D(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {-0.5, -1.0, -0.5},
    .comp = {{1.0, 0.0, 0.0}, {1.0, 1.0, 1.0}, {0.0, 0.0, 1.0}}

};

paramQ = (Param3D) {

    .type = FrameTetrahedron,
    .orig = {0.0, -0.5, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}}

};

correctBdgBox = (AABB3D) {

    .min = {0.0, -0.5, 0.0},
    .max = {0.5, -1.0 / 3.0, 0.5}

};

UnitTest3D(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// If we reached here, it means all the unit tests succeed
printf("All unit tests 3D have succeed.\n");

```



```

}

// Main function
int main(int argc, char** argv) {

    Test3D();

    return 0;

}

```

6.1.3 2D dynamic

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library
#include "fmb2dt.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

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

} Param2DTime;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection 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(&bdbgBox);
    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(&bdbgBox);
    printf("\n");

    // Terminate the unit tests
    exit(0);

}

// Else the Frames were not intersected,
// no need to check the bounding box
} else {

    // Display information
    printf(" Succeed (no inter)\n");

}

}

printf("\n");

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

}

}

void Test2DTime(void) {

    // Declare two variables to memorize the arguments to the
    // Validation function
    Param2DTime paramP;
    Param2DTime paramQ;

    // Declare a variable to memorize the correct bounding box
    AABB2DTime correctBdgBox;

    // Execute the unit test on various cases

```

```

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};

paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {-1.0, 0.0}

};

UnitTest2DTime(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};

paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {-1.01, -1.01},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {1.0, 0.0}

};

UnitTest2DTime(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};

paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {1.0, 0.0}

```

```

};
correctBdgBox = (AABB2DTime) {

    .min = {0.0, 0.0, 0.0},
    .max = {1.0, 1.0, 1.0}

};

UnitTest2DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};

paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.25},
    .comp = {{0.5, 0.0}, {0.0, 0.5}},
    .speed = {4.0, 0.0}

};

correctBdgBox = (AABB2DTime) {

    .min = {0.0, 0.25, 0.125},
    .max = {1.0, 0.75, 0.5}

};

UnitTest2DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};

paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.25, -1.0},
    .comp = {{0.5, 0.0}, {0.0, 0.5}},
    .speed = {0.0, 4.0}

};

correctBdgBox = (AABB2DTime) {

    .min = {0.25, 0.0, 0.125},

```

```

        .max = {0.75, 1.0, 0.5}

};
UnitTest2DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0},
    .comp = {{1.0, 0.0}, {0.0, 1.0}},
    .speed = {0.0, 0.0}

};
paramQ = (Param2DTime) {

    .type = FrameCuboid,
    .orig = {0.9, -1.0},
    .comp = {{0.5, 0.0}, {0.0, 0.5}},
    .speed = {0.0, 4.0}

};
correctBdgBox = (AABB2DTime) {

    .min = {0.9, 0.0, 0.125},
    .max = {1.0, 1.0, 0.5}

};
UnitTest2DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// If we reached here, it means all the unit tests succeed
printf("All unit tests 2DTime have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

    Test2DTime();

    return 0;

}

```

6.1.4 3D dynamic

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

// Include the FMB algorithm library

```

```

#include "fmb3dt.h"

// Epsilon for numerical precision
#define EPSILON 0.0001

// Helper structure to pass arguments to the UnitTest function
typedef struct {

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

} Param3DTime;

// Unit test function
// Takes two Frame definitions, the correct answer in term of
// intersection/no intersection and the correct bounding box
// Run the FMB intersection detection 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 pairs (that, tho) and (tho, that)
    Frame3DTime* that = &P;
    Frame3DTime* tho = &Q;

    // Loop on pairs of Frames
    for (
        int iPair = 2;
        iPair--;) {

        // Display the tested frames
        Frame3DTimePrint(that);
        printf("\nagainst\n");
        Frame3DTimePrint(tho);
        printf("\n");

        // Run the FMB intersection test
        bool isIntersecting =
            FMBTestIntersection3DTime(

```

```

        that,
        tho,
        &bdgBoxLocal);

// If the test hasn't given the expected answer about intersection
if (isIntersecting != correctAnswer) {

    // Display information about the failure
    printf(" Failed\n");
    printf("Expected : ");
    if (correctAnswer == false) printf("no ");
    printf("intersection\n");
    printf("Got : ");
    if (isIntersecting == false) printf("no ");
    printf("intersection\n");
    exit(0);

// Else, the test has given the expected answer about intersection
} else {

    // If the Frames were intersecting
    if (isIntersecting == true) {

        AABB3DTime bdgBox;
        Frame3DTimeExportBdgBox(
            tho,
            &bdgBoxLocal,
            &bdgBox);

        // Check the bounding box
        bool flag = true;
        for (
            int i = 4;
            i--;) {

            if (
                bdgBox.min[i] > correctBdgBox->min[i] + EPSILON ||
                bdgBox.max[i] < correctBdgBox->max[i] - EPSILON) {

                flag = false;
            }
        }

        // If the bounding box is the expected one
        if (flag == true) {

            // Display information
            printf("Succeed\n");
            AABB3DTimePrint(&bdgBox);
            printf("\n");

        // Else, the bounding box wasn't the expected one
        } else {

            // Display information
            printf("Failed\n");
            printf("Expected : ");
            AABB3DTimePrint(correctBdgBox);
            printf("\n");
            printf("      Got : ");

```



```

        AABB3DTimePrint(&bdbgBox);
        printf("\n");

        // Terminate the unit tests
        exit(0);

    }

    // Else the Frames were not intersected,
    // no need to check the bounding box
} else {

    // Display information
    printf(" Succeed (no inter)\n");

}

}

printf("\n");

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

}

}

void Test3DTime(void) {

    // Declare two variables to memorize the arguments to the
    // Validation function
    Param3DTime paramP;
    Param3DTime paramQ;

    // Declare a variable to memorize the correct bounding box
    AABB3DTime correctBdgBox;

    // Execute the unit test on various cases
    // -----
    paramP = (Param3DTime) {

        .type = FrameCuboid,
        .orig = {0.0, 0.0, 0.0},
        .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
        .speed = {0.0, 0.0, 0.0}

    };

    paramQ = (Param3DTime) {

        .type = FrameCuboid,
        .orig = {-1.0, 0.0, 0.0},
        .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
        .speed = {-1.0, 0.0, 0.0}

    };

    UnitTest3DTime(
        paramP,
        paramQ,
        false,
        NULL);
}

```

```

// -----
paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {-1.01, -1.01, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {1.0, 0.0, 0.0}

};
UnitTest3DTime(
    paramP,
    paramQ,
    false,
    NULL);

// -----
paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {1.0, 0.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

    .min = {0.0, 0.0, 0.0, 0.0},
    .max = {1.0, 1.0, 1.0, 1.0}

};
UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

};

```

```

paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {-1.0, 0.25, 0.0},
    .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {4.0, 0.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

    .min = {0.0, 0.25, 0.0, 0.125},
    .max = {1.0, 0.75, 1.0, 0.5}

};
UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.25, -1.0, 0.0},
    .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 4.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

    .min = {0.25, 0.0, 0.0, 0.125},
    .max = {0.75, 1.0, 1.0, 0.5}

};
UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// -----
paramP = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.0, 0.0, 0.0},
    .comp = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
    .speed = {0.0, 0.0, 0.0}

};
paramQ = (Param3DTime) {

    .type = FrameCuboid,
    .orig = {0.9, -1.0, 0.0},

```

```

        .comp = {{0.5, 0.0, 0.0}, {0.0, 0.5, 0.0}, {0.0, 0.0, 1.0}},
        .speed = {0.0, 4.0, 0.0}

};
correctBdgBox = (AABB3DTime) {

    .min = {0.9, 0.0, 0.0, 0.125},
    .max = {1.0, 1.0, 1.0, 0.5}

};
UnitTest3DTime(
    paramP,
    paramQ,
    true,
    &correctBdgBox);

// If we reached here, it means all the unit tests succeed
printf("All unit tests 3DTime have succeed.\n");

}

// Main function
int main(int argc, char** argv) {

    Test3DTime();

    return 0;

}

```

6.2 Results

6.2.1 2D static

```

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)

```

```

Succeed (no inter)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.250000,-0.250000) x(0.500000,0.000000) y(0.000000,2.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.250000,0.000000)-maxXY(0.750000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)

Co(-0.250000,0.250000) x(2.000000,0.000000) y(0.000000,0.500000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.250000)-maxXY(1.000000,0.750000)

Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(-0.500000,-0.500000) x(1.000000,1.000000) y(-1.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,1.000000)

Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
against
Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.000000) x(-1.000000,0.000000) y(0.000000,1.000000)

```

```

against
Co(1.500000,1.500000) x(1.000000,-1.000000) y(-1.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
against
Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,1.000000) x(1.000000,0.000000) y(0.000000,-1.000000)
against
Co(1.000000,0.500000) x(-0.500000,0.500000) y(-0.500000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
against
Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
Succeed
minXY(1.500000,0.000000)-maxXY(1.666667,1.000000)

Co(2.000000,-1.000000) x(0.000000,1.000000) y(-0.500000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(1.000000,1.000000)
Succeed
minXY(1.500000,0.500000)-maxXY(2.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.500000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)

```

```

Succeed
minXY(0.500000,0.500000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

To(1.000000,2.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,1.000000)-maxXY(1.000000,1.500000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,1.000000)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(1.000000,0.500000)

Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
against
To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

To(0.000000,-0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(-0.500000,0.000000) y(0.000000,-0.500000)
Succeed
minXY(0.000000,0.000000)-maxXY(0.500000,0.500000)

Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
Co(0.500000,0.500000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)

```

```

against
To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(1.500000,1.500000) x(-1.500000,0.000000) y(0.000000,-1.500000)
against
Co(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed
minXY(0.000000,0.500000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
against
To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
Succeed (no inter)

To(1.010000,1.010000) x(-1.000000,0.000000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.000000) y(0.000000,1.000000)
Succeed (no inter)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(1.000000,1.000000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed
minXY(0.500000,0.000000)-maxXY(1.000000,1.000000)

To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
against
To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
Succeed (no inter)

To(1.010000,1.500000) x(-0.500000,-0.500000) y(0.000000,-1.000000)
against
To(0.000000,0.000000) x(1.000000,0.500000) y(0.500000,1.000000)
Succeed (no inter)

All unit tests 2D have succeed.

```

6.2.2 3D static

```

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed

```



```

Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,-1.010000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
  (1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

```

```

Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,0.750000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(1.000000,0.000000,1.000000)

To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-1.000000,-1.000000,-1.000000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
against
To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

To(0.000000,-0.500000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
To(-0.500000,-1.000000,-0.500000) x(1.000000,0.000000,0.000000) y
(1.000000,1.000000,1.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,-0.500000,0.000000)-maxXYZ(0.500000,0.000000,0.500000)

All unit tests 3D have succeed.

```

6.2.3 2D dynamic

```

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed (no inter)

Co(-1.000000,0.000000) s(-1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y

```

```

(0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed (no inter)

Co(-1.010000,-1.010000) s(1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(2.000000,1.000000,1.000000)

Co(-1.000000,0.000000) s(1.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed
minXYT(-1.000000,0.000000,0.000000)-maxXYT(1.000000,1.000000,1.000000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
Succeed
minXYT(-1.500000,0.000000,0.125000)-maxXYT(2.500000,1.000000,0.500000)

Co(-1.000000,0.250000) s(4.000000,0.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed
minXYT(-0.500000,0.000000,0.125000)-maxXYT(1.500000,1.000000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.000000,2.500000,0.500000)

Co(0.250000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)

```

```

Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.000000,1.500000,0.500000)

Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
against
Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
Succeed
minXYT(0.000000,-1.500000,0.125000)-maxXYT(1.400000,2.500000,0.500000)

Co(0.900000,-1.000000) s(0.000000,4.000000) x(0.500000,0.000000) y
(0.000000,0.500000)
against
Co(0.000000,0.000000) s(0.000000,0.000000) x(1.000000,0.000000) y
(0.000000,1.000000)
Succeed
minXYT(0.000000,-0.500000,0.125000)-maxXYT(1.400000,1.500000,0.500000)

All unit tests 2DTime have succeed.

```

6.2.4 3D dynamic

```

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(-1.000000,0.000000,0.000000) s(-1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(-1.010000,-1.010000,0.000000) s(1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z

```

```

(0.000000,0.000000,1.000000)
against
Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
(2.000000,1.000000,1.000000,1.000000)

Co(-1.000000,0.000000,0.000000) s(1.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.000000,0.000000,0.000000,0.000000)-maxXYZT
(1.000000,1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(-1.500000,0.000000,0.000000,0.125000)-maxXYZT
(2.500000,1.000000,1.000000,0.500000)

Co(-1.000000,0.250000,0.000000) s(4.000000,0.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(-0.500000,0.000000,0.000000,0.125000)-maxXYZT
(1.500000,1.000000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
(1.000000,2.500000,1.000000,0.500000)

Co(0.250000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT

```

```

(1.000000,1.500000,1.000000,0.500000)

Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-1.500000,0.000000,0.125000)-maxXYZT
(1.400000,2.500000,1.000000,0.500000)

Co(0.900000,-1.000000,0.000000) s(0.000000,4.000000,0.000000) x
(0.500000,0.000000,0.000000) y(0.000000,0.500000,0.000000) z
(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) s(0.000000,0.000000,0.000000) x
(1.000000,0.000000,0.000000) y(0.000000,1.000000,0.000000) z
(0.000000,0.000000,1.000000)
Succeed
minXYZT(0.000000,-0.500000,0.000000,0.125000)-maxXYZT
(1.400000,1.500000,1.000000,0.500000)

All unit tests 3DTime have succeed.

```

7 Validation against SAT

In this section I introduce the code I've used to validate the algorithm and its implementation. The validation consists of running the FMB algorithm on randomly generated pairs of Frame and check that its result is equal to the one of running the SAT algorithm on the same pair of Frames. The code of the implementation of the SAT algorithm is given in annex (p.286)

7.1 Code

7.1.1 2D static

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include FMB and SAT algorithm library
#include "fmb2d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

```

```

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

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

} Param2D;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair2D(
    const Param2D paramP,
    const Param2D paramQ) {

    // Create the two Frames
    Frame2D P =
        Frame2DCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.comp);

    Frame2D Q =
        Frame2DCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame2D* that = &P;
    Frame2D* tho = &Q;

    // Loop on pairs of Frames
    for (
        int iPair = 2;
        iPair--;) {

        // Test intersection with FMB
        bool isIntersectingFMB =
            FMBTestIntersection2D(
                that,
                tho,
                NULL);

        // Test intersection with SAT
        bool isIntersectingSAT =
            SATTestIntersection2D(
                that,
                tho);
    }
}

```



```

// If the results are different
if (isIntersectingFMB != isIntersectingSAT) {

    // Print the disagreement
    printf("Validation2D has failed\n");
    Frame2DPrint(that);
    printf(" against ");
    Frame2DPrint(tho);
    printf("\n");
    printf("FMB : ");
    if (isIntersectingFMB == false) printf("no ");
    printf("intersection\n");
    printf("SAT : ");
    if (isIntersectingSAT == false) printf("no ");
    printf("intersection\n");

    // Stop the validation
    exit(0);

}

// If the Frames are in intersection
if (isIntersectingFMB == true) {

    // Update the number of intersection
    nbInter++;

// If the Frames are not in intersection
} else {

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

}

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

}

}

void Validate2D(void) {

    // Initialise the random generator
    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 here it means the validation was successfull
// Print results

```

```

    printf("Validation2D has succeed.\n");
    printf("Tested %lu intersections ", nbInter);
    printf("and %lu no intersections\n", nbNoInter);
}

int main(int argc, char** argv) {

    printf("==== 2D static =====\n");
    Validate2D();

    return 0;
}

```

7.1.2 3D static

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include FMB and SAT algorithm library
#include "fmb3d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

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

} Param3D;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair3D(
    const Param3D paramP,
    const Param3D paramQ) {

```

```

// Create the two Frames
Frame3D P =
    Frame3DCreateStatic(
        paramP.type,
        paramP.orig,
        paramP.comp);

Frame3D Q =
    Frame3DCreateStatic(
        paramQ.type,
        paramQ.orig,
        paramQ.comp);

// Helper variables to loop on the pair (that, tho) and (tho, that)
Frame3D* that = &P;
Frame3D* tho = &Q;

// Loop on pairs of Frames
for (
    int iPair = 2;
    iPair--;) {

    // Test intersection with FMB
    bool isIntersectingFMB =
        FMBTestIntersection3D(
            that,
            tho,
            NULL);

    // Test intersection with SAT
    bool isIntersectingSAT =
        SATTestIntersection3D(
            that,
            tho);

    // If the results are different
    if (isIntersectingFMB != isIntersectingSAT) {

        // Print the disagreement
        printf("Validation3D has failed\n");
        Frame3DPrint(that);
        printf(" against ");
        Frame3DPrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT == false) printf("no ");
        printf("intersection\n");

        // Stop the validation
        exit(0);

    }

    // If the Frames are in intersection
    if (isIntersectingFMB == true) {

        // Update the number of intersection
        nbInter++;
    }
}

```

```

    // If the Frames are not in intersection
    } else {

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

    }

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

}

}

void Validate3D(void) {

    // Initialise the random generator
    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 here it means the validation was successfull
// Print results
printf("Validation3D has succeed.\n");
printf("Tested %lu intersections ", nbInter);
printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

    printf("==== 3D static =====\n");
    Validate3D();

    return 0;

}

```

7.1.3 2D dynamic

```

// Include standard libraries

```

```

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

// Include the FMB and SAT algorithm library
#include "fmb2dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;
unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

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

} Param2DTime;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair2DTime(
    const Param2DTime paramP,
    const Param2DTime paramQ) {

    // Create the two Frames
    Frame2DTime P =
        Frame2DTimeCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.speed,
            paramP.comp);

    Frame2DTime Q =
        Frame2DTimeCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.speed,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame2DTime* that = &P;
    Frame2DTime* tho = &Q;

```

```

// Loop on pairs of Frames
for (
    int iPair = 2;
    iPair--;) {

    // Test intersection with FMB
    bool isIntersectingFMB =
        FMBTestIntersection2DTime(
            that,
            tho,
            NULL);

    // Test intersection with SAT
    bool isIntersectingSAT =
        SATTestIntersection2DTime(
            that,
            tho);

    // If the results are different
    if (isIntersectingFMB != isIntersectingSAT) {

        // Print the disagreement
        printf("Validation2D has failed\n");
        Frame2DTimePrint(that);
        printf(" against ");
        Frame2DTimePrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT == false) printf("no ");
        printf("intersection\n");

        // Stop the validation
        exit(0);

    }

    // If the Frames are in intersection
    if (isIntersectingFMB == true) {

        // Update the number of intersection
        nbInter++;

    }

    // If the Frames are not in intersection
    } else {

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

    }

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

}

}

void Validate2DTime(void) {

```



```

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

// Declare two variables to memorize the arguments to the
// Validation function
Param2DTime paramP;
Param2DTime paramQ;

// Initialize the number of intersection and no intersection
nbInter = 0;
nbNoInter = 0;

// Loop on the tests
for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param2DTime* param = &paramP;
    for (
        int iParam = 2;
        iParam--;) {

        // 50% chance of being a Cuboid or a Tetrahedron
        if (rnd() < 0.5) {

            param->type = FrameCuboid;

        } else {

            param->type = FrameTetrahedron;

        }

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

            param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;
            param->speed[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

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

                param->comp[iComp][iAxis] =
                    -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

            }

        }

        param = &paramQ;

    }

    // Calculate the determinant of the Frames' components matrix
    double detP =
        paramP.comp[0][0] * paramP.comp[1][1] -
        paramP.comp[1][0] * paramP.comp[0][1];

```

```

double detQ =
    paramQ.comp[0][0] * paramQ.comp[1][1] -
    paramQ.comp[1][0] * paramQ.comp[0][1];

// If the determinants are not null, ie the Frame are not degenerate
if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    ValidationOnePair2DTime(
        paramP,
        paramQ);

}

}

// If we reached here it means the validation was successfull
// Print results
printf("Validation2DTime has succeed.\n");
printf("Tested %lu intersections ", nbInter);
printf("and %lu no intersections\n", nbNoInter);

}

int main(int argc, char** argv) {

    printf("==== 2D dynamic =====\n");
    Validate2DTime();

    return 0;

}

```

7.1.4 3D dynamic

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>

// Include the FMB and SAT algorithm library
#include "fmb3dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of tests of the validation
#define NB_TESTS 1000000

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Global variables to count nb of tests resulting in intersection
// and no intersection
unsigned long int nbInter;

```

```

unsigned long int nbNoInter;

// Helper structure to pass arguments to the Validation function
typedef struct {

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

} Param3DTime;

// Validation function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and check the results are identical
void ValidationOnePair3DTime(
    const Param3DTime paramP,
    const Param3DTime paramQ) {

    // Create the two Frames
    Frame3DTime P =
        Frame3DTimeCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.speed,
            paramP.comp);

    Frame3DTime Q =
        Frame3DTimeCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.speed,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame3DTime* that = &P;
    Frame3DTime* tho = &Q;

    // Loop on pairs of Frames
    for (
        int iPair = 2;
        iPair--;) {

        // Test intersection with FMB
        bool isIntersectingFMB =
            FMBTestIntersection3DTime(
                that,
                tho,
                NULL);

        // Test intersection with SAT
        bool isIntersectingSAT =
            SATTestIntersection3DTime(
                that,
                tho);

        // If the results are different
        if (isIntersectingFMB != isIntersectingSAT) {

            // Print the disagreement
            printf("Validation3D has failed\n");
            Frame3DTimePrint(that);
        }
    }
}

```

```

        printf(" against ");
        Frame3DTimePrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT == false) printf("no ");
        printf("intersection\n");

        // Stop the validation
        exit(0);
    }

    // If the Frames are in intersection
    if (isIntersectingFMB == true) {

        // Update the number of intersection
        nbInter++;

    // If the Frames are not in intersection
    } else {

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

    }

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

}

}

void Validate3DTime(void) {

    // Initialise the random generator
    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 here it means the validation was successfull
// Print results
printf("Validation3DTime has succeed.\n");
printf("Tested %lu intersections ", nbInter);
printf("and %lu no intersections\n", nbNoInter);
}

int main(int argc, char** argv) {

    printf("==== 3D dynamic =====\n");
    Validate3DTime();

    return 0;
}

```

7.2 Results

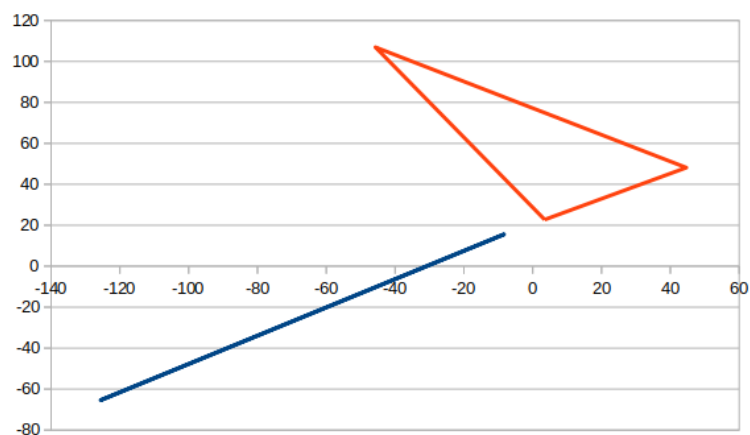
7.2.1 Failures

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

```

===== 2D static =====
Validation2D has failed
Co(-63.571705,-22.581119) x(55.239119,38.152177) y(-62.031537,-42.843548) against To(3.474294,22.751011)
x(-49.195251,84.166201) y(41.179031,-95.350316)
FMB : intersection
SAT : no intersection

```



This case can be detected and avoided prior to the intersection test by checking the determinant of the frame: degenerated frames have a null determinant. In the example above the determinant of the first frame is equal to -0.001667.

7.2.2 2D static

```
===== 2D static =====
Validation2D has succeed.
Tested 467076 intersections and 1532830 no intersections
```

7.2.3 2D dynamic

```
===== 2D dynamic =====
Validation2DTime has succeed.
Tested 744444 intersections and 1255474 no intersections
```

7.2.4 3D static

```
===== 3D static =====
Validation3D has succeed.
Tested 314980 intersections and 1685018 no intersections
```

7.2.5 3D dynamic

```
===== 3D dynamic =====
Validation3DTime has succeed.
Tested 523862 intersections and 1476138 no intersections
```

8 Qualification against SAT

In this section I introduce the code I've used to qualify the algorithm and its implementation. The qualification consists of running the FMB algorithm on randomly generated pairs of Frame, and check its execution time against the one of running the SAT algorithm on the same pair of Frames.

8.1 Code

8.1.1 2D static

```
// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
```

```

#include "fmb2d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_2D 1500

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

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

} Param2D;

// Type of qualification
typedef enum {
    typeQualif_all,
    typeQualif_nearCaseOnly
} TypeQualif;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;

```



```

double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification2DStatic(
    const Param2D paramP,
    const Param2D paramQ,
    TypeQualif typeQualif) {

    // Create the two Frames
    Frame2D P =
        Frame2DCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.comp);

    Frame2D Q =
        Frame2DCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame2D* that = &P;
    Frame2D* tho = &Q;

    // If the type of qualification is nearCaseOnly
    if (typeQualif == typeQualif_nearCaseOnly) {

        // If the AABBs of the two Frame are not in intersection
        bool isIntersectingAABB =
            AABBTestIntersection2D(
                &(that->bdgBox),
                &(tho->bdgBox));
        if (isIntersectingAABB == false) {

            // Skip the test on this pair to simulate pruning of pairs
            // of distant Frame by a prior step in a real collision
            // detection system.
            return;
        }
    }
}

```

```

    }

}

// Loop on pairs of Frames
for (
    int iPair = 2;
    iPair--;) {

    // Declare an array to memorize the results of the repeated
    // test on the same pair,
    // to prevent optimization from the compiler to remove the for loop
    bool isIntersectingFMB[NB_REPEAT_2D] = {false};

    // Start measuring time
    struct timeval start;
    gettimeofday(&start, NULL);

    // Run the FMB intersection test
    for (
        int i = NB_REPEAT_2D;
        i--;) {

        isIntersectingFMB[i] =
            FMBTestIntersection2D(
                that,
                tho,
                NULL);

    }

    // Stop measuring time
    struct timeval stop;
    gettimeofday(&stop, NULL);

    // Calculate the delay of execution
    unsigned long deltausFMB = 0;
    if (stop.tv_sec < start.tv_sec) {

        printf("time warps, try again\n");
        exit(0);

    }

    if (stop.tv_sec > start.tv_sec + 1) {

        printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
        exit(0);

    }

    if (stop.tv_usec < start.tv_usec) {

        deltausFMB = stop.tv_sec - start.tv_sec;
        deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

    } else {

        deltausFMB = stop.tv_usec - start.tv_usec;

    }
}

```

```

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_2D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
    int i = NB_REPEAT_2D;
    i--;) {

    isIntersectingSAT[i] =
        SATTestIntersection2D(
            that,
            tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
    exit(0);

}

if (stop.tv_usec < start.tv_usec) {

    deltausSAT = stop.tv_sec - start.tv_sec;
    deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

    deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

    // If FMB and SAT disagrees
    if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

        printf("Qualification has failed\n");
        Frame2DPrint(that);
        printf(" against ");
        Frame2DPrint(tho);
        printf("\n");
        printf("FMB : ");
    }
}

```

```

        if (isIntersectingFMB[0] == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT[0] == false) printf("no ");
        printf("intersection\n");

        // Stop the qualification test
        exit(0);
    }

    // Get the ratio of execution time
    double ratio = ((double)deltausFMB) / ((double)deltausSAT);

    // If the Frames intersect
    if (isIntersectingSAT[0] == true) {

        // Update the counters
        if (countInter == 0) {

            minInter = ratio;
            maxInter = ratio;

        } else {

            if (minInter > ratio) minInter = ratio;
            if (maxInter < ratio) maxInter = ratio;

        }

        sumInter += ratio;
        ++countInter;

        if (
            paramP.type == FrameCuboid &&
            paramQ.type == FrameCuboid) {

            if (countInterCC == 0) {

                minInterCC = ratio;
                maxInterCC = ratio;

            } else {

                if (minInterCC > ratio) minInterCC = ratio;
                if (maxInterCC < ratio) maxInterCC = ratio;

            }

            sumInterCC += ratio;
            ++countInterCC;

        } else if (
            paramP.type == FrameCuboid &&
            paramQ.type == FrameTetrahedron) {

            if (countInterCT == 0) {

                minInterCT = ratio;
                maxInterCT = ratio;

            } else {

```

```

        if (minInterCT > ratio) minInterCT = ratio;
        if (maxInterCT < ratio) maxInterCT = ratio;

    }

    sumInterCT += ratio;
    ++countInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countInterTC == 0) {

        minInterTC = ratio;
        maxInterTC = ratio;

    } else {

        if (minInterTC > ratio) minInterTC = ratio;
        if (maxInterTC < ratio) maxInterTC = ratio;

    }

    sumInterTC += ratio;
    ++countInterTC;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countInterTT == 0) {

        minInterTT = ratio;
        maxInterTT = ratio;

    } else {

        if (minInterTT > ratio) minInterTT = ratio;
        if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

}

// Else, the Frames do not intersect
} else {

    // Update the counters
    if (countNoInter == 0) {

        minNoInter = ratio;
        maxNoInter = ratio;

    } else {

        if (minNoInter > ratio) minNoInter = ratio;
        if (maxNoInter < ratio) maxNoInter = ratio;

    }

}

```

```

}

sumNoInter += ratio;
++countNoInter;

if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countNoInterCC == 0) {

        minNoInterCC = ratio;
        maxNoInterCC = ratio;

    } else {

        if (minNoInterCC > ratio) minNoInterCC = ratio;
        if (maxNoInterCC < ratio) maxNoInterCC = ratio;

    }

    sumNoInterCC += ratio;
    ++countNoInterCC;

} else if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterCT == 0) {

        minNoInterCT = ratio;
        maxNoInterCT = ratio;

    } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

    }

    sumNoInterCT += ratio;
    ++countNoInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

    } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

    }

    sumNoInterTC += ratio;
    ++countNoInterTC;

```

```

    } else if (
        paramP.type == FrameTetrahedron &&
        paramQ.type == FrameTetrahedron) {

        if (countNoInterTT == 0) {

            minNoInterTT = ratio;
            maxNoInterTT = ratio;

        } else {

            if (minNoInterTT > ratio) minNoInterTT = ratio;
            if (maxNoInterTT < ratio) maxNoInterTT = ratio;

        }

        sumNoInterTT += ratio;
        ++countNoInterTT;

    }

}

// Else, if time of execution for FMB was less than 10ms
} else if (deltausFMB < 10) {

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

// Else, if time of execution for SAT was less than 10ms
} else if (deltausSAT < 10) {

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

}

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

}

}

void Qualify2DStatic(TypeQualif typeQualif) {

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

    // Open the files to save the results
    FILE* fp = NULL;
    FILE* fpCC = NULL;
    FILE* fpCT = NULL;
    FILE* fpTC = NULL;
    FILE* fpTT = NULL;
    if (typeQualif == typeQualif_all) {

        fp = fopen("../Results/qualification2D.txt", "w");
        fpCC = fopen("../Results/qualification2DCC.txt", "w");
        fpCT = fopen("../Results/qualification2DCT.txt", "w");
    }
}

```

```

    fpTC = fopen("../Results/qualification2DTC.txt", "w");
    fpTT = fopen("../Results/qualification2DTT.txt", "w");

} else if (typeQualif == typeQualif_nearCaseOnly) {

    fp = fopen("../Results/qualification2Dnearcaseonly.txt", "w");
    fpCC = fopen("../Results/qualification2DCCnearcaseonly.txt", "w");
    fpCT = fopen("../Results/qualification2DCTnearcaseonly.txt", "w");
    fpTC = fopen("../Results/qualification2DTCnearcaseonly.txt", "w");
    fpTT = fopen("../Results/qualification2DTTnearcaseonly.txt", "w");

} else {

    printf("Unimplemented typeQualif in Qualifiy2DStatic\n");
    exit(1);

}

// Loop on runs
for (
    int iRun = 0;
    iRun < NB_RUNS;
    ++iRun) {

    // Ratio intersection/no intersection for the displayed results
    double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

    // Initialize counters
    minInter = 0.0;
    maxInter = 0.0;
    sumInter = 0.0;
    countInter = 0;
    minNoInter = 0.0;
    maxNoInter = 0.0;
    sumNoInter = 0.0;
    countNoInter = 0;

    minInterCC = 0.0;
    maxInterCC = 0.0;
    sumInterCC = 0.0;
    countInterCC = 0;
    minNoInterCC = 0.0;
    maxNoInterCC = 0.0;
    sumNoInterCC = 0.0;
    countNoInterCC = 0;

    minInterCT = 0.0;
    maxInterCT = 0.0;
    sumInterCT = 0.0;
    countInterCT = 0;
    minNoInterCT = 0.0;
    maxNoInterCT = 0.0;
    sumNoInterCT = 0.0;
    countNoInterCT = 0;

    minInterTC = 0.0;
    maxInterTC = 0.0;
    sumInterTC = 0.0;
    countInterTC = 0;
    minNoInterTC = 0.0;
    maxNoInterTC = 0.0;
    sumNoInterTC = 0.0;

```



```

countNoInterTC = 0;

minInterTT = 0.0;
maxInterTT = 0.0;
sumInterTT = 0.0;
countInterTT = 0;
minNoInterTT = 0.0;
maxNoInterTT = 0.0;
sumNoInterTT = 0.0;
countNoInterTT = 0;

// Declare two variables to memorize the arguments to the
// Qualification function
Param2D paramP;
Param2D paramQ;

// Loop on the number of tests
for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param2D* param = &paramP;
    for (
        int iParam = 2;
        iParam--;) {

        // 50% chance of being a Cuboid or a Tetrahedron
        if (rnd() < 0.5) {

            param->type = FrameCuboid;

        } else {

            param->type = FrameTetrahedron;

        }

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

            param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

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

                param->comp[iComp][iAxis] =
                    -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

            }

        }

        param = &paramQ;

    }

    // Calculate the determinant of the Frames' components matrix
    double detP =
        paramP.comp[0][0] * paramP.comp[1][1] -

```

```

        paramP.comp[1][0] * paramP.comp[0][1];

double detQ =
    paramQ.comp[0][0] * paramQ.comp[1][1] -
    paramQ.comp[1][0] * paramQ.comp[0][1];

// If the determinants are not null, ie the Frame are not degenerate
if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification2DStatic(
        paramP,
        paramQ,
        typeQualif);

}

}

// Save the results
if (iRun == 0) {

    fprintf(fp, "percPairInter,");
    fprintf(fp, "countInterTo,countNoInterTo,");
    fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
    fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
    fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

    fprintf(fpCC, "percPairInter,");
    fprintf(fpCC, "countInterCC,countNoInterCC,");
    fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
    fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
    fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

    fprintf(fpCT, "percPairInter,");
    fprintf(fpCT, "countInterCT,countNoInterCT,");
    fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
    fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
    fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

    fprintf(fpTC, "percPairInter,");
    fprintf(fpTC, "countInterTC,countNoInterTC,");
    fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
    fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
    fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

    fprintf(fpTT, "percPairInter,");
    fprintf(fpTT, "countInterTT,countNoInterTT,");
    fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
    fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
    fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}

fprintf(
    fp,
    "%.1f,",
    ratioInter);
fprintf(
    fp,
    "%lu,%lu,",
    countInter,

```

```

        countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minInter,
    avgInter,
    maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minNoInter,
    avgNoInter,
    maxNoInter);
double avg =
    ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
    fp,
    "%f,%f,%f",
    (minNoInter < minInter ? minNoInter : minInter),
    avg,
    (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

    fprintf(fp, "\n");

}

fprintf(
    fpCC,
    "%.1f,",
    ratioInter);
fprintf(
    fpCC,
    "%lu,%lu,",
    countInterCC,
    countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
    fpCC,
    "%f,%f,%f,",
    minInterCC,
    avgInterCC,
    maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f,",
    minNoInterCC,
    avgNoInterCC,
    maxNoInterCC);
double avgCC =
    ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f",
    (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
    avgCC,
    (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

```

```

        fprintf(fpCC, "\n");
    }

    fprintf(
        fpCT,
        "%.1f,",
        ratioInter);
    fprintf(
        fpCT,
        "%lu,%lu,",
        countInterCT,
        countNoInterCT);
    double avgInterCT = sumInterCT / (double)countInterCT;
    fprintf(
        fpCT,
        "%f,%f,%f,",
        minInterCT,
        avgInterCT,
        maxInterCT);
    double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
    fprintf(
        fpCT,
        "%f,%f,%f,",
        minNoInterCT,
        avgNoInterCT,
        maxNoInterCT);
    double avgCT =
        ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
    fprintf(
        fpCT,
        "%f,%f,%f",
        (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
        avgCT,
        (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
    if (iRun < NB_RUNS - 1) {

        fprintf(fpCT, "\n");
    }

    fprintf(
        fpTC,
        "%.1f,",
        ratioInter);
    fprintf(
        fpTC,
        "%lu,%lu,",
        countInterTC,
        countNoInterTC);
    double avgInterTC = sumInterTC / (double)countInterTC;
    fprintf(
        fpTC,
        "%f,%f,%f,",
        minInterTC,
        avgInterTC,
        maxInterTC);
    double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
    fprintf(
        fpTC,
        "%f,%f,%f,",
        minNoInterTC,

```

```

        avgNoInterTC,
        maxNoInterTC);
double avgTC =
    ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
    avgTC,
    (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTC, "\n");

}

fprintf(
    fpTT,
    "%.1f,",
    ratioInter);
fprintf(
    fpTT,
    "%lu,%lu,",
    countInterTT,
    countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    minInterTT,
    avgInterTT,
    maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    minNoInterTT,
    avgNoInterTT,
    maxNoInterTT);
double avgTT =
    ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
    avgTT,
    (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTT, "\n");

}

}

// Close the files
fclose(fp);
fclose(fpCC);
fclose(fpCT);
fclose(fpTC);
fclose(fpTT);

```

```

}

int main(int argc, char** argv) {

    TypeQualif typeQualif = typeQualif_all;

    for (
        int iArg = 0;
        iArg < argc;
        ++iArg) {

        if (strcmp(argv[iArg], "-nearCaseOnly") == 0) {

            typeQualif = typeQualif_nearCaseOnly;

        }

    }

    Qualify2DStatic(typeQualif);

    return 0;

}

```

8.1.2 3D static

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb3d.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_3D 800

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

```

```

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

} Param3D;

// Type of qualification
typedef enum {
    typeQualif_all,
    typeQualif_nearCaseOnly
} TypeQualif;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each

```

```

void Qualification3DStatic(
    const Param3D paramP,
    const Param3D paramQ,
    TypeQualif typeQualif) {

    // Create the two Frames
    Frame3D P =
        Frame3DCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.comp);

    Frame3D Q =
        Frame3DCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame3D* that = &P;
    Frame3D* tho = &Q;

    // If the type of qualification is nearCaseOnly
    if (typeQualif == typeQualif_nearCaseOnly) {

        // If the AABBs of the two Frame are not in intersection
        bool isIntersectingAABB =
            AABBTestIntersection3D(
                &(that->bdgBox),
                &(tho->bdgBox));
        if (isIntersectingAABB == false) {

            // Skip the test on this pair to simulate pruning of pairs
            // of distant Frame by a prior step in a real collision
            // detection system.
            return;

        }

    }

    // Loop on pairs of Frames
    for (
        int iPair = 2;
        iPair--;) {

        // Declare an array to memorize the results of the repeated
        // test on the same pair,
        // to prevent optimization from the compiler to remove the for loop
        bool isIntersectingFMB[NB_REPEAT_3D] = {false};

        // Start measuring time
        struct timeval start;
        gettimeofday(&start, NULL);

        // Run the FMB intersection test
        for (
            int i = NB_REPEAT_3D;
            i--;) {

            isIntersectingFMB[i] =
                FMBTestIntersection3D(

```



```

        that,
        tho,
        NULL);
    }

    // Stop measuring time
    struct timeval stop;
    gettimeofday(&stop, NULL);

    // Calculate the delay of execution
    unsigned long deltausFMB = 0;
    if (stop.tv_sec < start.tv_sec) {

        printf("time warps, try again\n");
        exit(0);
    }

    if (stop.tv_sec > start.tv_sec + 1) {

        printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
        exit(0);
    }

    if (stop.tv_usec < start.tv_usec) {

        deltausFMB = stop.tv_sec - start.tv_sec;
        deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;
    } else {

        deltausFMB = stop.tv_usec - start.tv_usec;
    }

    // Declare an array to memorize the results of the repeated
    // test on the same pair,
    // to prevent optimization from the compiler to remove the for loop
    bool isIntersectingSAT[NB_REPEAT_3D] = {false};

    // Start measuring time
    gettimeofday(&start, NULL);

    // Run the FMB intersection test
    for (
        int i = NB_REPEAT_3D;
        i--;) {

        isIntersectingSAT[i] =
            SATTestIntersection3D(
                that,
                tho);
    }

    // Stop measuring time
    gettimeofday(&stop, NULL);

    // Calculate the delay of execution
    unsigned long deltausSAT = 0;

```

```

if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
    exit(0);

}

if (stop.tv_usec < start.tv_usec) {

    deltausSAT = stop.tv_sec - start.tv_sec;
    deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

    deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

    // If FMB and SAT disagrees
    if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

        printf("Qualification has failed\n");
        Frame3DPrint(that);
        printf(" against ");
        Frame3DPrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB[0] == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT[0] == false) printf("no ");
        printf("intersection\n");

        // Stop the qualification test
        exit(0);

    }

    // Get the ratio of execution time
    double ratio = ((double)deltausFMB) / ((double)deltausSAT);

    // If the Frames intersect
    if (isIntersectingSAT[0] == true) {

        // Update the counters
        if (countInter == 0) {

            minInter = ratio;
            maxInter = ratio;

        } else {

```

```

        if (minInter > ratio) minInter = ratio;
        if (maxInter < ratio) maxInter = ratio;
    }

    sumInter += ratio;
    ++countInter;

    if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameCuboid) {

        if (countInterCC == 0) {

            minInterCC = ratio;
            maxInterCC = ratio;

        } else {

            if (minInterCC > ratio) minInterCC = ratio;
            if (maxInterCC < ratio) maxInterCC = ratio;

        }

        sumInterCC += ratio;
        ++countInterCC;

    } else if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameTetrahedron) {

        if (countInterCT == 0) {

            minInterCT = ratio;
            maxInterCT = ratio;

        } else {

            if (minInterCT > ratio) minInterCT = ratio;
            if (maxInterCT < ratio) maxInterCT = ratio;

        }

        sumInterCT += ratio;
        ++countInterCT;

    } else if (
        paramP.type == FrameTetrahedron &&
        paramQ.type == FrameCuboid) {

        if (countInterTC == 0) {

            minInterTC = ratio;
            maxInterTC = ratio;

        } else {

            if (minInterTC > ratio) minInterTC = ratio;
            if (maxInterTC < ratio) maxInterTC = ratio;

        }

    }

```

```

        sumInterTC += ratio;
        ++countInterTC;
    } else if (
        paramP.type == FrameTetrahedron &&
        paramQ.type == FrameTetrahedron) {

        if (countInterTT == 0) {

            minInterTT = ratio;
            maxInterTT = ratio;

        } else {

            if (minInterTT > ratio) minInterTT = ratio;
            if (maxInterTT < ratio) maxInterTT = ratio;

        }

        sumInterTT += ratio;
        ++countInterTT;
    }

    // Else, the Frames do not intersect
} else {

    // Update the counters
    if (countNoInter == 0) {

        minNoInter = ratio;
        maxNoInter = ratio;

    } else {

        if (minNoInter > ratio) minNoInter = ratio;
        if (maxNoInter < ratio) maxNoInter = ratio;

    }

    sumNoInter += ratio;
    ++countNoInter;

    if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameCuboid) {

        if (countNoInterCC == 0) {

            minNoInterCC = ratio;
            maxNoInterCC = ratio;

        } else {

            if (minNoInterCC > ratio) minNoInterCC = ratio;
            if (maxNoInterCC < ratio) maxNoInterCC = ratio;

        }

        sumNoInterCC += ratio;
        ++countNoInterCC;
    }
}

```

```

} else if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterCT == 0) {

        minNoInterCT = ratio;
        maxNoInterCT = ratio;

    } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

    }

    sumNoInterCT += ratio;
    ++countNoInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

    } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

    }

    sumNoInterTC += ratio;
    ++countNoInterTC;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterTT == 0) {

        minNoInterTT = ratio;
        maxNoInterTT = ratio;

    } else {

        if (minNoInterTT > ratio) minNoInterTT = ratio;
        if (maxNoInterTT < ratio) maxNoInterTT = ratio;

    }

    sumNoInterTT += ratio;
    ++countNoInterTT;

}

}

// Else, if time of execution for FMB was less than 10ms

```

```

    } else if (deltausFMB < 10) {

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

        // Else, if time of execution for SAT was less than 10ms
    } else if (deltausSAT < 10) {

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

    }

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

}

}

void Qualify3DStatic(TypeQualif typeQualif) {

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

    // Open the files to save the results
    FILE* fp = NULL;
    FILE* fpCC = NULL;
    FILE* fpCT = NULL;
    FILE* fpTC = NULL;
    FILE* fpTT = NULL;
    if (typeQualif == typeQualif_all) {

        fp = fopen("../Results/qualification3D.txt", "w");
        fpCC = fopen("../Results/qualification3DCC.txt", "w");
        fpCT = fopen("../Results/qualification3DCT.txt", "w");
        fpTC = fopen("../Results/qualification3DTC.txt", "w");
        fpTT = fopen("../Results/qualification3DTT.txt", "w");

    } else if (typeQualif == typeQualif_nearCaseOnly) {

        fp = fopen("../Results/qualification3Dnearcaseonly.txt", "w");
        fpCC = fopen("../Results/qualification3DCCnearcaseonly.txt", "w");
        fpCT = fopen("../Results/qualification3DCTnearcaseonly.txt", "w");
        fpTC = fopen("../Results/qualification3DTCnearcaseonly.txt", "w");
        fpTT = fopen("../Results/qualification3DTTnearcaseonly.txt", "w");

    } else {

        printf("Unimplemented typeQualif in Qualify3DStatic\n");
        exit(1);

    }

    // Loop on runs
    for (
        int iRun = 0;
        iRun < NB_RUNS;
        ++iRun) {

        // Ratio intersection/no intersection for the displayed results

```

```

double ratioInter = 0.1 + 0.8 * (double)iRun / (double)(NB_RUNS - 1);

// Initialize counters
minInter = 0.0;
maxInter = 0.0;
sumInter = 0.0;
countInter = 0;
minNoInter = 0.0;
maxNoInter = 0.0;
sumNoInter = 0.0;
countNoInter = 0;

minInterCC = 0.0;
maxInterCC = 0.0;
sumInterCC = 0.0;
countInterCC = 0;
minNoInterCC = 0.0;
maxNoInterCC = 0.0;
sumNoInterCC = 0.0;
countNoInterCC = 0;

minInterCT = 0.0;
maxInterCT = 0.0;
sumInterCT = 0.0;
countInterCT = 0;
minNoInterCT = 0.0;
maxNoInterCT = 0.0;
sumNoInterCT = 0.0;
countNoInterCT = 0;

minInterTC = 0.0;
maxInterTC = 0.0;
sumInterTC = 0.0;
countInterTC = 0;
minNoInterTC = 0.0;
maxNoInterTC = 0.0;
sumNoInterTC = 0.0;
countNoInterTC = 0;

minInterTT = 0.0;
maxInterTT = 0.0;
sumInterTT = 0.0;
countInterTT = 0;
minNoInterTT = 0.0;
maxNoInterTT = 0.0;
sumNoInterTT = 0.0;
countNoInterTT = 0;

// Declare two variables to memorize the arguments to the
// Qualification function
Param3D paramP;
Param3D paramQ;

// Loop on the number of tests
for (
    unsigned long iTest = NB_TESTS;
    iTest--;) {

    // Create two random Frame definitions
    Param3D* param = &paramP;
    for (
        int iParam = 2;

```

```

iParam--;) {

// 50% chance of being a Cuboid or a Tetrahedron
if (rnd() < 0.5) {

    param->type = FrameCuboid;

} else {

    param->type = FrameTetrahedron;

}

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

    param->orig[iAxis] = -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

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

        param->comp[iComp][iAxis] =
            -RANGE_AXIS + 2.0 * rnd() * RANGE_AXIS;

    }

}

param = &paramQ;

}

// Calculate the determinant of the Frames' components matrix
double detP =
    paramP.comp[0][0] * (paramP.comp[1][1] * paramP.comp[2][2] -
    paramP.comp[1][2] * paramP.comp[2][1]) -
    paramP.comp[1][0] * (paramP.comp[0][1] * paramP.comp[2][2] -
    paramP.comp[0][2] * paramP.comp[2][1]) +
    paramP.comp[2][0] * (paramP.comp[0][1] * paramP.comp[1][2] -
    paramP.comp[0][2] * paramP.comp[1][1]);

double detQ =
    paramQ.comp[0][0] * (paramQ.comp[1][1] * paramQ.comp[2][2] -
    paramQ.comp[1][2] * paramQ.comp[2][1]) -
    paramQ.comp[1][0] * (paramQ.comp[0][1] * paramQ.comp[2][2] -
    paramQ.comp[0][2] * paramQ.comp[2][1]) +
    paramQ.comp[2][0] * (paramQ.comp[0][1] * paramQ.comp[1][2] -
    paramQ.comp[0][2] * paramQ.comp[1][1]);

// If the determinants are not null, ie the Frame are not degenerate
if (fabs(detP) > EPSILON && fabs(detQ) > EPSILON) {

    // Run the validation on the two Frames
    Qualification3DStatic(
        paramP,
        paramQ,
        typeQualif);

}

```



```

}

// Save the results
if (iRun == 0) {

    fprintf(fp, "percPairInter,");
    fprintf(fp, "countInterTo,countNoInterTo,");
    fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
    fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
    fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

    fprintf(fpCC, "percPairInter,");
    fprintf(fpCC, "countInterCC,countNoInterCC,");
    fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
    fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
    fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

    fprintf(fpCT, "percPairInter,");
    fprintf(fpCT, "countInterCT,countNoInterCT,");
    fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
    fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
    fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

    fprintf(fpTC, "percPairInter,");
    fprintf(fpTC, "countInterTC,countNoInterTC,");
    fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
    fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
    fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

    fprintf(fpTT, "percPairInter,");
    fprintf(fpTT, "countInterTT,countNoInterTT,");
    fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
    fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
    fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");

}

fprintf(
    fp,
    "%.1f,",
    ratioInter);
fprintf(
    fp,
    "%lu,%lu,",
    countInter,
    countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minInter,
    avgInter,
    maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minNoInter,
    avgNoInter,
    maxNoInter);
double avg =
    ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;

```

```

fprintf(
    fp,
    "%f,%f,%f",
    (minNoInter < minInter ? minNoInter : minInter),
    avg,
    (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

    fprintf(fp, "\n");

}

fprintf(
    fpCC,
    "%.1f,",
    ratioInter);
fprintf(
    fpCC,
    "%lu,%lu,",
    countInterCC,
    countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(
    fpCC,
    "%f,%f,%f",
    minInterCC,
    avgInterCC,
    maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f",
    minNoInterCC,
    avgNoInterCC,
    maxNoInterCC);
double avgCC =
    ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f",
    (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
    avgCC,
    (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpCC, "\n");

}

fprintf(
    fpCT,
    "%.1f,",
    ratioInter);
fprintf(
    fpCT,
    "%lu,%lu,",
    countInterCT,
    countNoInterCT);
double avgInterCT = sumInterCT / (double)countInterCT;
fprintf(
    fpCT,
    "%f,%f,%f",

```

```

        minInterCT,
        avgInterCT,
        maxInterCT);
double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
fprintf(
    fpCT,
    "%f,%f,%f,",
    minNoInterCT,
    avgNoInterCT,
    maxNoInterCT);
double avgCT =
    ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
    fpCT,
    "%f,%f,%f",
    (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
    avgCT,
    (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpCT, "\n");

}

fprintf(
    fpTC,
    "%.1f,",
    ratioInter);
fprintf(
    fpTC,
    "%lu,%lu,",
    countInterTC,
    countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
    fpTC,
    "%f,%f,%f,",
    minInterTC,
    avgInterTC,
    maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f,",
    minNoInterTC,
    avgNoInterTC,
    maxNoInterTC);
double avgTC =
    ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
    avgTC,
    (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTC, "\n");

}

fprintf(

```

```

        fpTT,
        "%.1f,",
        ratioInter);
fprintf(
    fpTT,
    "%lu,%lu,",
    countInterTT,
    countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minInterTT,
    avgInterTT,
    maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minNoInterTT,
    avgNoInterTT,
    maxNoInterTT);
double avgTT =
    ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
    avgTT,
    (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTT, "\n");

}

}

// Close the files
fclose(fp);
fclose(fpCC);
fclose(fpCT);
fclose(fpTC);
fclose(fpTT);

}

int main(int argc, char** argv) {

    TypeQualif typeQualif = typeQualif_all;

    for (
        int iArg = 0;
        iArg < argc;
        ++iArg) {

        if (strcmp(argv[iArg], "-nearCaseOnly") == 0) {

            typeQualif = typeQualif_nearCaseOnly;

        }

    }

}

```

```

    }

    Qualify3DStatic(typeQualif);

    return 0;
}

```

8.1.3 2D dynamic

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb2dt.h"
#include "sat.h"

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_2D 1500

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

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

} Param2DTime;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

```

```

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;
double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification2DDynamic(
    const Param2DTime paramP,
    const Param2DTime paramQ) {

    // Create the two Frames
    Frame2DTime P =
        Frame2DTimeCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.speed,
            paramP.comp);

    Frame2DTime Q =
        Frame2DTimeCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.speed,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame2DTime* that = &P;
    Frame2DTime* tho = &Q;

```

```

// Loop on pairs of Frames
for (
    int iPair = 2;
    iPair--;) {

    // Declare an array to memorize the results of the repeated
    // test on the same pair,
    // to prevent optimization from the compiler to remove the for loop
    bool isIntersectingFMB[NB_REPEAT_2D] = {false};

    // Start measuring time
    struct timeval start;
    gettimeofday(&start, NULL);

    // Run the FMB intersection test
    for (
        int i = NB_REPEAT_2D;
        i--;) {

        isIntersectingFMB[i] =
            FMBTestIntersection2DTime(
                that,
                tho,
                NULL);

    }

    // Stop measuring time
    struct timeval stop;
    gettimeofday(&stop, NULL);

    // Calculate the delay of execution
    unsigned long deltausFMB = 0;
    if (stop.tv_sec < start.tv_sec) {

        printf("time warps, try again\n");
        exit(0);

    }

    if (stop.tv_sec > start.tv_sec + 1) {

        printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
        exit(0);

    }

    if (stop.tv_usec < start.tv_usec) {

        deltausFMB = stop.tv_sec - start.tv_sec;
        deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

    } else {

        deltausFMB = stop.tv_usec - start.tv_usec;

    }

    // Declare an array to memorize the results of the repeated
    // test on the same pair,
    // to prevent optimization from the compiler to remove the for loop

```

```

bool isIntersectingSAT[NB_REPEAT_2D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
    int i = NB_REPEAT_2D;
    i--;) {

    isIntersectingSAT[i] =
        SATTestIntersection2DTime(
            that,
            tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
    exit(0);

}

if (stop.tv_usec < start.tv_usec) {

    deltausSAT = stop.tv_sec - start.tv_sec;
    deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;

} else {

    deltausSAT = stop.tv_usec - start.tv_usec;

}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

    // If FMB and SAT disagrees
    if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

        printf("Qualification has failed\n");
        Frame2DTimePrint(that);
        printf(" against ");
        Frame2DTimePrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB[0] == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
    }
}

```



```

    if (isIntersectingSAT[0] == false) printf("no ");
    printf("intersection\n");

    // Stop the qualification test
    exit(0);
}

// Get the ratio of execution time
double ratio = ((double)deltausFMB) / ((double)deltausSAT);

// If the Frames intersect
if (isIntersectingSAT[0] == true) {

    // Update the counters
    if (countInter == 0) {

        minInter = ratio;
        maxInter = ratio;

    } else {

        if (minInter > ratio) minInter = ratio;
        if (maxInter < ratio) maxInter = ratio;

    }

    sumInter += ratio;
    ++countInter;

    if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameCuboid) {

        if (countInterCC == 0) {

            minInterCC = ratio;
            maxInterCC = ratio;

        } else {

            if (minInterCC > ratio) minInterCC = ratio;
            if (maxInterCC < ratio) maxInterCC = ratio;

        }

        sumInterCC += ratio;
        ++countInterCC;

    } else if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameTetrahedron) {

        if (countInterCT == 0) {

            minInterCT = ratio;
            maxInterCT = ratio;

        } else {

            if (minInterCT > ratio) minInterCT = ratio;
            if (maxInterCT < ratio) maxInterCT = ratio;

        }

    }
}

```

```

    }

    sumInterCT += ratio;
    ++countInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countInterTC == 0) {

        minInterTC = ratio;
        maxInterTC = ratio;

    } else {

        if (minInterTC > ratio) minInterTC = ratio;
        if (maxInterTC < ratio) maxInterTC = ratio;

    }

    sumInterTC += ratio;
    ++countInterTC;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countInterTT == 0) {

        minInterTT = ratio;
        maxInterTT = ratio;

    } else {

        if (minInterTT > ratio) minInterTT = ratio;
        if (maxInterTT < ratio) maxInterTT = ratio;

    }

    sumInterTT += ratio;
    ++countInterTT;

}

// Else, the Frames do not intersect
} else {

    // Update the counters
    if (countNoInter == 0) {

        minNoInter = ratio;
        maxNoInter = ratio;

    } else {

        if (minNoInter > ratio) minNoInter = ratio;
        if (maxNoInter < ratio) maxNoInter = ratio;

    }

}

```

```

sumNoInter += ratio;
++countNoInter;

if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameCuboid) {

    if (countNoInterCC == 0) {

        minNoInterCC = ratio;
        maxNoInterCC = ratio;

    } else {

        if (minNoInterCC > ratio) minNoInterCC = ratio;
        if (maxNoInterCC < ratio) maxNoInterCC = ratio;

    }

    sumNoInterCC += ratio;
    ++countNoInterCC;

} else if (
    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterCT == 0) {

        minNoInterCT = ratio;
        maxNoInterCT = ratio;

    } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

    }

    sumNoInterCT += ratio;
    ++countNoInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

    } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

    }

    sumNoInterTC += ratio;
    ++countNoInterTC;

} else if (
    paramP.type == FrameTetrahedron &&

```

```

        paramQ.type == FrameTetrahedron) {

        if (countNoInterTT == 0) {

            minNoInterTT = ratio;
            maxNoInterTT = ratio;

        } else {

            if (minNoInterTT > ratio) minNoInterTT = ratio;
            if (maxNoInterTT < ratio) maxNoInterTT = ratio;

        }

        sumNoInterTT += ratio;
        ++countNoInterTT;

    }

}

// Else, if time of execution for FMB was less than 10ms
} else if (deltausFMB < 10) {

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

// Else, if time of execution for SAT was less than 10ms
} else if (deltausSAT < 10) {

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

}

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

}

}

void Qualify2DDynamic(void) {

    // Initialise the random generator
    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);

}

}

// Save the results
if (iRun == 0) {

    fprintf(fp, "percPairInter,");
    fprintf(fp, "countInterTo,countNoInterTo,");
    fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");

```

```

fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

fprintf(fpCC, "percPairInter,");
fprintf(fpCC, "countInterCC,countNoInterCC,");
fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

fprintf(fpCT, "percPairInter,");
fprintf(fpCT, "countInterCT,countNoInterCT,");
fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");

fprintf(fpTC, "percPairInter,");
fprintf(fpTC, "countInterTC,countNoInterTC,");
fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

fprintf(fpTT, "percPairInter,");
fprintf(fpTT, "countInterTT,countNoInterTT,");
fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");
}

fprintf(
    fp,
    "%.1f,",
    ratioInter);
fprintf(
    fp,
    "%lu,%lu,",
    countInter,
    countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minInter,
    avgInter,
    maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minNoInter,
    avgNoInter,
    maxNoInter);
double avg =
    ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
    fp,
    "%f,%f,%f",
    (minNoInter < minInter ? minNoInter : minInter),
    avg,
    (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

```

```

        fprintf(fp, "\n");
    }

    fprintf(
        fpCC,
        "%.1f,",
        ratioInter);
    fprintf(
        fpCC,
        "%lu,%lu,",
        countInterCC,
        countNoInterCC);
    double avgInterCC = sumInterCC / (double)countInterCC;
    fprintf(
        fpCC,
        "%f,%f,%f,",
        minInterCC,
        avgInterCC,
        maxInterCC);
    double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
    fprintf(
        fpCC,
        "%f,%f,%f,",
        minNoInterCC,
        avgNoInterCC,
        maxNoInterCC);
    double avgCC =
        ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
    fprintf(
        fpCC,
        "%f,%f,%f",
        (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
        avgCC,
        (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
    if (iRun < NB_RUNS - 1) {

        fprintf(fpCC, "\n");
    }

    fprintf(
        fpCT,
        "%.1f,",
        ratioInter);
    fprintf(
        fpCT,
        "%lu,%lu,",
        countInterCT,
        countNoInterCT);
    double avgInterCT = sumInterCT / (double)countInterCT;
    fprintf(
        fpCT,
        "%f,%f,%f,",
        minInterCT,
        avgInterCT,
        maxInterCT);
    double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
    fprintf(
        fpCT,
        "%f,%f,%f,",
        minNoInterCT,

```



```

        avgNoInterCT,
        maxNoInterCT);
double avgCT =
    ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
    fpCT,
    "%f,%f,%f",
    (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
    avgCT,
    (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpCT, "\n");

}

fprintf(
    fpTC,
    "%.1f,",
    ratioInter);
fprintf(
    fpTC,
    "%lu,%lu,",
    countInterTC,
    countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    minInterTC,
    avgInterTC,
    maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    minNoInterTC,
    avgNoInterTC,
    maxNoInterTC);
double avgTC =
    ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
    avgTC,
    (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTC, "\n");

}

fprintf(
    fpTT,
    "%.1f,",
    ratioInter);
fprintf(
    fpTT,
    "%lu,%lu,",
    countInterTT,
    countNoInterTT);

```

```

double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minInterTT,
    avgInterTT,
    maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minNoInterTT,
    avgNoInterTT,
    maxNoInterTT);
double avgTT =
    ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
    avgTT,
    (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTT, "\n");

}

}

// Close the files
fclose(fp);
fclose(fpCC);
fclose(fpCT);
fclose(fpTC);
fclose(fpTT);

}

int main(int argc, char** argv) {

    Qualify2DDynamic();

    return 0;

}

```

8.1.4 3D dynamic

```

// Include standard libraries
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <time.h>
#include <sys/time.h>

// Include FMB and SAT algorithm library
#include "fmb3dt.h"
#include "sat.h"

```

```

// Epsilon to detect degenerated triangles
#define EPSILON 0.1

// Range of values for the random generation of Frames
#define RANGE_AXIS 100.0

// Nb of run
#define NB_RUNS 9

// Nb of tests per run
#define NB_TESTS 500000

// Nb of times the test is run on one pair of frame, used to
// slow down the processus and be able to measure time
#define NB_REPEAT_3D 800

// Helper macro to generate random number in [0.0, 1.0]
#define rnd() (double)(rand())/(double)(RAND_MAX)

// Helper structure to pass arguments to the Qualification function
typedef struct {

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

} Param3DTime;

// Global variables to count nb of tests resulting in intersection
// and no intersection, and min/max/total time of execution for each
double minInter;
double maxInter;
double sumInter;
unsigned long countInter;
double minNoInter;
double maxNoInter;
double sumNoInter;
unsigned long countNoInter;

double minInterCC;
double maxInterCC;
double sumInterCC;
unsigned long countInterCC;
double minNoInterCC;
double maxNoInterCC;
double sumNoInterCC;
unsigned long countNoInterCC;

double minInterCT;
double maxInterCT;
double sumInterCT;
unsigned long countInterCT;
double minNoInterCT;
double maxNoInterCT;
double sumNoInterCT;
unsigned long countNoInterCT;

double minInterTC;
double maxInterTC;
double sumInterTC;
unsigned long countInterTC;

```

```

double minNoInterTC;
double maxNoInterTC;
double sumNoInterTC;
unsigned long countNoInterTC;

double minInterTT;
double maxInterTT;
double sumInterTT;
unsigned long countInterTT;
double minNoInterTT;
double maxNoInterTT;
double sumNoInterTT;
unsigned long countNoInterTT;

// Qualification function
// Takes two Frame definition as input, run the intersection test on
// them with FMB and SAT, and measure the time of execution of each
void Qualification3DDynamic(
    const Param3DTime paramP,
    const Param3DTime paramQ) {

    // Create the two Frames
    Frame3DTime P =
        Frame3DTimeCreateStatic(
            paramP.type,
            paramP.orig,
            paramP.speed,
            paramP.comp);

    Frame3DTime Q =
        Frame3DTimeCreateStatic(
            paramQ.type,
            paramQ.orig,
            paramQ.speed,
            paramQ.comp);

    // Helper variables to loop on the pair (that, tho) and (tho, that)
    Frame3DTime* that = &P;
    Frame3DTime* tho = &Q;

    // Loop on pairs of Frames
    for (
        int iPair = 2;
        iPair--;) {

        // Declare an array to memorize the results of the repeated
        // test on the same pair,
        // to prevent optimization from the compiler to remove the for loop
        bool isIntersectingFMB[NB_REPEAT_3D] = {false};

        // Start measuring time
        struct timeval start;
        gettimeofday(&start, NULL);

        // Run the FMB intersection test
        for (
            int i = NB_REPEAT_3D;
            i--;) {

            isIntersectingFMB[i] =
                FMBTestIntersection3DTime(
                    that,

```

```

        tho,
        NULL);

}

// Stop measuring time
struct timeval stop;
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausFMB = 0;
if (stop.tv_sec < start.tv_sec) {

    printf("time warps, try again\n");
    exit(0);

}

if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausFMB >> 1s, decrease NB_REPEAT\n");
    exit(0);

}

if (stop.tv_usec < start.tv_usec) {

    deltausFMB = stop.tv_sec - start.tv_sec;
    deltausFMB += stop.tv_usec + 1000000 - start.tv_usec;

} else {

    deltausFMB = stop.tv_usec - start.tv_usec;

}

// Declare an array to memorize the results of the repeated
// test on the same pair,
// to prevent optimization from the compiler to remove the for loop
bool isIntersectingSAT[NB_REPEAT_3D] = {false};

// Start measuring time
gettimeofday(&start, NULL);

// Run the FMB intersection test
for (
    int i = NB_REPEAT_3D;
    i--;) {

    isIntersectingSAT[i] =
        SATTestIntersection3DTime(
            that,
            tho);

}

// Stop measuring time
gettimeofday(&stop, NULL);

// Calculate the delay of execution
unsigned long deltausSAT = 0;
if (stop.tv_sec < start.tv_sec) {

```

```

    printf("time warps, try again\n");
    exit(0);
}

if (stop.tv_sec > start.tv_sec + 1) {

    printf("deltausSAT >> 1s, decrease NB_REPEAT\n");
    exit(0);
}

if (stop.tv_usec < start.tv_usec) {

    deltausSAT = stop.tv_sec - start.tv_sec;
    deltausSAT += stop.tv_usec + 1000000 - start.tv_usec;
} else {

    deltausSAT = stop.tv_usec - start.tv_usec;
}

// If the delays are greater than 10ms
if (deltausFMB >= 10 && deltausSAT >= 10) {

    // If FMB and SAT disagrees
    if (isIntersectingFMB[0] != isIntersectingSAT[0]) {

        printf("Qualification has failed\n");
        Frame3DTimePrint(that);
        printf(" against ");
        Frame3DTimePrint(tho);
        printf("\n");
        printf("FMB : ");
        if (isIntersectingFMB[0] == false) printf("no ");
        printf("intersection\n");
        printf("SAT : ");
        if (isIntersectingSAT[0] == false) printf("no ");
        printf("intersection\n");

        // Stop the qualification test
        exit(0);
    }

    // Get the ratio of execution time
    double ratio = ((double)deltausFMB) / ((double)deltausSAT);

    // If the Frames intersect
    if (isIntersectingSAT[0] == true) {

        // Update the counters
        if (countInter == 0) {

            minInter = ratio;
            maxInter = ratio;
        } else {

            if (minInter > ratio) minInter = ratio;

```

```

        if (maxInter < ratio) maxInter = ratio;
    }

    sumInter += ratio;
    ++countInter;

    if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameCuboid) {

        if (countInterCC == 0) {

            minInterCC = ratio;
            maxInterCC = ratio;

        } else {

            if (minInterCC > ratio) minInterCC = ratio;
            if (maxInterCC < ratio) maxInterCC = ratio;

        }

        sumInterCC += ratio;
        ++countInterCC;

    } else if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameTetrahedron) {

        if (countInterCT == 0) {

            minInterCT = ratio;
            maxInterCT = ratio;

        } else {

            if (minInterCT > ratio) minInterCT = ratio;
            if (maxInterCT < ratio) maxInterCT = ratio;

        }

        sumInterCT += ratio;
        ++countInterCT;

    } else if (
        paramP.type == FrameTetrahedron &&
        paramQ.type == FrameCuboid) {

        if (countInterTC == 0) {

            minInterTC = ratio;
            maxInterTC = ratio;

        } else {

            if (minInterTC > ratio) minInterTC = ratio;
            if (maxInterTC < ratio) maxInterTC = ratio;

        }

        sumInterTC += ratio;

```

```

        ++countInterTC;

    } else if (
        paramP.type == FrameTetrahedron &&
        paramQ.type == FrameTetrahedron) {

        if (countInterTT == 0) {

            minInterTT = ratio;
            maxInterTT = ratio;

        } else {

            if (minInterTT > ratio) minInterTT = ratio;
            if (maxInterTT < ratio) maxInterTT = ratio;

        }

        sumInterTT += ratio;
        ++countInterTT;

    }

    // Else, the Frames do not intersect
} else {

    // Update the counters
    if (countNoInter == 0) {

        minNoInter = ratio;
        maxNoInter = ratio;

    } else {

        if (minNoInter > ratio) minNoInter = ratio;
        if (maxNoInter < ratio) maxNoInter = ratio;

    }

    sumNoInter += ratio;
    ++countNoInter;

    if (
        paramP.type == FrameCuboid &&
        paramQ.type == FrameCuboid) {

        if (countNoInterCC == 0) {

            minNoInterCC = ratio;
            maxNoInterCC = ratio;

        } else {

            if (minNoInterCC > ratio) minNoInterCC = ratio;
            if (maxNoInterCC < ratio) maxNoInterCC = ratio;

        }

        sumNoInterCC += ratio;
        ++countNoInterCC;

    } else if (

```



```

    paramP.type == FrameCuboid &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterCT == 0) {

        minNoInterCT = ratio;
        maxNoInterCT = ratio;

    } else {

        if (minNoInterCT > ratio) minNoInterCT = ratio;
        if (maxNoInterCT < ratio) maxNoInterCT = ratio;

    }

    sumNoInterCT += ratio;
    ++countNoInterCT;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameCuboid) {

    if (countNoInterTC == 0) {

        minNoInterTC = ratio;
        maxNoInterTC = ratio;

    } else {

        if (minNoInterTC > ratio) minNoInterTC = ratio;
        if (maxNoInterTC < ratio) maxNoInterTC = ratio;

    }

    sumNoInterTC += ratio;
    ++countNoInterTC;

} else if (
    paramP.type == FrameTetrahedron &&
    paramQ.type == FrameTetrahedron) {

    if (countNoInterTT == 0) {

        minNoInterTT = ratio;
        maxNoInterTT = ratio;

    } else {

        if (minNoInterTT > ratio) minNoInterTT = ratio;
        if (maxNoInterTT < ratio) maxNoInterTT = ratio;

    }

    sumNoInterTT += ratio;
    ++countNoInterTT;

}

}

// Else, if time of execution for FMB was less than 10ms
} else if (deltausFMB < 10) {

```

```

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

// Else, if time of execution for SAT was less than 10ms
} else if (deltausSAT < 10) {

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

}

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

}

}

void Qualify3DDynamic(void) {

    // Initialise the random generator
    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);

}

}

// Save the results
if (iRun == 0) {

    fprintf(fp, "percPairInter,");
    fprintf(fp, "countInterTo,countNoInterTo,");
    fprintf(fp, "minInterTo,avgInterTo,maxInterTo,");
    fprintf(fp, "minNoInterTo,avgNoInterTo,maxNoInterTo,");
    fprintf(fp, "minTotalTo,avgTotalTo,maxTotalTo\n");

    fprintf(fpCC, "percPairInter,");
    fprintf(fpCC, "countInterCC,countNoInterCC,");
    fprintf(fpCC, "minInterCC,avgInterCC,maxInterCC,");
    fprintf(fpCC, "minNoInterCC,avgNoInterCC,maxNoInterCC,");
    fprintf(fpCC, "minTotalCC,avgTotalCC,maxTotalCC\n");

    fprintf(fpCT, "percPairInter,");
    fprintf(fpCT, "countInterCT,countNoInterCT,");
    fprintf(fpCT, "minInterCT,avgInterCT,maxInterCT,");
    fprintf(fpCT, "minNoInterCT,avgNoInterCT,maxNoInterCT,");
    fprintf(fpCT, "minTotalCT,avgTotalCT,maxTotalCT\n");
}

```

```

    fprintf(fpTC, "percPairInter,");
    fprintf(fpTC, "countInterTC,countNoInterTC,");
    fprintf(fpTC, "minInterTC,avgInterTC,maxInterTC,");
    fprintf(fpTC, "minNoInterTC,avgNoInterTC,maxNoInterTC,");
    fprintf(fpTC, "minTotalTC,avgTotalTC,maxTotalTC\n");

    fprintf(fpTT, "percPairInter,");
    fprintf(fpTT, "countInterTT,countNoInterTT,");
    fprintf(fpTT, "minInterTT,avgInterTT,maxInterTT,");
    fprintf(fpTT, "minNoInterTT,avgNoInterTT,maxNoInterTT,");
    fprintf(fpTT, "minTotalTT,avgTotalTT,maxTotalTT\n");
}

fprintf(
    fp,
    "%.1f,",
    ratioInter);
fprintf(
    fp,
    "%lu,%lu,",
    countInter,
    countNoInter);
double avgInter = sumInter / (double)countInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minInter,
    avgInter,
    maxInter);
double avgNoInter = sumNoInter / (double)countNoInter;
fprintf(
    fp,
    "%f,%f,%f,",
    minNoInter,
    avgNoInter,
    maxNoInter);
double avg =
    ratioInter * avgInter + (1.0 - ratioInter) * avgNoInter;
fprintf(
    fp,
    "%f,%f,%f",
    (minNoInter < minInter ? minNoInter : minInter),
    avg,
    (maxNoInter > maxInter ? maxNoInter : maxInter));
if (iRun < NB_RUNS - 1) {

    fprintf(fp, "\n");
}

fprintf(
    fpCC,
    "%.1f,",
    ratioInter);
fprintf(
    fpCC,
    "%lu,%lu,",
    countInterCC,
    countNoInterCC);
double avgInterCC = sumInterCC / (double)countInterCC;
fprintf(

```

```

        fpCC,
        "%f,%f,%f,",
        minInterCC,
        avgInterCC,
        maxInterCC);
double avgNoInterCC = sumNoInterCC / (double)countNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f,",
    minNoInterCC,
    avgNoInterCC,
    maxNoInterCC);
double avgCC =
    ratioInter * avgInterCC + (1.0 - ratioInter) * avgNoInterCC;
fprintf(
    fpCC,
    "%f,%f,%f",
    (minNoInterCC < minInterCC ? minNoInterCC : minInterCC),
    avgCC,
    (maxNoInterCC > maxInterCC ? maxNoInterCC : maxInterCC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpCC, "\n");

}

fprintf(
    fpCT,
    "%.1f,",
    ratioInter);
fprintf(
    fpCT,
    "%lu,%lu,",
    countInterCT,
    countNoInterCT);
double avgInterCT = sumInterCT / (double)countInterCT;
fprintf(
    fpCT,
    "%f,%f,%f,",
    minInterCT,
    avgInterCT,
    maxInterCT);
double avgNoInterCT = sumNoInterCT / (double)countNoInterCT;
fprintf(
    fpCT,
    "%f,%f,%f,",
    minNoInterCT,
    avgNoInterCT,
    maxNoInterCT);
double avgCT =
    ratioInter * avgInterCT + (1.0 - ratioInter) * avgNoInterCT;
fprintf(
    fpCT,
    "%f,%f,%f",
    (minNoInterCT < minInterCT ? minNoInterCT : minInterCT),
    avgCT,
    (maxNoInterCT > maxInterCT ? maxNoInterCT : maxInterCT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpCT, "\n");

}

```

```

fprintf(
    fpTC,
    "%.1f,",
    ratioInter);
fprintf(
    fpTC,
    "%lu,%lu,",
    countInterTC,
    countNoInterTC);
double avgInterTC = sumInterTC / (double)countInterTC;
fprintf(
    fpTC,
    "%f,%f,%f,",
    minInterTC,
    avgInterTC,
    maxInterTC);
double avgNoInterTC = sumNoInterTC / (double)countNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f,",
    minNoInterTC,
    avgNoInterTC,
    maxNoInterTC);
double avgTC =
    ratioInter * avgInterTC + (1.0 - ratioInter) * avgNoInterTC;
fprintf(
    fpTC,
    "%f,%f,%f",
    (minNoInterTC < minInterTC ? minNoInterTC : minInterTC),
    avgTC,
    (maxNoInterTC > maxInterTC ? maxNoInterTC : maxInterTC));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTC, "\n");

}

fprintf(
    fpTT,
    "%.1f,",
    ratioInter);
fprintf(
    fpTT,
    "%lu,%lu,",
    countInterTT,
    countNoInterTT);
double avgInterTT = sumInterTT / (double)countInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minInterTT,
    avgInterTT,
    maxInterTT);
double avgNoInterTT = sumNoInterTT / (double)countNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f,",
    minNoInterTT,
    avgNoInterTT,
    maxNoInterTT);
double avgTT =

```

```

        ratioInter * avgInterTT + (1.0 - ratioInter) * avgNoInterTT;
fprintf(
    fpTT,
    "%f,%f,%f",
    (minNoInterTT < minInterTT ? minNoInterTT : minInterTT),
    avgTT,
    (maxNoInterTT > maxInterTT ? maxNoInterTT : maxInterTT));
if (iRun < NB_RUNS - 1) {

    fprintf(fpTT, "\n");

}

}

// Close the files
fclose(fp);
fclose(fpCC);
fclose(fpCT);
fclose(fpTC);
fclose(fpTT);

}

int main(int argc, char** argv) {

    Qualify3DDynamic();

    return 0;

}

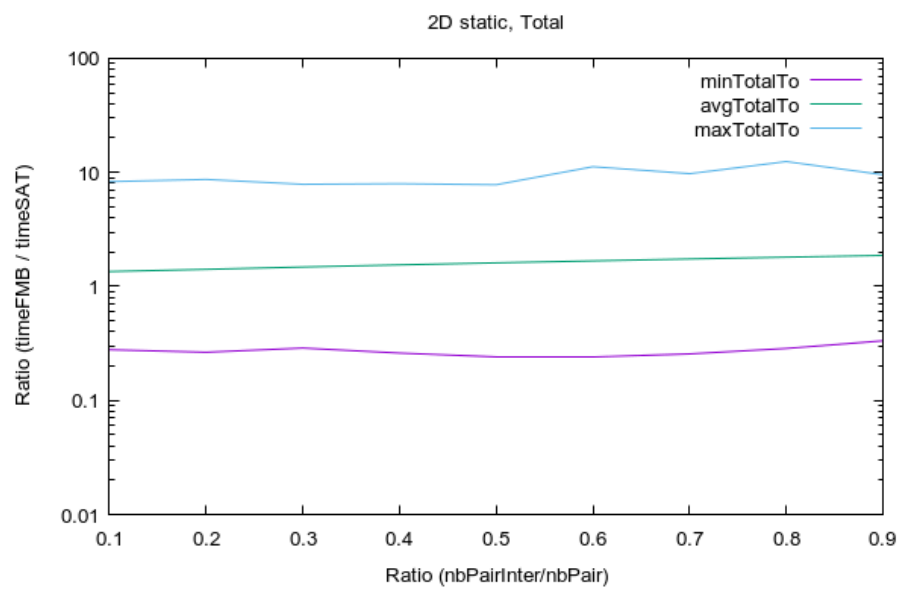
```

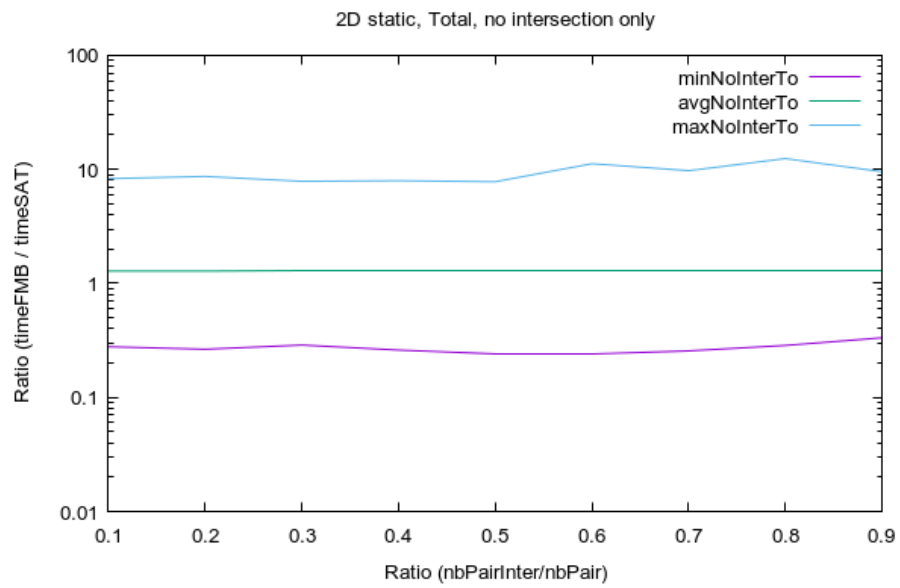
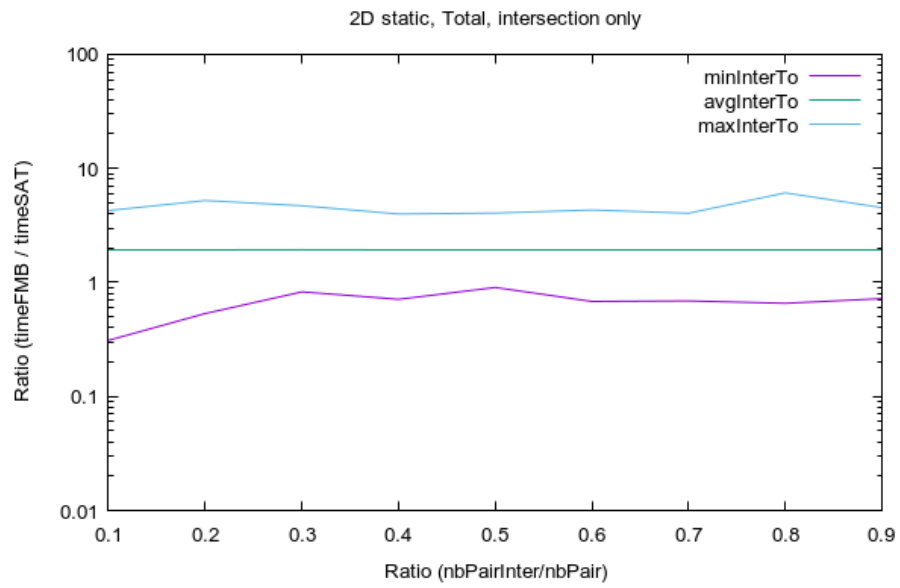
8.2 Results

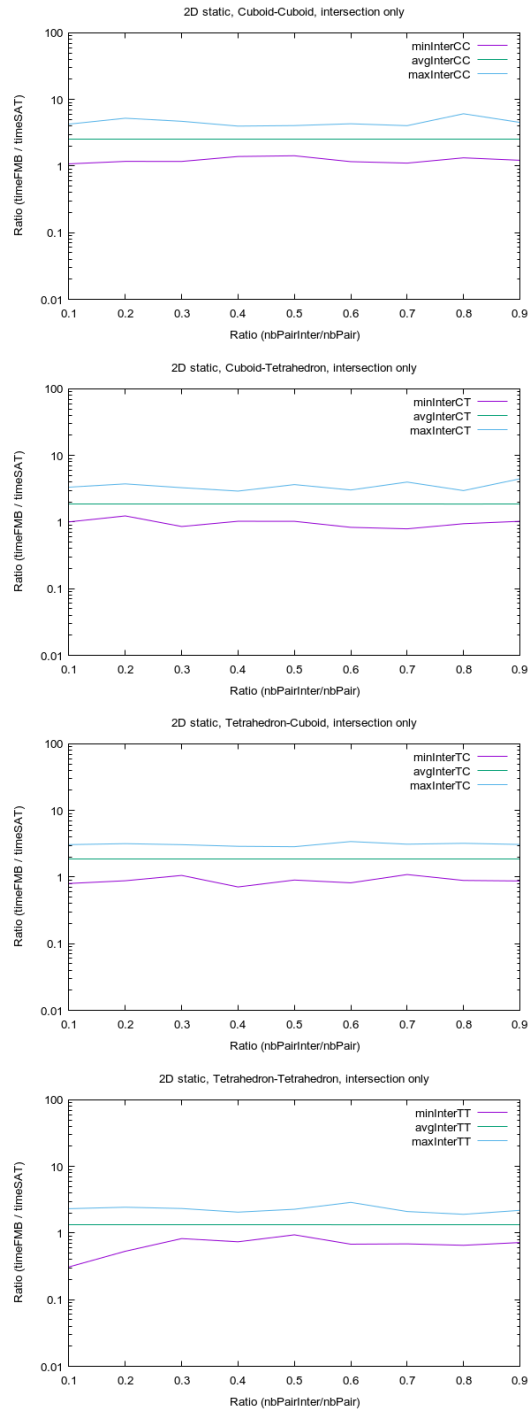
In this subsection I give the results of the qualification for each case. These results are commented in the next section.

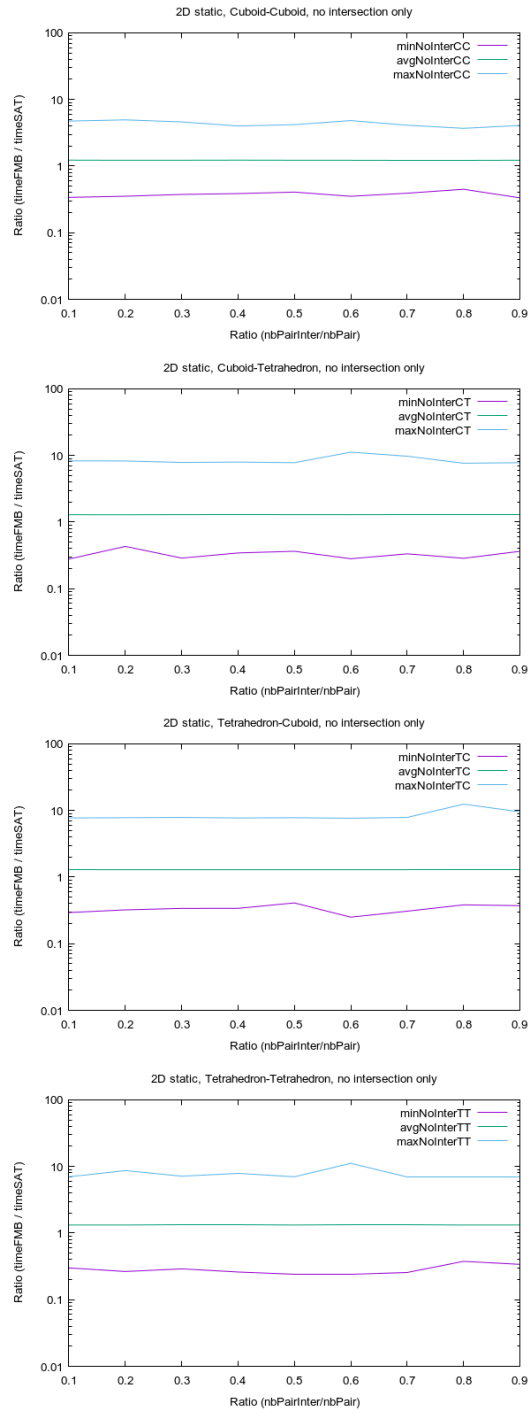
8.2.1 2D static

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	235620	766438	0.728682	1.945942	4.885246	0.233010	1.294967	11.825000	0.233010	1.351065	11.825000
0.2	233656	766392	0.817391	1.945636	5.277778	0.301587	1.285719	7.785714	0.301587	1.41718	7.785714
0.3	234656	766298	0.681159	1.946502	3.882353	0.244681	1.286115	7.857143	0.244681	1.484231	7.857143
0.4	233502	766452	0.768000	1.944899	4.933333	0.170330	1.284528	11.394737	0.170330	1.548676	11.394737
0.5	234228	765736	0.842105	1.946938	3.980769	0.310811	1.285151	7.785714	0.310811	1.615925	7.785714
0.6	234820	765146	0.736842	1.945708	4.021978	0.280000	1.284432	0.280000	0.280000	1.681198	7.785714
0.7	233966	766004	0.728682	1.947499	4.509434	0.302632	1.285356	7.714286	0.302632	1.748866	7.714286
0.8	234632	765342	0.761905	1.944300	4.166667	0.285714	1.285189	10.857143	0.285714	1.812678	10.857143
0.9	234556	765402	0.758065	1.946977	3.961538	0.225000	1.285286	8.533333	0.225000	1.880808	8.533333
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	65410	184798	1.046875	2.533837	4.885246	0.306452	1.226296	5.125000	0.306452	1.357050	5.125000
0.2	65232	184332	1.304762	2.534244	5.277778	0.315068	1.226965	4.350000	0.315068	1.488421	5.277778
0.3	65008	183450	1.196581	2.534663	3.882353	0.244681	1.226894	5.121951	0.244681	1.620484	5.121951
0.4	65376	184470	0.887417	2.533973	4.172414	0.227941	1.225641	4.087302	0.227941	1.748974	4.172414
0.5	65974	184610	1.381443	2.534094	3.980769	0.343750	1.226408	4.047619	0.343750	1.880251	4.047619
0.6	65960	183758	1.516854	2.534026	4.021978	0.352113	1.226213	3.866667	0.352113	2.010501	4.021978
0.7	65832	183978	1.273585	2.533832	4.509434	0.380952	1.228026	4.384615	0.380952	2.142090	4.509434
0.8	65374	182886	1.387755	2.534558	4.166667	0.337838	1.225012	4.375000	0.337838	2.272649	4.375000
0.9	65962	183850	1.231481	2.533907	3.961538	0.363636	1.225573	4.800000	0.363636	2.403073	4.800000
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	58610	191158	0.762821	1.880856	3.438596	0.233010	1.292530	7.785714	0.233010	1.351363	7.785714
0.2	59004	191640	1.125000	1.880279	2.915254	0.338028	1.295207	7.785714	0.338028	1.412221	7.785714
0.3	58584	191990	0.842857	1.880691	3.807018	0.323529	1.295542	7.857143	0.323529	1.471087	7.857143
0.4	58126	192034	1.048969	1.881102	4.166667	0.170330	1.293605	8.640000	0.170330	1.528604	8.640000
0.5	58268	191058	0.953488	1.880220	3.017544	0.333333	1.293637	7.785714	0.333333	1.588928	7.785714
0.6	58742	191244	1.000000	1.880406	2.758065	0.289474	1.294135	7.785714	0.289474	1.645897	7.785714
0.7	59224	191360	0.875000	1.880198	2.966102	0.302632	1.294686	7.714286	0.302632	1.704544	7.714286
0.8	58710	192062	1.090909	1.880490	3.140625	0.295455	1.297139	7.714286	0.295455	1.768320	7.714286
0.9	58836	192062	1.027273	1.880313	3.140351	0.225000	1.294409	8.533333	0.225000	1.821723	8.533333
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	58540	191268	0.930435	1.872295	3.483871	0.328125	1.291334	11.825000	0.328125	1.349430	11.825000
0.2	58534	191474	1.138298	1.872086	3.241935	0.323529	1.292820	7.642857	0.323529	1.408673	7.642857
0.3	58706	191722	0.832117	1.871947	3.368421	0.380000	1.291753	7.642857	0.380000	1.465811	7.642857
0.4	58660	190726	0.872000	1.872148	4.933333	0.185185	1.291549	11.394737	0.185185	1.523789	11.394737
0.5	58774	191268	0.963333	1.872414	3.067797	0.310811	1.290295	7.785714	0.310811	1.581354	7.785714
0.6	58590	191608	0.902439	1.872386	3.175439	0.250000	1.288717	7.785714	0.250000	1.638919	7.785714
0.7	58298	192024	0.899160	1.872224	3.080645	0.361702	1.289644	7.642857	0.361702	1.697450	7.642857
0.8	59060	191682	1.034783	1.872066	3.203390	0.285714	1.293018	7.642857	0.285714	1.756256	7.642857
0.9	58778	190622	1.037037	1.872167	3.032787	0.259259	1.292851	7.571429	0.259259	1.814255	7.571429
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	50960	199214	0.728682	1.350806	2.594203	0.271605	1.326022	8.266667	0.271605	1.328501	8.266667
0.2	50766	198946	0.817391	1.350303	2.514286	0.301587	1.324259	7.071429	0.301587	1.329488	7.071429
0.3	51358	198136	0.681159	1.350361	2.952224	0.258924	1.324691	7.133333	0.258924	1.329542	7.133333
0.4	51340	199222	0.768000	1.350128	3.416667	0.209677	1.323853	7.812500	0.209677	1.334201	7.812500
0.5	51192	198760	0.842105	1.350688	2.191176	0.339623	1.326601	7.000000	0.339623	1.338634	7.000000
0.6	51528	198536	0.736842	1.350429	2.220588	0.278481	1.325761	7.071429	0.278481	1.340562	7.071429
0.7	50612	198682	0.728682	1.350305	2.716418	0.303448	1.325326	7.000000	0.303448	1.342811	7.000000
0.8	51488	198712	0.761905	1.350469	2.136364	0.306452	1.325324	10.857143	0.306452	1.345440	10.857143
0.9	50980	198868	0.758065	1.350749	2.420230	0.358491	1.324428	7.000000	0.358491	1.348117	7.000000



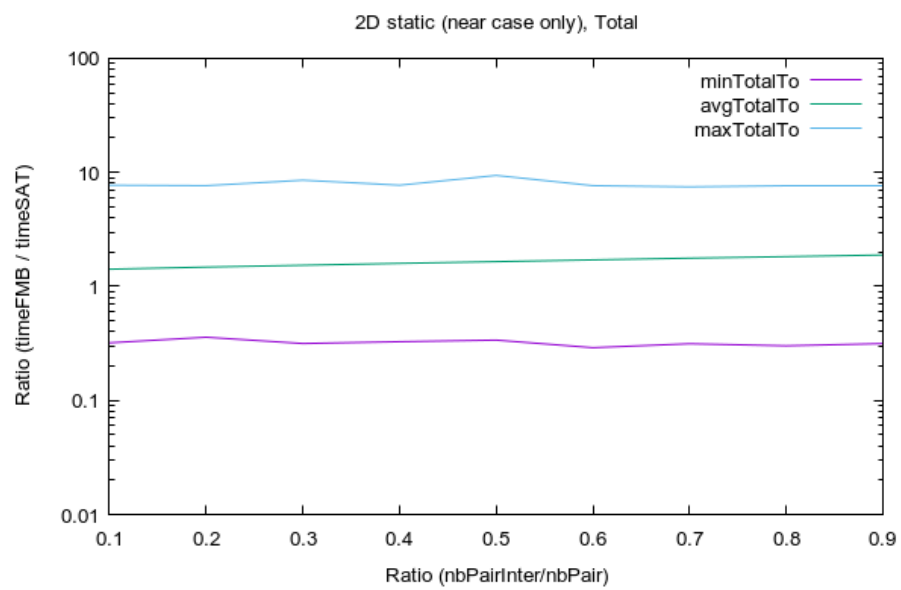


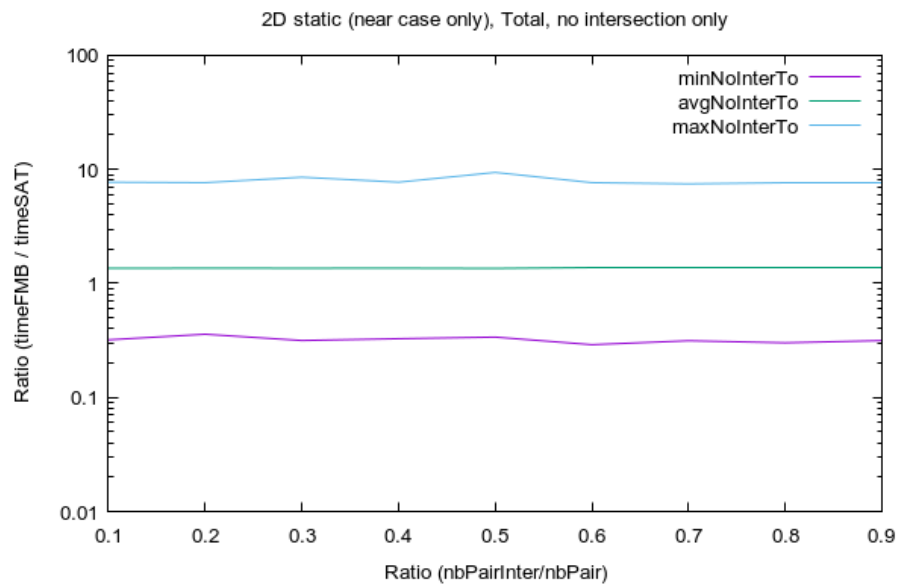
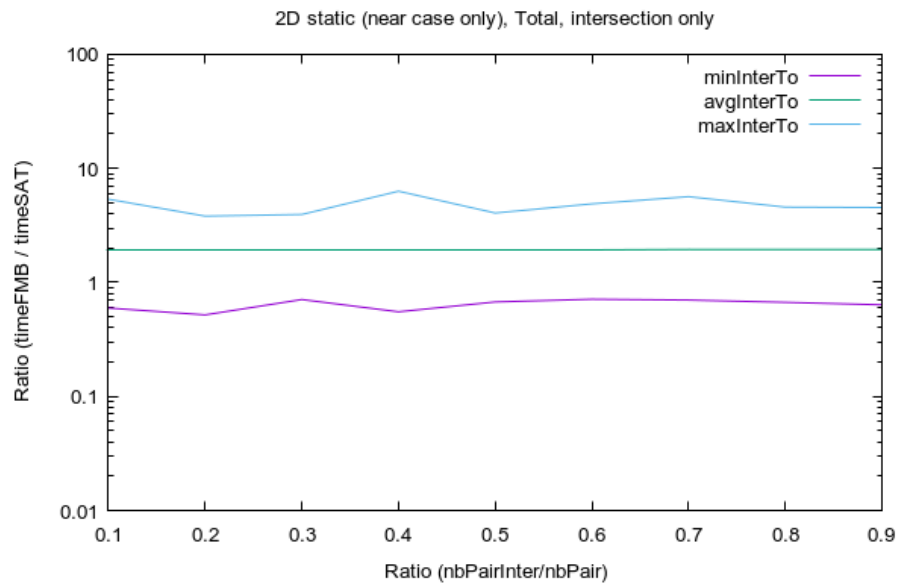


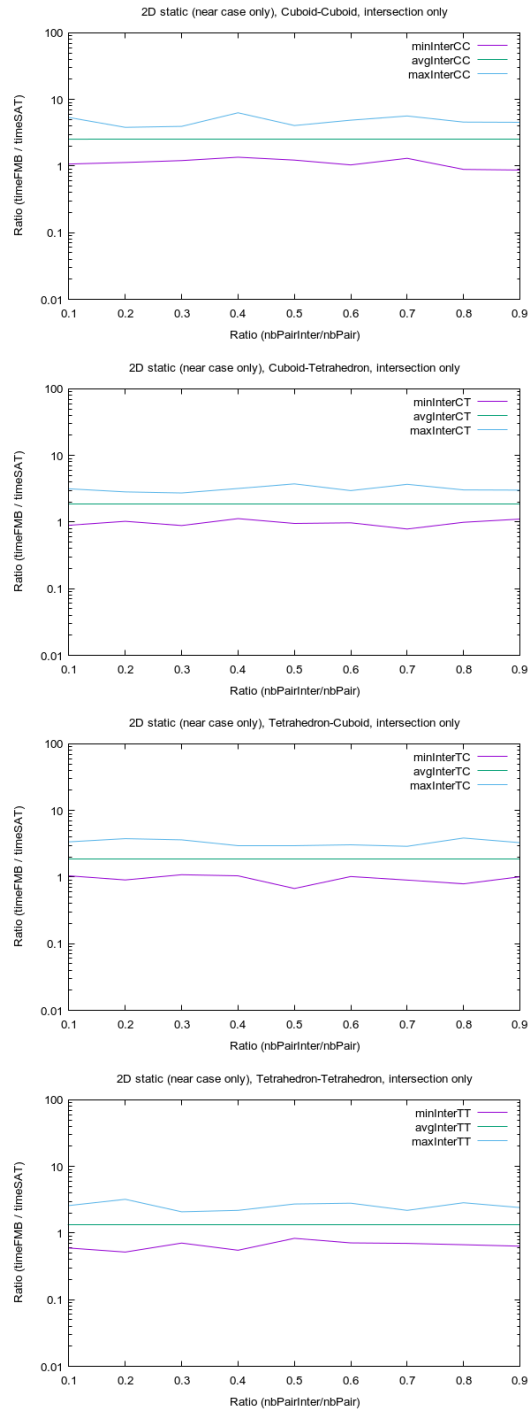


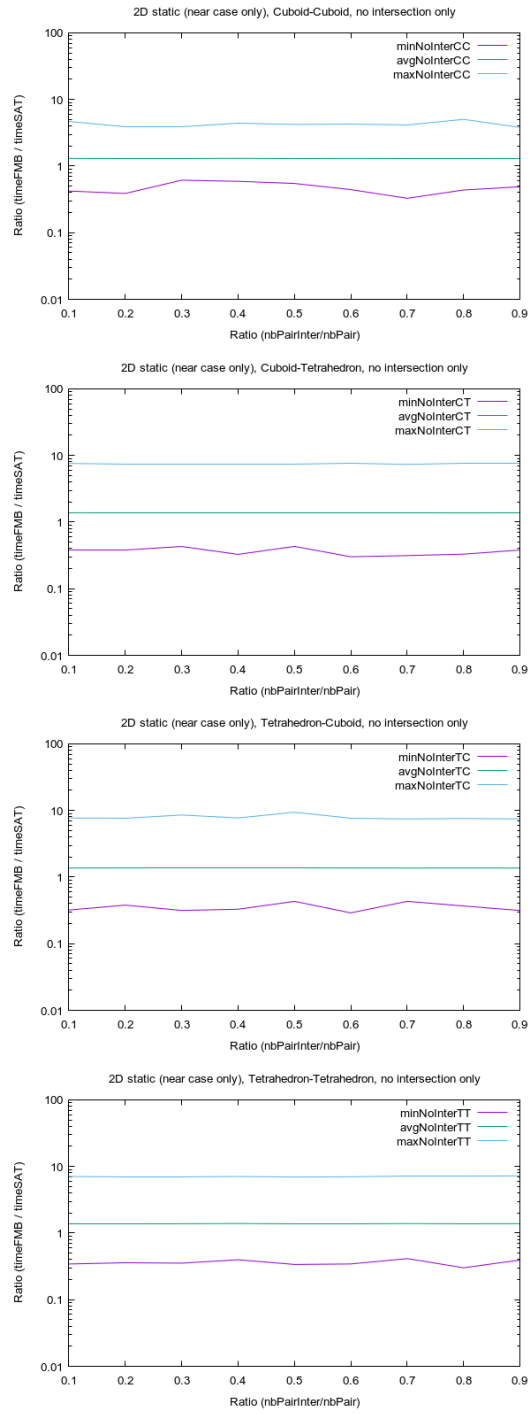
8.2.2 2D static (near case only)

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	235300	156646	0.596273	1.945464	5.358491	0.320000	1.356756	7.714286	0.320000	1.414727	7.714286
0.2	234826	156102	0.518819	1.943909	3.811321	0.358491	1.358491	7.642857	0.358491	1.475911	7.642857
0.3	235224	156760	0.708029	1.942258	3.944444	0.315789	1.357215	8.533333	0.315789	1.537778	8.533333
0.4	233866	156936	0.552486	1.942595	6.307692	0.327273	1.359726	7.714286	0.327273	1.592874	7.714286
0.5	234690	158024	0.674157	1.943335	4.056604	0.338028	1.353723	9.400000	0.338028	1.648529	9.400000
0.6	234548	157068	0.712121	1.942905	4.868792	0.290323	1.357182	7.642857	0.290323	1.708616	7.642857
0.7	234100	157050	0.700000	1.943835	5.648148	0.314286	1.358402	7.646667	0.314286	1.768205	7.646667
0.8	234046	156780	0.669065	1.942640	4.574074	0.301370	1.355981	7.642857	0.301370	1.825309	7.642857
0.9	234516	158010	0.635762	1.944781	4.537037	0.315068	1.356276	7.642857	0.315068	1.885931	7.642857
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	66632	39324	1.073171	2.532359	5.358491	0.422018	1.301348	4.675000	0.422018	1.424449	5.358491
0.2	66066	39326	1.132231	2.532038	3.811321	0.387755	1.302074	3.875000	0.387755	1.548067	3.875000
0.3	66524	39916	1.209091	2.531373	3.944444	0.614035	1.302599	3.875000	0.614035	1.671231	3.944444
0.4	66698	39462	1.357143	2.531975	6.307692	0.589744	1.304384	4.400000	0.589744	1.795420	6.307692
0.5	65752	40204	1.229358	2.531935	4.056604	0.548387	1.293296	4.219512	0.548387	1.912616	4.219512
0.6	65700	39482	1.040201	2.531815	4.868792	0.443038	1.301947	4.268293	0.443038	2.039528	4.868792
0.7	65766	39790	1.307692	2.532131	5.648148	0.328358	1.303837	4.146341	0.328358	2.163643	5.648148
0.8	65590	39136	0.891156	2.531899	4.574074	0.437500	1.303607	5.025000	0.437500	2.286241	5.025000
0.9	66406	40122	0.870968	2.531746	4.537037	0.489796	1.298901	3.804878	0.489796	2.408462	4.537037
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	58566	39454	0.896000	1.878667	3.157895	0.379310	1.378299	7.600000	0.379310	1.428336	7.600000
0.2	58724	39168	1.026549	1.878797	2.838710	0.378947	1.379948	7.400000	0.378947	1.479718	7.400000
0.3	59146	39340	0.888000	1.878639	2.736842	0.430108	1.380767	7.466667	0.430108	1.530129	7.466667
0.4	58410	39302	1.127451	1.878855	3.192982	0.327273	1.377444	7.466667	0.327273	1.578008	7.466667
0.5	58568	39434	0.953488	1.878781	3.741379	0.431034	1.372395	7.400000	0.431034	1.625888	7.400000
0.6	58786	39420	0.974790	1.878773	2.964912	0.301370	1.371848	7.642857	0.301370	1.676003	7.642857
0.7	58692	39136	0.786765	1.878639	3.687500	0.314286	1.375680	7.333333	0.314286	1.727511	7.333333
0.8	58680	39542	0.991736	1.878517	3.041667	0.330097	1.374359	7.642857	0.330097	1.777685	7.642857
0.9	58204	39716	1.107438	1.878591	3.017241	0.380435	1.375622	7.642857	0.380435	1.828294	7.642857
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	58818	39488	1.043103	1.866572	3.377049	0.320000	1.366067	7.714286	0.320000	1.416117	7.714286
0.2	58712	39088	0.905172	1.866707	3.774194	0.378947	1.375253	7.642857	0.378947	1.473544	7.642857
0.3	59062	38996	1.084906	1.866393	3.618421	0.315789	1.366739	8.533333	0.315789	1.516635	8.533333
0.4	58164	39320	1.043860	1.866604	2.928255	0.328767	1.366777	7.714286	0.328767	1.566708	7.714286
0.5	59152	39054	0.674157	1.866728	2.931054	0.431579	1.365602	9.400000	0.431579	1.616165	9.400000
0.6	58740	38984	1.018182	1.866775	3.050847	0.290323	1.370959	7.642857	0.290323	1.668448	7.642857
0.7	58572	39408	0.901639	1.866642	2.894737	0.431818	1.366942	7.466667	0.431818	1.716732	7.466667
0.8	58422	39114	0.791946	1.866388	3.854839	0.369421	1.367591	7.571429	0.369421	1.766628	7.571429
0.9	58600	39436	1.009174	1.866603	3.288136	0.315068	1.366840	7.466667	0.315068	1.816627	7.466667
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	51284	38380	0.596273	1.349689	2.588235	0.343284	1.377721	7.071429	0.343284	1.374918	7.071429
0.2	51324	38520	0.518919	1.349663	3.214286	0.358491	1.378968	7.000000	0.358491	1.373107	7.000000
0.3	51364	38508	0.708029	1.349763	2.088235	0.353535	1.380121	7.000000	0.353535	1.371014	7.000000
0.4	51594	38852	0.552486	1.349929	2.191176	0.397727	1.390879	7.071429	0.397727	1.374499	7.071429
0.5	51218	39326	0.833333	1.350000	2.731343	0.338028	1.354989	6.928571	0.338028	1.369494	6.928571
0.6	51322	39182	0.712121	1.349604	2.808234	0.343750	1.364480	7.000000	0.343750	1.363555	7.000000
0.7	51070	38716	0.700000	1.349707	2.191176	0.413793	1.388324	7.071429	0.413793	1.374299	7.071429
0.8	51354	38988	0.669065	1.350050	2.850746	0.301370	1.388269	7.071429	0.301370	1.356944	7.071429
0.9	51306	38736	0.635762	1.349447	2.417910	0.393258	1.385112	7.214286	0.393258	1.353013	7.214286



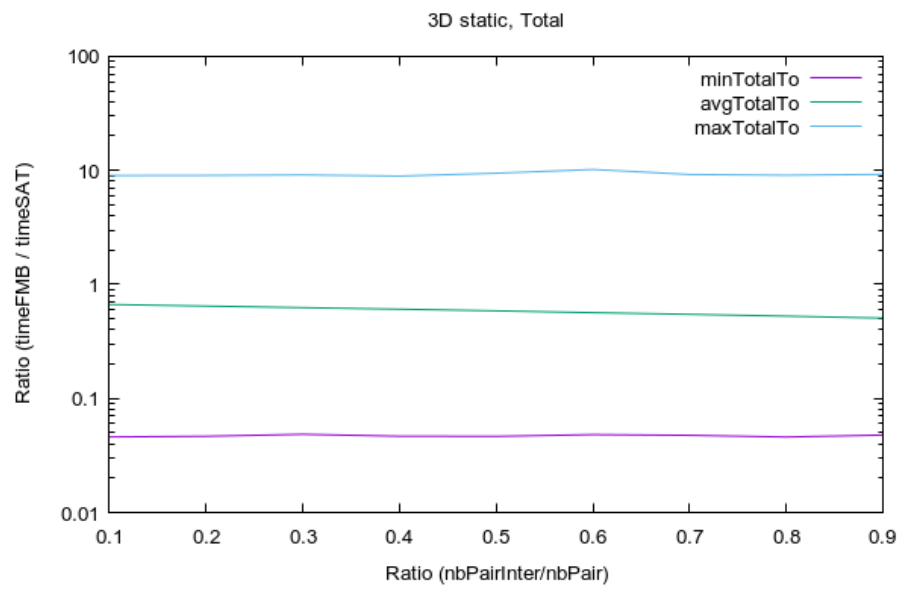


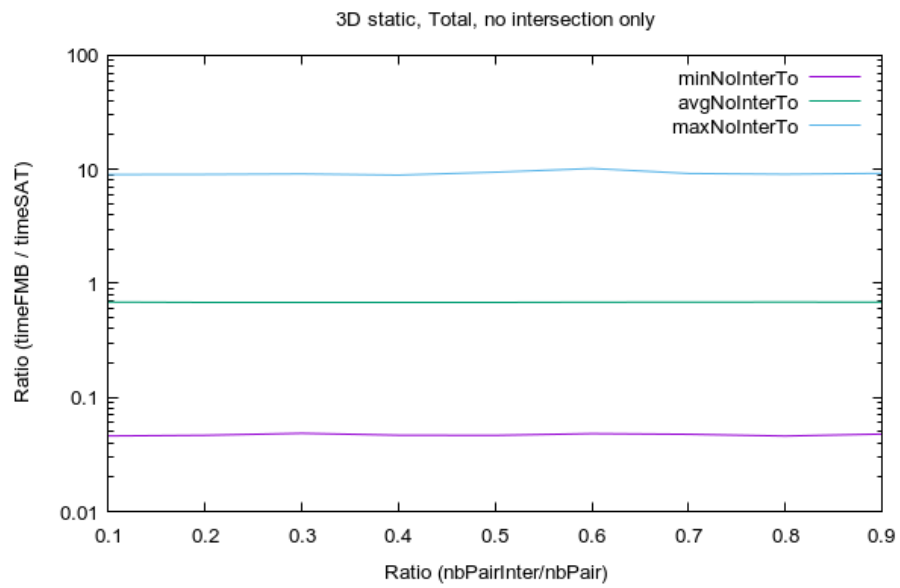
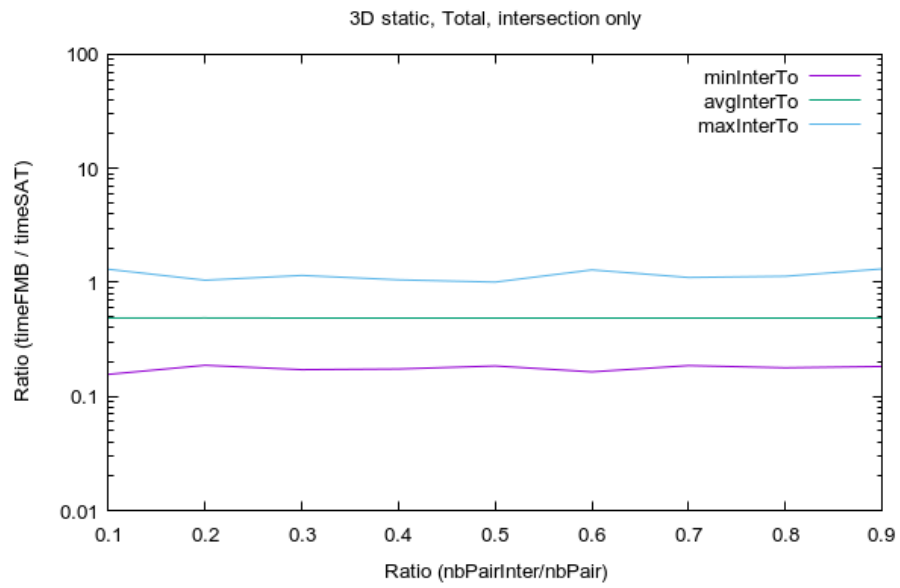


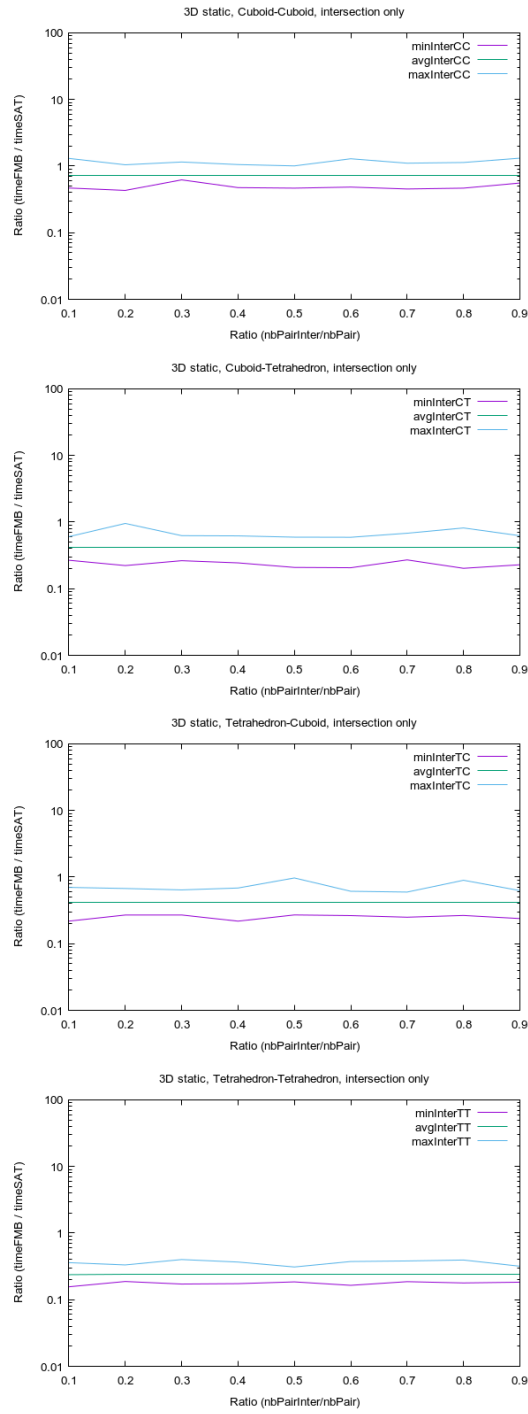


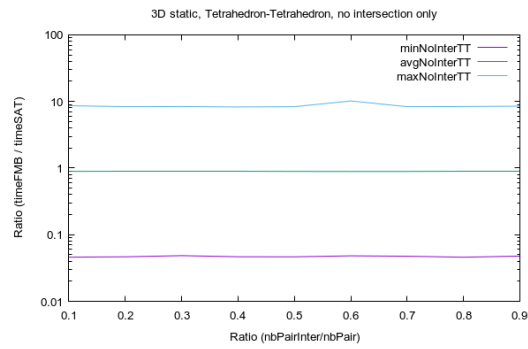
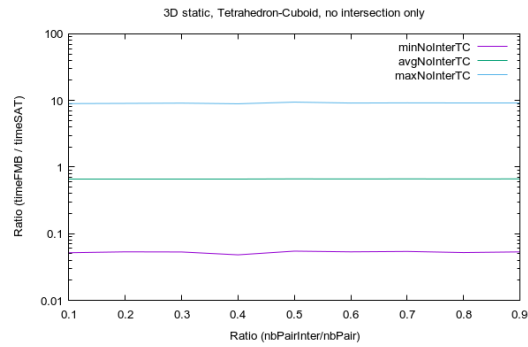
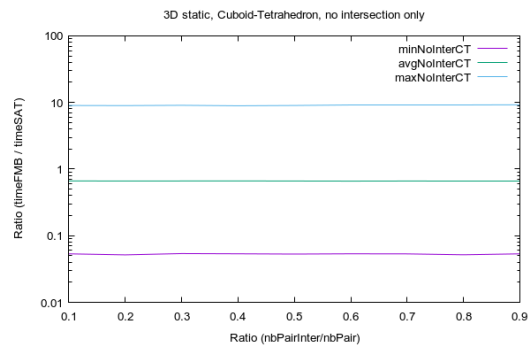
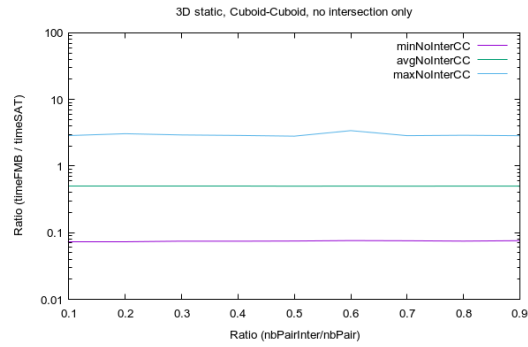
8.2.3 3D static

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	157948	842052	0.156250	0.487254	1.303819	0.045977	0.636466	9.000000	0.045977	0.636466	9.000000
0.2	842264	0.187699	0.146375	0.046605	0.635255	0.046605	0.635255	9.029412	0.046605	0.635255	9.029412
0.3	158814	841186	0.117101	0.486531	1.151724	0.046637	0.635131	9.117647	0.046637	0.635131	9.117647
0.4	157914	842084	0.174074	0.487662	1.053082	0.046667	0.635311	8.911765	0.046667	0.635311	8.911765
0.5	158068	841930	0.185146	0.487248	1.006885	0.046512	0.635591	9.441176	0.046512	0.635591	9.441176
0.6	158172	841828	0.164311	0.486540	1.287415	0.048193	0.634193	10.178571	0.048193	0.634193	10.178571
0.7	156462	843536	0.186441	0.487472	1.105341	0.047490	0.635271	9.205882	0.047490	0.635271	9.205882
0.8	158104	841896	0.178368	0.487436	1.132653	0.045860	0.635932	9.058824	0.045860	0.635932	9.058824
0.9	157922	842008	0.183221	0.486243	1.312715	0.047745	0.635672	9.235294	0.047745	0.635672	9.235294
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	52952	197382	0.468610	0.718566	1.303819	0.074148	0.500861	2.867925	0.074148	0.522641	2.867925
0.2	53434	196902	0.431211	0.718460	1.046375	0.074148	0.500623	3.057692	0.074148	0.544190	3.057692
0.3	53262	196728	0.620896	0.718637	1.151724	0.074656	0.500689	2.932642	0.074656	0.566073	2.932642
0.4	53194	196706	0.474972	0.718509	1.053082	0.074447	0.500934	2.882353	0.074447	0.587964	2.882353
0.5	53168	196712	0.466073	0.718508	1.006885	0.074899	0.499391	2.814103	0.074899	0.608949	2.814103
0.6	52946	196932	0.463759	0.718585	1.287415	0.076132	0.500995	3.405063	0.076132	0.631549	3.405063
0.7	52640	197904	0.452563	0.718558	1.105341	0.075665	0.499523	2.860759	0.075665	0.652847	2.860759
0.8	53254	197410	0.466298	0.718531	1.132653	0.074747	0.500948	2.899371	0.074747	0.675015	2.899371
0.9	52590	197002	0.557029	0.718541	1.312715	0.075820	0.500102	2.853503	0.075820	0.696697	2.853503
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	39402	210000	0.267073	0.414551	0.604426	0.053412	0.633866	9.000000	0.053412	0.633953	9.000000
0.2	39336	209718	0.222222	0.414530	0.953425	0.051502	0.632085	8.970588	0.051502	0.612574	8.970588
0.3	39348	210330	0.263094	0.414414	0.625509	0.053857	0.632857	9.058824	0.053857	0.587924	9.058824
0.4	39088	210874	0.244121	0.414463	0.620408	0.053492	0.633645	8.911765	0.053492	0.563972	8.911765
0.5	39304	210870	0.208606	0.414445	0.593284	0.052941	0.661224	9.000000	0.052941	0.537835	9.000000
0.6	39242	211414	0.206290	0.414482	0.591022	0.053492	0.659726	9.058824	0.053492	0.512579	9.058824
0.7	38862	210496	0.270574	0.414270	0.680297	0.053333	0.632058	9.058824	0.053333	0.488606	9.058824
0.8	39120	209694	0.202237	0.414500	0.815860	0.051650	0.632584	9.058824	0.051650	0.464077	9.058824
0.9	39738	210748	0.228733	0.414889	0.624654	0.053412	0.662841	9.235294	0.053412	0.439504	9.235294
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	39494	210806	0.218235	0.414549	0.700375	0.051799	0.633922	8.970588	0.051799	0.633985	8.970588
0.2	38704	211012	0.269231	0.414296	0.673943	0.053412	0.633866	9.029412	0.053412	0.613952	9.029412
0.3	39278	210008	0.289663	0.414540	0.642247	0.053097	0.632761	9.117647	0.053097	0.588295	9.117647
0.4	39554	211146	0.217972	0.414305	0.685259	0.048209	0.663129	8.911765	0.048209	0.563600	8.911765
0.5	39260	210140	0.270101	0.414346	0.696967	0.054711	0.665216	9.441176	0.054711	0.539826	9.441176
0.6	39424	210086	0.263990	0.414378	0.612273	0.053412	0.632286	9.142857	0.053412	0.513541	9.142857
0.7	39090	210916	0.249720	0.414441	0.595862	0.054135	0.664241	9.205882	0.054135	0.489381	9.205882
0.8	39416	210426	0.285031	0.414439	0.895408	0.052098	0.630600	9.058824	0.052098	0.464164	9.058824
0.9	39324	211006	0.239403	0.414455	0.627095	0.053333	0.664374	9.058824	0.053333	0.439446	9.058824
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	26100	223864	0.156250	0.237556	0.359564	0.045977	0.892526	8.640000	0.045977	0.827029	8.640000
0.2	26262	224630	0.187699	0.237765	0.332544	0.046605	0.888154	8.346154	0.046605	0.758567	8.346154
0.3	26926	224120	0.171701	0.237807	0.400982	0.048452	0.889393	8.384615	0.048452	0.693917	8.384615
0.4	26078	223358	0.174074	0.237764	0.367273	0.046667	0.888433	8.269231	0.046667	0.628165	8.269231
0.5	26336	223208	0.185146	0.237565	0.309724	0.046512	0.890969	8.346154	0.046512	0.564287	8.346154
0.6	26560	223396	0.164311	0.237549	0.374136	0.048193	0.894946	10.178571	0.048193	0.498308	10.178571
0.7	26870	224220	0.186441	0.237576	0.381356	0.047490	0.890793	8.346154	0.047490	0.433591	8.346154
0.8	26314	224366	0.178368	0.237596	0.394221	0.045860	0.891962	8.384615	0.045860	0.389469	8.384615
0.9	26340	223252	0.183221	0.237568	0.317857	0.047745	0.891102	8.500000	0.047745	0.302922	8.500000



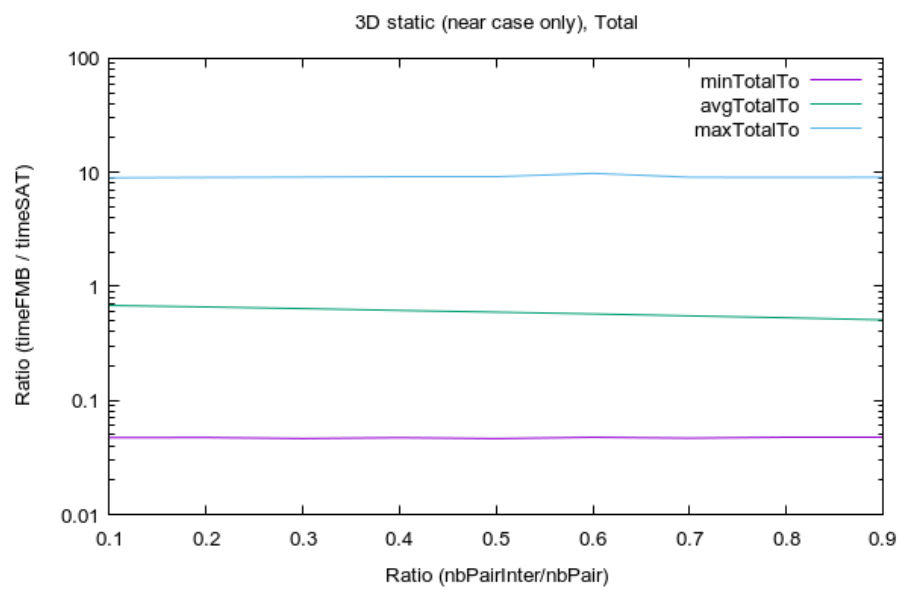


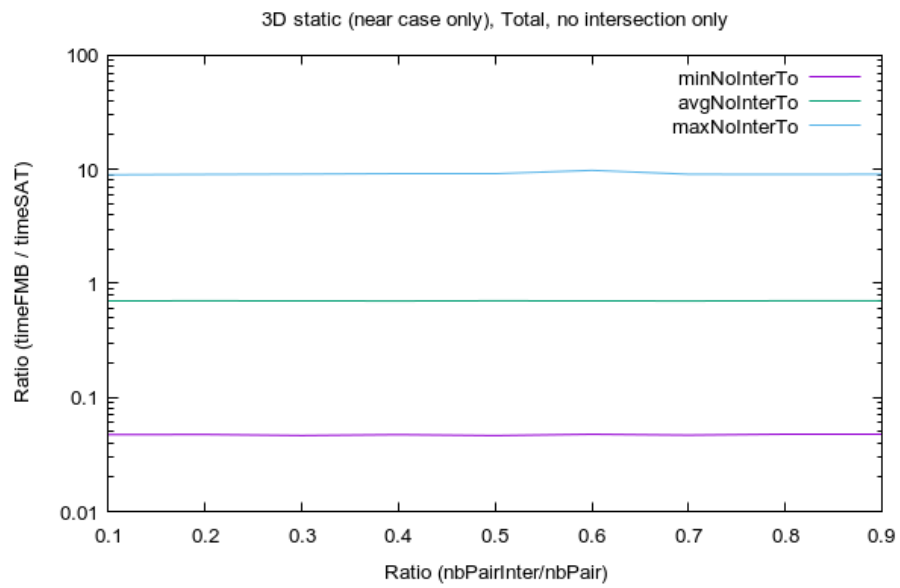
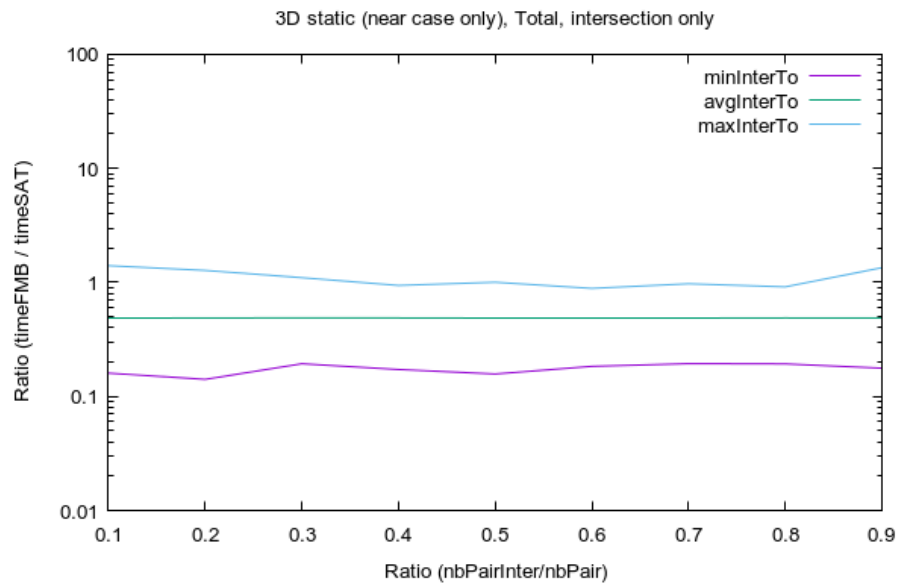


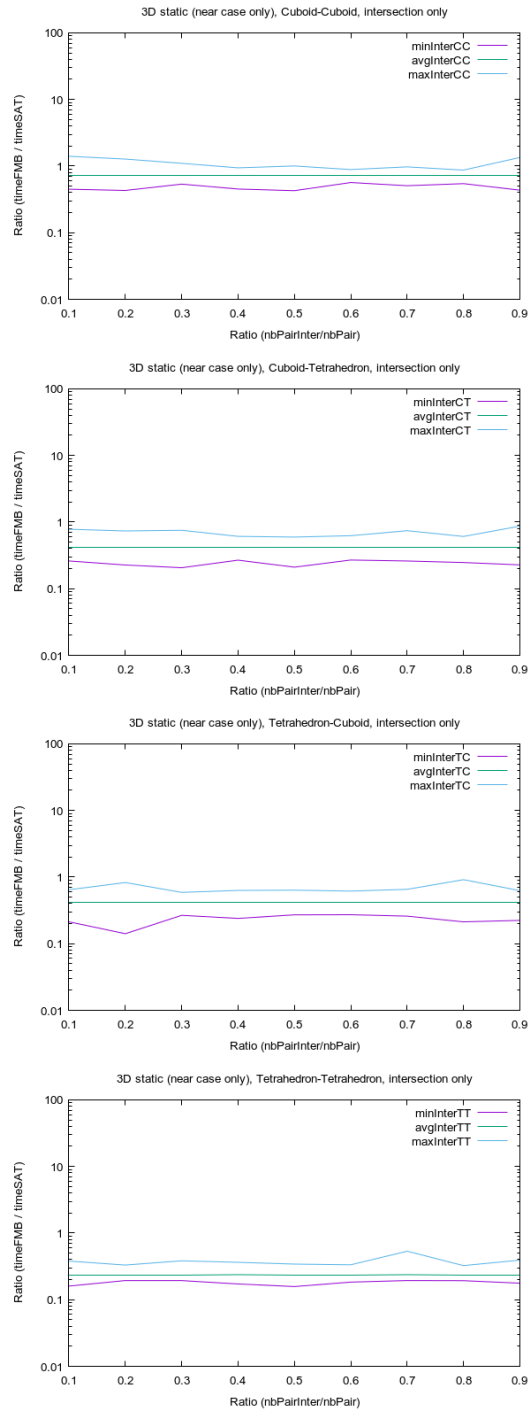


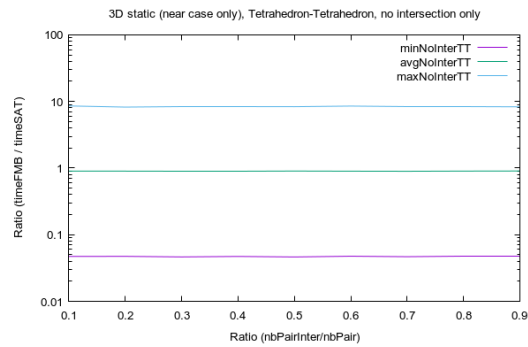
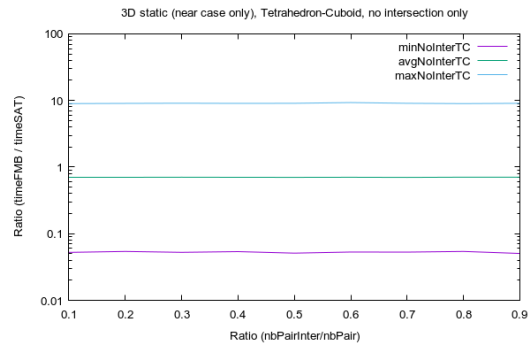
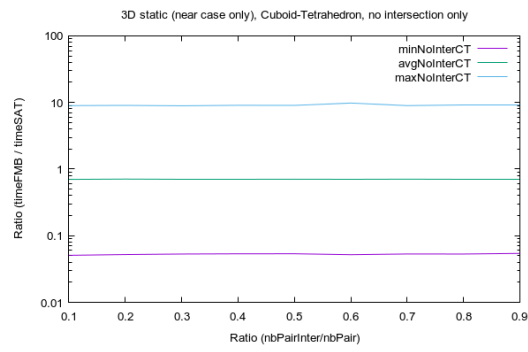
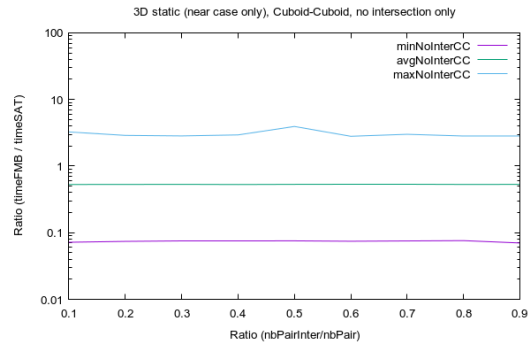
8.2.4 3D static (near case only)

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	158388	306704	0.160169	0.486416	1.401709	0.047120	0.702200	8.970588	0.047120	0.680621	8.970588
0.2	158482	304272	0.141229	0.487075	1.273973	0.047306	0.702927	9.029412	0.047306	0.660567	9.029412
0.3	158146	306448	0.193443	0.488531	1.098958	0.046392	0.702742	9.090909	0.046392	0.638479	9.090909
0.4	158014	305320	0.172227	0.487713	0.940226	0.047182	0.701391	9.058824	0.047182	0.615920	9.058824
0.5	157548	305080	0.157313	0.486485	1.003339	0.046296	0.703164	9.058824	0.046296	0.594824	9.058824
0.6	157996	304416	0.183604	0.487514	0.886441	0.047493	0.702662	9.800000	0.047493	0.573573	9.800000
0.7	158458	304628	0.193619	0.487026	0.973262	0.071342	0.701342	9.058824	0.046753	0.513231	9.058824
0.8	158318	304094	0.192810	0.488178	0.913272	0.047306	0.704110	9.029412	0.047306	0.511719	9.029412
0.9	158148	306596	0.177021	0.486958	1.345455	0.047306	0.704110	9.058824	0.047306	0.508673	9.058824
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	52820	81278	0.449134	0.718774	1.401709	0.071804	0.529669	3.251572	0.071804	0.548579	3.251572
0.2	53338	80720	0.430584	0.718478	1.273973	0.074074	0.530195	2.884146	0.074074	0.567852	2.884146
0.3	53522	81076	0.535576	0.718742	1.098958	0.074297	0.531762	2.834395	0.074297	0.587866	2.834395
0.4	53450	81100	0.461788	0.718488	0.940226	0.074297	0.528288	2.935897	0.074297	0.604368	2.935897
0.5	52634	81588	0.427544	0.718441	1.003339	0.075356	0.531720	3.930693	0.075356	0.625080	3.930693
0.6	53278	80392	0.565860	0.718374	0.886441	0.074380	0.531219	2.801282	0.074380	0.643512	2.801282
0.7	53194	81502	0.507299	0.718535	0.973262	0.075203	0.530926	3.000000	0.075203	0.662253	3.000000
0.8	53708	80846	0.544980	0.718575	0.870017	0.075975	0.529703	2.825806	0.075975	0.680800	2.825806
0.9	53074	80762	0.435451	0.718650	1.345455	0.070076	0.530595	2.831169	0.070076	0.699844	2.831169
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	39474	76954	0.260819	0.414680	0.780933	0.050754	0.700736	8.970588	0.050754	0.672130	8.970588
0.2	39630	76840	0.226847	0.414414	0.735493	0.052098	0.707476	9.029412	0.052098	0.648884	9.029412
0.3	39634	76372	0.206157	0.414422	0.752294	0.053019	0.702836	8.911765	0.053019	0.616312	8.911765
0.4	39012	77056	0.267990	0.414469	0.609890	0.053492	0.702303	9.058824	0.053492	0.587169	9.058824
0.5	38976	76782	0.210427	0.414498	0.595716	0.053571	0.704798	9.029412	0.053571	0.589648	9.029412
0.6	39002	76946	0.269327	0.414374	0.624675	0.051873	0.702696	9.800000	0.051873	0.529703	9.800000
0.7	39546	76612	0.260241	0.414375	0.741758	0.053097	0.705489	8.970588	0.053097	0.501709	8.970588
0.8	39172	76592	0.246925	0.414370	0.608458	0.053019	0.702691	9.029412	0.053019	0.472034	9.029412
0.9	39372	77146	0.227768	0.414593	0.868914	0.054173	0.702877	9.029412	0.054173	0.443422	9.029412
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	39700	76770	0.213185	0.414318	0.646116	0.052402	0.703555	8.970588	0.052402	0.674631	8.970588
0.2	39030	76588	0.141229	0.414171	0.827238	0.054094	0.703784	9.029412	0.054094	0.645862	9.029412
0.3	39266	77440	0.266178	0.414178	0.589212	0.052478	0.704352	9.090909	0.052478	0.611299	9.090909
0.4	39182	76432	0.239654	0.414352	0.630155	0.053812	0.702652	9.029412	0.053812	0.587332	9.029412
0.5	39702	76668	0.271040	0.414329	0.635744	0.051064	0.701723	9.058824	0.051064	0.588026	9.058824
0.6	39440	77212	0.272050	0.414165	0.615979	0.053019	0.703640	9.308300	0.053019	0.529955	9.308300
0.7	39378	76496	0.259259	0.414223	0.654422	0.052863	0.700219	9.058824	0.052863	0.500022	9.058824
0.8	39216	76528	0.212786	0.414192	0.913272	0.054094	0.705127	8.970588	0.054094	0.472379	8.970588
0.9	39320	77108	0.223896	0.414329	0.626050	0.050477	0.704519	9.058824	0.050477	0.443438	9.058824
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	26394	70702	0.160169	0.237149	0.379562	0.047120	0.900661	8.560000	0.047120	0.834310	8.560000
0.2	26484	70124	0.193665	0.237205	0.330952	0.047306	0.900178	8.230769	0.047306	0.767583	8.230769
0.3	25724	70560	0.193443	0.237228	0.383061	0.046392	0.897335	8.269231	0.046392	0.699303	8.269231
0.4	26370	70732	0.172227	0.237311	0.363855	0.047182	0.897513	8.269231	0.047182	0.633432	8.269231
0.5	26236	70072	0.157313	0.237276	0.342324	0.046296	0.902495	8.346154	0.046296	0.638855	8.346154
0.6	26176	69866	0.183604	0.237125	0.334146	0.047493	0.898816	8.520000	0.047493	0.501801	8.520000
0.7	26340	70018	0.193619	0.237404	0.534279	0.046753	0.896396	8.269231	0.046753	0.435102	8.269231
0.8	26222	70128	0.192810	0.237184	0.325175	0.047306	0.901590	8.269231	0.047306	0.370065	8.269231
0.9	26382	70580	0.177021	0.237093	0.392272	0.047306	0.903559	8.307692	0.047306	0.303759	8.307692



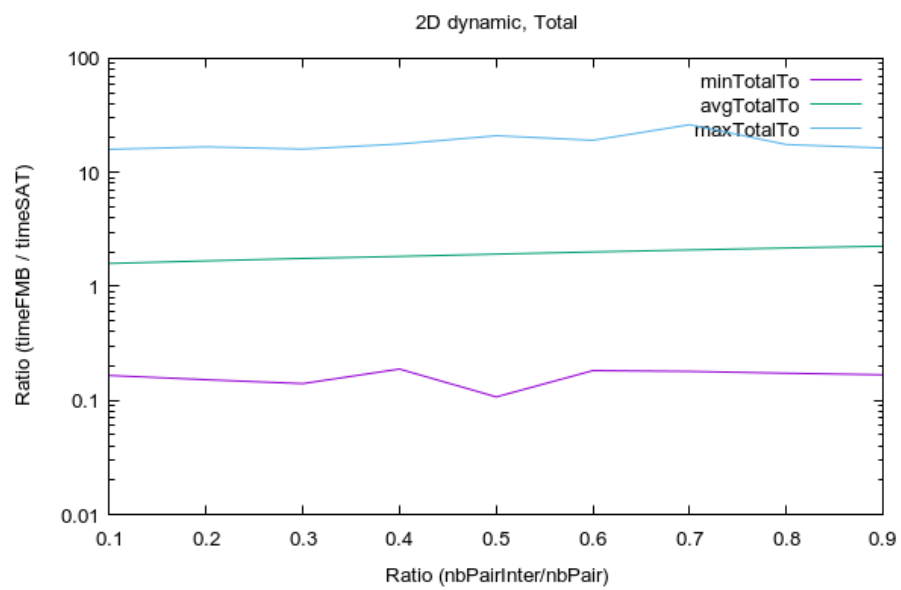


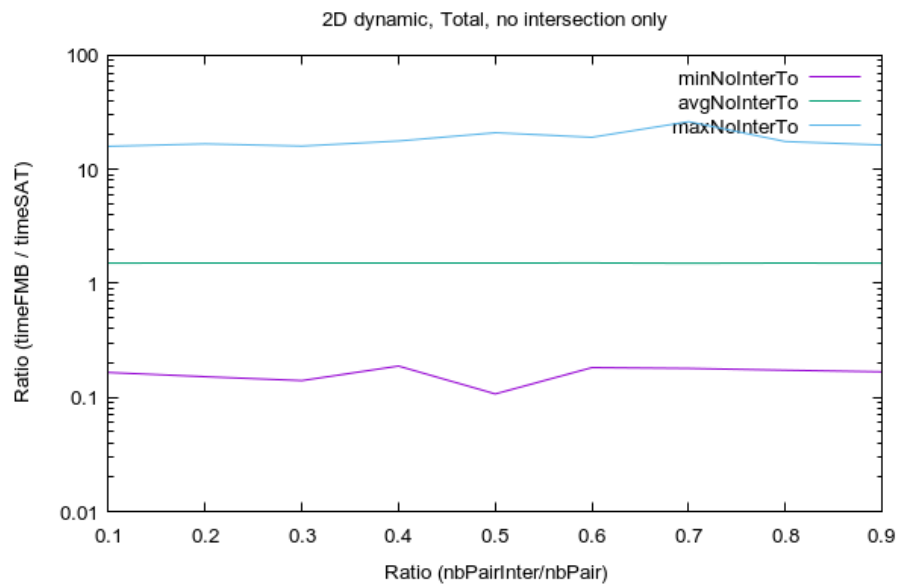
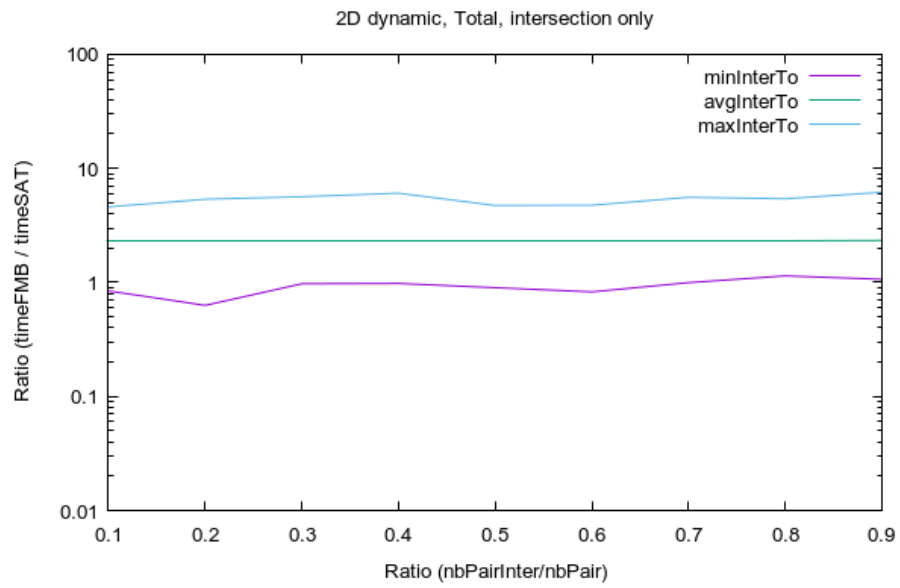


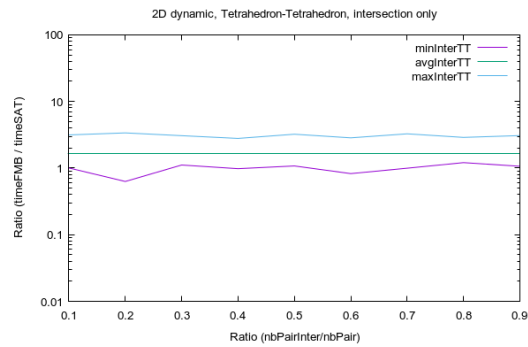
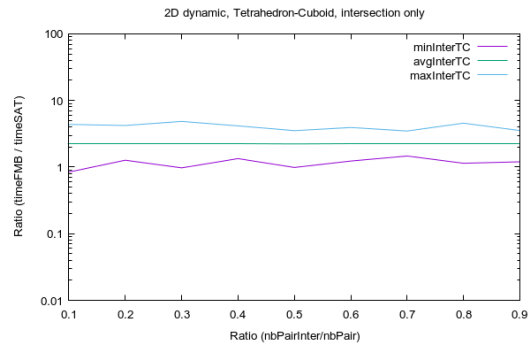
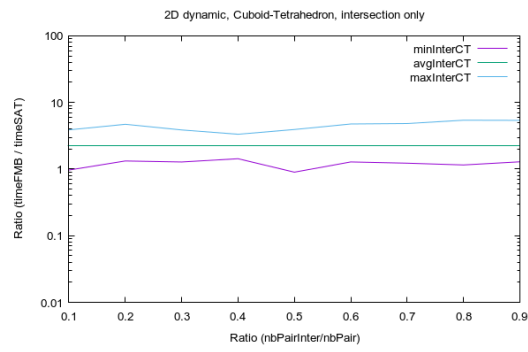
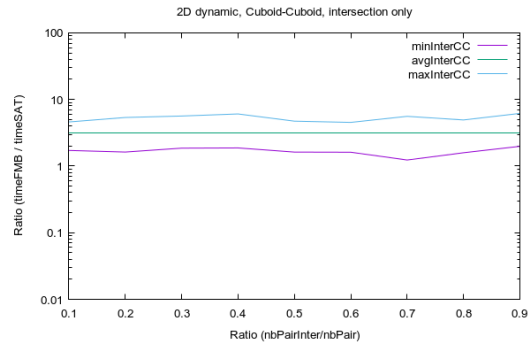


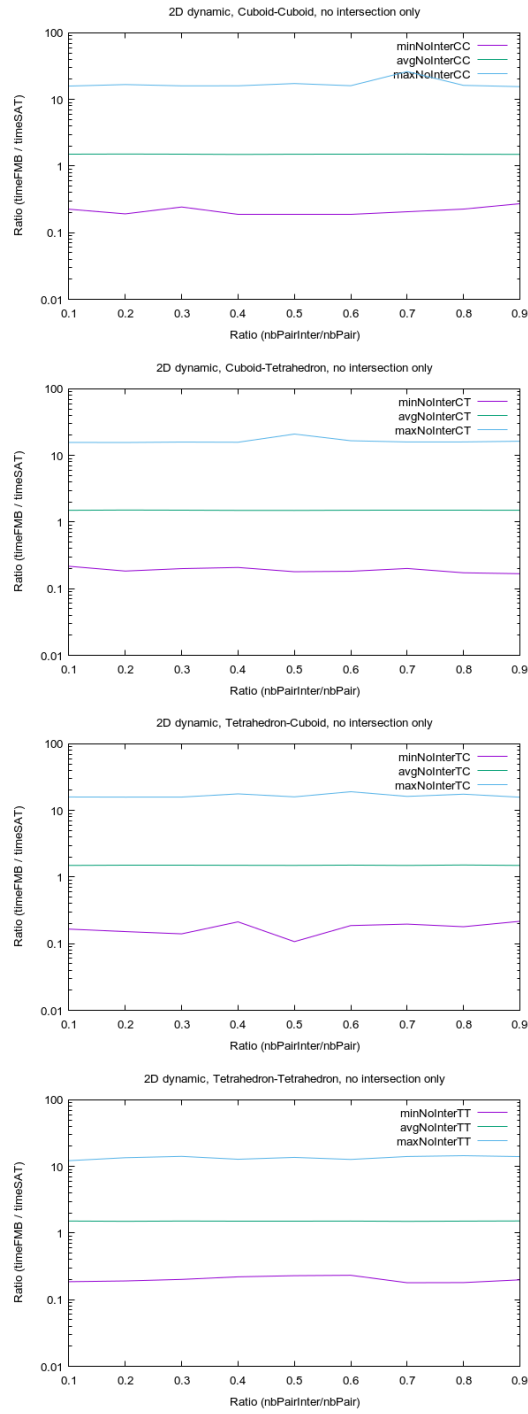
8.2.5 2D dynamic

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	373524	626430	0.821875	2.326440	6.324561	0.141975	1.494309	21.192308	0.141975	1.577522	21.192308
0.2	371444	628528	1.019380	6.377193	6.377193	0.203947	1.495697	17.676923	0.203947	1.661795	17.676923
0.3	371916	628040	0.831210	2.326918	5.593220	0.174497	1.493185	15.846154	0.174497	1.743305	15.846154
0.4	372754	627190	1.050388	2.326438	5.730435	0.183673	1.497271	18.538462	0.183673	1.828937	18.538462
0.5	372294	627672	0.925532	2.327966	6.241379	0.176971	1.499245	18.440000	0.176971	1.908606	18.440000
0.6	372120	627834	0.890785	2.323543	5.434783	0.180556	1.491836	17.000000	0.180556	1.990860	17.000000
0.7	370892	628072	0.834955	2.325518	5.813008	0.167883	1.493973	26.666667	0.167883	2.076055	26.666667
0.8	372228	627742	0.916968	2.326824	4.901639	0.153061	1.494070	23.538462	0.153061	2.160113	23.538462
0.9	372778	627186	1.010676	2.326066	6.250000	0.150602	1.488978	16.040000	0.150602	2.242357	16.040000
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	100110	150170	1.808824	3.107638	6.324561	0.250000	1.505446	21.192308	0.250000	1.665755	21.192308
0.2	99634	150590	1.377301	3.107698	6.377193	0.215517	1.513250	17.576923	0.215517	1.832140	17.576923
0.3	99514	151202	1.617117	3.107715	5.593220	0.254902	1.491580	15.846154	0.254902	1.976420	15.846154
0.4	99808	149840	1.756303	3.107587	5.730435	0.242991	1.501855	18.538462	0.242991	2.144148	18.538462
0.5	99906	149110	1.596774	3.107761	6.241379	0.221154	1.511386	18.440000	0.221154	2.309563	18.440000
0.6	98698	149940	1.967219	3.107978	5.434783	0.211864	1.497662	16.360000	0.211864	2.463851	16.360000
0.7	98790	151770	1.806931	3.107804	5.813008	0.257143	1.514075	22.730769	0.257143	2.629666	22.730769
0.8	99784	150360	1.925532	3.108612	4.901639	0.242718	1.504069	20.307692	0.242718	2.787703	20.307692
0.9	99508	150878	1.335766	3.107952	6.250000	0.206612	1.478662	16.040000	0.206612	2.945023	16.040000
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	93208	157340	0.919614	2.211428	4.850000	0.179310	1.492824	16.608696	0.179310	1.564684	16.608696
0.2	92534	157090	1.308756	2.211770	6.008000	0.209302	1.494973	16.043478	0.209302	1.630333	16.043478
0.3	93330	156280	1.028986	2.211419	5.314050	0.211009	1.505492	15.782609	0.211009	1.717270	15.782609
0.4	93060	157482	1.276018	2.211596	3.965517	0.196970	1.490961	15.739130	0.196970	1.779125	15.739130
0.5	93114	157308	1.035971	2.211075	3.517241	0.185484	1.469340	15.652174	0.185484	1.840207	15.652174
0.6	92928	157176	1.035461	2.211384	3.874074	0.200000	1.478300	17.000000	0.200000	1.918150	17.000000
0.7	93008	156680	1.071429	2.211352	5.076740	0.184000	1.480378	15.545455	0.184000	1.992060	15.545455
0.8	92384	156872	1.353774	2.212108	4.308219	0.218182	1.471512	17.826087	0.218182	2.063989	17.826087
0.9	93200	156712	1.010676	2.211503	3.870968	0.150602	1.497304	15.708333	0.150602	2.140083	15.708333
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	93442	156020	1.352381	2.211175	4.236641	0.141975	1.480832	15.916667	0.141975	1.533866	15.916667
0.2	92804	157064	1.252708	2.211406	5.505746	0.203947	1.492018	15.739130	0.203947	1.635895	15.739130
0.3	93234	156718	1.257778	2.211311	3.500000	0.192593	1.476263	15.608696	0.192593	1.696778	15.608696
0.4	93360	156570	1.280543	2.211357	3.900000	0.183673	1.468337	18.468333	0.183673	1.777545	18.468333
0.5	93468	157310	1.032491	2.211963	5.315385	0.176571	1.484293	15.590909	0.176571	1.846128	15.590909
0.6	93704	157214	1.062500	2.211289	4.531250	0.180556	1.489399	16.217391	0.180556	1.922749	16.217391
0.7	93228	157354	1.044118	2.211540	4.124031	0.203704	1.483814	26.666667	0.203704	1.993222	26.666667
0.8	93158	156976	0.938719	2.211884	4.541353	0.216981	1.493866	23.538462	0.216981	2.069284	23.538462
0.9	93728	156112	1.162264	2.211704	3.575163	0.188406	1.476626	15.434783	0.188406	2.138196	15.434783
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	86764	162900	0.821875	1.672772	3.017857	0.212963	1.498294	13.681818	0.212963	1.515742	13.681818
0.2	86572	163784	1.019380	1.673021	3.446541	0.218487	1.493370	12.125000	0.218487	1.529120	12.125000
0.3	86380	163940	0.831210	1.672871	3.112583	0.174497	1.499115	12.857143	0.174497	1.551242	12.857143
0.4	86526	163298	1.050388	1.673062	2.237500	0.211009	1.507714	18.000000	0.211009	1.573853	18.000000
0.5	86786	163944	0.925532	1.673116	3.096774	0.187050	1.492978	14.545455	0.187050	1.583047	14.545455
0.6	86790	163504	0.890785	1.672767	2.577922	0.200000	1.501330	12.181818	0.200000	1.604192	12.181818
0.7	86866	163268	0.834955	1.672902	3.907609	0.167883	1.498123	13.086957	0.167883	1.620468	13.086957
0.8	86902	163534	0.916968	1.673456	3.318182	0.153061	1.506693	16.083333	0.153061	1.640103	16.083333
0.9	86342	163484	1.073469	1.672760	3.354430	0.207207	1.502315	13.500000	0.207207	1.655716	13.500000



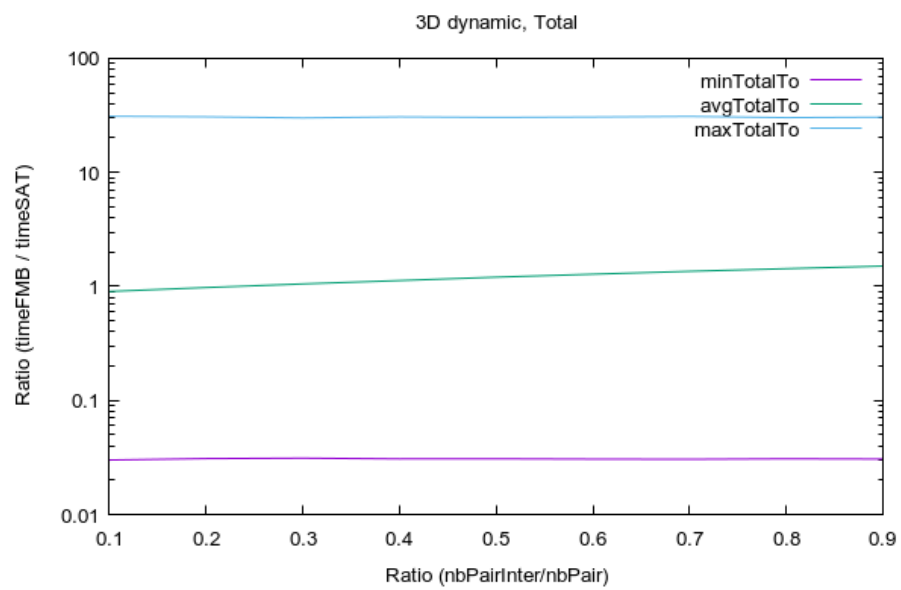


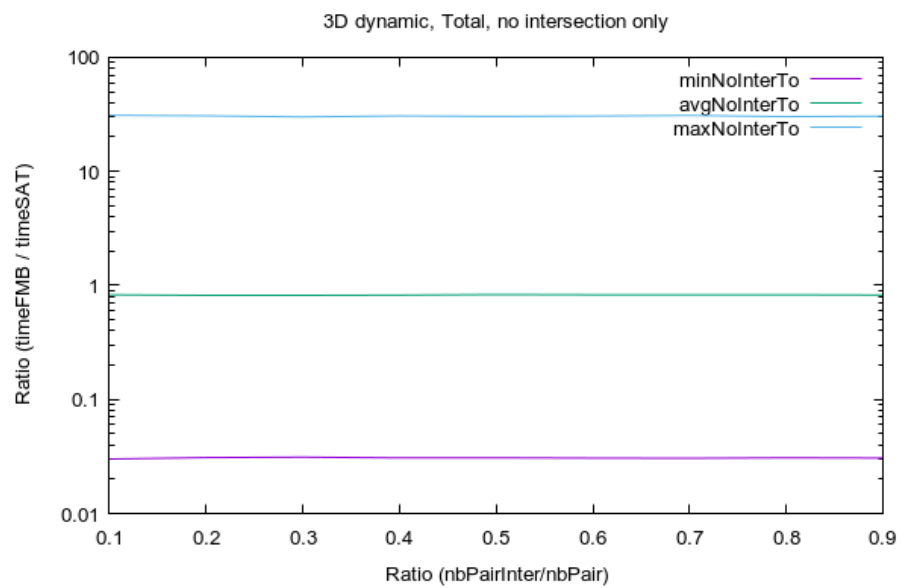
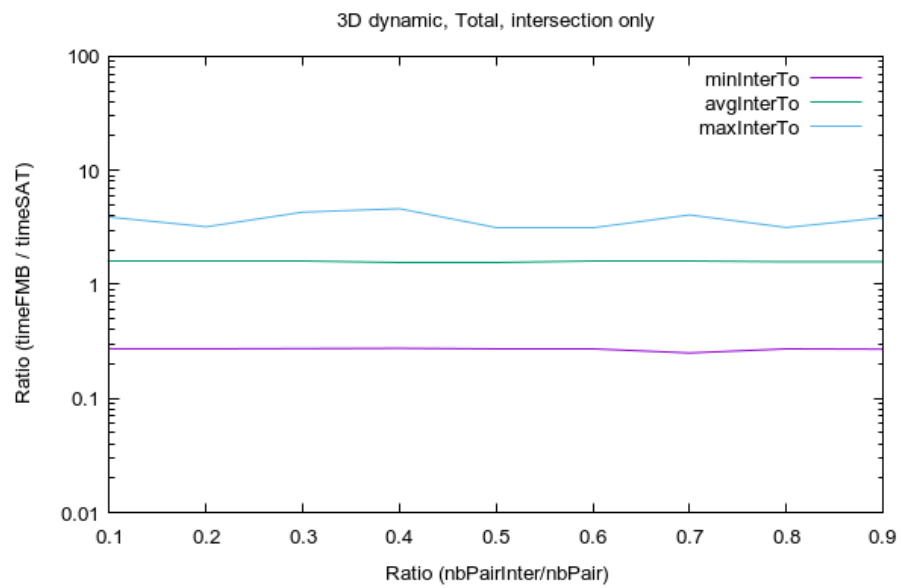


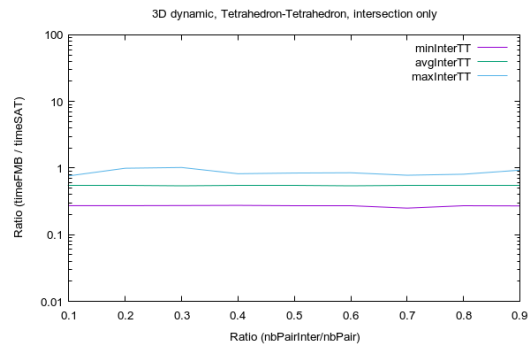
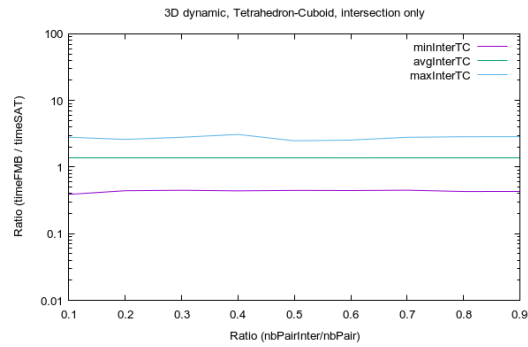
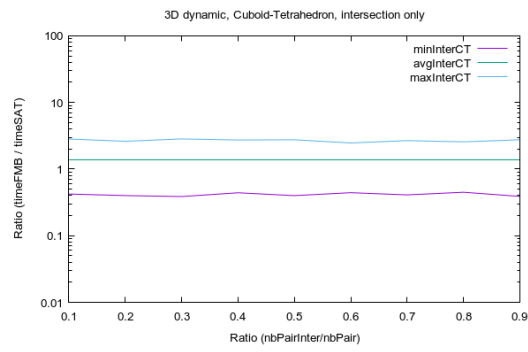
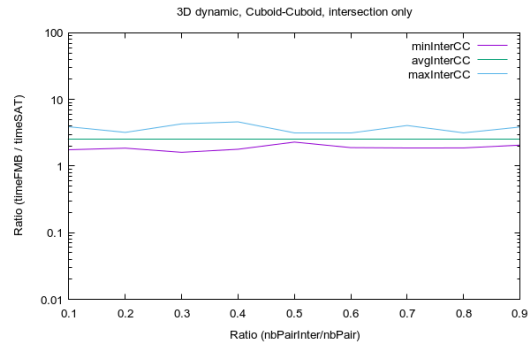


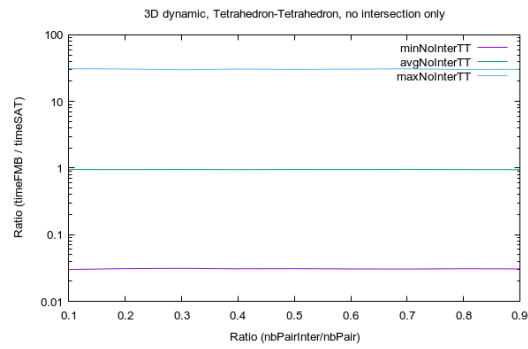
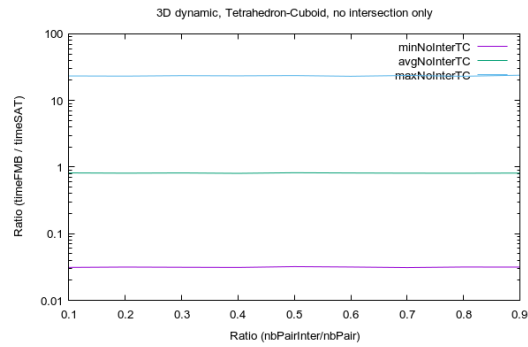
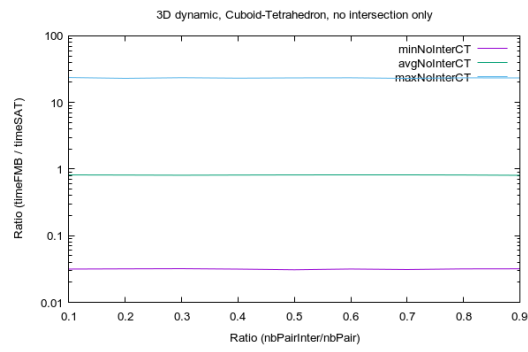
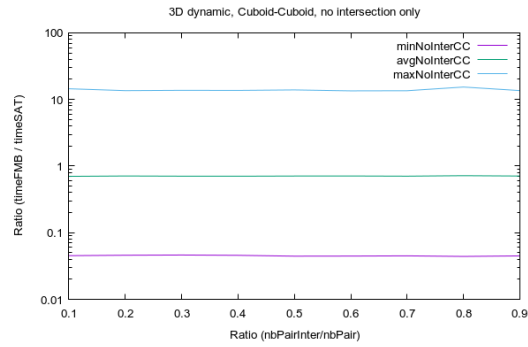
8.2.6 3D dynamic

perPairInter	countInterTo	countNoInterTo	minInterTo	avgInterTo	maxInterTo	minNoInterTo	avgNoInterTo	maxNoInterTo	minTotalTo	avgTotalTo	maxTotalTo
0.1	263100	736898	0.272840	1.561274	3.888405	0.030021	0.826719	31.058824	0.030021	0.920215	31.058824
0.2	261908	736092	0.272670	1.562669	3.204188	0.030928	0.825578	30.705882	0.030928	0.977160	30.705882
0.3	261512	738488	0.273651	1.581505	4.301226	0.031204	0.825611	30.028571	0.031204	1.052379	30.028571
0.4	261676	738324	0.275510	1.578566	4.606457	0.030724	0.820552	30.647059	0.030724	1.124660	30.647059
0.5	263036	736964	0.271644	1.578455	3.138502	0.030769	0.830502	30.342857	0.030769	1.204479	30.342857
0.6	261666	738334	0.271636	1.582467	3.139757	0.030590	0.826732	30.542857	0.030590	1.280173	30.542857
0.7	262700	737298	0.265030	1.561929	4.064198	0.030501	0.825445	30.941176	0.030501	1.354984	30.941176
0.8	261788	738212	0.271930	1.560961	3.153713	0.030792	0.827578	30.200000	0.030792	1.430205	30.200000
0.9	261446	738552	0.270507	1.582064	3.851097	0.030657	0.823369	30.470588	0.030657	1.506195	30.470588
perPairInter	countInterCC	countNoInterCC	minInterCC	avgInterCC	maxInterCC	minNoInterCC	avgNoInterCC	maxNoInterCC	minTotalCC	avgTotalCC	maxTotalCC
0.1	80510	169652	1.757151	2.576507	3.888405	0.045168	0.701647	14.462555	0.045168	0.889133	14.462555
0.2	80068	169344	1.857585	2.575961	3.204188	0.045881	0.709939	13.558036	0.045881	1.083143	13.558036
0.3	79920	171196	1.609437	2.575821	4.301226	0.046218	0.705785	13.709821	0.046218	1.266796	13.709821
0.4	79450	170060	1.789272	2.575948	4.606457	0.045738	0.704660	13.669643	0.045738	1.453175	13.669643
0.5	80006	169574	2.290867	2.575838	3.138502	0.044400	0.708803	13.900433	0.044400	1.642320	13.900433
0.6	80168	169750	1.692830	2.575839	3.139757	0.044534	0.710288	13.461538	0.044534	1.829618	13.461538
0.7	80146	169696	1.868863	2.576050	4.064198	0.044898	0.705756	13.528634	0.044898	2.014962	13.528634
0.8	79882	169488	1.872865	2.576015	3.153713	0.044044	0.717938	15.358079	0.044044	2.204400	15.358079
0.9	79948	170784	2.063019	2.576160	3.851097	0.044852	0.709212	13.578475	0.044852	2.389465	13.578475
perPairInter	countInterCT	countNoInterCT	minInterCT	avgInterCT	maxInterCT	minNoInterCT	avgNoInterCT	maxNoInterCT	minTotalCT	avgTotalCT	maxTotalCT
0.1	65772	184162	0.422029	1.377077	2.825038	0.031548	0.816739	23.625000	0.031548	0.872773	23.625000
0.2	65588	184144	0.399866	1.375866	2.614317	0.031770	0.813767	23.000000	0.031770	0.926187	23.000000
0.3	65822	184558	0.386951	1.377168	2.834586	0.031970	0.807982	23.479167	0.031970	0.978767	23.479167
0.4	65736	184546	0.440220	1.376676	2.732479	0.031508	0.812901	23.145833	0.031508	1.038411	23.145833
0.5	65980	183972	0.399425	1.376893	2.753569	0.030769	0.817254	23.354167	0.030769	1.097074	23.354167
0.6	65072	184054	0.441748	1.377508	2.467057	0.031579	0.811526	23.437500	0.031579	1.151115	23.437500
0.7	65856	184532	0.409984	1.377702	2.674735	0.031019	0.810963	22.959184	0.031019	1.207651	22.959184
0.8	65598	184002	0.448743	1.376599	2.562735	0.031722	0.815030	23.404255	0.031722	1.264285	23.404255
0.9	65264	184734	0.389841	1.376281	2.756757	0.031818	0.806429	23.333333	0.031818	1.319296	23.333333
perPairInter	countInterTC	countNoInterTC	minInterTC	avgInterTC	maxInterTC	minNoInterTC	avgNoInterTC	maxNoInterTC	minTotalTC	avgTotalTC	maxTotalTC
0.1	65582	183946	0.389928	1.375972	2.802360	0.031157	0.816626	23.326531	0.031157	0.872561	23.326531
0.2	65964	184362	0.440896	1.375880	2.602056	0.031555	0.809641	23.166667	0.031555	0.922889	23.166667
0.3	65128	184432	0.448031	1.376825	2.804787	0.031320	0.816360	23.520833	0.031320	0.984433	23.520833
0.4	65670	184124	0.438875	1.376058	3.087842	0.031114	0.804804	23.387755	0.031114	1.033305	23.387755
0.5	65748	184292	0.445885	1.375641	2.479042	0.032037	0.822639	23.604167	0.032037	1.099140	23.604167
0.6	65806	185352	0.443908	1.375559	2.541979	0.031603	0.814424	23.061224	0.031603	1.151105	23.061224
0.7	65942	184196	0.449612	1.377563	2.801187	0.030973	0.810005	23.680851	0.030973	1.207296	23.680851
0.8	65564	184696	0.430012	1.376377	2.848915	0.031571	0.808994	23.080000	0.031571	1.262900	23.080000
0.9	65726	184066	0.430467	1.376366	2.860674	0.031484	0.812602	23.895833	0.031484	1.319899	23.895833
perPairInter	countInterTT	countNoInterTT	minInterTT	avgInterTT	maxInterTT	minNoInterTT	avgNoInterTT	maxNoInterTT	minTotalTT	avgTotalTT	maxTotalTT
0.1	51236	199138	0.272840	0.542323	0.764244	0.030021	0.951824	31.058824	0.030021	0.910874	31.058824
0.2	50288	200220	0.272670	0.542131	0.993947	0.030928	0.949663	30.705882	0.030928	0.988156	30.705882
0.3	50642	198302	0.273651	0.541412	1.019949	0.031204	0.954078	30.028571	0.031204	0.830279	30.028571
0.4	50820	199894	0.275510	0.542126	0.822251	0.030724	0.946459	30.647059	0.030724	0.947726	30.647059
0.5	51302	199126	0.271644	0.542179	0.842747	0.030860	0.953568	30.342857	0.030860	0.747918	30.342857
0.6	50620	199178	0.271636	0.541696	0.849780	0.030590	0.951476	30.542857	0.030590	0.705608	30.542857
0.7	50766	198874	0.265030	0.542665	0.781110	0.030501	0.955403	30.941176	0.030501	0.666488	30.941176
0.8	50744	200026	0.271930	0.542535	0.808280	0.030792	0.949183	30.200000	0.030792	0.623885	30.200000
0.9	50508	198968	0.270507	0.542111	0.923974	0.030657	0.947044	30.470588	0.030657	0.582604	30.470588









9 Comments about the qualification results

For the 2D static case:

- FMB is in average 1.3 times slower than SAT to detect intersection and non intersection between Tetrahedrons.
- FMB is in average 1.9 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.3 times slower to detect non intersection.
- FMB is in average 2.5 times slower than SAT to detect intersection between Cuboids, and 1.2 times slower to detect non intersection.

FMB is then in average always slower (from 2.5 times to 1.2 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

For the 3D static case:

- FMB is in average 4.2 times faster than SAT to detect intersection between Tetrahedrons, and 1.1 times faster to detect non intersection.
- FMB is in average 2.4 times faster than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.5 times faster to detect non intersection.
- FMB is in average 1.4 times faster than SAT to detect intersection between Cuboids, and 2.0 times faster to detect non intersection.

FMB is then in average always faster (from 4.2 times to 1.1 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

For the 2D dynamic case:

- FMB is in average 1.7 times slower than SAT to detect intersection between Tetrahedrons, and 1.5 times slower to detect non intersection.
- FMB is in average 2.2 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.5 times slower to detect non intersection.
- FMB is in average 3.1 times slower than SAT to detect intersection between Cuboids, and 1.5 times slower to detect non intersection.

FMB is then in average always slower (from 3.1 times to 1.5 times) than SAT whatever the combinaison of Tetrahedron and Cuboid and the percentage of intersection.

For the 3D dynamic case:

- FMB is in average 1.8 times faster than SAT to detect intersection between Tetrahedrons, and 1.05 times faster to detect non intersection.
- FMB is in average 1.4 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.2 times faster to detect non intersection.
- FMB is in average 2.6 times slower than SAT to detect intersection between Cuboids, and 1.4 times faster to detect non intersection.

FMB is then in average always faster (from 1.8 times to 1.05 times) than SAT for a set of Tetrahedron, and faster than SAT for a combinaison of Tetrahedrons and Cuboids containing less than around 35% of intersection, or a combinaison of Cuboids containing less than around 15% of intersection.

Overall, FMB is faster than SAT, at least if the percentage of intersecting Frames is low, for the 3D cases. But it is always slower for the 2D dynamic cases. In practice, for example in applications where the Frames represents real world objects supposedly normally not in intersection, FMB would be a

better choice than SAT for 3D applications.

In a real world collision detection system the pair of Frames would be pruned by first applying a rough but fast detection algorithm. The slower but accurate SAT and FMB algorithms would be then only applied to pairs of Frames closed together. To ensure the results stay valid on this subset of possible cases, I've also run the qualification for 2D and 3D static cases in 'near case only' mode. In this mode, only pairs of Frames whose AABBs are intersecting are used. The results are as follow.

For the 2D static near only case:

- FMB is in average 1.3 times slower than SAT to detect intersection between Tetrahedrons, and 1.4 times slower to detect non intersection.
- FMB is in average 1.9 times slower than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.4 times slower to detect non intersection.
- FMB is in average 2.5 times slower than SAT to detect intersection between Cuboids, and 1.3 times slower to detect non intersection.

There is no significant difference with the general cases.

For the 3D static near only case:

- FMB is in average 4.3 times faster than SAT to detect intersection between Tetrahedrons, and 1.1 times faster to detect non intersection.
- FMB is in average 2.4 times faster than SAT to detect intersection between a Tetrahedron and a Cuboid, and 1.5 times faster to detect non intersection.
- FMB is in average 1.4 times faster than SAT to detect intersection between Cuboids, and 2.0 times faster to detect non intersection.

There is no significant difference with the general cases.

SAT and FMB follows the same strategy: assume that the pair of Frames is in intersection and try to prove it is false by checking a list of conditions. These conditions are the difference between the two algorithms. The results of the qualification show that in average the conditions used by FMB allows to detect a non intersection faster than those of SAT.

For one given pair in intersection, all the conditions must be checked before the algorithms give their answer. The algorithm with the smallest execution time of all these conditions is then the fastest, and the results shows that this is in general SAT (the exceptions are the 3D static case and 3D dynamic case for Tetrahedrons pairs). This is shown in the results by the low variability of the ratio $\text{timeFMB}/\text{timeSAT}$ for intersecting pairs.

For one given pair not in intersection, the algorithms reply as soon as one condition is verified. This may be the first one, as it may be the last one depending on the geometry of the pair of Frames. Then, the variability of the ratio $\text{timeFMB}/\text{timeSAT}$ varies widely as shown in the results, from 50 times faster to 29 times slower, but the results shows that in general the advantage goes to the FMB algorithm (the exception is the 2D dynamic case).

In the SAT algorithm, one must perform the projection of all vertices on one axis and then check the result which is the intersection condition. Every axis comes from the geometry of the Frames and one cannot preview which one will be lead to the checked condition for a non intersecting pair. In the FMB algorithm, the conditions depends on the way the system of linear inequation is built. Then, for best performances, it must be done in such a way that inequality (45) is encountered as soon as possible. With the FMB representation, contrary to the SAT one, it is possible to do so independantly of the geometry of the pair of Frames by reordering the inequalities of the system. For example, the $X_i \leq 1.0$ inequalities must be moved down to the end of the system of the linear inequalities for better performance, as they will never lead to '(45) is true' at the first step of the Fourier-Motzkin algorithm.

Looking for other rearrangement of the inequations, I've come to the conclusion that the best possible case (in term of speed) is, when checking Frame A against Frame B, to have:

- B's origin is the nearest vertex of B relative to A's origin
- the projection of B's origin in A's coordinate system is such as components of $\overrightarrow{AB_A}$ are all positive

This the best possible case because it minimised the a_i in (45) in the initial system or during Fourier-Motzkin algorithm, which leads quickly to '(45) is true' if the Frames are not in intersection. The Frame representation is invariant of the vertex chosen as origin, so it's possible to rearrange them to try to fit the conditions above (however it's not always possible to fit both). I've checked that it effectively leads to slightly better performances by first modifying the qualification program to generate only these cases, and then by adding a rearrangement of the origins at the beginning of the FMB algorithm. Unfortunately, the cost of the origin rearrangement is heavier than its benefit. Still, I believe one may find some clever rearrangement which would lead to even better performance for the FMB algorithm.

10 Conclusion

In this paper I've introduced the FMB algorithm which solve efficiently the intersection detection problem of 2D/3D static/dynamic cuboid/tetrahedron by using the Fourier-Motzkin elimination method. All information necessary to implement and use the FMB algorithm, or reproduce the results introduced in this paper are included in this paper, and available on the GitHub repository <https://github.com/BayashiPascal/FMB/>.

Validation and qualification against the SAT algorithm prove the correctness of the results from the FMB algorithm and prove it's a valid alternative in term of performance to the SAT algorithm, especially when applied to tetrahedrons and/or in the 3D static case. It is also important to note its simplicity to implement, and the fact that the FMB algorithm returns a bounding box of the intersection, if any, while the SAT algorithm only returns a boolean answer.

Idea on direction to explore with the view to improve the FMB algorithm is given. Steps of the Fourier-Motzkin could also be easily parallelized on an appropriate architecture to improve performance. Tests of implementation with others programming languages, or on other runtime environments, or against other algorithms (such as CJK) would also be interesting to perform. Finally, while the algorithm is introduced here in 2D and 3D, its extension

to upper dimensions is straightforward.

11 Annex

11.1 Runtime environment

Results introduced 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);

// Test for intersection between 3D Frame that and 3D Frame tho,
// both faces
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3DFace(
    const Frame3D* const that,
    const Frame3D* const tho);

// Check the intersection constraint for Frames that and tho,
// both faces, relatively to axis
bool CheckAxis3DFace(
    const Frame3D* const that,
    const Frame3D* const tho,
    const double* const axis);

#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 jFrame = 2;
            jFrame--;) {

            // 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 = 0;

    if (iEdge == 3) {

        edge = relSpeed;

    } else if (iEdge == 2) {

        if (frameEdgeType == FrameTetrahedron) {

            edge = thirdEdge;

        } else {

            edge = relSpeed;

        }

    } else {

        edge = frameEdge->comp[iEdge];

    }

    // Declare variables to memorize the boundaries of projection
    // of the two frames on the current edge
    double bdgBoxA[2];
    double bdgBoxB[2];

```

```

// Declare two variables to loop on Frames and commonalize code
const Frame2DTime* frame = that;
double* bdgBox = bdgBoxA;

// Loop on Frames
for (
    int jFrame = 2;
    jFrame--;) {

    // Shortcuts
    const double* frameOrig = frame->orig;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    FrameType frameType = frame->type;

    // Get the number of vertices of frame
    int nbVertices = (frameType == FrameTetrahedron ? 3 : 4);

    // Declare a variable to memorize if the current vertex is
    // the first in the loop, used to initialize the boundaries
    bool firstVertex = true;

    // Loop on vertices of the frame
    for (
        int iVertex = nbVertices;
        iVertex--;) {

        // Get the vertex
        double vertex[2];
        vertex[0] = frameOrig[0];
        vertex[1] = frameOrig[1];
        switch (iVertex) {

            case 3:
                vertex[0] += frameCompA[0] + frameCompB[0];
                vertex[1] += frameCompA[1] + frameCompB[1];
                break;
            case 2:
                vertex[0] += frameCompA[0];
                vertex[1] += frameCompA[1];
                break;
            case 1:
                vertex[0] += frameCompB[0];
                vertex[1] += frameCompB[1];
                break;
            default:
                break;
        }

        // Get the projection of the vertex on the normal of the edge
        // Orientation of the normal doesn't matter, so we
        // use arbitrarily the normal (edge[1], -edge[0])
        double proj = vertex[0] * edge[1] - vertex[1] * edge[0];

        // If it's the first vertex
        if (firstVertex == true) {

            // Initialize the boundaries of the projection of the
            // Frame on the edge
            bdgBox[0] = proj;
            bdgBox[1] = proj;
        }
    }
}

```

```

        // Update the flag to memorize we did the first vertex
        firstVertex = false;

    // Else, it's not the first vertex
    } else {

        // Update the boundaries of the projection of the Frame on
        // the edge
        if (bdgBox[0] > proj) {

            bdgBox[0] = proj;

        }

        if (bdgBox[1] < proj) {

            bdgBox[1] = proj;

        }

    }

    // If we are checking the second frame's vertices
    if (frame == tho) {

        // Check also the vertices moved by the relative speed
        vertex[0] += relSpeed[0];
        vertex[1] += relSpeed[1];

        proj = vertex[0] * edge[1] - vertex[1] * edge[0];

        if (bdgBox[0] > proj) {

            bdgBox[0] = proj;

        }

        if (bdgBox[1] < proj) {

            bdgBox[1] = proj;

        }

    }

}

// Switch the frame to check the vertices of the second Frame
frame = tho;
bdgBox = bdgBoxB;

}

// If the projections of the two frames on the edge are
// not intersecting
if (
    bdgBoxB[1] < bdgBoxA[0] ||
    bdgBoxA[1] < bdgBoxB[0]) {

    // There exists an axis which separates the Frames,
    // thus they are not in intersection

```

```

        return false;

    }

}

// Switch the frames to test against the second Frame's edges
frameEdge = tho;

}

// If we reaches here, it means the two Frames are intersecting
return true;

}

// Test for intersection between 3D Frame that and 3D Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3D(
    const Frame3D* const that,
    const Frame3D* const tho) {

    // Declare two variables to memorize the opposite edges in case
    // of tetrahedron
    double oppEdgesThat[3][3];
    double oppEdgesTho[3][3];

    // Declare two variables to memorize the number of edges, by default 3
    int nbEdgesThat = 3;
    int nbEdgesTho = 3;

    // If the first Frame is a tetrahedron
    if (that->type == FrameTetrahedron) {

        // Shortcuts
        const double* frameCompA = that->comp[0];
        const double* frameCompB = that->comp[1];
        const double* frameCompC = that->comp[2];

        // Initialise the opposite edges
        oppEdgesThat[0][0] = frameCompB[0] - frameCompA[0];
        oppEdgesThat[0][1] = frameCompB[1] - frameCompA[1];
        oppEdgesThat[0][2] = frameCompB[2] - frameCompA[2];

        oppEdgesThat[1][0] = frameCompB[0] - frameCompC[0];
        oppEdgesThat[1][1] = frameCompB[1] - frameCompC[1];
        oppEdgesThat[1][2] = frameCompB[2] - frameCompC[2];

        oppEdgesThat[2][0] = frameCompC[0] - frameCompA[0];
        oppEdgesThat[2][1] = frameCompC[1] - frameCompA[1];
        oppEdgesThat[2][2] = frameCompC[2] - frameCompA[2];

        // Correct the number of edges
        nbEdgesThat = 6;

    }

    // If the second Frame is a tetrahedron
    if (tho->type == FrameTetrahedron) {

        // Shortcuts
        const double* frameCompA = tho->comp[0];

```

```

const double* frameCompB = tho->comp[1];
const double* frameCompC = tho->comp[2];

// Initialise the opposite edges
oppEdgesTho[0][0] = frameCompB[0] - frameCompA[0];
oppEdgesTho[0][1] = frameCompB[1] - frameCompA[1];
oppEdgesTho[0][2] = frameCompB[2] - frameCompA[2];

oppEdgesTho[1][0] = frameCompB[0] - frameCompC[0];
oppEdgesTho[1][1] = frameCompB[1] - frameCompC[1];
oppEdgesTho[1][2] = frameCompB[2] - frameCompC[2];

oppEdgesTho[2][0] = frameCompC[0] - frameCompA[0];
oppEdgesTho[2][1] = frameCompC[1] - frameCompA[1];
oppEdgesTho[2][2] = frameCompC[2] - frameCompA[2];

// Correct the number of edges
nbEdgesTho = 6;
}

// Declare variables to loop on Frames and commonalize code
const Frame3D* frame = that;
const double (*oppEdgesA)[3] = oppEdgesThat;

// Loop to commonalize code when checking SAT based on that's edges
// and then tho's edges
for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    FrameType frameType = frame->type;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    const double* frameCompC = frame->comp[2];

    // Declare a variable to memorize the number of faces, by default 3
    int nbFaces = 3;

    // Declare a variable to memorize the normal to faces
    // Arrangement is normFaces[iFace][iAxis]
    double normFaces[4][3];

    // Initialise the normal to faces
    normFaces[0][0] =
        frameCompA[1] * frameCompB[2] -
        frameCompA[2] * frameCompB[1];
    normFaces[0][1] =
        frameCompA[2] * frameCompB[0] -
        frameCompA[0] * frameCompB[2];
    normFaces[0][2] =
        frameCompA[0] * frameCompB[1] -
        frameCompA[1] * frameCompB[0];

    normFaces[1][0] =
        frameCompA[1] * frameCompC[2] -
        frameCompA[2] * frameCompC[1];
    normFaces[1][1] =
        frameCompA[2] * frameCompC[0] -
        frameCompA[0] * frameCompC[2];
    normFaces[1][2] =

```

```

    frameCompA[0] * frameCompC[1] -
    frameCompA[1] * frameCompC[0];

normFaces[2][0] =
    frameCompC[1] * frameCompB[2] -
    frameCompC[2] * frameCompB[1];
normFaces[2][1] =
    frameCompC[2] * frameCompB[0] -
    frameCompC[0] * frameCompB[2];
normFaces[2][2] =
    frameCompC[0] * frameCompB[1] -
    frameCompC[1] * frameCompB[0];

// If the frame is a tetrahedron
if (frameType == FrameTetrahedron) {

    // Shortcuts
    const double* oppEdgeA = oppEdgesA[0];
    const double* oppEdgeB = oppEdgesA[1];

    // Initialise the normal to the opposite face
    normFaces[3][0] =
        oppEdgeA[1] * oppEdgeB[2] -
        oppEdgeA[2] * oppEdgeB[1];
    normFaces[3][1] =
        oppEdgeA[2] * oppEdgeB[0] -
        oppEdgeA[0] * oppEdgeB[2];
    normFaces[3][2] =
        oppEdgeA[0] * oppEdgeB[1] -
        oppEdgeA[1] * oppEdgeB[0];

    // Correct the number of faces
    nbFaces = 4;

}

// Loop on the frame's faces
for (
    int iFace = nbFaces;
    iFace--;) {

    // Check against the current face's normal
    bool isIntersection =
        CheckAxis3D(
            that,
            tho,
            normFaces[iFace]);

    // If the axis is separating the Frames
    if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;

    }

}

// Switch the frame to test against the second Frame
frame = tho;
oppEdgesA = oppEdgesTho;

```



```

}

// Loop on the pair of edges between the two frames
for (
    int iEdgeThat = nbEdgesThat;
    iEdgeThat--;) {

    // Get the first edge
    const double* edgeThat = NULL;
    if (iEdgeThat < 3) {

        edgeThat = that->comp[iEdgeThat];

    } else {

        edgeThat = oppEdgesThat[iEdgeThat - 3];

    }

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

        // Get the second edge
        const double* edgeTho = NULL;
        if (iEdgeTho < 3) {

            edgeTho = tho->comp[iEdgeTho];

        } else {

            edgeTho = oppEdgesTho[iEdgeTho - 3];

        }

        // Get the cross product of the two edges
        double axis[3];
        axis[0] = edgeThat[1] * edgeTho[2] - edgeThat[2] * edgeTho[1];
        axis[1] = edgeThat[2] * edgeTho[0] - edgeThat[0] * edgeTho[2];
        axis[2] = edgeThat[0] * edgeTho[1] - edgeThat[1] * edgeTho[0];

        // Check against the cross product of the two edges
        bool isIntersection =
            CheckAxis3D(
                that,
                tho,
                axis);

        // If the axis is separating the Frames
        if (isIntersection == false) {

            // The Frames are not in intersection,
            // terminate the test
            return false;

        }

    }

}

}

```

```

    // If we reaches here, it means the two Frames are intersecting
    return true;
}

// Test for intersection between moving 3D Frame that and 3D
// Frame tho
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3DTime(
    const Frame3DTime* const that,
    const Frame3DTime* const tho) {

    // Declare two variables to memorize the opposite edges in case
    // of tetrahedron
    double oppEdgesThat[3][3];
    double oppEdgesTho[3][3];

    // Declare a variable to memorize the speed of tho relative to that
    double relSpeed[3];
    relSpeed[0] = tho->speed[0] - that->speed[0];
    relSpeed[1] = tho->speed[1] - that->speed[1];
    relSpeed[2] = tho->speed[2] - that->speed[2];

    // Declare two variables to memorize the number of edges, by default 3
    int nbEdgesThat = 3;
    int nbEdgesTho = 3;

    // If the first Frame is a tetrahedron
    if (that->type == FrameTetrahedron) {

        // Shortcuts
        const double* frameCompA = that->comp[0];
        const double* frameCompB = that->comp[1];
        const double* frameCompC = that->comp[2];

        // Initialise the opposite edges
        oppEdgesThat[0][0] = frameCompB[0] - frameCompA[0];
        oppEdgesThat[0][1] = frameCompB[1] - frameCompA[1];
        oppEdgesThat[0][2] = frameCompB[2] - frameCompA[2];

        oppEdgesThat[1][0] = frameCompB[0] - frameCompC[0];
        oppEdgesThat[1][1] = frameCompB[1] - frameCompC[1];
        oppEdgesThat[1][2] = frameCompB[2] - frameCompC[2];

        oppEdgesThat[2][0] = frameCompC[0] - frameCompA[0];
        oppEdgesThat[2][1] = frameCompC[1] - frameCompA[1];
        oppEdgesThat[2][2] = frameCompC[2] - frameCompA[2];

        // Correct the number of edges
        nbEdgesThat = 6;
    }

    // If the second Frame is a tetrahedron
    if (tho->type == FrameTetrahedron) {

        // Shortcuts
        const double* frameCompA = tho->comp[0];
        const double* frameCompB = tho->comp[1];
        const double* frameCompC = tho->comp[2];

        // Initialise the opposite edges

```

```

    oppEdgesTho[0][0] = frameCompB[0] - frameCompA[0];
    oppEdgesTho[0][1] = frameCompB[1] - frameCompA[1];
    oppEdgesTho[0][2] = frameCompB[2] - frameCompA[2];

    oppEdgesTho[1][0] = frameCompB[0] - frameCompC[0];
    oppEdgesTho[1][1] = frameCompB[1] - frameCompC[1];
    oppEdgesTho[1][2] = frameCompB[2] - frameCompC[2];

    oppEdgesTho[2][0] = frameCompC[0] - frameCompA[0];
    oppEdgesTho[2][1] = frameCompC[1] - frameCompA[1];
    oppEdgesTho[2][2] = frameCompC[2] - frameCompA[2];

    // Correct the number of edges
    nbEdgesTho = 6;
}

// Declare variables to loop on Frames and commonalize code
const Frame3DTime* frame = that;
const double (*oppEdgesA)[3] = oppEdgesThat;

// Loop to commonalize code when checking SAT based on that's edges
// and then tho's edges
for (
    int iFrame = 2;
    iFrame--;) {

    // Shortcuts
    FrameType frameType = frame->type;
    const double* frameCompA = frame->comp[0];
    const double* frameCompB = frame->comp[1];
    const double* frameCompC = frame->comp[2];

    // Declare a variable to memorize the number of faces, by default 3
    int nbFaces = 3;

    // Declare a variable to memorize the normal to faces
    // Arrangement is normFaces[iFace][iAxis]
    double normFaces[10][3];

    // Initialise the normal to faces
    normFaces[0][0] =
        frameCompA[1] * frameCompB[2] -
        frameCompA[2] * frameCompB[1];
    normFaces[0][1] =
        frameCompA[2] * frameCompB[0] -
        frameCompA[0] * frameCompB[2];
    normFaces[0][2] =
        frameCompA[0] * frameCompB[1] -
        frameCompA[1] * frameCompB[0];

    normFaces[1][0] =
        frameCompA[1] * frameCompC[2] -
        frameCompA[2] * frameCompC[1];
    normFaces[1][1] =
        frameCompA[2] * frameCompC[0] -
        frameCompA[0] * frameCompC[2];
    normFaces[1][2] =
        frameCompA[0] * frameCompC[1] -
        frameCompA[1] * frameCompC[0];

    normFaces[2][0] =

```

```

        frameCompC[1] * frameCompB[2] -
        frameCompC[2] * frameCompB[1];
normFaces[2][1] =
        frameCompC[2] * frameCompB[0] -
        frameCompC[0] * frameCompB[2];
normFaces[2][2] =
        frameCompC[0] * frameCompB[1] -
        frameCompC[1] * frameCompB[0];

// If the frame is a tetrahedron
if (frameType == FrameTetrahedron) {

    // Shortcuts
    const double* oppEdgeA = oppEdgesA[0];
    const double* oppEdgeB = oppEdgesA[1];

    // Initialise the normal to the opposite face
    normFaces[3][0] =
        oppEdgeA[1] * oppEdgeB[2] -
        oppEdgeA[2] * oppEdgeB[1];
    normFaces[3][1] =
        oppEdgeA[2] * oppEdgeB[0] -
        oppEdgeA[0] * oppEdgeB[2];
    normFaces[3][2] =
        oppEdgeA[0] * oppEdgeB[1] -
        oppEdgeA[1] * oppEdgeB[0];

    // Correct the number of faces
    nbFaces = 4;
}

// If we are checking the frame 'tho'
if (frame == tho) {

    // Add the normal to the virtual faces created by the speed
    // of tho relative to that
    normFaces[nbFaces][0] =
        relSpeed[1] * frameCompA[2] -
        relSpeed[2] * frameCompA[1];
    normFaces[nbFaces][1] =
        relSpeed[2] * frameCompA[0] -
        relSpeed[0] * frameCompA[2];
    normFaces[nbFaces][2] =
        relSpeed[0] * frameCompA[1] -
        relSpeed[1] * frameCompA[0];
    if (
        fabs(normFaces[nbFaces][0]) > EPSILON ||
        fabs(normFaces[nbFaces][1]) > EPSILON ||
        fabs(normFaces[nbFaces][2]) > EPSILON) {

        ++nbFaces;
    }

    normFaces[nbFaces][0] =
        relSpeed[1] * frameCompB[2] -
        relSpeed[2] * frameCompB[1];
    normFaces[nbFaces][1] =
        relSpeed[2] * frameCompB[0] -
        relSpeed[0] * frameCompB[2];
    normFaces[nbFaces][2] =

```

```

        relSpeed[0] * frameCompB[1] -
        relSpeed[1] * frameCompB[0];
    if (
        fabs(normFaces[nbFaces][0]) > EPSILON ||
        fabs(normFaces[nbFaces][1]) > EPSILON ||
        fabs(normFaces[nbFaces][2]) > EPSILON) {

        ++nbFaces;

    }

    normFaces[nbFaces][0] =
        relSpeed[1] * frameCompC[2] -
        relSpeed[2] * frameCompC[1];
    normFaces[nbFaces][1] =
        relSpeed[2] * frameCompC[0] -
        relSpeed[0] * frameCompC[2];
    normFaces[nbFaces][2] =
        relSpeed[0] * frameCompC[1] -
        relSpeed[1] * frameCompC[0];
    if (
        fabs(normFaces[nbFaces][0]) > EPSILON ||
        fabs(normFaces[nbFaces][1]) > EPSILON ||
        fabs(normFaces[nbFaces][2]) > EPSILON) {

        ++nbFaces;

    }

    if (frameType == FrameTetrahedron) {

        const double* oppEdgeA = oppEdgesA[0];
        const double* oppEdgeB = oppEdgesA[1];
        const double* oppEdgeC = oppEdgesA[2];

        normFaces[nbFaces][0] =
            relSpeed[1] * oppEdgeA[2] -
            relSpeed[2] * oppEdgeA[1];
        normFaces[nbFaces][1] =
            relSpeed[2] * oppEdgeA[0] -
            relSpeed[0] * oppEdgeA[2];
        normFaces[nbFaces][2] =
            relSpeed[0] * oppEdgeA[1] -
            relSpeed[1] * oppEdgeA[0];
        if (
            fabs(normFaces[nbFaces][0]) > EPSILON ||
            fabs(normFaces[nbFaces][1]) > EPSILON ||
            fabs(normFaces[nbFaces][2]) > EPSILON) {

            ++nbFaces;

        }

        normFaces[nbFaces][0] =
            relSpeed[1] * oppEdgeB[2] -
            relSpeed[2] * oppEdgeB[1];
        normFaces[nbFaces][1] =
            relSpeed[2] * oppEdgeB[0] -
            relSpeed[0] * oppEdgeB[2];
        normFaces[nbFaces][2] =
            relSpeed[0] * oppEdgeB[1] -
            relSpeed[1] * oppEdgeB[0];
    }

```

```

        if (
            fabs(normFaces[nbFaces][0]) > EPSILON ||
            fabs(normFaces[nbFaces][1]) > EPSILON ||
            fabs(normFaces[nbFaces][2]) > EPSILON) {

            ++nbFaces;

        }

        normFaces[nbFaces][0] =
            relSpeed[1] * oppEdgeC[2] -
            relSpeed[2] * oppEdgeC[1];
        normFaces[nbFaces][1] =
            relSpeed[2] * oppEdgeC[0] -
            relSpeed[0] * oppEdgeC[2];
        normFaces[nbFaces][2] =
            relSpeed[0] * oppEdgeC[1] -
            relSpeed[1] * oppEdgeC[0];
        if (
            fabs(normFaces[nbFaces][0]) > EPSILON ||
            fabs(normFaces[nbFaces][1]) > EPSILON ||
            fabs(normFaces[nbFaces][2]) > EPSILON) {

            ++nbFaces;

        }

    }

}

// Loop on the frame's faces
for (
    int iFace = nbFaces;
    iFace--;) {

    // Check against the current face's normal
    bool isIntersection =
        CheckAxis3DTime(
            that,
            tho,
            normFaces[iFace],
            relSpeed);

    // If the axis is separating the Frames
    if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;

    }

}

// Switch the frame to test against the second Frame
frame = tho;
oppEdgesA = oppEdgesTho;

}

// Loop on the pair of edges between the two frames

```

```

for (
    int iEdgeThat = nbEdgesThat;
    iEdgeThat--;) {

    // Get the first edge
    const double* edgeThat = NULL;
    if (iEdgeThat < 3) {

        edgeThat = that->comp[iEdgeThat];

    } else {

        edgeThat = oppEdgesThat[iEdgeThat - 3];

    }

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

        // Get the second edge
        const double* edgeTho = NULL;
        if (iEdgeTho == nbEdgesTho) {

            edgeTho = relSpeed;

        } else if (iEdgeTho < 3) {

            edgeTho = tho->comp[iEdgeTho];

        } else {

            edgeTho = oppEdgesTho[iEdgeTho - 3];

        }

        // Get the cross product of the two edges
        double axis[3];
        axis[0] = edgeThat[1] * edgeTho[2] - edgeThat[2] * edgeTho[1];
        axis[1] = edgeThat[2] * edgeTho[0] - edgeThat[0] * edgeTho[2];
        axis[2] = edgeThat[0] * edgeTho[1] - edgeThat[1] * edgeTho[0];

        // Check against the cross product of the two edges
        bool isIntersection =
            CheckAxis3DTime(
                that,
                tho,
                axis,
                relSpeed);

        // If the axis is separating the Frames
        if (isIntersection == false) {

            // The Frames are not in intersection,
            // terminate the test
            return false;

        }

    }

}

}

```

```

    // If we reaches here, it means the two Frames are intersecting
    return true;
}

// Test for intersection between 3D Frame that and 3D Frame tho,
// both faces
// Return true if the two Frames are intersecting, else false
bool SATTestIntersection3DFace(
    const Frame3D* const that,
    const Frame3D* const tho) {

    // Check against that's normal
    bool isIntersection =
        CheckAxis3DFace(
            that,
            tho,
            that->comp[2]);

    // If the axis is separating the Frames
    if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;
    }

    // Check against tho's normal
    isIntersection =
        CheckAxis3DFace(
            that,
            tho,
            tho->comp[2]);

    // If the axis is separating the Frames
    if (isIntersection == false) {

        // The Frames are not in intersection,
        // terminate the test
        return false;
    }

    // Declare two variables to memorize the opposite edges in case
    // of tetrahedron
    double oppEdgeThat[3];
    double oppEdgeTho[3];

    // Declare two variables to memorize the number of edges, by default 3
    int nbEdgesThat = 2;
    int nbEdgesTho = 2;

    // If the first Frame is a tetrahedron
    if (that->type == FrameTetrahedron) {

        // Shortcuts
        const double* frameCompA = that->comp[0];
        const double* frameCompB = that->comp[1];

        // Initialise the opposite edges

```



```

    oppEdgeThat[0] = frameCompB[0] - frameCompA[0];
    oppEdgeThat[1] = frameCompB[1] - frameCompA[1];
    oppEdgeThat[2] = frameCompB[2] - frameCompA[2];

    // Correct the number of edges
    nbEdgesThat = 3;
}

// If the second Frame is a tetrahedron
if (tho->type == FrameTetrahedron) {

    // Shortcuts
    const double* frameCompA = tho->comp[0];
    const double* frameCompB = tho->comp[1];

    // Initialise the opposite edges
    oppEdgeTho[0] = frameCompB[0] - frameCompA[0];
    oppEdgeTho[1] = frameCompB[1] - frameCompA[1];
    oppEdgeTho[2] = frameCompB[2] - frameCompA[2];

    // Correct the number of edges
    nbEdgesTho = 3;
}

// Loop on the pair of edges between the two frames
for (
    int iEdgeThat = nbEdgesThat;
    iEdgeThat--;) {

    // Get the first edge
    const double* edgeThat = NULL;
    if (iEdgeThat < 2) {

        edgeThat = that->comp[iEdgeThat];

    } else {

        edgeThat = oppEdgeThat;

    }

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

        // Get the second edge
        const double* edgeTho = NULL;
        if (iEdgeTho < 2) {

            edgeTho = tho->comp[iEdgeTho];

        } else {

            edgeTho = oppEdgeTho;

        }

        // 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 =
    CheckAxis3DFace(
        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;

}

// 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,
// both faces, relatively to axis
bool CheckAxis3DFace(
    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];
    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[3];
        vertex[0] = frameOrig[0];
        vertex[1] = frameOrig[1];
        vertex[2] = frameOrig[2];
        switch (iVertex) {

            case 3:
                vertex[0] +=
                    frameCompA[0] + frameCompB[0];
                vertex[1] +=
                    frameCompA[1] + frameCompB[1];
                vertex[2] +=
                    frameCompA[2] + frameCompB[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 : cd2DTime; makemainOPTIMIZATION =
        (OPTIMIZATION); cd -

main3D:
    cd 3D; make main OPTIMIZATION=
        (OPTIMIZATION); cd - main3DTime : cd3DTime; makemainOPTIMIZATION =
        (OPTIMIZATION); cd -

unitTests : unitTests2D unitTests2DTime unitTests3D unitTests3DTime

unitTests2D:
    cd 2D; make unitTests OPTIMIZATION=
        (OPTIMIZATION); cd - unitTests2DTime : cd2DTime; makeunitTestsOPTIMIZATION =
        (OPTIMIZATION); cd -

unitTests3D:
    cd 3D; make unitTests OPTIMIZATION=
        (OPTIMIZATION); cd - unitTests3DTime : cd3DTime; makeunitTestsOPTIMIZATION =
        (OPTIMIZATION); cd -

validation : validation2D validation2DTime validation3D validation3DTime

validation2D:
    cd 2D; make validation OPTIMIZATION=
        (OPTIMIZATION); cd - validation2DTime : cd2DTime; makevalidationOPTIMIZATION =
        (OPTIMIZATION); cd -

validation3D:
    cd 3D; make validation OPTIMIZATION=
        (OPTIMIZATION); cd - validation3DTime : cd3DTime; makevalidationOPTIMIZATION =
        (OPTIMIZATION); cd -

qualification : qualification2D qualification2DTime qualification3D
                qualification3DTime

qualification2D:
    cd 2D; make qualification OPTIMIZATION=
        (OPTIMIZATION); cd - qualification2DTime : cd2DTime; makequalificationOPTIMIZATION =
        (OPTIMIZATION); cd -

qualification3D:
    cd 3D; make qualification OPTIMIZATION=
        (OPTIMIZATION); cd - qualification3DTime : cd3DTime; makequalificationOPTIMIZATION =
        (OPTIMIZATION); cd -

clean : clean2D clean2DTime clean3D clean3DTime

```

```

clean2D:
    cd 2D; make clean; cd -

clean2DTime:
    cd 2DTime; make clean; cd -

clean3D:
    cd 3D; make clean; cd -

clean3DTime:
    cd 3DTime; make clean; cd -

valgrind : valgrind2D valgrind2DTime valgrind3D valgrind3DTime

valgrind2D:
    cd 2D; make valgrind; cd -

valgrind2DTime:
    cd 2DTime; make valgrind; cd -

valgrind3D:
    cd 3D; make valgrind; cd -

valgrind3DTime:
    cd 3DTime; make valgrind; cd -

cppcheck : cppcheck2D cppcheck2DTime cppcheck3D cppcheck3DTime

cppcheck2D:
    cd 2D; make cppcheck; cd -

cppcheck2DTime:
    cd 2DTime; make cppcheck; cd -

cppcheck3D:
    cd 3D; make cppcheck; cd -

cppcheck3DTime:
    cd 3DTime; make cppcheck; cd -

run : run2D run2DNearCaseOnly run2DTime run3D run3DNearCaseOnly run3DTime

run2D:
    cd 2D; ./main > ../Results/main2D.txt; ./unitTests > ../Results/
    unitTests2D.txt; ./validation > ../Results/validation2D.txt; ./
    qualification; cd -

run3D:
    cd 3D; ./main > ../Results/main3D.txt; ./unitTests > ../Results/
    unitTests3D.txt; ./validation > ../Results/validation3D.txt; ./
    qualification; cd -

run2DNearCaseOnly:
    cd 2D; ./qualification -nearCaseOnly; cd -

run3DNearCaseOnly:
    cd 3D; ./qualification -nearCaseOnly; cd -

run2DTime:
    cd 2DTime; ./main > ../Results/main2DTime.txt; ./unitTests > ../
    Results/unitTests2DTime.txt; ./validation > ../Results/

```

```

validation2DTime.txt; ./qualification; cd -

run3DTime:
    cd 3DTime; ./main > ../Results/main3DTime.txt; ./unitTests > ../
    Results/unitTests3DTime.txt; ./validation > ../Results/
    validation3DTime.txt; ./qualification; cd -

plot: cleanPlot plot2D plot2DNearCaseOnly plot2DTime plot3D
    plot3DNearCaseOnly plot3DTime

cleanPlot:
    rm -f Results/*.png

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

plot2DNearCaseOnly:
    cd Results; gnuplot qualification2DNearCaseOnly.gnu; cd -

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

plot3D:
    cd Results; gnuplot qualification3D.gnu; cd -

plot3DNearCaseOnly:
    cd Results; gnuplot qualification3DNearCaseOnly.gnu; cd -

plot3DTime:
    cd Results; gnuplot qualification3DTime.gnu; cd -

doc:
    cd Doc; make latex; cd -

getRuntimeEnvironment:
    echo "uname -v\n" > runtimeEnv.txt; uname -v >> runtimeEnv.txt; echo
    "\n=====\n" >> runtimeEnv.txt; echo "lshw -short\n" >>
    runtimeEnv.txt; sudo lshw -short >> runtimeEnv.txt; echo "\n
    =====\n" >> runtimeEnv.txt; echo "lscpu\n" >> runtimeEnv
    .txt; lscpu >> runtimeEnv.txt; echo "\n=====\n" >>
    runtimeEnv.txt; echo "(COMPILER) -v" >> runtimeEnv.txt; (COMPILER)
    -v 1>> runtimeEnv.txt 2>> runtimeEnv.txt

dynamicAnalysis:
    make valgrind 1> dynamicAnalysis.txt 2> dynamicAnalysis.txt

staticAnalysis:
    make cppcheck 1> staticAnalysis.txt 2> staticAnalysis.txt

```

11.3.1 2D static

```

all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=(OPTIMIZATION) -I../SAT -I../Frame main : main.ofmb2d.oframe.o Makefile(
    COMPILER) -o main main.o fmb2d.o frame.o

main.o : main.c fmb2d.h ../Frame/frame.h Makefile

```

```

        (COMPILER) - cmain.c(BUILD_ARG)

unitTests : unitTests.o fmb2d.o frame.o Makefile
        (COMPILER) - ounitTestsunitTests.ofmb2d.oframe.o(LINK_ARG)

unitTests.o : unitTests.c fmb2d.h ../Frame/frame.h Makefile
        (COMPILER) - cunitTests.c(BUILD_ARG)

validation : validation.o fmb2d.o sat.o frame.o Makefile
        (COMPILER) - ovalidationvalidation.ofmb2d.osat.oframe.ovalidation.o : validation.cfmb2d.h../SAT/sat.h../Frame/
        (COMPILER) -c validation.c
        (BUILD_ARG)qualification : qualification.ofmb2d.osat.oframe.oMakefile(
        COMPILER) -o qualification qualification.o fmb2d.o sat.o frame.o

        (LINK_ARG)qualification.o : qualification.cfmb2d.h../SAT/sat.h../Frame/frame.hMakefile
        (COMPILER) -c qualification.c
        (BUILD_ARG)fmb2d.o : fmb2d.cfmb2d.h../Frame/frame.hMakefile(COMPILER) -
        c fmb2d.c
        (BUILD_ARG)sat.o : ../SAT/sat.c../SAT/sat.h../Frame/frame.hMakefile(
        COMPILER) -c ../SAT/sat.c
        (BUILD_ARG)frame.o : ../Frame/frame.c../Frame/frame.hMakefile(COMPILER)
        -c ../Frame/frame.c

```

11.3.2 3D static

```

all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=(OPTIMIZATION) - I../SAT - I../Framemain : main.ofmb3d.oframe.oMakefile(
        COMPILER) -o main main.o fmb3d.o frame.o

main.o : main.c fmb3d.h ../Frame/frame.h Makefile
        (COMPILER) - cmain.c(BUILD_ARG)

unitTests : unitTests.o fmb3d.o frame.o Makefile
        (COMPILER) - ounitTestsunitTests.ofmb3d.oframe.o(LINK_ARG)

unitTests.o : unitTests.c fmb3d.h ../Frame/frame.h Makefile
        (COMPILER) - cunitTests.c(BUILD_ARG)

validation : validation.o fmb3d.o sat.o frame.o Makefile
        (COMPILER) - ovalidationvalidation.ofmb3d.osat.oframe.ovalidation.o : validation.cfmb3d.h../SAT/sat.h../Frame/
        (COMPILER) -c validation.c
        (BUILD_ARG)qualification : qualification.ofmb3d.osat.oframe.oMakefile(
        COMPILER) -o qualification qualification.o fmb3d.o sat.o frame.o

        (LINK_ARG)qualification.o : qualification.cfmb3d.h../SAT/sat.h../Frame/frame.hMakefile
        (COMPILER) -c qualification.c
        (BUILD_ARG)fmb3d.o : fmb3d.cfmb3d.h../Frame/frame.hMakefile(COMPILER) -
        c fmb3d.c
        (BUILD_ARG)sat.o : ../SAT/sat.c../SAT/sat.h../Frame/frame.hMakefile(
        COMPILER) -c ../SAT/sat.c
        (BUILD_ARG)frame.o : ../Frame/frame.c../Frame/frame.hMakefile(COMPILER)
        -c ../Frame/frame.c

```

11.3.3 2D dynamic

```

all : main unitTests validation qualification

COMPILER?=gcc

```

```

OPTIMIZATION?=-O3
BUILD_ARG=(OPTIMIZATION) - I../SAT - I../Frame main : main.ofmb2dt.oframe.o Makefile(
    COMPILER) -o main main.o fmb2dt.o frame.o

main.o : main.c fmb2dt.h ../Frame/frame.h Makefile
    (COMPILER) - cmain.c(BUILD_ARG)

unitTests : unitTests.o fmb2dt.o frame.o Makefile
    (COMPILER) - ounitTests unitTests.ofmb2dt.oframe.o(LINK_ARG)

unitTests.o : unitTests.c fmb2dt.h ../Frame/frame.h Makefile
    (COMPILER) - cunitTests.c(BUILD_ARG)

validation : validation.o fmb2dt.o sat.o frame.o Makefile
    (COMPILER) - ovalidation validation.ofmb2dt.osat.oframe.ovalidation.o : validation.cfmb2dt.h../SAT/sat.h../Frame
    (COMPILER) -c validation.c
    (BUILD_ARG) qualification : qualification.ofmb2dt.osat.oframe.o Makefile(
    COMPILER) -o qualification qualification.o fmb2dt.o sat.o frame.
    o
    (LINK_ARG) qualification.o : qualification.cfmb2dt.h../SAT/sat.h../Frame/frame.h Makefile
    (COMPILER) -c qualification.c
    (BUILD_ARG) fmb2dt.o : fmb2dt.cfmb2dt.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

```

11.3.4 3D dynamic

```

all : main unitTests validation qualification

COMPILER?=gcc
OPTIMIZATION?=-O3
BUILD_ARG=(OPTIMIZATION) - I../SAT - I../Frame main : main.ofmb3dt.oframe.o Makefile(
    COMPILER) -o main main.o fmb3dt.o frame.o

main.o : main.c fmb3dt.h ../Frame/frame.h Makefile
    (COMPILER) - cmain.c(BUILD_ARG)

unitTests : unitTests.o fmb3dt.o frame.o Makefile
    (COMPILER) - ounitTests unitTests.ofmb3dt.oframe.o(LINK_ARG)

unitTests.o : unitTests.c fmb3dt.h ../Frame/frame.h Makefile
    (COMPILER) - cunitTests.c(BUILD_ARG)

validation : validation.o fmb3dt.o sat.o frame.o Makefile
    (COMPILER) - ovalidation validation.ofmb3dt.osat.oframe.ovalidation.o : validation.cfmb3dt.h../SAT/sat.h../Frame
    (COMPILER) -c validation.c
    (BUILD_ARG) qualification : qualification.ofmb3dt.osat.oframe.o Makefile(
    COMPILER) -o qualification qualification.o fmb3dt.o sat.o frame.
    o
    (LINK_ARG) qualification.o : qualification.cfmb3dt.h../SAT/sat.h../Frame/frame.h Makefile
    (COMPILER) -c qualification.c
    (BUILD_ARG) fmb3dt.o : fmb3dt.cfmb3dt.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

```

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
==8646== Memcheck, a memory error detector
==8646== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8646== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8646== Command: ./unitTests
==8646==
--8646-- Valgrind options:
--8646--      -v
--8646--      --track-origins=yes
--8646--      --leak-check=full
--8646--      --gen-suppressions=yes
--8646--      --show-leak-kinds=all
--8646-- Contents of /proc/version:
--8646--      Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
        version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
        Mar 31 04:17:56 UTC 2020
--8646--
--8646-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
        avx-avx2-bmi
--8646-- Page sizes: currently 4096, max supported 4096
--8646-- Valgrind library directory: /usr/lib/valgrind
--8646-- Reading syms from /home/bayashi/GitHub/FMB/2D/unitTests
--8646-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--8646--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--8646--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--8646--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--8646--   .. CRC is valid
--8646-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--8646--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--8646--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--8646--   object doesn't have a symbol table
--8646--   object doesn't have a dynamic symbol table
--8646-- Scheduler: using generic scheduler lock implementation.
--8646-- Reading suppressions file: /usr/lib/valgrind/default.supp
==8646== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-8646-
        by-bayashi-on-???
==8646== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-8646-
        by-bayashi-on-???
==8646== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
        -8646-by-bayashi-on-???
==8646==
==8646== TO CONTROL THIS PROCESS USING vgdb (which you probably
==8646== don't want to do, unless you know exactly what you're doing,
==8646== or are doing some strange experiment):
==8646==   /usr/lib/valgrind/../../bin/vgdb --pid=8646 ...command...
==8646==
==8646== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==8646==   /path/to/gdb ./unitTests
==8646== and then give GDB the following command
==8646==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=8646
```

```

==8646== --pid is optional if only one valgrind process is running
==8646==
--8646-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
x580608c1 (???)
--8646-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
x580608db (???)
--8646-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--8646-- Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--8646-- .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--8646-- object doesn't have a symbol table
--8646-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--8646-- Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--8646-- .. CRC mismatch (computed f893b962 wanted 95ee359e)
--8646-- object doesn't have a symbol table
==8646== WARNING: new redirection conflicts with existing -- ignoring it
--8646-- old: 0x0401f2f0 (strlen ) R-> (0000.0) 0x580608c1
???
--8646-- new: 0x0401f2f0 (strlen ) R-> (2007.0) 0x04c32db0
strlen
--8646-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
x4c33ee0 (strcmp)
--8646-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
x4c374f0 (mempcpy)
--8646-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--8646-- Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--8646-- .. CRC mismatch (computed b1c74187 wanted 042cc048)
--8646-- Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--8646-- .. CRC is valid
--8646-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)

```



```

--8646-- REDIR: 0x4edb0e0 (libc.so.6:memcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9da0 (libc.so.6:strpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edaef0 (libc.so.6:stpcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--8646-- REDIR: 0x4fca3c0 (libc.so.6:__strrchr_avx2) redirected to 0x4c32730
(rindex)
--8646-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--8646-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--8646-- REDIR: 0x4fcaab0 (libc.so.6:__memcpy_avx_unaligned_erms)
redirected to 0x4c37130 (memcpy)
--8646-- 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.
--8646-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==8646==
==8646== HEAP SUMMARY:
==8646==       in use at exit: 0 bytes in 0 blocks
==8646==     total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==8646==
==8646== All heap blocks were freed -- no leaks are possible
==8646==
==8646== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==8646== 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
==8650== Memcheck, a memory error detector

```

```

==8650== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8650== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8650== Command: ./unitTests
==8650==
--8650-- Valgrind options:
--8650--     -v
--8650--     --track-origins=yes
--8650--     --leak-check=full
--8650--     --gen-suppressions=yes
--8650--     --show-leak-kinds=all
--8650-- Contents of /proc/version:
--8650--     Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
        version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
        Mar 31 04:17:56 UTC 2020
--8650--
--8650-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
        avx-avx2-bmi
--8650-- Page sizes: currently 4096, max supported 4096
--8650-- Valgrind library directory: /usr/lib/valgrind
--8650-- Reading syms from /home/bayashi/GitHub/FMB/2DTime/unitTests
--8650-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--8650--     Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--8650--     .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--8650--     Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--8650--     .. CRC is valid
--8650-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--8650--     Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--8650--     .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--8650--     object doesn't have a symbol table
--8650--     object doesn't have a dynamic symbol table
--8650-- Scheduler: using generic scheduler lock implementation.
--8650-- Reading suppressions file: /usr/lib/valgrind/default.supp
==8650== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-8650-
        by-bayashi-on-???
==8650== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-8650-
        by-bayashi-on-???
==8650== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
        -8650-by-bayashi-on-???
==8650==
==8650== TO CONTROL THIS PROCESS USING vgdb (which you probably
==8650== don't want to do, unless you know exactly what you're doing,
==8650== or are doing some strange experiment):
==8650==   /usr/lib/valgrind/../../bin/vgdb --pid=8650 ...command...
==8650==
==8650== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==8650==   /path/to/gdb ./unitTests
==8650== and then give GDB the following command
==8650==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=8650
==8650== --pid is optional if only one valgrind process is running
==8650==
--8650-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
        x580608c1 (???)
--8650-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
        x580608db (???)
--8650-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--8650--     Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--8650--     .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--8650--     object doesn't have a symbol table
--8650-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
        so
--8650--     Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
        ..

```

```

--8650--    .. CRC mismatch (computed f893b962 wanted 95ee359e)
--8650--    object doesn't have a symbol table
==8650== WARNING: new redirection conflicts with existing -- ignoring it
--8650--    old: 0x0401f2f0 (strlen          ) R-> (0000.0) 0x580608c1
--8650--    ???
--8650--    new: 0x0401f2f0 (strlen          ) R-> (2007.0) 0x04c32db0
--8650--    strlen
--8650-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
--8650--    x4c33ee0 (strcmp)
--8650-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
--8650--    x4c374f0 (mempcpy)
--8650-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--8650--    Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--8650--    .. CRC mismatch (computed b1c74187 wanted 042cc048)
--8650--    Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--8650--    .. CRC is valid
--8650-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edb0e0 (libc.so.6:mempcpy@@GLIBC_2.14) redirected to 0
--8650--    x4a2a6e0 (_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
--8650--    _vgnU_ifunc_wrapper)

```

```

--8650-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--8650-- REDIR: 0x4fca3c0 (libc.so.6:__strchr_avx2) redirected to 0x4c32730
(rindex)
--8650-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--8650-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--8650-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
redirected to 0x4c37130 (mempcpy)
--8650-- 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.
--8650-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==8650==
==8650== HEAP SUMMARY:
==8650==      in use at exit: 0 bytes in 0 blocks
==8650==    total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==8650==
==8650== All heap blocks were freed -- no leaks are possible
==8650==
==8650== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

```

==8650== 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
==8653== Memcheck, a memory error detector
==8653== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8653== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8653== Command: ./unitTests
==8653==
--8653-- Valgrind options:
--8653--   -v
--8653--   --track-origins=yes
--8653--   --leak-check=full
--8653--   --gen-suppressions=yes
--8653--   --show-leak-kinds=all
--8653-- Contents of /proc/version:
--8653--   Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
        version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
        Mar 31 04:17:56 UTC 2020
--8653--
--8653-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-
        avx-avx2-bmi
--8653-- Page sizes: currently 4096, max supported 4096
--8653-- Valgrind library directory: /usr/lib/valgrind
--8653-- Reading syms from /home/bayashi/GitHub/FMB/3D/unitTests
--8653-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--8653--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--8653--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--8653--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--8653--   .. CRC is valid
--8653-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--8653--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--8653--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--8653--   object doesn't have a symbol table
--8653--   object doesn't have a dynamic symbol table
--8653-- Scheduler: using generic scheduler lock implementation.
--8653-- Reading suppressions file: /usr/lib/valgrind/default.supp
==8653== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-8653-
        by-bayashi-on-???
==8653== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-8653-
        by-bayashi-on-???
==8653== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
        -8653-by-bayashi-on-???
==8653==
==8653== TO CONTROL THIS PROCESS USING vgdb (which you probably
==8653== don't want to do, unless you know exactly what you're doing,
==8653== or are doing some strange experiment):
==8653==   /usr/lib/valgrind/../../bin/vgdb --pid=8653 ...command...
==8653==
==8653== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==8653==   /path/to/gdb ./unitTests
==8653== and then give GDB the following command
==8653==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=8653
==8653== --pid is optional if only one valgrind process is running
==8653==
--8653-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
        x580608c1 (???)
--8653-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
        x580608db (???)

```

```

--8653-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--8653--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--8653--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--8653--   object doesn't have a symbol table
--8653-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--8653--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--8653--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--8653--   object doesn't have a symbol table
==8653== WARNING: new redirection conflicts with existing -- ignoring it
--8653--   old: 0x0401f2f0 (strlen          ) R-> (0000.0) 0x580608c1
???
--8653--   new: 0x0401f2f0 (strlen          ) R-> (2007.0) 0x04c32db0
strlen
--8653-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
x4c33ee0 (strcmp)
--8653-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
x4c374f0 (mempcpy)
--8653-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--8653--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--8653--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--8653--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--8653--   .. CRC is valid
--8653-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edb0e0 (libc.so.6:mempcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)

```

```

--8653-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--8653-- REDIR: 0x4fca3c0 (libc.so.6:__strchr_avx2) redirected to 0x4c32730
(rindex)
--8653-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)
--8653-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--8653-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
redirected to 0x4c37130 (mempcpy)
--8653-- REDIR: 0x4fca590 (libc.so.6:__strlen_avx2) redirected to 0x4c32cf0
(strlen)
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.500000,0.500000,0.500000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
(0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

```

```

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)

Co(-0.500000,-0.500000,-0.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.000000,0.000000,0.000000)-maxXYZ(0.500000,0.500000,0.500000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(1.500000,1.500000,1.500000) x(-1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,0.500000)-maxXYZ(1.000000,1.000000,1.000000)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed (no inter)

Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
against
Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

Co(0.500000,1.500000,-1.500000) x(1.000000,0.000000,0.000000) y
  (0.000000,-1.000000,0.000000) z(0.000000,0.000000,1.000000)
against
Co(0.000000,0.000000,0.000000) x(1.000000,0.000000,0.000000) y
  (0.000000,1.000000,0.000000) z(0.000000,0.000000,-1.000000)
Succeed
minXYZ(0.500000,0.500000,-1.000000)-maxXYZ(1.000000,1.000000,-0.500000)

```



```

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.
--8653-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==8653==
==8653== HEAP SUMMARY:
==8653==      in use at exit: 0 bytes in 0 blocks
==8653==    total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==8653==
==8653== All heap blocks were freed -- no leaks are possible
==8653==
==8653== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==8653== 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
==8656== Memcheck, a memory error detector
==8656== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8656== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8656== Command: ./unitTests
==8656==
--8656-- Valgrind options:
--8656--      -v
--8656--      --track-origins=yes
--8656--      --leak-check=full
--8656--      --gen-suppressions=yes
--8656--      --show-leak-kinds=all
--8656-- Contents of /proc/version:
--8656--      Linux version 5.3.0-46-generic (buildd@lcy01-amd64-013) (gcc
        version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38~18.04.1-Ubuntu SMP Tue
        Mar 31 04:17:56 UTC 2020
--8656--
--8656-- Arch and hwcaps: AMD64, LittleEndian, amd64-cx16-lzcnt-rdtscp-sse3-

```

```

    avx-avx2-bmi
--8656-- Page sizes: currently 4096, max supported 4096
--8656-- Valgrind library directory: /usr/lib/valgrind
--8656-- Reading syms from /home/bayashi/GitHub/FMB/3DTime/unitTests
--8656-- Reading syms from /lib/x86_64-linux-gnu/ld-2.27.so
--8656--   Considering /lib/x86_64-linux-gnu/ld-2.27.so ..
--8656--   .. CRC mismatch (computed 1b7c895e wanted 2943108a)
--8656--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/ld-2.27.so ..
--8656--   .. CRC is valid
--8656-- Reading syms from /usr/lib/valgrind/memcheck-amd64-linux
--8656--   Considering /usr/lib/valgrind/memcheck-amd64-linux ..
--8656--   .. CRC mismatch (computed 41ddb025 wanted 9972f546)
--8656--   object doesn't have a symbol table
--8656--   object doesn't have a dynamic symbol table
--8656-- Scheduler: using generic scheduler lock implementation.
--8656-- Reading suppressions file: /usr/lib/valgrind/default.supp
==8656== embedded gdbserver: reading from /tmp/vgdb-pipe-from-vgdb-to-8656-
by-bayashi-on-???
==8656== embedded gdbserver: writing to   /tmp/vgdb-pipe-to-vgdb-from-8656-
by-bayashi-on-???
==8656== embedded gdbserver: shared mem   /tmp/vgdb-pipe-shared-mem-vgdb
-8656-by-bayashi-on-???
==8656==
==8656== TO CONTROL THIS PROCESS USING vgdb (which you probably
==8656== don't want to do, unless you know exactly what you're doing,
==8656== or are doing some strange experiment):
==8656==   /usr/lib/valgrind/../../bin/vgdb --pid=8656 ...command...
==8656==
==8656== TO DEBUG THIS PROCESS USING GDB: start GDB like this
==8656==   /path/to/gdb ./unitTests
==8656== and then give GDB the following command
==8656==   target remote | /usr/lib/valgrind/../../bin/vgdb --pid=8656
==8656== --pid is optional if only one valgrind process is running
==8656==
--8656-- REDIR: 0x401f2f0 (ld-linux-x86-64.so.2:strlen) redirected to 0
x580608c1 (???)
--8656-- REDIR: 0x401f0d0 (ld-linux-x86-64.so.2:index) redirected to 0
x580608db (???)
--8656-- Reading syms from /usr/lib/valgrind/vgpreload_core-amd64-linux.so
--8656--   Considering /usr/lib/valgrind/vgpreload_core-amd64-linux.so ..
--8656--   .. CRC mismatch (computed 50df1b30 wanted 4800a4cf)
--8656--   object doesn't have a symbol table
--8656-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.
so
--8656--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
..
--8656--   .. CRC mismatch (computed f893b962 wanted 95ee359e)
--8656--   object doesn't have a symbol table
==8656== WARNING: new redirection conflicts with existing -- ignoring it
--8656--   old: 0x0401f2f0 (strlen                          ) R-> (0000.0) 0x580608c1
???
--8656--   new: 0x0401f2f0 (strlen                          ) R-> (2007.0) 0x04c32db0
strlen
--8656-- REDIR: 0x401d360 (ld-linux-x86-64.so.2:strcmp) redirected to 0
x4c33ee0 (strcmp)
--8656-- REDIR: 0x401f830 (ld-linux-x86-64.so.2:mempcpy) redirected to 0
x4c374f0 (mempcpy)
--8656-- Reading syms from /lib/x86_64-linux-gnu/libc-2.27.so
--8656--   Considering /lib/x86_64-linux-gnu/libc-2.27.so ..
--8656--   .. CRC mismatch (computed b1c74187 wanted 042cc048)
--8656--   Considering /usr/lib/debug/lib/x86_64-linux-gnu/libc-2.27.so ..
--8656--   .. CRC is valid

```



```

--8656-- REDIR: 0x4edac70 (libc.so.6:memmove) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9d40 (libc.so.6:strncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edaf50 (libc.so.6:strcasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9790 (libc.so.6:strcat) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9d70 (libc.so.6:rindex) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edc7c0 (libc.so.6:rawmemchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edade0 (libc.so.6:mempcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edac10 (libc.so.6:bcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9d00 (libc.so.6:strncmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9800 (libc.so.6:strcmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edad40 (libc.so.6:memset) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ef80f0 (libc.so.6:wcschr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9ca0 (libc.so.6:strnlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9870 (libc.so.6:strcspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edafa0 (libc.so.6:strncasecmp) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9840 (libc.so.6:strcpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edb0e0 (libc.so.6:mempcpy@@GLIBC_2.14) redirected to 0
x4a2a6e0 (_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9da0 (libc.so.6:stpbrk) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed97c0 (libc.so.6:index) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ed9c70 (libc.so.6:strlen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ee46c0 (libc.so.6:memrchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edaff0 (libc.so.6:strcasecmp_l) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edabe0 (libc.so.6:memchr) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4ef8eb0 (libc.so.6:wcslen) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4eda050 (libc.so.6:strspn) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edaf20 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edaef0 (libc.so.6:stpncpy) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edc7f0 (libc.so.6:strchrnul) redirected to 0x4a2a6e0 (
_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4edb040 (libc.so.6:strncasecmp_l) redirected to 0x4a2a6e0
(_vgnU_ifunc_wrapper)
--8656-- REDIR: 0x4fca3c0 (libc.so.6:__strchr_avx2) redirected to 0x4c32730
(rindex)
--8656-- REDIR: 0x4ed3070 (libc.so.6:malloc) redirected to 0x4c2faa0 (malloc
)

```

```

--8656-- REDIR: 0x4fca1d0 (libc.so.6:__strchrnul_avx2) redirected to 0
x4c37020 (strchrnul)
--8656-- REDIR: 0x4fcaab0 (libc.so.6:__mempcpy_avx_unaligned_erms)
redirected to 0x4c37130 (mempcpy)
--8656-- 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.
--8656-- REDIR: 0x4ed3950 (libc.so.6:free) redirected to 0x4c30cd0 (free)
==8656==
==8656== HEAP SUMMARY:
==8656==       in use at exit: 0 bytes in 0 blocks
==8656==     total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==8656==
==8656== All heap blocks were freed -- no leaks are possible
==8656==
==8656== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==8656== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3DTime'
/home/bayashi/GitHub/FMB
make[1]: Leaving directory '/home/bayashi/GitHub/FMB'

```

11.5 Static analysis

```

make[1]: Entering directory '/home/bayashi/GitHub/FMB'
cd 2D; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2D'
cppcheck --enable=all ./
Checking fmb2d.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2D'
/home/bayashi/GitHub/FMB
cd 2DTime; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/2DTime'
cppcheck --enable=all ./
Checking fmb2dt.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/2DTime'
/home/bayashi/GitHub/FMB
cd 3D; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3D'
cppcheck --enable=all ./

```

```

Checking fmb3d.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3D'
/home/bayashi/GitHub/FMB
cd 3DTime; make cppcheck; cd -
make[2]: Entering directory '/home/bayashi/GitHub/FMB/3DTime'
cppcheck --enable=all ./
Checking fmb3dt.c ...
1/5 files checked 20% done
Checking main.c ...
2/5 files checked 40% done
Checking qualification.c ...
3/5 files checked 60% done
Checking unitTests.c ...
4/5 files checked 80% done
Checking validation.c ...
5/5 files checked 100% done
(information) Cppcheck cannot find all the include files (use --check-config
for details)
make[2]: Leaving directory '/home/bayashi/GitHub/FMB/3DTime'
/home/bayashi/GitHub/FMB
make[1]: Leaving directory '/home/bayashi/GitHub/FMB'

```

11.6 Versions of this paper

This paper has been revised has follow:

- February 5th, 2020 : Original version
- April 5th, 2020 : Addition of the 'near-case only' results in qualification.
- April 25th, 2020 : Update of the qualification results after moving the computation of the inverse of components matrix from outside to inside the intersection detection test for a better comparison with SAT.
- May 6th, 2020 : Add image on the first page. Correct erroneous indices notation in the formulation of the problem. Add generic algorithm.
- May 13th, 2020 : Correction of indices in the code of the SAT for 2D static and 2D dynamic. No effect on results, just cleaning.

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.