Efficient Conflict Detection for Conflict Resolution

Richard Alligier, Nicolas Durand, Gregory Alligier

Efficient Conflict Detection for Conflict Resolution

Richard Alligier, Nicolas Durand
ENAC Lab
7 av. Édouard Belin
31 055 Toulouse, France
firstname.surname@enac.fr

Gregory Alligier
DSNA/DTI
1 av. du Dr Maurice Grynfogel
31 035 Toulouse, France
firstname.surname@aviation-civile.gouv.fr

***Abstract*—Accurate tools to detect and solve conflicts are becoming necessary to assist air traffic controllers in their task. Air traffic controllers will eventually rely on tools to test and choose alternative trajectories. Enabling such tools demands a near real-time conflict detection algorithm.**

A previous publication ([1]) proposed optimization methods to perform conflict resolution in real time on moderate size problems. However, this previous publication only considered the time to solve the associated combinatorial optimization problem. It did not take into account the time to compute the conflicts between the alternative trajectories. This time can be high in the scenarios envisioned in [1]. For each aircraft, 161 alternative trajectories were considered. Detecting all the conflicts required to compare 2,721,705 pairs of trajectories for a 15 aircraft scenario and 128,308,950 pairs for a 100 aircraft scenario.

The conflict detection procedure uses predicted trajectories which are inherently entangled with uncertainties. A seamlessly way to handle these uncertainties is to bound the future positions in a sequence of volume. This is how the uncertainties are modeled in the scenarios. However, this uncertainty model makes the conflict computation more time consuming.

In this paper we propose a Graphics Processing Unit (GPU) implementation of a conflict detection algorithm. Compared with a CPU implementation, the proposed algorithm reduces the computation time by two orders of magnitude. The 15 aircraft scenarios, as described in [1], are computed in 30 ms and the 100 aircraft scenarios are computed in 1 s.

Keywords: conflict detection, conflict resolution, GPU

I. Introduction

To sustain air traffic growth, accurate tools to detect conflicts are becoming necessary to assist air traffic controllers in their task. Many uncertainty models have been proposed in the literature to make future tools realistic and trustable for their users. We can for example cite Erzberger conflict probe model as one of the pioneer approaches [2], [3] in the USA. In Europe the HIPS project [4], [5], representation of the conflicting zones were taking into account uncertainties and they had to compute in real time complex detection. This is also the case in approaches proposed by French CENA[1] [6], [7] where aircraft are not modeled with points, instead trajectories are discretized in time and aircraft are represented by convex hulls that expand with time.

In 2017, Allignol et al. [1] proposed a framework to compare conflict resolution algorithms that precalculated the conflicting alternative trajectories. The model used discretized maneuver options for 161 alternative trajectories taking into account an uncertainty model able to handle various uncertainties. The number of alternative maneuvers seems high but it is quite small in comparison to all the possible combinations of parameters: different times for maneuver start and maneuver end, different types of maneuvers using speed change, altitude change, direct route, or heading changes.

The optimization methods proposed in [1] are able to efficiently solve moderate size conflict resolutions problems in real time. However, it did not take into account the time to compute the conflict between the alternative trajectories. This time can be high in the scenarios envisioned in [1]. For each aircraft, 161 alternative trajectories were considered. Detecting all the conflicts requires comparing 2,721,705 pairs of trajectories for a 15 aircraft scenario and 128,308,950 pairs for a 100 aircraft scenario.

In the context of calculating many alternative options in real time for applications such as conflict resolution, it seems very useful to focus on minimizing the time spent to detect potential conflicts between diverse options. This article proposes a method to reduce drastically the time spent to determine the conflicting alternative trajectories.

A. Related work

In this article, we implement a conflict detection algorithm on Graphics Processing Unit (GPU). Several articles report a GPU implementation to accelerate computation in the Air Traffic Management (ATM) domain. Tandale *et al.* [8] used a GPU to accelerate the computation of trajectory predictions. A two orders of magnitude speedup was obtained. The detection of aircraft entering in a restricted airspace has been implemented in a GPU by Thompson *et al.* [9]. In [10], a GPU is used to compute the conflict between two trajectories that are hours long but described with few points. The two trajectories are interpolated using a GPU and the distances are computed between the points in order to detect if the two trajectories are in conflict.

For a conflict detection problem involving n trajectories discretized in T time steps, $\frac{n(n-1)}{2}T$ distance computations are required to detect all the possible conflict between the trajectories. However the number of distance computation can be drastically reduced. Many articles rely on spatial partition to avoid testing all the possible pairs of positions. Isaacson *et al.* [11] discarded pairs according with their altitude. A spatial

[1]Centre d'Études de la Navigation Aérienne

grid was used to reduce the number of pairs in [12], [13]. In [14], [15], this idea was improved by using a spatiotemporal partition to reduce even more the number of pairs to be tested. Ruiz *et al.* [15] uses a uniform grid, and Kuenz *et al.* [14] uses a 4D partitioning tree.

These articles all model the future aircraft positions as points. Hence, uncertainties on the future positions are not modeled. Some articles [16], [17] model the position as a bivariate Gaussian random variable. Hence, the position is associated with a probability distribution with a specific Gaussian shape.

In this paper, we consider the scenarios described in [1]. We consider a conflict detection problem involving n aircraft. For each aircraft, we consider $m = 161$ alternative trajectories discretized in T time steps. Each position is modeled as a volume. We assume that this volume surely contains the actual future position of the aircraft. This volume is a convex polygon in the horizontal plane, and a minimum and maximum altitude in the vertical plane. This model is flexible and can take into account a large spectrum of uncertainty sources. However computing conflicts with such a model is very computationally demanding as we have to compute distances between convex polygons. We propose an almost brute-force GPU implemented algorithm to detect conflicts between a large number of alternative trajectories.

B. Outline

In part II, we describe the models involved in the conflict detection problem. Part III details the conflict detection algorithm used and part IV explains how it was parallelized. Part V details the experimentation made and part VI analyses the results. We conclude and bring some insight of further work to be done.

II. Description of the problem

In this section, we describe the models involved in the conflict detection problem addressed in this paper. We detail the trajectory prediction model used and we explain why an efficient conflict detection algorithm is required for conflict resolution.

A. Trajectory prediction model

In order to detect a conflict, we need to predict the future trajectories and check if there is a loss of separation between these predicted trajectories. Due to the erroneous nature of the predictions, the trajectory prediction model should handle uncertainties.

In our model, the time horizon is divided into steps of duration τ. The predicted trajectory is modeled as a sequence of positions, one position for one time step. However, in order to take into account the uncertainty on the predicted position, each position is actually modeled as a volume. This volume supposedly always contains the actual future position. In our model this volume is defined by a minimum altitude, a maximum altitude and a 2D convex in the horizontal plane. The Figure 1 illustrates such a volume.

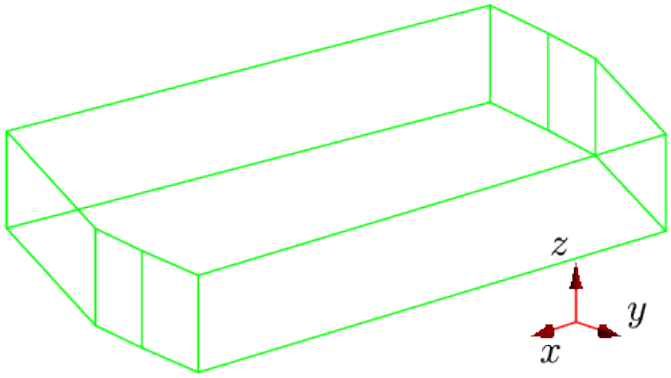

Fig. 1. In our trajectory prediction model, each position is a volume. This volume supposedly always contains the actual future position.

No assumptions are made on how these volumes are built. The trajectory prediction module only has to generate the sequence of volumes. The volume can take into account a full spectrum of uncertainties.

For instance, the size of the volume can be the result of the latency of a pilot to follow an ATC instruction, the uncertainty on the speed and the aircraft performance if a climb maneuver is involved. We can compute the predicted point-wise trajectory for different combination of values for these three uncertainty sources. Then, for each time step, we build a volume by computing the convex of the points of the different predicted point-wise trajectories. Using this procedure, we obtain the desired sequence of volumes.

Figure 2 illustrates the sequence one can obtain by applying such a procedure. In this example, we have an uncertainty on the speed, the heading change angle and the time at which this heading change is performed.

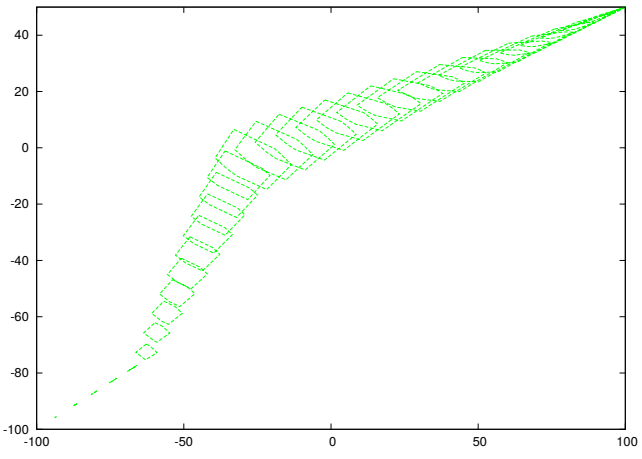

Fig. 2. The predicted trajectory is modeled as a sequence of positions. This figure shows such a sequence in the horizontal plane.

B. Conflict detection

ATC must ensure a vertical and a horizontal separation norm between the aircraft. Two predicted trajectories are said in con-

flict if there is a time at which the predicted aircraft position are neither vertically separated by $NORM_{vertical} = 1000\,\text{ft}$ nor horizontally separated by $NORM_{horizontal} = 5\,\text{NM}$.

1) Conflict between two trajectories: We want to decide if two trajectories are in conflict. With our trajectory prediction model, two trajectories p and q are in conflict if there is a time step t at which the two volumes $p(t)$ and $q(t)$ are not vertically nor horizontally separated. Two volumes $p(t)$ and $q(t)$ are not separated if and only if there exist $(x_p, y_p, z_p) \in p(t)$ and $(x_q, y_q, z_q) \in q(t)$ such that $|z_p - z_q| < NORM_{vertical}$ and $(x_p - x_q)^2 + (y_p - y_q)^2 < NORM^2{}_{horizontal}$.

We have assumed that the actual future trajectories will be always in the predicted volumes. Thus, if the predicted trajectories are separated then the actual future trajectories will be also separated.

However, one must be aware that depending on the aircraft speed, the step duration τ must be small enough to not miss any conflict.

2) Conflict between two aircraft: For a conflict between two aircraft, we have to consider alternative trajectories for these two aircraft. Assuming that we consider m alternative trajectories for each aircraft, we have to determine which ordered pair (k, l) of alternative trajectories are in conflict. These results can be stored in a conflict matrix $(C_{k,l})_{1 \leq k,l \leq m}$. If we consider trajectories with T timesteps, this matrix can be built by testing the separation of at most m^2T pairs of volumes.

3) Conflict between n aircraft: Even if being in conflict is a binary relationship, one aircraft can be in conflict with several aircraft. One way to solve this complex situation is to group the aircraft in conflict in a cluster. A cluster is a transitive closing on conflicting pairs of aircraft.

Because one maneuver of one aircraft might create conflicts with the other aircraft involved in the cluster, we need to know all the conflict matrices of all the pairs of aircraft in the cluster. Thus, with n aircraft, we have to compute $\frac{n(n-1)}{2}$ conflict matrices. In the end, it leads to compare $\frac{n(n-1)}{2}m^2T$ volumes at most. This number can be huge even with a limited number of aircraft involved. For the smallest scenario in this paper with $n = 15$ aircraft, $m = 161$ alternatives trajectories and $T = 150$ time steps, we have 408,255,750 volumes comparisons. Because conflict detection and resolution problems have to be solved quickly, there is a need to build these conflict matrices as fast as possible.

III. Conflict detection algorithm

The conflict detection problem can be seen as a collision detection problem where the moving objects are the aircraft positions increased by half the separation norm. Detecting if moving objects will collide is a problem occurring in video games, physical simulations and robotics. This problem can be decomposed in three phases [18]. The broad-phase is a phase eliminating pairs of objects that are too far away too collide. Then, for the remaining pairs we can apply a cheap collision test on bounding volumes, this is called the mid-phase. Finally, when the mid-phase failed to discard the pair, a fine-grained and costly colliding test between the two objects is done during the narrow-phase.

1) Broad-Phase: When n objects are moving, $n(n-1)/2$ pairs of potential collision need to be tested. However using a spatial partitioning or temporal coherence, one can cull away some of these pairs. One example of spatial partitioning is the uniform grid [19]. The 3D space is split in cells. Objects that are not in adjacent cells are not colliding. An example of algorithm exploiting temporal coherence is the sweep and prune algorithm [20]. In this type of algorithm, a structure is incrementally updated at each time step. In this algorithm, the objects are sorted along the x-axis, the y-axis and the z-axis. These sorted axes are used to discard non colliding pairs. If the objects are moving slowly from one frame to another, then the sorted axes do not change between two time steps. In this case, sorts are performed in $\mathcal{O}(n)$ using an insertion sort for instance.
2) Mid-Phase: In order to avoid time-consuming colliding tests, objects are usually bounded by simpler bounding volumes. The idea is that deciding if two simple volumes will collide can be done with a cheap test. In general, the simpler the bounding volume is the cheaper the collision test is. However, if the bounding volume is too simple compared to the object then the bounding volume will be unnecessary large. In this case the bounding volumes might not discard a non-colliding object pair. Thus, there is a trade-off between the cost of the colliding test and how the bounding volume fits the object. Common bounding volumes are Axis-Aligned Bounding Boxes (AABBs), oriented bounding boxes and spheres.
3) Narrow-Phase: Colliding test between two objects can be done by different algorithms depending on the shape of the objects (convex or not) and if the objects can be preprocessed (rigid or not). The choice of the algorithm depends also on the envisioned application. When required by the application, some algorithm can also provide a penetration depth and contact points if the two objects are colliding.

A uniform grid algorithm was successfully applied for conflict detection in [15] but the objects in this publication are typically 5 NM large. However, a uniform grid does not suit our problem. In order to not miss any collision, the grid's cells must be larger than the largest object[2]. However, some objects in our problem are quite large making the cell quite large too[3]. This drastically reduces the filtering power of the uniform grid. In the end, the conflict detection algorithm presented in this paper does not use a broad-phase algorithm. All the object pairs are considered.

[2]More specifically, a missed collision is possible if, along one axis, the sum of the size of the two largest object is longer than two times the size of the cell.

[3]Some objects are 40 NM large and the objects are all in a 200 NM-side square.

A. Mid-Phase: Axis-Aligned Bounding Boxes

In our algorithm we chose Axis-Aligned Bounding Boxes (AABBs) as bounding volumes. Building an AABB of a volume can be efficiently computed, we only have to compute the minimum and maximum coordinates of the volume. As two position volumes must be separated by a norm, half the norm is added to each side of the AABB. Figures 3 and 4 illustrate how the final AABB is built. First we compute the AABB (in blue) of the position volume and then the AABB is increased by half the norm.

The collision test between AABBs is cheap as it requires six floating point comparisons at most.

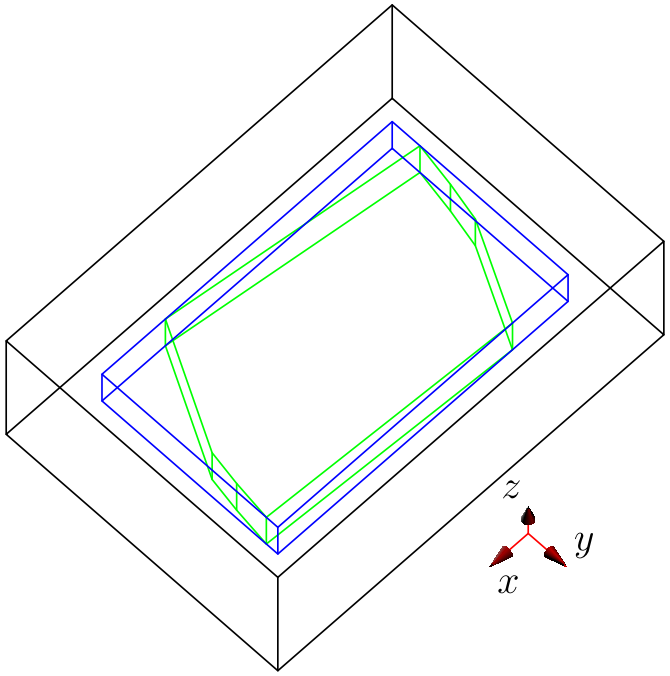

Fig. 3. The aircraft position volume is in green, its bounding box in blue, and the bounding box actually used in black.

B. Narrow-Phase

If the test between the AABBs has failed to discard a collision then we perform a fine test between the increased position volume. A position volume is defined by a convex polygon on the horizontal plane and two values on the z-axis, a z_{min} and a z_{max}. If the mid-phase failed to discard the conflict, then we only have to test if the convexes on the horizontal plane are separated by $NORM_{horizontal}$.

1) ISA-GJK: The ISA-GJK algorithm [21] can be used to decide if two arbitrary convex shapes intersects. It is a variant of the GJK algorithm [22] which is widely used in robotics and computational geometric to determine the distance between two convex shapes. This is one of the algorithm implemented on GPU in the physics library Bullet [23].

The GJK algorithm relies on the Minkowski sum. The Minkowski sum, denoted by the operator $\oplus$, of two sets of points A and B is defined by the following equation:

$$A \oplus B = \{a + b \mid a \in A,\ b \in B\}$$

Considering that $-B = \{-b \mid b \in B\}$, A and B intersect if and only if the origin point O is in $A \oplus (-B)$.[4] The goal of the ISA-GJK algorithm is to determine if the point O is in this sum. Actually, GJK uses *support functions* in order to avoid computing this sum.

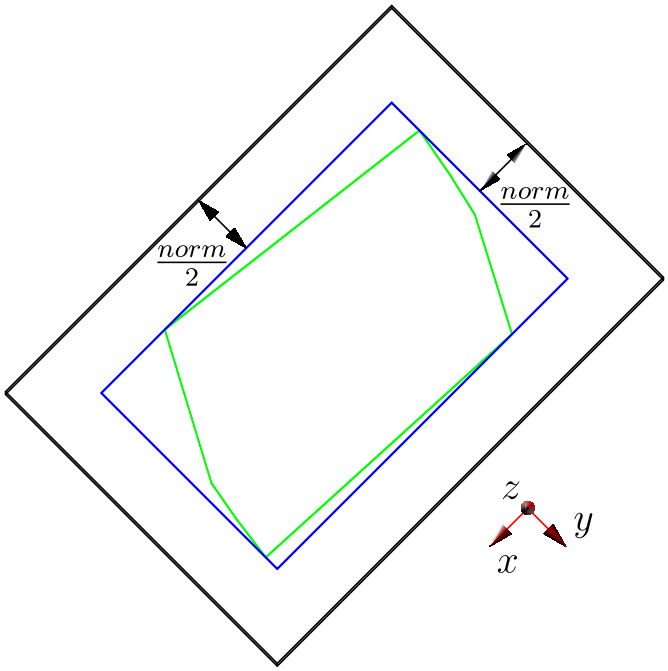

Fig. 4. The aircraft position volume is in green, its bounding box in blue, and the bounding box actually used in black.

In GJK, each shape is solely described by its *support function*. The input of a *support function* is a direction and its output is the farthest point in this direction. For any shape S, the farthest point in the direction $\overrightarrow{d}$ is simply $\underset{P \in S}{argmax}\ \overrightarrow{d} \cdot \overrightarrow{OP}$, where O is the origin. With this definition, one can notice that any shape has the same support function as its convex hull. Actually, with the support functions, computing explicitly the convex hull is not required although it might speed up the computation of the farthest point. If the considered shapes are not convex, the ISA-GJK algorithm will compute the intersection between the two shapes' convex hull. For any shapes A, B and any direction $\overrightarrow{d}$, we have:

$$support_{A \oplus B}(\overrightarrow{d}) = support_A(\overrightarrow{d}) + support_B(\overrightarrow{d})$$

$$support_{-B}(\overrightarrow{d}) = -support_B(-\overrightarrow{d})$$

$$support_{A \oplus (-B)}(\overrightarrow{d}) = support_A(\overrightarrow{d}) - support_B(-\overrightarrow{d})$$

With this last equation, we are able to compute the support function of the set $A \oplus (-B)$ by only using the support

[4] This happens if and only if $O \in \{a - b | a \in A,\ b \in B\}$.

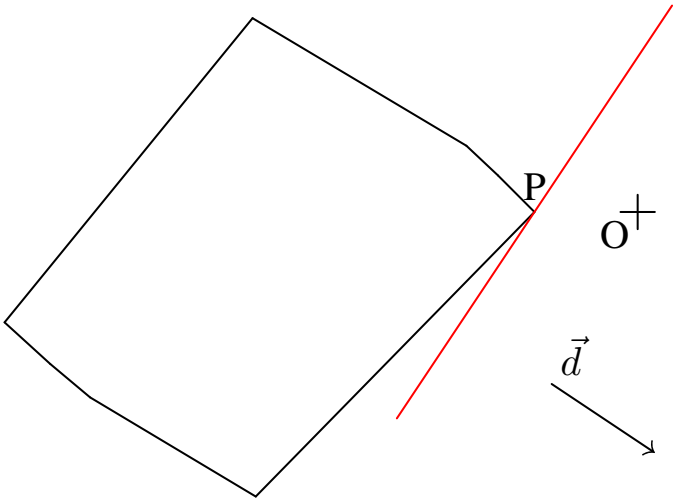

Fig. 5. The black shape is $A \oplus (-B)$ and the axis $(P, \vec{d})$ in red is a separating axis. In such a case, A and B are not colliding.

functions of A and B without computing the actual Minkowski sum. The next step consists in deciding if a shape described by its support function contains the origin O.

For the ISA-GJK algorithm, there are only two termination cases:

1) An axis separating the shape $A \oplus (-B)$ from the origin point O is found. In that case, the shapes A and B are not colliding. This case is illustrated by the Figure 5;
2) A simplex, a triangle in 2D, included in the shape $A \oplus (-B)$ contains the origin. In that case, the shapes A and B are colliding.

The idea of ISA-GJK is moving a simplex towards O. This simplex is moved by replacing one of its point. Then we check if the newly introduced point defines a separating axis or if the new simplex contains the origin O.

We have described ISA-GJK, an algorithm used to determine if two convexes intersects. However, we want to decide if two convexes are closer than $NORM_{horizontal}$. This can be easily done by using $A \oplus Circle(NORM_{horizontal}) \oplus (-B)$ instead of $A \oplus (-B)$ in ISA-GJK. The result of $A \oplus Circle(NORM_{horizontal})$ is illustrated by the Figure 6.

One advantage of ISA-GJK is its capability to work with non-convex shapes without explicitly computing convex hulls. Using ISA-GJK, we might skip the convex hull computation phase. This advantage was not used in our experiment because the shapes provided by the trajectory prediction module were convexes.

IV. Parallelization

In this paper we compare a CPU based implementation and a GPU based parallel implementation of a conflict detection algorithm. In these two implementations, the position volumes are bounded by bounding volumes, the AABBs described in subsection III-A. This helps to avoid a time consuming test as the one described in subsection III-B.

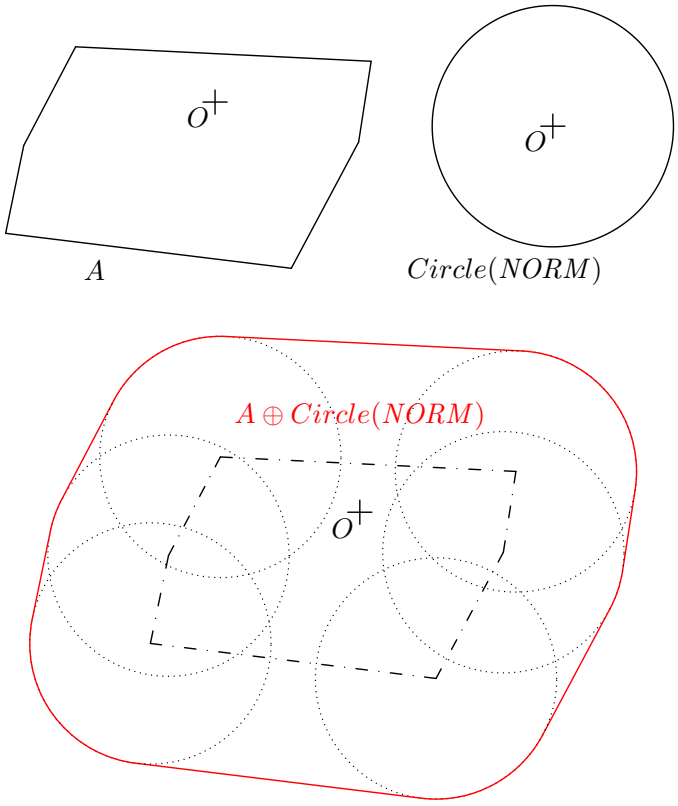

Fig. 6. The result of $A \oplus Circle(NORM_{horizontal})$ is the red shape.

A. CPU based implementation

We need to compute $\frac{n(n-1)}{2}$ conflict matrices. Each conflict matrix is computed using Algorithm 1. The method used to decide if two position volumes intersect is the ISA-GJK method described in subsection III-B1.

Algorithm 1 Computation on CPU of the conflict matrix between aircraft i and j

```
Initialize the matrix C with false (no conflicts)
for each alternative trajectory k for aircraft i do
    for each alternative trajectory l for aircraft j do
        Initialize the time step t at 0
        while t < T and not C_{k,l} do
            if AABB_{k,t} and AABB_{l,t} are not separated then
                if k(t) and l(t) are not separated then
                    C_{k,l} = true
                end if
            end if
            Increment t
        end while
    end for
end for
return Conflict matrix C
```

B. GPU based parallel implementation

The implementation was completed using the NVIDIA Compute Unified Device Architecture (CUDA) [24], a general-purpose parallel computing platform and programming model. This platform provides a "CUDA C/C++" compiler `nvcc` and tools to fully take advantage of the GPU.

In CUDA, a kernel is a function that is executed N times by N different CUDA threads. The threads might be executed in parallel depending how they are scheduled. Each thread is given a unique `ThreadId`. The programmer can access this value inside the kernel. Using this `ThreadId`, each CUDA thread can perform a specific computation on a specific data. The threads are grouped in blocks. Each block has a unique `BlockId`. Each thread has its own *local memory* and the block threads share a *shared memory*. All threads of all the blocks have access to a *global memory*.

Algorithm 2 is the algorithm dispatching the computation on the GPU. This algorithm transfers the predicted trajectories to the *global memory*. This data is not directly available, we need to wait the transfer to be completed, hence the line 2. At line 3, the kernel computes the AABBs The resulting AABBs are stored in the *global memory*. Then, a loop computes all the conflict matrices using one kernel call per conflict matrix. These matrices are first stored in the *global memory*. After all the computation are completed, the conflict matrices are transferred back to the host memory.

Algorithm 2 Dispatching the computation on the GPU

```
1: Transfer predicted trajectories to GPU device
2: Wait until end of all GPU tasks
3: Launch kernel computing AABBs
4: for i = 0 to n − 1 do
5:     for j = i + 1 to n − 1 do
6:         Launch kernel computing conflict matrix for aircraft
           i and j
7:     end for
8: end for
9: Wait until end of all GPU tasks
10: Transfer conflict matrices from GPU device to host
11: Wait until end of all GPU tasks
12: return Conflict matrices
```

The kernel used to compute the matrix is described in the Algorithm 3. It computes one conflict matrix. At line 6, the method used to decide if two position volumes intersect is the ISA-GJK method described in subsection III-B1.

A running thread uses registers, however thread registers are allocated from a global register pool on the GPU. The global register pool has a limited size. Using the NVIDIA Visual Profiler, a tool used to profile CUDA applications, it was found that the number of register used by this kernel was limiting the number of threads running simultaneously on the GPU. To solve this issue, we used the compilation option `--maxrregcount=32` which limits the number of registers used to 32 by thread. This limitation of the register number might slow down the thread execution but more threads are running simultaneously. In the end, this option has reduced the computation time by 25 %.

Algorithm 3 Kernel computing conflict between $k(t)$ and $l(t)$

```
1: k = "trajectory (blockId_x × blockDim_x + threadId_x)"
2: l = "trajectory (blockId_y × blockDim_y + threadId_y)"
3: t = blockId_z × blockDim_z + threadId_z
4: if AABB_{k,t} and AABB_{l,t} are not separated then
5:     if C_{k,l} is false then
6:         if k(t) and l(t) are not separated then
7:             C_{k,l} = true
8:         end if
9:     end if
10: end if
```

V. Experiments

In order to compare the GPU and CPU implementations, we consider different conflict detection scenarios. We also describe the hardware used for the experiments.

We consider 11 sizes of scenarios, involving 15, 20, 25, 30, 40, 50, 60, 70, 80, 90 and 100 aircraft, with three levels of uncertainties ($\varepsilon \in \{1, 2, 3\}$). For each combination, 10 scenarios of aircraft converging to the center of the considered airspace volume were randomly built. For each scenario, speeds are chosen from 384 kn to 576 kn (i.e. 20 % variation around a typical speed of 480 kn). The nominal vertical speed for maneuvers is set to 600 ft min^{-1}. The aircraft initial positions are chosen on a 100 NM radius circle and are noised within a 20 NM-side square. The initial heading is also noised with a value chosen in $[-1, 1]$ radians ($\approx \pm 60°$). In the vertical plane, aircraft are equally dispatched on 5 different levels from $FL280$ to $FL320$. A total of 11×10 scenarios are built. Figure 7 illustrates these dimensions.

For the each aircraft, $m = 161$ alternatives trajectories are considered. Each trajectory is a sequence of $T = 150$ position volumes with one position each $\tau = 10$ s. These trajectories are the result of a heading change angle or a Flight Level change. A comprehensive description of how these trajectories are built can be found in [1].

For each scenario, three levels of uncertainties are defined. The size of the convex hulls approximating the possible positions of aircraft increases with the uncertainty level, creating more conflicts for the same scenario. A total of $11 \times 3 \times 10 = 330$ scenarios were thus built and tested with two different approaches in the next section. These scenarios are available online at http://clusters.recherche.enac.fr. The available files might be useful for other researchers to compare their conflict detection method with ours.

The experiments were completed on a personal computer consisting of a quad-core Intel® i7-6700K equipped with 16 GB of memory and a Palit® GTX 1080 GameRock GPU. Both the GPU and the CPU are not overclocked.

VI. Results

In this section we have tested our two algorithms on the scenarios described in the previous section.

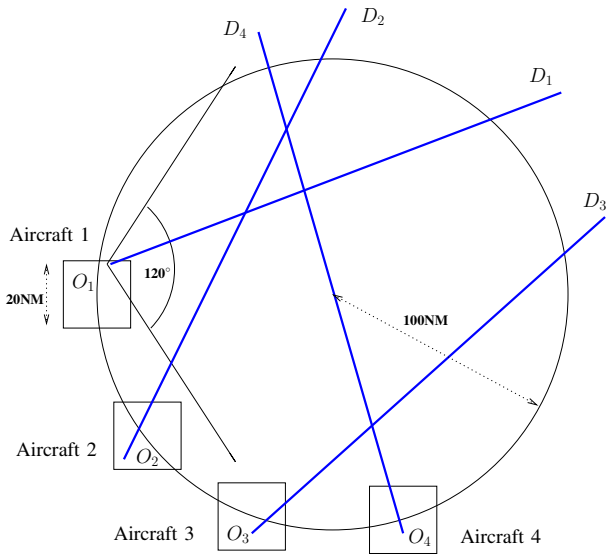

Fig. 7. Geometry of conflict scenario generation.

A. Results for the CPU version

The conflict computation time is plotted on the Figure 8. The computation time ranges from 3 s for 15 aircraft to 149 s for 100 aircraft. The increase is consistent with the fact that $\frac{n(n-1)}{2}m^2$ pairs of trajectories are compared. The computation time increases with the uncertainty level. When the uncertainty level increase, the position volumes increases and more pairs of trajectories are in conflict. As a consequence, more fine-grained collision tests are executed. For example, using a 100 aircraft problem, when the uncertainty levels are 1, 2, 3, the numbers of conflicting pairs are respectively 4.6×10^6, 7.7×10^6 and 8.8×10^6.

B. Results for the GPU parallel version

For this version, we have measured two durations on Algorithm 2: the time to transfer the trajectory data to the GPU (first line to line 2) and the time to actually compute the conflicts (line 3 to last line). The duration of the complete algorithm is the sum of these two durations.

The transfer time is plotted on Figure 9. The duration of the trajectory data transfer to GPU is almost linear with the amount of data to transfer which is proportional to nmT.

The conflict computation time is plotted on Figure 10. As with the CPU version, the time increases proportionaly to $\frac{n(n-1)}{2}m^2$. However, the 100 aircraft problem that requires the comparison of 128,308,950 pairs of trajectories is handled in 1090 ms. For the 15 aircraft problem, the 2,721,705 pairs are processed in 27 ms. On average, the trajectory pairs are processed at a rate of 117,000 pairs per millisecond.

If we consider the transfer time and the conflict computation time, the GPU implementation is two order of magnitude faster. It is 90 times faster for the 15 aircraft problem and 140 times faster for the 100 aircraft problem. This difference in speed-up between the small and large problem is mainly due to the transfer time.

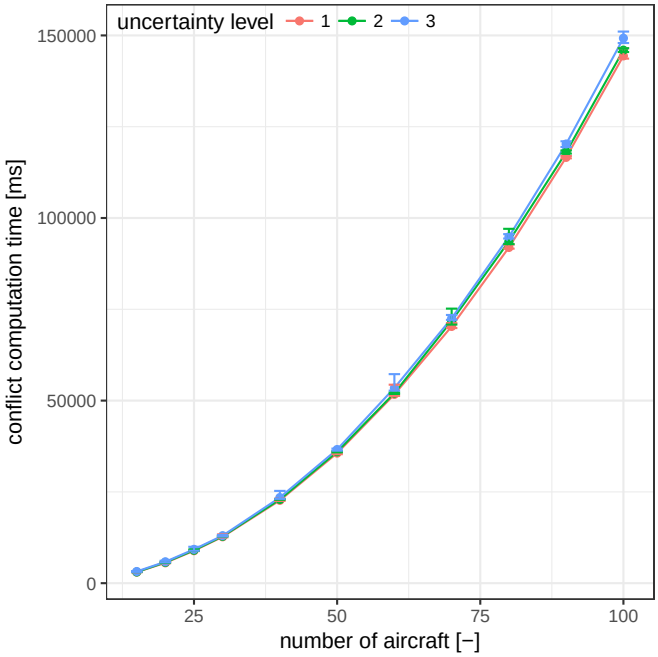

Fig. 8. Conflict computation time using the CPU implementation.

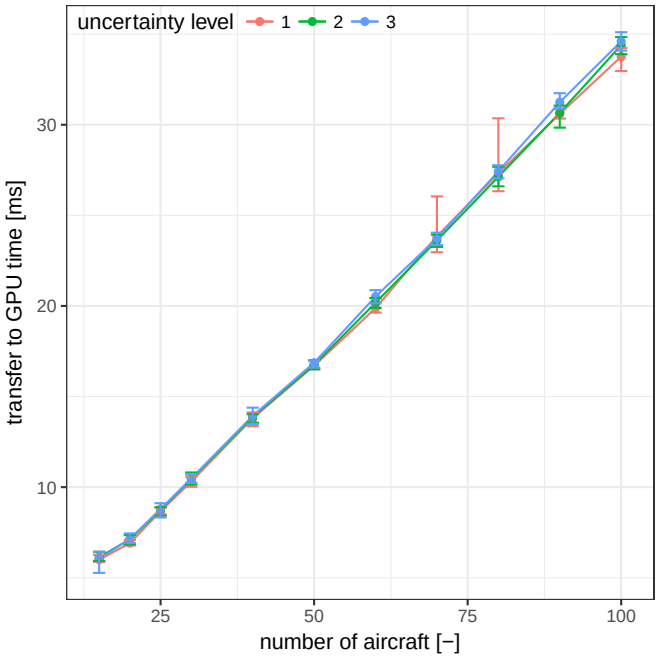

Fig. 9. Transfer time from host to the device, the GTX1080 GPU.

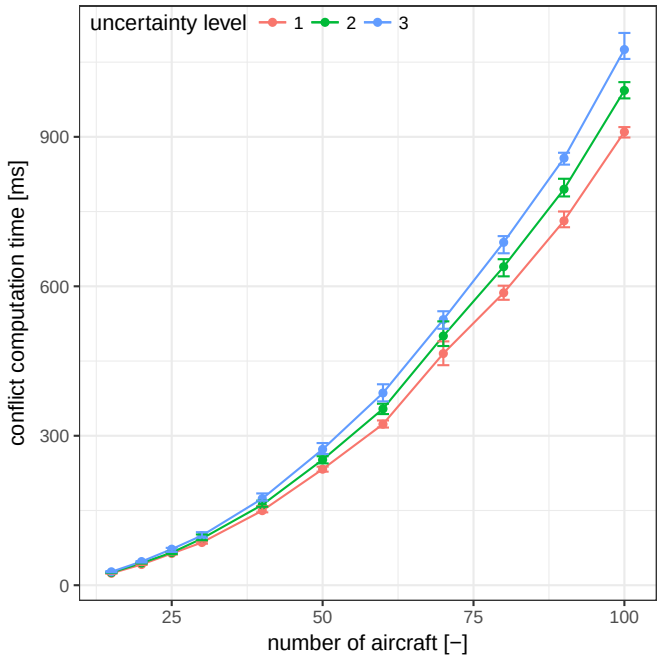

Fig. 10. Conflict computation time using the GPU implementation.

VII. Conclusion and Further Work

We have shown in this article that the time spent to detect conflicts could be reduced by two order of magnitude when we use a GPU parallel implementation. A previous paper ([1]) demonstrated that the optimization problem associated to conflict resolution can be solved in real time for medium size problems. This paper demonstrates that the detection phase required by the algorithms in [1] can also be done in real time. This might enable the use of conflict resolution tools in real time.

This algorithm could be very useful to improve different levels of Air Traffic Management. At a short term level (30 min or less), it can accelerate the necessary detection phase to find alternative trajectories for conflicting situations involving many aircraft. It could also be useful in order to make tools representing dynamically conflicting zones. Air Traffic Controllers will eventually rely on tools to test alternative trajectories and it seems reasonable that these tools instantly show the effect of a chosen option. A fast detection tool able to handle complex uncertainty models can be very useful.

The next step of our research will be to focus on the trajectory prediction time which can also be parallelized and reduced. We also plan to integrate our algorithm in a ATC assistant tool. This integration should help building a reliable efficient tool able to handle realistic uncertainty models and offer real time answers to the controllers.

References

[1] C. Allignol, N. Barnier, N. Durand, A. Gondran, and R. Wang, "Comparing conflict resolution algorithms on large-size opensource 3d-problems," in *12th USA/Europe Air Traffic Management Research and Developpment Seminar*, 2017.

[2] H. Erzberger, "Conflict probing and resolution in the presence of errors," in *Procceedings of the 1st USA/Europe Seminar*, 1997.

[3] W. C. Arthur and M. P. McLaughlin, "User request evaluation tool (uret). interfacility conflict probe performance assessment," in *Procceedings of the 2nd USA/Europe Seminar*, 1998.

[4] D. C. Meckiff and D. P. Gibbs, "PHARE : Highly interactive problem solver," tech. rep., Eurocontrol, 1994.

[5] A. Price and C. Meckiff, "Hips and its application to oceanic control," in *1st ATM R&D Seminar*, 1997.

[6] N. Durand, J.-M. Alliot, and J. Noailles, "Automatic aircraft conflict resolution using genetic algorithms," in *Proceedings of the Symposium on Applied Computing, Philadelphia*, ACM, 1996.

[7] G. Granger, N. Durand, and J. Alliot, "Optimal resolution of en-route conflicts," in *4th ATM R&D Seminar*, 2001.

[8] M. Tandale, S. Wiraatmadja, P. Menon, and J. Rios, "High-speed prediction of air traffic for real-time decision support," in *AIAA Guidance, Navigation, and Control Conference*, p. 6660, 2011.

[9] E. Thompson, N. Clem, D. A. Peter, J. Bryan, B. I. Peterson, and D. Holbrook, "Parallel cuda implementation of conflict detection for application to airspace deconfliction," *The Journal of Supercomputing*, vol. 71, no. 10, pp. 3787–3810, 2015.

[10] E. de la Iglesia, G. Botella, C. Garcia, and M. Prieto, "Parallel trajectory synchronization for aircraft conflicts resolution," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, (New York, NY, USA), pp. 1339–1341, ACM, 2015.

[11] D. R. Isaacson and H. Erzberger, "Design of a conflict detection algorithm for the center/tracon automation system," in *Digital Avionics Systems Conference, 1997. 16th DASC., AIAA/IEEE*, vol. 2, pp. 9–3, IEEE, 1997.

[12] B. Sridhar and G. B. Chatterji, "Computationally efficient conflict detection methods for air traffic management," in *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, vol. 2, pp. 1126–1130 vol.2, Jun 1997.

[13] F. Wieland, D. Carnes, and G. Schultz, "Using quad trees for parallelizing conflict detection in a sequential simulation," in *Proceedings of the Fifteenth Workshop on Parallel and Distributed Simulation*, PADS '01, (Washington, DC, USA), pp. 117–123, IEEE Computer Society, 2001.

[14] A. Kuenz and N. Peinecke, "Tiling the world—efficient 4d conflict detection for large scale scenarios," in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, pp. 3–B, IEEE, 2009.

[15] S. Ruiz, M. Piera, and C. Zúñiga, "Relational time-space data structure to speed up conflict detection under heavy traffic conditions," *SESAR Innovation Days (SID)*, 2011.

[16] M. Prandini, J. Lygeros, A. Nilim, and S. Sastry, "A probabilistic framework for aircraft conflict detection," in *Guidance, Navigation, and Control Conference and Exhibit*, p. 4144, 1999.

[17] M. Prandini, J. Hu, J. Lygeros, and S. Sastry, "A probabilistic approach to aircraft conflict detection," *IEEE Transactions on intelligent transportation systems*, vol. 1, no. 4, pp. 199–220, 2000.

[18] C. Ericson, *Real-time collision detection*. CRC Press, 2004.

[19] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys (CSUR)*, vol. 11, no. 4, pp. 397–409, 1979.

[20] D. Bara and A. Witkin, "Dynamic simulation of non-penetrating rigid bodies," *Computer Graphics (SIGGRAPH'92)*, pp. 303–308, 1992.

[21] G. V. d. Bergen, "A fast and robust gjk implementation for collision detection of convex objects," *Journal of graphics tools*, vol. 4, no. 2, pp. 7–25, 1999.

[22] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, pp. 193–203, Apr 1988.

[23] E. Coumans, "Bullet physics simulation," in *ACM SIGGRAPH 2015 Courses*, p. 7, ACM, 2015.

[24] Nvidia, "Compute unified device architecture programming guide," 2016.