Numerical Study of Ice Particle Shedding: Interpolation Methods and 2D Trajectories

Kevin Ignatowicz and François Morency
kevin.ignatowicz.1@ens.etsmtl.ca- 514 396-8800
Mechanical Engineering Department, École de Technologie Supérieure,
1100 Rue Notre-Dame Ouest, Montréal, Québec, H3C1K3, Canada

Abstract

Flying in icing conditions can be dangerous and de-icing systems can raise safety concerns: ice blocks shed can impact elements like the engines. Numerical prediction of these blocks trajectories still lack of data, mainly due to the random aspect of ice shapes and initial positions. The first step will be to determine the flow field around an aircraft geometry using the RANS approach and the Finite Volume Method with the open source SU2 CFD code. Using an interpolation method, we will next be able to compute the velocity at every spatial location inside the discretized computation domain. The aerodynamic behaviour of typical spherical and flat plate shaped ice blocks will be introduced into that velocity field to follow their trajectories along the airplane. In this paper, we will focus on interpolation and compare the effect of interpolation methods available on MatLab on the computation, in first attempt, of 2D trajectories. The aspects we will focus on will be the trajectory paths and the computational resources -memory and execution time- used for each interpolation method. The linear and natural methods are faster than the nearest one and compute similar trajectory paths, but with some differences. Future work will be to implement it in 3D and the use of a Monte-Carlo method to create the probability map will be the last step. It will lead us to find some critical zones with high chances of ice transit and thus help designers to find a safe location to position the aft-mounted engines among others.

1. Introduction

In the aeronautic domain, icing is one of the major concerns during airplane exploitation. Icing events contribute to 17% of the loss of control in-flight according to the IATA 2015 Safety Report[1]. Icing can happen on the ground and also during the flight. Several devices are installed on aircraft to protect it against icing and are separated in two categories: anti-icing systems to prevent ice accretion and de-icing systems to remove ice already appeared on vital components[2]. The second category of device has raised another concern that can be as important as ice accretion: it created shed ice particles that can become hazardous projectiles if they enter in a collision with critical aircraft's components such as the engines or the rear stabiliser[3].

To study this ice shedding and particle trajectories, numerical simulations have become preponderant due to the costs and risks engendered by real test flights. Nevertheless, both experimental and numerical approaches are used, especially for certifications requirements[3,4]. One part of that simulation work is to interpolate velocities in the flow field given by a CFD code to compute the trajectories at each time step. The objective of this paper is to compare the interpolation methods available regarding trajectory paths and computation time. The flow field data are commonly stored in a data sheet read by the interpolation function.

Our work, focused on 2D trajectories for the moment, will be carried on MatLab using *linear*, *nearest* and *natural* interpolation methods[7]. Are the methods computing the same trajectory patterns and are they similar for the use of computational resources, such as memory and execution time? The models and methodology used will be detailed in the first part of the present document, followed by the results and the analysis.

2. Case of study

2.1 Mathematical models

In section 1, the names of the main MatLab entities used in the function performing the interpolation were given: *scatteredInterpolant* and its associated interpolation methods *linear*, *nearest* and *natural.* Now let's describe the mathematical equations hidden behind these terms.
Amidror[6], made a survey on the interpolation functions and methods used by MatLab and details us the mathematical models governing them.

- *scatteredInterpolant*

  This function, which is the heart of the interpolation task in our case is based on the Delaunay Triangulation. This particular triangulation of a set of scattered points of a plane (in 2D case) is made so every point is on the circumcircles of all the triangles and not inside the disk. In other words, one circumcircle contains only one triangle. (Figure 1)

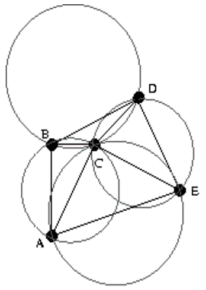

Figure 1: Example of a Delaunay Triangulation for a set of 5 scattered points. The circumcircles are also represented[8].

  The bold points of Figure 1 represent the nodes of the mesh where the velocity components are known. The purpose of the triangulation is to help to determine these components at any location on the plane. To visualize geometrically, for each point of the plane of coordinates (x,y), the value u is represented by a 3rd dimension as a bar chart, thus each point of the node has a "height" corresponding to the velocity's component value u, v or w depending on which is investigated. The syntax of scatteredInterpolant used in the case of our project is as described in the previous section:

```
scatteredInterpolant(P,U,method)
```

Three methods are available to perform the interpolation, and are described in the following list.

- *Linear* Interpolation Method

  As described just before, the set of data can be represented as special location (x,y) having a z altitude corresponding to the value, in our case, of one of the velocity components. (Figure 2)

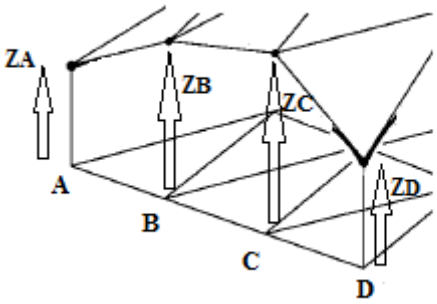

Figure 2: Representation of the data sets of points. The spatial locations A, B, C, D have their associated Z value. (Figure modified from Amidror)

The *linear* interpolation method is based on the equations of the planes containing each triangle whose vertex known coordinates are, for example, (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3). The equation of the plane containing such a triangle is given by solving:

$$\begin{aligned} z_1 &= ax_1 + by_1 + c \\ z_2 &= ax_2 + by_2 + c \\ z_3 &= ax_3 + by_3 + c \end{aligned} \qquad \textbf{(1)}$$

The interpolated value of the velocity z_q in a point (x_q, y_q) located within that triangle is then simply given by $\mathbf{z_q = ax_q + by_q + c}$.

The *linear* interpolation method thus solves the equations **(1)** before calculating the z_q value, which is the interpolated value at the query point of coordinates (x_q, y_q).

- *Nearest* Interpolation Method

This second method also uses the Delaunay Triangulation but interpolates with a different interpolant function. For each point of the data set, a quadratic polynomial expression is calculated which passes through the point interesting us and its 5 nearest neighbours:

$$z = ax^2+bxy+cy^2+dx+ey+f = F(x,y) \qquad \textbf{(2)}$$

Solving equation **(2)** for the 6 points give an explicit expression for each point.

At the interpolation stage itself, the triangle where the query point is located gives the three polynomial expressions of the three vertices of that triangle. Then, the interpolated value is calculated as a weighted average value of the three polynomials:

$$\mathbf{z_q = w_1F_1(x_q,y_q)+w_2F_2(x_q,y_q)+w_3F_3(x_q,y_q)} \qquad \textbf{(3)}$$

In the equation **(3),** the weight w_i are calculated so w_i equals 0 on the edge opposite to the i^{th} vertex and $d_i = 1$ on the i^{th} vertex. In practice it takes the following form, with k=3 :

$$w_i = \frac{d_i^k}{d_1^k + d_2^k + d_3^k} \qquad \textbf{(4)}$$

We can see that the pre-processing stage calculating the interpolant expression is heavier in that case, compared to the one in the *linear* method.

- *Natural* Interpolation Method

That method differs from the two previous ones by not directly using the Delaunay Triangulation of the scattered points set. In this case, the Voronï tessellation, dual representation of the Delaunay Triangulation made with the perpendicular bisectors of the edges of the triangles (Figure 3).

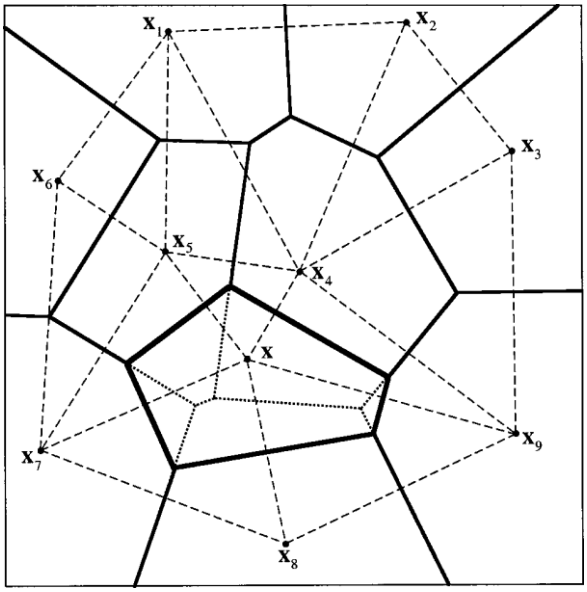

Figure 3: Voronoï tessellation (Amidror).

The Voronoï tessellation defines a set of polygons, each containing one point of the data set. The point A of the set is called *Natural Neighbour* of the point B if it shares a common polygon edge: for example in Figure 3, point x has as natural neighbours x_4, x_5, x_7, x_8 and x_9.

Interpolation consists of incorporating the query point and then creates its own polygon in the original tessellation. Then, the interpolated value is calculated as follows, being once again a weighted average value based on the gradients on the vertices, on the ratio of the area T of the polygons of the natural neighbours and the distance d between the query point and its natural neighbours. The sum is done on the natural neighbours of the query point:

$$f(x) = \sum_i w_i(x) g_i(x) = \sum_i \frac{h_i(x) d(x, x_i i)^{-1}}{\sum_i h_i i(x) d(x, x_i i)^{-1}} \boldsymbol{g}_i(x) \quad \textbf{(5)}$$

with

$$g_i(x) = z_i + \nabla z(x_i)^T (x - x_i)$$

$$h_i(x) = \frac{area[T_i(x)]}{area[T(x)]} \qquad 0 \le h_i(x) \le 1 \qquad \sum_i h_i(x) = 1$$

Even if the interpolant looks complex, this method computes faster than the nearest one.

2.2 Methodology

To carry out these numerical simulations, the methodology used is a typical CFD analysis beginning with a simulation done with the open-source SU2 code developed by Stanford University[5]. This simulation basically generates a data sheet giving us the flow field around our geometry and importantly for the present analysis, the 3 velocity components on each node of the mesh. Then, this data will be processed with MatLab, incorporating an interpolation function, in an existing in house trajectory code, solving the equation of motion **(6)** of a particle in the flow field. The airfoil geometry used to begin is the known NACA0012 airfoil (Figure 4), and trajectories will be studied around that airfoil at varying angle of attack.

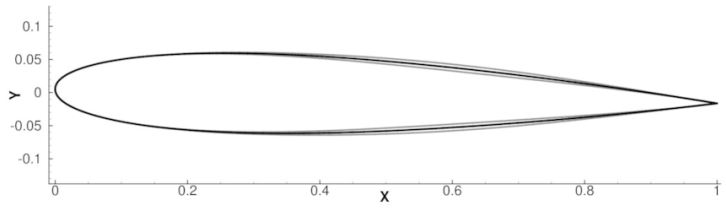

Figure 4: NACA0012 airfoil (X and Y are in meters).

$$m\frac{dU}{dt} = \frac{1}{2}\rho Vrel^2 S(Cd\ cos(\alpha) - Cl\sin(\alpha))$$
$$m\frac{dW}{dt} = \frac{1}{2}\rho Vrel^2 S(Cl\ cos(\alpha) + Cd\sin(\alpha)) - m\ g \quad \textbf{(6)}$$

In equation **(6)**, Vrel is the relative velocity and equals $((U_q\text{-}u)^2+(W_q\text{-}w)^2)^{0.5}$. U_q and W_q are the local velocity components of the fluid, which are precisely the value interpolated. Cl and Cd, lift and drag coefficients respectively, depend on correlations given by Holmes for the flat plate and Cd depend on the Reynolds number for the sphere[9] while Cl is 0.

The data sheet given by SU2 contains, among others in a row, the Conservative1 (ρ), the Conservative2 (ρu), the Conservative3 (ρv), the Conservative4 (ρw) and the three coordinate x, y and z of the corresponding node. To extract the velocity components, the *velocity* function will divide each Conservative 2, 3 and 4 by Conservative 1. This interpolation function, *velocity*, accomplishes the tasks illustrated on figure 5:

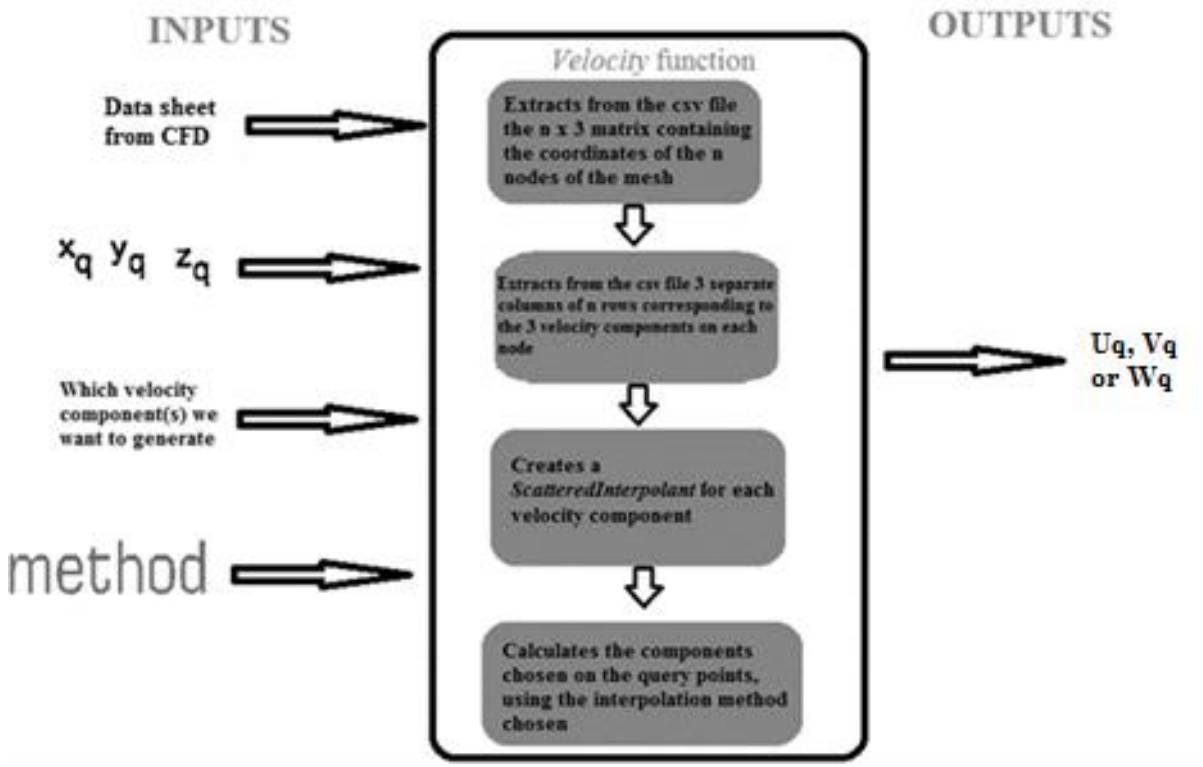

Figure 5: Illustration of the working of the Velocity function.

To sum up this function, it takes as inputs the data csv file directly generated by the CFD, the coordinates of the point where we want to compute the velocity, which component we want to generate – this input has been added to permit an easier incorporation of the result of that function in an equation-, and the interpolation we want to use. This function is based on the MatLab *scatteredInterpolant* function, since our mesh is an unstructured one, and the interpolation methods available with it are *Linear*, *Nearest* and *Natural*. The syntax use for this interpolant is:

```
scatteredInterpolant(P,U,method)
```

where P is the matrix containing on each row the 3 coordinates of the mesh nodes, and U the matrix listing the value of the first component of the velocity on each corresponding node. The same syntax is used for V and W, 2nd and 3rd velocity components. The mathematical models used by *ScatteredInterpolant* and the methods are detailed in the next paragraph.

This particular function will be grafted in a complete program computing trajectories of a sphere in a fluid in motion. This work done, the trajectories will be computed and the differences between the 3 interpolation methods analyzed. It will give us a clue regarding the fluctuations due to the method. For all of the previous calculations, memory needed and calculation time will be evaluated and compared. The trajectory patterns will be the next study, and the impact of velocities values fluctuations on the patterns will be investigated.

3. Results

3.1 Trajectory patterns

Simulations have been carried to compute trajectories of particles around a 2D NACA0012 airfoil. Confronting the results with the calculations of Holmes and Shimoï2 did the validation of the sphere and flat plate simulation.

The conditions of the simulation were the ones in the SU2 Quick Start Inviscid NACA0012 Test Case[6]: Mach 0.8, far field temperature 273.15K, far field pressure 101325.0Pa. In first attempts, a sphere of 4 centimetres diameter was studied around the airfoil, at angles of attack (AoA) of 1.25, 5, 8 and 13 degrees.

The mesh used was not modified from the Test Case and was composed of 10216 cells and 5233 nodes (Figure 6). Figure 7 and 8 present respectively the velocity field and the trajectories obtained for each interpolation method for AoA = 13°.

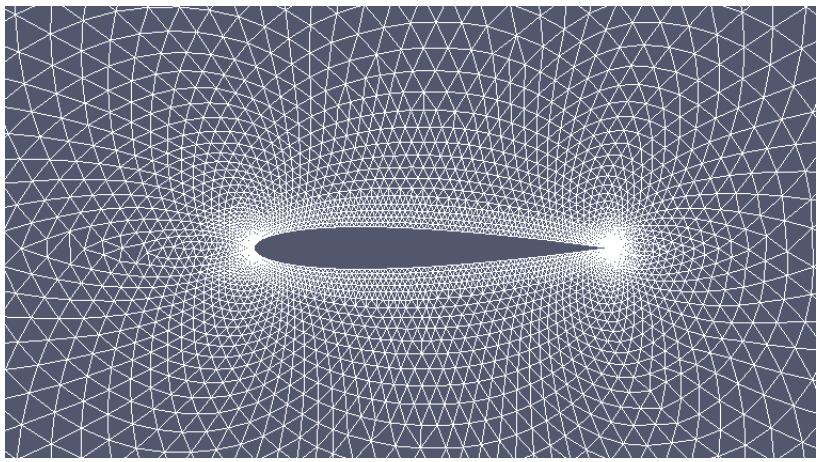

Figure 6: Mesh used for the simulations.

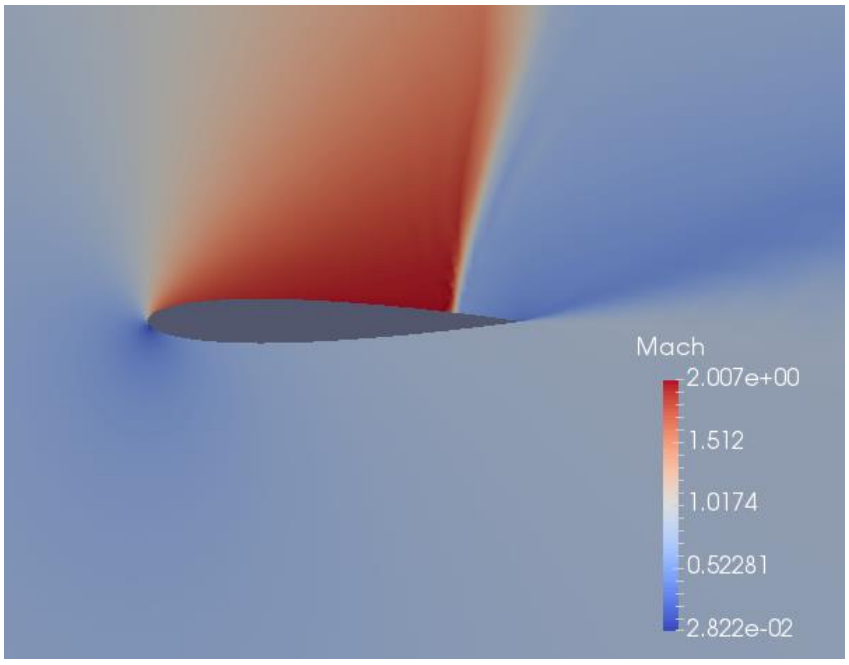

Figure 7: Mach distribution around the airfoil.

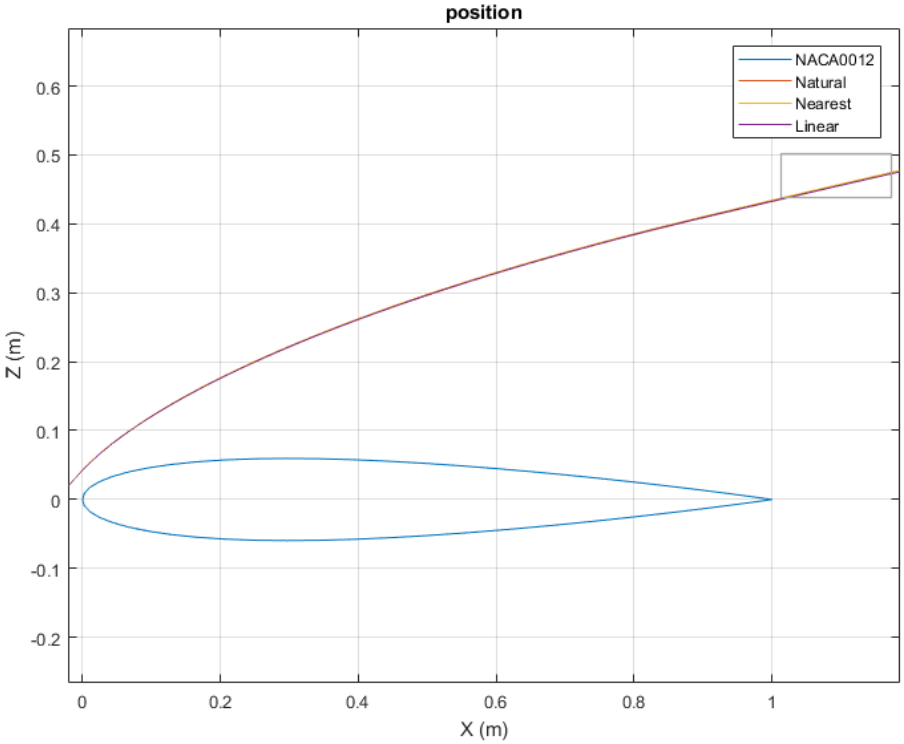

Figure 8: Trajectory patterns of a sphere around the NACA0012 at 13° AoA.

Figure 8 shows that the trajectories computed are globally the same and don't seem to depend on the interpolation method. Nevertheless, a close view on the rectangle drawn on the top right shows slights differences (Figure 9):

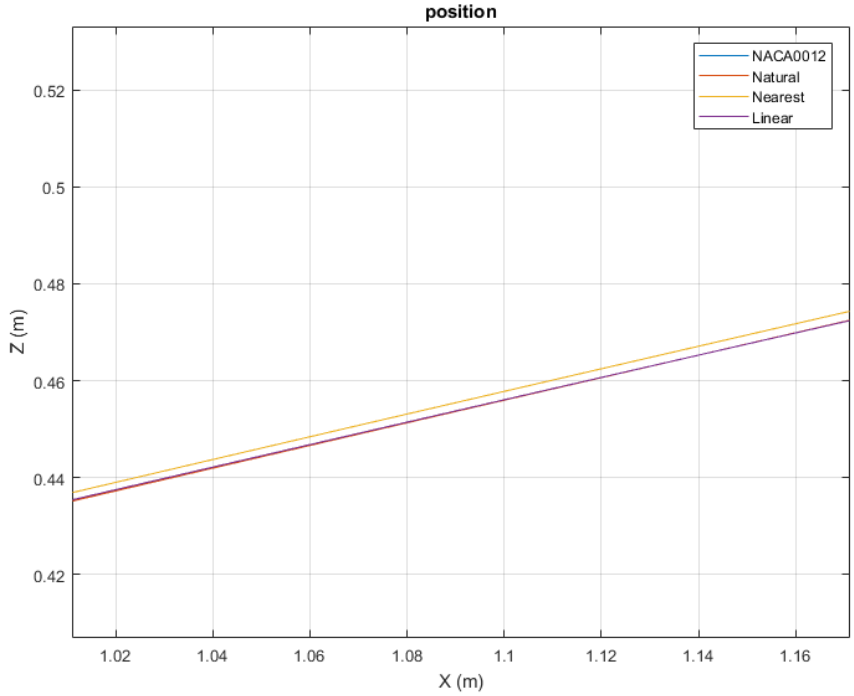

Figure 9: Close view on the trajectories.

By calculating the gap in the Z direction along X, it is obvious that the trajectory obtained with the *Nearest* method tend to diverge from the *Natural* one, which present almost no differences with the *Linear* method. Figure 10 represents the difference in the z coordinate for a given x between the *natural* method and the *linear* (blue curve) and between the *nearest* method and the *linear* (brown curve). X is the longitudinal axis in the direction of the cord and z ascendant vertical axis.

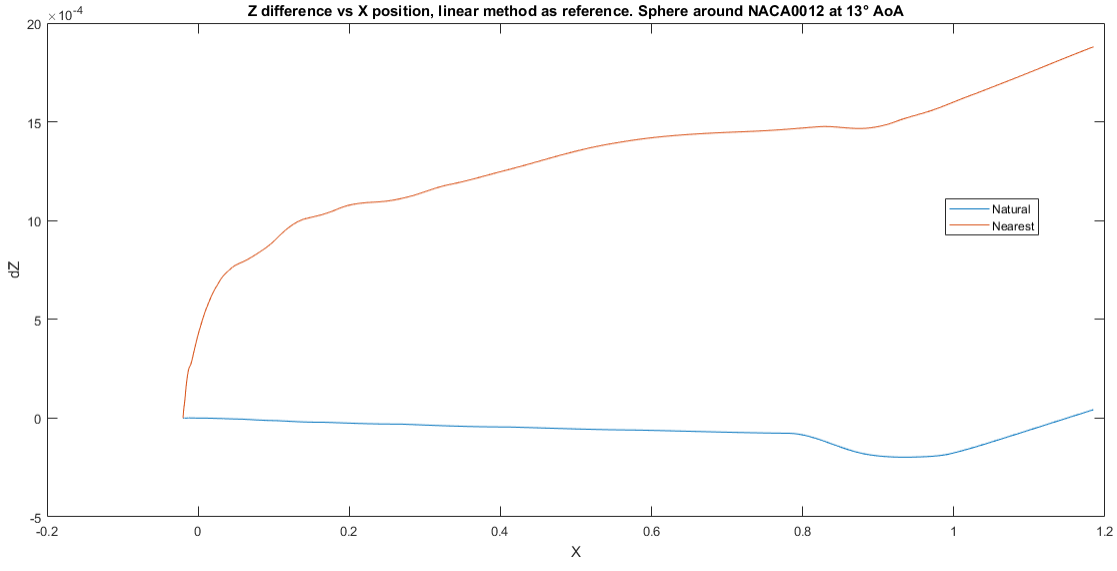

Figure 10: Gap in Z direction between interpolations methods with Linear as reference

Natural and *Linear* methods give, with a precision of 0.0001m, the same trajectory pattern while the *Nearest* one has some differences, about 2 mm 1.2 m after the leading edge.

In the flat plate simulation, once again the global patterns are the same and differences are observed while getting a close view on some location of the path (Figure 11). The flat plate used to carry out the simulation was a square of 0.152m edge length, and 0.001m thickness.

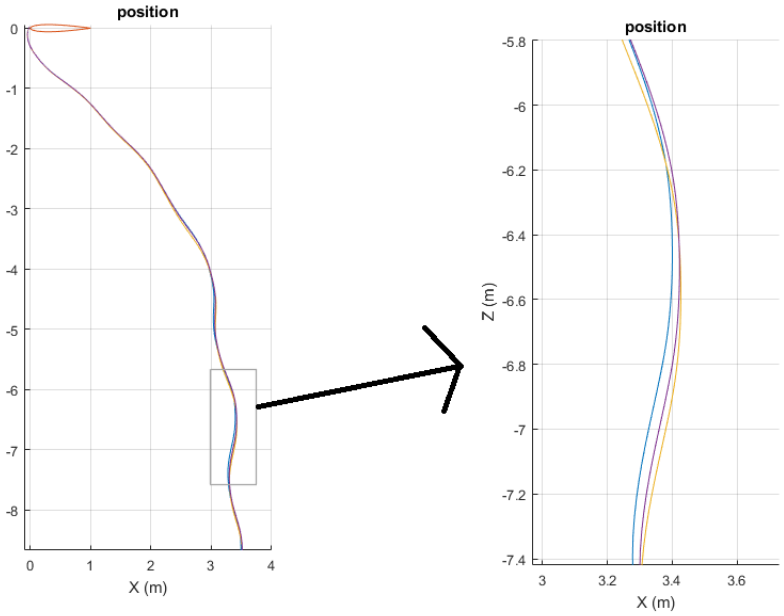

Figure 11: Global path and close view on the trajectory of a flat plate around the NACA0012.

To explain the differences, the 3 interpolation methods used here don't have the same interpolant expression. For a location, computed velocities are not exactly the same and lead to different values of calculated forces on the particles that finally lead to a different trajectory pattern once the equations of motion are solved. Table 1 shows an example of computed velocities at the same point for each of the interpolation method. This specific point was chosen because it corresponds to the beginning of a separation between the trajectory paths of the *linear* and *nearest* methods in the case of the flat plate.

Table 1: Computed axial velocity at the point (2.4;-3) in the flowfield using axis of Figure 11..

Method	U (m/s)
Linear	254,599
Nearest	255,353
Natural	254,591

Linear and *Natural* methods give very close velocity values and it explains why both of these methods give similar trajectory paths compare to the *Nearest* one.

3.2 Calculation time

One of the aims of this paper is to compare the calculation time of the interpolation methods available with scatteredInterpolant.

The results are presented in the Table 2. Absolute calculation time obviously depends on the computer used to carry out the simulations, so the times have been scaled, taking as a reference the calculation time of the linear method at 1.25° AoA. Once again, as well as for the trajectory paths, *Linear* and *Natural* methods present similarities while *Nearest* is 60% slower.

Table 2: Comparison of calculation times.

Method	CPU time (1,25°AoA)	CPU time (5° AoA)	CPU time (8° AoA)	CPU time (13°AoA)
Linear	1 (reference)	0,98	0,97	1,03
Nearest	1,66	1,57	1,51	1,82
Natural	1,01	1	0,96	0,96

Except for the *Nearest* method, calculation time do not depend on the flow field as it remains globally constant for a same method at various AoA.

4. Conclusion

The preliminary results presented here tend to show that among the three interpolation methods available with scatteredInterpolant, the *linear* and *natural* ones have the same behaviour in terms of computes trajectory patterns and calculation time. The *nearest* one present the same trajectory paths but has some slight differences compared to the trajectory (which is the same at 0.0001m) obtained for the other methods. It is also 60 to 80% slower to compute the trajectory in the same simulation case. That can be explained by the pre-processing stage prior to the interpolation itself, that has to compute 6 quadratic polynomial expressions, which is more time consuming that the linear equations of the *linear* method, for example.

Acknowledgements

Thanks to Le Décanat Des Études of ÉTS for the funding of the travel from Montréal to Toronto to attend to the conference and thanks to the CASI committee for the organisation of the CASI AERO 2017.

References

1. *Safety Report 2015*. 2016, International Air Transport Association: Montreal, Quebec.
2. Védie, L., F. Morency, and N. Kubler, *Sensitivity analysis of ice piece trajectory calculation*, in *8th AIAA Atmospheric and Space Environments Conference*. 2016, American Institute of Aeronautics and Astronautics.
3. Michael Papadakis, H.-W.Y.a.K.S., *Parametric Investigation of Ice Shedding from a Business Jet Aircraft*, in *SAE Aircraft & Engine Icing International Conference*. 2007, SAE International: Seville, Espagne. p. 17.
4. Widhalm, M., *Lagrangian Trajectory Simulation of Rotating Regular Shaped Ice Particles.* SAE Technical Paper 2015: p. 14.
5. Amidror, I., *Scattered data interpolation methods for electronic imaging systems: a survey.* Journal of Electronic Imaging., 2002. **11**(2): p. 20.
6. Palacios, F., T.D. Economon, and J.J. Alonso, *Large-scale aircraft design using SU2*, in *53rd AIAA Aerospace Sciences Meeting*. 2015, American Institute of Aeronautics and Astronautics.
7. Mathworks. (2017). *Mathematics Toolbox: User's Guide* (r2017a).
8. Computing constrained Delaunay Triangulations retrieved at http://www.geom.uiuc.edu/~samuelp/del_project.html
9. Clift, R., Grace, J.R., and Weber M.E., Bubbles, Drops and Particles, 1978