Development of an Aircraft Aero-Icing Suite Using Chapel Programming Language

Hélène Papillon Laroche
Simon Bourgault-Côté
Matthieu Parenteau
Eric Laurendeau
helene.papillon-laroche@polymtl.ca
simon.bourgault-cote@polymtl.ca
matthieu.parenteau@polymtl.ca
eric.laurendeau@polymtl.ca
Polytechnique Montréal
Montréal, Québec, Canada

ABSTRACT

This paper presents an aircraft ice accretion simulation suite implemented in the Chapel programming language for deterministic and stochastic ice accretion in two (2D) and three (3D) dimensions. The work is performed inside the CHApel Multi-Physics Simulation software (CHAMPS) developed at Polytechnique Montreal since 2019. Different physical models are added to the flow solver to simulate the droplet trajectories, the surface thermodynamic exchanges, and the surface deformation. The object-oriented approach used in the development of CHAMPS, combined with the generic functions and types from Chapel, allowed the development of a code that is easy to maintain and that still has high growth potential. The latest extension to CHAMPS is the capability to perform stochastic ice accretion using an advancing front grid methodology at the core and by randomly distributing the droplets, like in a cloud. Although stochastic ice accretion is not new, this paper presents an original methodology that has advantages over other methods from the literature, such as conserving a valid surface mesh from the beginning to the end of the stochastic accretion. Multi-layer ice accretion results are presented in 2D and 3D for a deterministic methodology, whereas single-layer 2D results are presented for the stochastic method.

CCS CONCEPTS

• **Applied computing** → **Aerospace**.

KEYWORDS

Computational Fluid Dynamics, Aircraft Icing, Icing Prediction, Multi-Physics Simulation Software, Chapel Programming Language

CHIUW 2021, June 04, 2021, Online

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

ACM Reference Format:
Hélène Papillon Laroche, Simon Bourgault-Côté, Matthieu Parenteau, and Eric Laurendeau. 2021. Development of an Aircraft Aero-Icing Suite Using Chapel Programming Language. In *Chapel Implementers and Users Workshop 2021, June 04, 2021, Virtual format*. 9 pages.

1 INTRODUCTION

Aircraft icing involves complex multi-physical phenomena and is a field of great interest in terms of in-flight safety. The modeling of these phenomena using numerical simulations allows a better understanding of the degradation of the aerodynamic performances due to ice accretion on exposed surfaces. These numerical simulations are even more important in the certification of an aircraft since it is unlikely, if not impossible, to evaluate all the icing conditions foreseen by the Appendix C of FAR part 25 [9] using in-flight tests.

Ice accretion happens as an aircraft passes through a cloud of supercooled droplets which leads to thermodynamic exchanges. Therefore, the numerical simulations are complex since the physical phenomenon involves the airflow, the impingement of the droplets, the thermodynamic exchanges, and the ice growth. Traditionally, multi-physics Computational Fluid Dynamics (CFD) software use Fortran, C and C++ languages coupled with Single Program Multiple Data (SPMD) framework to achieve high performance [18] such as IGLOO3D [20], LEWICE3D [29], NSCODE-ICE [7] and FENSAPE-ICE [3]. The challenge is to balance the computational and the implementation costs while obtaining the required fidelity, as the concept of SPMD can represent additional difficulties in an academic research laboratory. Parenteau et al. [18] presented a new CFD software named Chapel Multi-Physics Simulation (CHAMPS), which uses the Chapel programming language as an alternative to more traditional programming languages. This paper presents an extension of CHAMPS for ice accretion predictions over aircraft, highlighting the modularity and the highly productive framework provided by the Chapel programming language.

Section 2 presents an overview of a typical multi-layer icing process to introduce the context to an unfamiliar reader. Section 3 continues with a brief overview of the implementation of CHAMPS and is followed in Section 4 by the additional considerations for the data structure brought by the multi-physics simulations. Then,

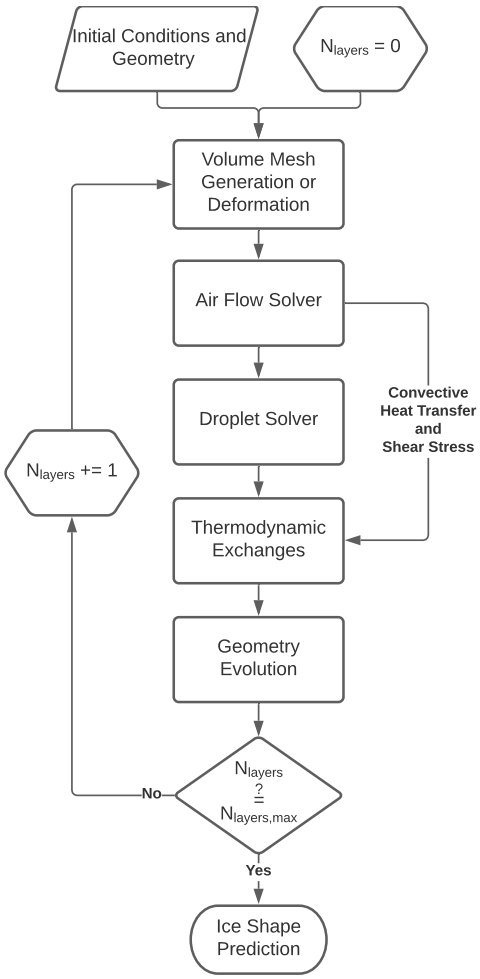


Figure 1: Multi-layer Icing Process, adapted from [7, 11]

a stochastic icing approach within CHAMPS is presented along with how Chapel's features are used. Finally, CHAMPS is compared to experimental data to access its capability to simulate real icing conditions.

2 OVERVIEW OF THE MULTI-LAYER ICING PROCESS

The multi-layer icing process is declined in five main components which are performed sequentially and repeated until the icing simulation is completed. The complete icing process is well described by Bourgault-Côté [7]. The flow chart of the process is presented at Figure 1.

2.1 Continuous Phase : Air Flow Solver

The aerodynamic flow field is involved in the icing process since it drives the droplet impingement, the convective heat transfer, and the water runback flow on the geometry [7]. The state-of-the-art is to use the Reynolds-averaged Navier–Stokes (RANS) equations to solve the continuous phase, as it allows to model compressible effects as well as viscous and turbulent flow [7, 11].

This aerodynamic framework requires the discretization of the computational domain into a high-quality volume mesh, in order to well resolve the flow field. If the aerodynamic degradation due to ice contamination is sought, as in a multi-layer approach, re-meshing capacities are needed.

2.2 Disperse Phase : Droplet Solver

The disperse phase consists of the liquid water droplets transported by the airflow. Ice accretion can occur when these droplets impinge a surface, that is any exposed surfaces of the aircraft in the present context (wings, fuselage, nacelle). Therefore, the impingement map is needed on the surface. The droplet trajectories can be modeled using Lagrangian or Eulerian methods, the first being the resolution of the equation of motion for each particle and the second being the resolution of a partial differential equation system, as proposed by Bourgault et al. [6].

The latter approach is now the state-of-the-art as the mesh used by the RANS flow solver is reused for the droplet field resolution, as well as the parallelization of the computations [7].

2.3 Thermodynamic Exchanges

The impingement of the water droplets on the surface of the airfoil and the convective heat transfer induced by the flow result in thermodynamic exchanges. The resolution of these exchanges on the surface, with mass and energy balances, allows obtaining the accumulated ice thickness. Unlike the volume resolution of the continuous and disperse phases, the thermodynamic model uses the surface discretization of the geometry. The mass and energy balances are described by Messinger [15], and the resolution of the thermodynamic exchanges can be completed using the Messinger model, extensions of this model [16, 17, 30] or PDE systems as the Shallow-Water Icing model [5, 11].

2.4 Geometry Evolution

The geometry is deformed by considering the ice accretion map on the surface to obtain the final ice shape. One of the methods used to deform the surface grid is the algebraic method, or Langrangian method, that moves each node of the surface discretization following the ice accumulation map. Other methods include implicit methods, such as the level-set equation, and grid approaches such as the hyperbolic grid generation method or advancing front techniques [7]. Note that the word geometry is employed here even if only the surface grid is deformed, as this represents a discrete geometry. An underlying continuous geometry can be re-generated or deformed based on that surface grid deformation.

2.5 Mesh Regeneration

Once a new geometry definition, discrete or continuous, is obtained, a corresponding volume mesh can be obtained by deformation of the previous grid or by full or partial regeneration. This allows performing a multi-layer simulation to gradually accumulate the ice by smaller layers and to include the impact of that ice in the next flow and droplet trajectory simulations. When regenerating

the grids, the computational domain changes at each layer, which increases the complexity.

3 CHAPEL MULTI-PHYSICS SIMULATION (CHAMPS)

The development of the flow solver of CHAMPS, as well as its performances, are well described by Parenteau et al. [18]. Since the scope of this paper is toward the icing simulations, only a brief overview of the implementation of CHAMPS is presented.

The flow and the droplet fields are solved with a typical cell-centered finite volume approach over 2D and 3D unstructured multi-zone grids. The aerodynamic field is obtained by resolving the RANS equations, which are closed with the Spalart-Allmaras [23, 24] or the K - ω SST-V [14] turbulence models. Roe [22] or AUSM [4] schemes are used for the fluxes discretization and the second-order spatial accuracy is reached with the computation of the gradients of the flow variables, with the Green-Gauss or Weighted Least Square formulations [4]. Gradient limiters available to reach second-order accuracy include the formulation of Barth and Jespersen [2], Venkatakrishnan [27], Van Leer and Van Albada [4]. Time integration is completed with a hybrid Runge-Kutta scheme [13], a Block Symmetric Gauss-Seidel scheme, or a GMRES method.

The droplet impingement field is obtained via the Eulerian droplet equations [6], using an upwind scheme for the convective fluxes discretization. The spatial second-order accuracy is reached in a similar way as for the flow equations. The gradient, limiters and linear solvers usable for the flow solver are also available to the droplet solver. In fact, in CHAMPS, solvers and gradients are generalized so that any method developed for a module is immediately available for the other modules, thus highlighting the reusability and modularity of the original structure [18].

The thermodynamic exchanges are solved with an Iterative Messinger model [30], implemented following the new model structure presented at Section 4.3. The geometry evolution is performed with the Lagrangian method, presented at Section 2.4, or an hyperbolic scheme [7].

To achieve multi-layer icing simulations, two techniques are available for now: i) complete 2D grid regeneration through a 2D hyperbolic mesh generation method [8], or ii) 2D/3D volume mesh deformation through a radial basis function approach [28].

4 DATA STRUCTURE FOR MULTI-PHYSICS SIMULATIONS

Compared with specialized software for flow simulations, the multi-physics simulations performed with CHAMPS require a more complex computational structure.

4.1 Computational domains

CHAMPS is a multi-zone software, meaning that the computational domain, i.e. the mesh, is divided into multiple zones to reduce the computational time on multi-core systems. As described by Parenteau et al. [18], parallelism on distributed memory is applied to these zones, and exchanges are performed at the interfaces between the zones to obtain a consistent solution. Therefore, a suitable structure has initially been implemented for the flow solver in CHAMPS, taking the requirements of the icing process into account.

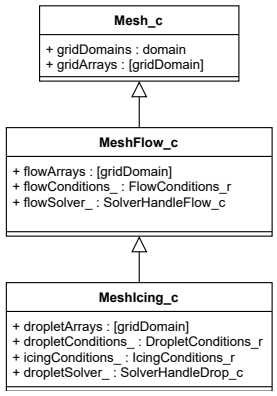


Figure 2: Overview of the *MeshIcing_c* class in CHAMPS

Indeed, the generic class *GlobalHandle_c* handles the distribution of any type of zone over the computational resources [18]. The type field *zoneType_t* is therefore used to define the type of simulation to perform, e.g. the resolution of the flow or an icing simulation. Exploiting the common characteristics of the grids used, as the node coordinates, the grid metrics, and the connectivity, inheritance of a base class, *Mesh_c*, is used to ease the multi-physics developments. There are two examples of this handling within CHAMPS that worth mentioning: i) the *MeshIcing_c* class and ii) the volume and surface meshes.

MeshIcing_c class. This class makes use of the fact that the flow and the droplet impingement are solved on the same computational grid. Therefore, the *MeshIcing_c* class inherits of the *MeshFlow_c* class, and the required fields for the resolution for the droplet impingement are added, as the associated arrays and the icing conditions. This structure, illustrated in Figure 2, shows the reusability of the implemented structure in CHAMPS and allows to access easily the flow variables while solving the droplet field.

Volume and surface meshes. The icing process involves both a volume mesh and a surface mesh. The difference between those two components is mainly the topological number of dimensions. Therefore, the specialized computational grids (volume or surface) inherit from the *Mesh_c* class and the specialized procedures based on the topological number of dimensions without further addition. In fact, once the computational grid is initialized, the different solvers such as the flow solver or the droplet solver are developed to compute on any grid that is loaded. This allows to solve a volume equation, such as the RANS equations, on a planar grid, which is topologically 2D, and on its extruded 3D equivalent with the same code and the same functions to obtain exactly the same solution. The possibility to develop for a 3D volume and test on a 2D plane, or the opposite, is provided by object-oriented programming and by the generic features of Chapel and is quite useful in a research laboratory.

4.2 Distributed Memory Parallelism

The use of both volume and surface meshes for the same simulations brings challenges when distributing the memory to conserve efficiency and scalability. As presented by [18], the distribution of

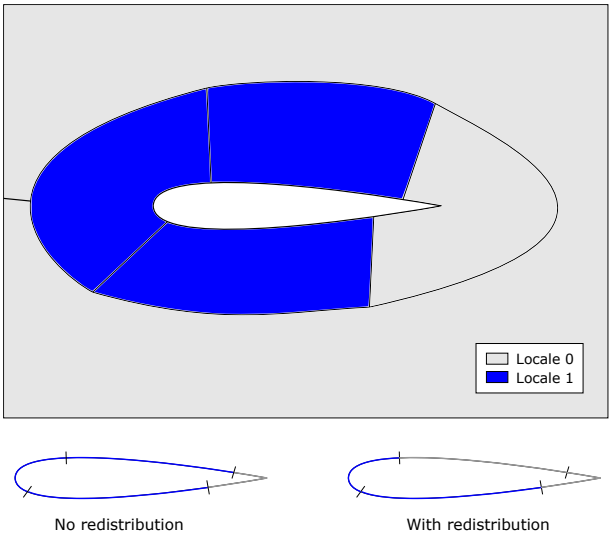


Figure 3: Volume (top) and surface (bottom) zones distribution across the computational nodes

the zones across the computational nodes is handled by the *GlobalHandle_c* class to best distribute the volume grid. The additional considerations brought by the use of the surface come from the fact that the surface mesh associated with the volume mesh on a locale does not necessarily contain the same number of zones as its volume counter part. An example of this situation is illustrated in Figure 3, where the computational domain is divided in six volume zones. Three volume zones are distributed on each locale (0 and 1), however, with no redistribution, the local 0 contains only one surface, while the locale 1 contains three surfaces.

Indeed, since the distribution of the zones across the computational nodes is based on the volume mesh, the parallelism can be unbalanced with under-utilized resources, when the solver reaches the thermodynamic and geometric modules.

To overcome this issue and the associated performance losses, the zones of the surface mesh are redistributed across the computational nodes by first deleting the volume *GlobalHandle_c* object, and all the volume computational domain, and reading again the grid file, but for the surface that time. A new *GlobalHandle_c* object is then created and the surface zones are automatically distributed with a Block distribution across all the Locales. This allows to minimize the under-usage of the resources by weakly ensuring that all the Locales contain a part of the surface zones. It is considered highly improbable that a volume grid distributed in N Locales containing M cores would contain less than N surface zones, thus the weak condition.

Another way to overcome this issue is to split the mesh in a way that the volume and surface partitioning are balanced. Coincidentally, this is achieved when completely regenerating a 2D planar grid with the hyperbolic mesh generation approach due to the numbering used that is appropriate for the grid splitting algorithm.

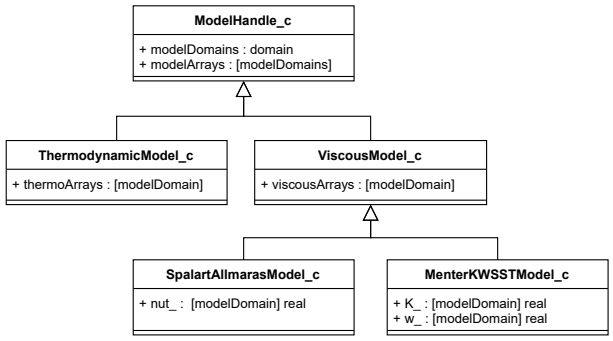


Figure 4: Overview of the classes for models in CHAMPS

4.3 Addition of Models

With the multiplication of the modeled physics in CHAMPS, a new structure was introduced: the *ModelHandle_c* class. First used for the turbulence models, this structure was then used in the thermodynamic module as well as in the geometry evolution module.

The rationale behind the *ModelHandle_c* class is that some physics can be resolved with multiple models involving different variables and numerical schemes. Therefore, it is more convenient to wrap those models in a class and, following the user inputs, instantiate the right model as a field of the generic *Mesh_c* class (or its children).

The *ModelHandle_c* class is the base class, as the *Mesh_c* class for the computational grids, and specialized models inherit from this class. The fields in *ModelHandle_c* represent the simplest definition of a model, as the number of variables, the number of elements, and the associated domain and arrays. The fields of the derived class depend on the model implemented. For example, a turbulence model is needed when resolving the flow. In CHAMPS, few models were implemented: the Spalart-Almaras (SA), K-ω STT, Langtry-Menter transitional, and variations of these models. The structure for those classes is presented in Figure 4.

Despite the modularity brought by this structure, compilation difficulties were encountered throughout the implementation of the new models. One of the main concerns during the growth of CHAMPS to multi-physic simulations was to keep the generic character of the structure. However, due to the use of object-oriented programming with generic objects for the zones and the models, the definition of the methods of the children objects often require a tight control with `where` statements to avoid compilation errors due to non-existing fields or methods in parents or in siblings (other children from same parents). In Chapel, `where` statements allow to define conditions tested at compilation to implement a particular function or method. For example, a function could be defined with a `where` statement on the type of a generic argument to be instantiated only when the argument is an integer. Two examples from CHAMPS of `where` statements can be seen below, the first being more complex as we are forced to use the == operators.

```
1 where globalHandle.type == GlobalHandleFlow_t ||
      globalHandle.type == BorrowedGlobalHandleFlow_t ||
      globalHandle.type == GlobalHandleIcing_t ||
      globalHandle.type == BorrowedGlobalHandleIcing_t
2 where zone.type < MeshFlow_c
```

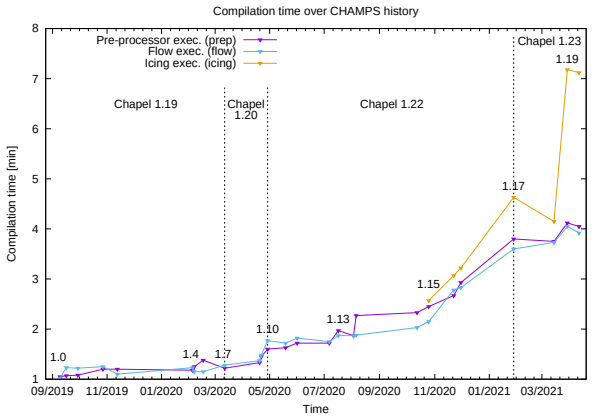


Figure 5: Compilation time of CHAMPS exec along version history

4.4 Compilation time

One of the main drawbacks of the growth of CHAMPS with new components or modules is the increase in compilation time and required memory, especially for the icing executable. Due to the hard limit in memory usage per process on the front nodes of the Compute Canada HPCs, the compilation process was split in two phases, namely i) the generation of the C code from the Chapel files and ii) the compilation of the C code. During the history of CHAMPS, work has been performed at different moments to reduce the resources required to compile CHAMPS in terms of time and memory. At its highest point, compilation time for the icing executable could take around 15 minutes, whereas memory usage was seen to reach up to 30GB of memory (RAM). The latter was mainly due to i) the overuse of generic function arguments in some modules, where even integers and reals were left without types in function declarations, and ii) the `use` of too many modules everywhere even when they were not required. Solving these two issues, which were observed for all executables when new modules were added by students unfamiliar with the Chapel language, also helped in reducing the compilation time to acceptable levels in general.

However, for the icing executable into which almost all modules are involved, more work was required. In particular, it was necessary to properly use `where` statements to limit the duplication of generic functions for multiple flavors of the mesh and model objects that are not related, as mentioned before. It restricted the generated C code and thus reduced the compilation time and memory. Figure 5 shows the compilation time for some executables through the history of CHAMPS versions. The Chapel version used is also indicated.

In Figure 5, each abrupt increase in compilation time is due to the addition of new components in the code, but for each successive version, the length of the code increases. The number of code lines went from around 8k for version 1.0 to around 48k in version 1.19. The drop in compilation time seen after version 1.17 is due to the specific modification of the code in order to reduce compilation time and memory consumption. It can be seen that the effect of that optimization operation was lost for some reasons for version 1.19. One hypothesis lies within the `where` statements discussed before, but it is not yet verified.

5 STOCHASTIC ICING APPROACH

Another feature added for the multi-physics simulations is the stochastic icing approach. Initially presented by Szilder [25] as the *Morphogenetic model*, the rationale of the method has been re-examined by Bourgault-Côté [7]. The approach aims to reproduce the chaotic character of the icing process that is mainly due to the cloud droplet distribution and the surface irregular roughness. The method is Lagrangian, meaning that it models the ice accretion by growing each iced element individually. Furthermore, to reduce the computational cost, the droplets are gathered in clusters, with the assumption that the droplets in a single cluster follow the same evolution (trajectory and freezing location). The steps are the following:

(1) Insert a droplet cluster upstream of the airfoil at a random position;
(2) Resolve the cluster trajectory;
  (a) If the trajectory impacts the airfoil, generate an element of ice;
(3) Return to step (1) until reaching the stop criterion, which is the mass of the accumulated ice.

Stochasticity is introduced through the impingement (step (1)) and the freezing (step (2a)) via probabilities and pseudo-random numbers generation.

Szilder and Lozowski [25] initially proposed to generate the element of ice (step (2a)) with a random walk model on a Cartesian grid. Then, Bourgault-Côté proposed a Cartesian advancing front [7].

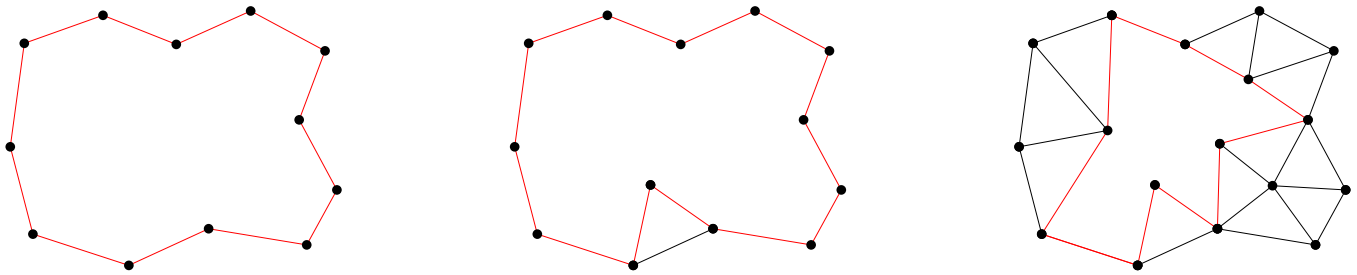
5.1 Advancing Front Algorithm

The implementation in CHAMPS differs from Bourgault-Côté's [7] method as the Cartesian advancing front is replaced by an unstructured advancing front. Therefore, an advancing front grid generator is implemented in CHAMPS, following the algorithms proposed by Löhner and Parikh [12], Peraire et al. [19] and Jin and Tanner [10]. The method is based on the dynamic creation of a grid by the generation of triangular (or tetrahedral) elements.

The steps required to complete the process are the following:

(1) Discretize the boundaries, which form the initial front;
(2) Generate the next element, based on the predefined order of treatment;
  (a) Select an existing node or create a new node;
  (b) Check intersections with existing elements, if true, return to step (2a);
(3) Update the front;
(4) Repeat (2) and (3) until reaching the stop criterion.

For the stochastic icing, the initial front is the surface discretization of the studied geometry. Since the process involves the generation of a mesh, a derived class of *Mesh_c*, *MeshStochastic_c*, is created to take advantage of the connectivity and metrics computations already available. Furthermore, CHAMPS' surface and volume meshes compatibility (Section 4.1) is used as the front initialization is performed in the same way as a surface mesh would be initialized



(a) Initial front

(b) Generation of one element and update of the front

(c) Front and mesh after several iterations

Figure 6: Unstructured advancing front process. The front is represented in red

even if the generated mesh is volumic. The front consists of the facets which are available to form a new element. These are the front facets from which the new elements are created.

As seen in Figure 6, the front and the mesh are structures that change throughout the generation of the mesh. This involves that the data structure enables the insertion and deletion of elements. To that end, the *var* keyword is used to declare the associated lists, domains, and arrays of *MeshStochastic_c*. The front is a field of *MeshStochastic_c* and takes the form of a *list* of facet indexes, and the mesh uses variable domains and arrays. As the element are added, the mesh domains are extended by redefining the domain assignment and, using the logical re-allocation of Chapel, the associated arrays are automatically resized. The front is managed through the use of *list.append()* and *list.remove()* procedures. Further developments are planned to overcome the computational cost of these memory manipulations, as the use of tree structures like Chapel's *heap* module.

5.2 Stochastic Impingement and Freezing

Randomness is first introduced as the droplets are inserted in the computational domain. Since the method is Lagrangian, the droplet trajectories are extracted from the Eulerian droplet velocity field via the computation of streamlines from random insertion points. The resolution of the trajectories is performed using the volumic RANS mesh, on which the droplet velocity field was obtained.

The process goes as follows, based on the method proposed by Rendall and Allen [21] to compute the droplet "streamline", which is its trajectory. A seeding plane is defined upstream of the geometry. Then, a cluster of droplets is inserted on this plane and the initial cell in the RANS mesh is found to extract the initial velocity vector of the cluster. From there, the entry point to the next cell is defined as the position of the closest downstream intersection of the velocity vector and the cell facets. The next cell index is obtained from the mesh cell connectivity, and the process is repeated.

The initial position of the droplet is generated with the *PCGRandom* module. A *PCGRandomStream* class is instantiated as a field of the *MeshStochastic_c* so that the stream of pseudorandom numbers stays consistent during the entire process (the seed do not change). To obtain the next initial position, *PCGRandomStream.getnext(min, max)* is simply called. The *min, max* fields are used so the pseudorandom number represents a plausible initial position in the computation domain. To limit unnecessary computations, the *min, max* fields are derived from the Eulerian droplet velocity field using an inverse trajectory computation.

Thermodynamic exchanges have yet to be computed, thus, the cluster of droplets freezes upon impact. This limits the framework to rime ice cases only.

5.3 Evolution of the Stochastic Process

If the trajectory of the cluster intersects a facet of the front, the trajectory computation stops and a new element is generated with the advancing front algorithm. The process is presented in Figure 7. The size of the created elements is given as an input. The process stops when the accumulated mass of ice (the sum of the mass of each created element) reaches the targeted mass, obtained with the icing conditions.

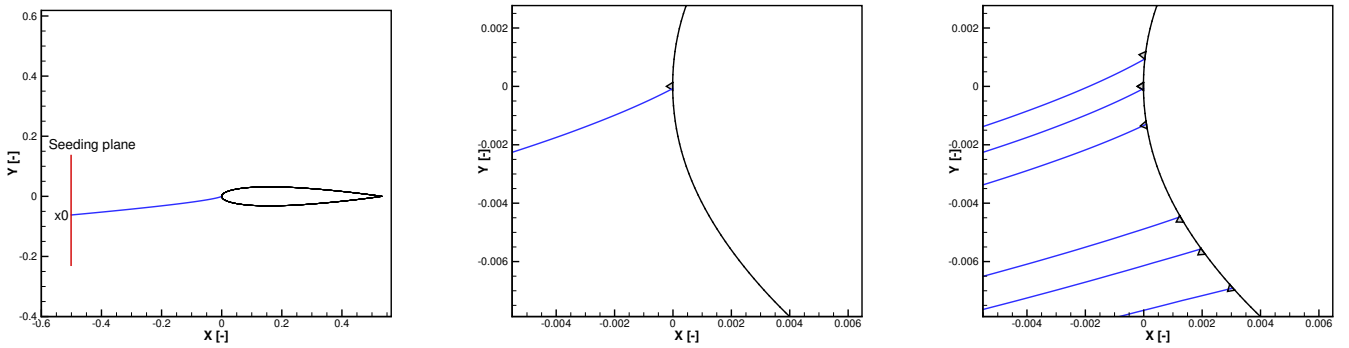
6 NUMERICAL RESULTS

6.1 2D Deterministic Icing

Rime ice is characterized by lower temperatures, at which the droplets freeze upon impact. The ice tends to grow as an extension of the geometry following the stagnation line. The test case presented in this paper is case 241 selected for the 1^{st} AIAA Ice Prediction Workshop (IPW1) [1]. The conditions for this case are presented in Table 1 and results obtained with CHAMPS are presented in Figure 8 for five ice layers performed using full mesh re-generation using the hyperbolic mesh generation method, as described in Section 3.

6.2 3D Deterministic Icing

Recently, the addition of the mesh deformation method using radial basis functions (RBF) allowed to perform multi-layer ice accretion simulations in 3D. Considering the complexity of ice shapes, such simulations are not often seen in the literature. The current methodology still has some limitations and for now, only rime cases have been tested due to their simpler shapes. Case 241 presented previously in 2D is shown in Figure 9 for five ice layers. The experimental test was performed on an extruded airfoil, which allows us to perform the 3D simulation by extruding the previous 2D grid to obtain 10 cells in the third direction with a cell width of approximately 0.5% the chord length.



(a) Insertion and trajectory computation

(b) Generation of an element upon impingement, zoom on the leading edge

(c) Generation of several elements, zoom on the leading edge

Figure 7: Evolution of the Stochastic Process

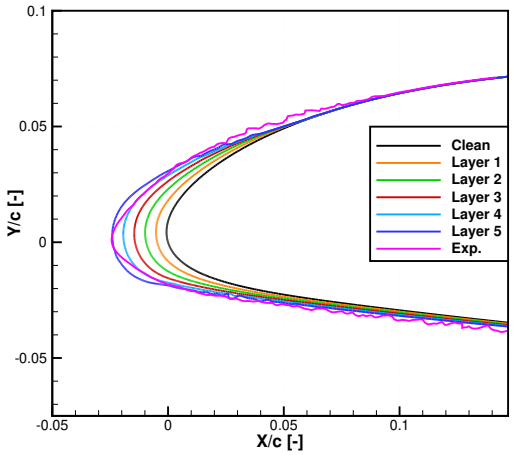


Figure 8: 2D rime ice: Case 241 from IPW1 [1]

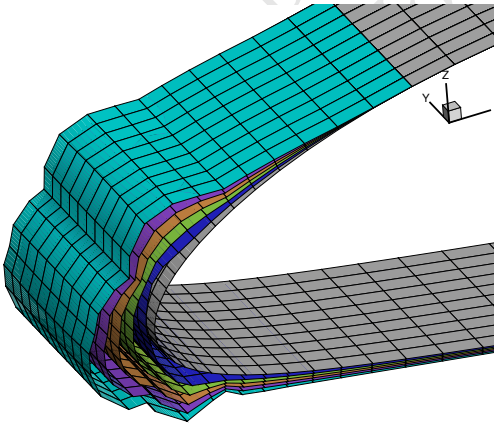


Figure 9: 3D rime ice: Case 241 from IPW1 [1]

Table 1: Test Cases Input Parameters

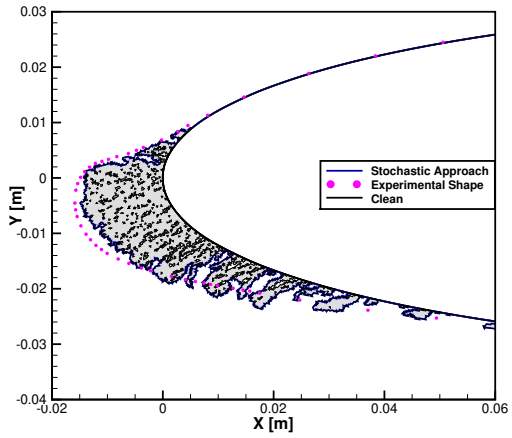
Test Case	Case 241, IPW1	Case 01, Trontin et al.
Geometry	NACA0012	NACA0012
Chord [m]	0.4572	0.5334
AoA [°]	2.0	4.0
Mach [-]	0.325	0.325
Temperature [K]	255.20	250.7
Pressure [kPa]	92.5	101.325
LWC [g/m^3]	0.42	0.55
MVD[μm]	30	20
Icing Time [min]	5.0	7.0

6.3 2D Stochastic Icing

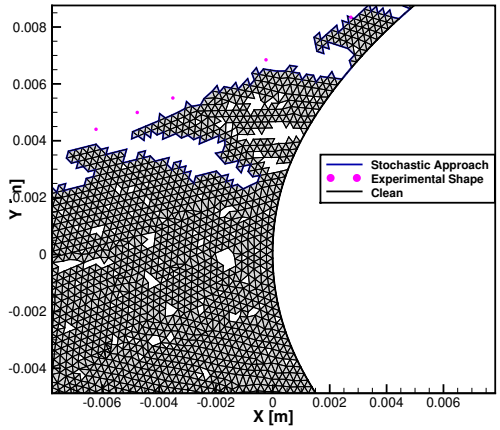
Stochastic icing results are presented on the rime case 241 [1] and rime case 01 of Trontin et al. [26]. Conditions are presented in Table 1.

6.3.1 Trontin's Rime Case 01. The results are presented in Figure 10. The obtained ice shape is compared to the experimental results [26], showing that the stochastic icing process leads in great agreement with the experiment. Figure 10b is zoomed on the leading edge of the geometry to highlight mesh generated with the advancing front method. Features like holes in the ice and ice feathers are captured, as observed in the experiments, which can not be model by deterministic approaches.

6.3.2 IPW1 Case 241. Figure 11 presents the results of ten different seeds for PRN generator. Ten different ice shapes are overlaid, with the bluescale corresponding to the likelihood of the results : the darker the area is, the most probable it is to obtain ice at this position. Thus, the impact of the stochasticity is mainly downstream of the stagnation point, especially on the intrados where ice feathers are observed. The obtained ice shapes are similar to the experimental envelope, which is the maximum combined cross section depicted in pink.



(a) Ice Shape



(b) Zoom on the leading edge

Figure 10: 2D stochastic rime ice: Case 01 from Trontin et al. [26]

7 CONCLUSION

In this paper, a multi-physics CFD software is implemented with the Chapel programming language. The flow solver was previously developed using Chapel's features to provide a flexible framework for multi-physics simulations. Some considerations are added to the data structures to meet the requirements of an icing simulation while ensuring the proper use of the computational resources. Additional modules are added, such as the droplet solver, the thermodynamic computations, the geometry deformation process, and the mesh regeneration, using inheritance to ease the development and to make the most of the generic functions and types from Chapel. A stochastic icing is also presented, which uses Chapel's standard module to generate PRN. Furthermore, memory manipulations as the ice grows are easily performed using domain resizing and list operations, enabling fast prototyping of the unstructured advancing front technique.

CHAMPS is validated against rime ice cases of the literature for the deterministic and stochastic approaches, showing results

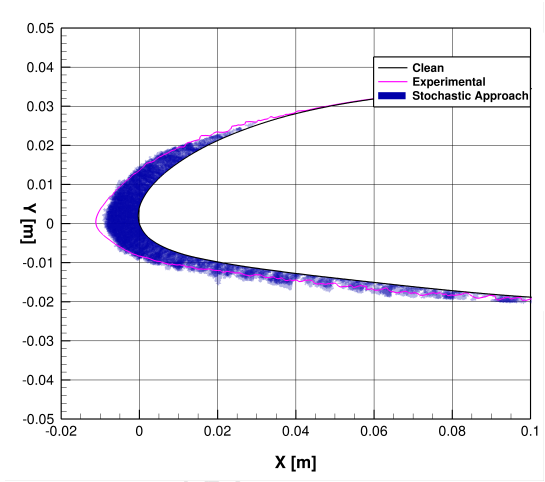


Figure 11: 2D stochastic rime ice: 10 runs of the case 241 from IPW1 [1], the bluescale represents the likelihood of the result

in great agreement with the experimental data. Future works are expected to assess the validity of the software on more icing conditions.

ACKNOWLEDGEMENTS

This work is supported by the National Sciences and Engineering Research Council of Canada (NSERC), the Fonds de Recherche du Québec Nature et technologies (FRQNT) and the Canada Research Chair in Modelling and Control of Unsteady Aircraft Aerodynamics. This research was enabled in part by support provided by Calcul Québec (www.calculquebec.ca) and Compute Canada (www.computecanada.ca).

REFERENCES

[1] AIAA. [n.d.]. 1^{st} *AIAA Ice Prediction Workshop*. https://folk.ntnu.no/richahan/IPW/.

[2] Timothy J. Barth and Jespersen Dennis C. 1989. The design and application of upwind schemes on unstructured meshes. In 27^{th} *Aerospace Sciences Meeting*. AIAA Paper 1989-0366. https://doi.org/10.2514/6.1989-366

[3] Héloïse Beaugendre, François Morency, and Wagdi G. Habashi. 2005. Development of a Second Generation In-Flight Icing Simulation Code. *Journal of Fluids Engineering* 128, 2 (02 2005), 378–387. https://doi.org/10.1115/1.2169807 arXiv:https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/128/2/378/5645781/378_1.pdf

[4] J. Blazek. 2015. *Computational Fluid Dynamics: Principles and Application* (3^{rd} ed.). Elsevier.

[5] Y. Bourgault, H. Beaugendre, and W. G. Habashi. 2000. Development of a Shallow-Water Icing Model in FENSAP-ICE. *Journal of Aircraft* 37, 4 (2000), 640–646. https://doi.org/10.2514/6.1999-246

[6] Yves Bourgault, Wagdi G. Habashi, Julien Dompierre, and Guido S. Baruzzi. 1999. A finite element method study of Eulerian droplets impingement models. *International Journal for Numerical Methods in Fluids* 29, 4 (1999), 429–449. https://doi.org/10.1002/(SICI)1097-0363(19990228)29:4<429::AID-FLD795>3.0.CO;2-F

[7] Simon Bourgault-Côté. 2019. *Ice Interface Evolution Modelling Algorithms for Aircraft Icing*. Ph.D. Dissertation. Polytechnique Montréal, Montréal, Canada.

[8] W. M. Chan. 1999. Hyperbolic Methods for Surface and Field Grid Generation. In *Handbook of Grid Generation* (1^{st} ed.), J. F. Thompson, B. K. Soni, and N. P. Weatherill (Eds.). CRC Press.

[9] Federal Aviation Administration. [n.d.]. *Eletronic Code of Federal Regulations, Appendix C to Part 25.* https://www.ecfr.gov/cgi-bin/text-idx?SID=bbf24e9c782b2057583037ed7c2efe26&mc=true&node=pt14.1.25&rgn=div5#ap14.1.25.0000_0nbspnbspnbsp.c Last accessed: 2018-08-16.

[10] H. Jin and R. I. Tanner. 1993. Generation of unstructured tetrahedral meshes by advancing front technique. *Internat. J. Numer. Methods Engrg.* 36, 11 (1993), 1805–1823. https://doi.org/10.1002/nme.1620361103 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620361103

[11] Pierre Lavoie. 2017. *Modeling of Thin Water Films on Swept Wings in Icing Condition.* Master's thesis. Polytechnique Montréal, MOntréal, Canada.

[12] Rainald Löhner and Paresh Parikh. 1988. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids* 8, 10 (1988), 1135–1149. https://doi.org/10.1002/fld.1650081003 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650081003

[13] D. J. Mavriplis and A. Jameson. 1990. Multigrid Solution of the Navier-Stokes Equations on Triangular Meshes. *AIAA Journal* 28, 8 (1990), 1415–1425. https://doi.org/10.2514/3.25233

[14] F. R. Menter. 1992. *Improved Two-Equation k-omega Turbulence Models for Aerodynamic Flows.* Technical Report. NASA TM 103975.

[15] B.L. Messinger. 1953. Equilibrium Temperature of an Unheated Icing Surface as a Function of Airspeed. *Journal of the Aeronautical Sciences* 1, 20 (Jan. 1953), 29–42. https://doi.org/10.2514/8.2520

[16] T. G. Myers. 2001. Extension to the Messinger Model for Aircraft Icing. *AIAA Journal* 39, 2 (2001), 211–218. https://doi.org/10.2514/2.1312

[17] S. Özgen and M. Canibek. 2009. Ice Accretion Simulation on Multi-Element Airfoils Using Extended Messinger Model. *Heat and Mass Transfer* 45, 3 (2009), 305–322. https://doi.org/10.1007/s00231-008-0430-4

[18] Matthieu Parenteau, Simon Bourgault-Côté, Frédéric Plante, Engin Kayraklioglu, and Éric Laurendeau. 2021. Development of Parallel CFD Applications with the Chapel Programming Language. (Jan. 2021). https://doi.org/10.2514/6.2021-0749

[19] J Peraire, J Peiró, and K Morgan. 1992. Adaptive remeshing for three-dimensional compressible flow computations. *J. Comput. Phys.* 103, 2 (1992), 269–285. https://doi.org/10.1016/0021-9991(92)90401-J

[20] Emmanuel Radenac, Helene Gaible, Herve Bezard, and Philippe Reulet. 2019. IGLOO3D Computations of the Ice Accretion on Swept-Wings of the SUNSET2 Database, In SAE Technical Paper. https://doi.org/10.4271/2019-01-1935

[21] T.C.S. Rendall and C.B. Allen. 2014. Finite-volume droplet trajectories for icing simulation. *International Journal of Multiphase Flow* 58 (jan 2014), 185–194. https://doi.org/10.1016/j.ijmultiphaseflow.2013.08.007

[22] P.L. Roe. 1981. Approximate Riemann solvers, parameter vectors, and difference schemes. *J. Comput. Phys.* 43, 2 (1981), 357–372. https://doi.org/10.1016/0021-9991(81)90128-5

[23] Philippe R. Spalart and Steve R. Allmaras. 1992. A One-Equation Turbulence Model for Aerodynamic Flows. In *30th Aerospace Sciences Meeting and Exhibit.* AIAA Paper 1992-0439. https://doi.org/10.2514/6.1992-439

[24] P. R. Spalart and S. R. Allmaras. 1994. A One-Equation Turbulence Model for Aerodynamic Flows. *Recherche Aerospatiale* 94, 1 (1994), 5–21.

[25] Krzysztof Szilder and Edward P. Lozowski. 2004. Novel Two-Dimensional Modeling Approach for Aircraft Icing. *Journal of Aircraft* 41, 4 (July 2004), 854–861. https://doi.org/10.2514/1.470

[26] Pierre Trontin, Ghislain Blanchard, Alexandros Kontogiannis, and Philippe Villedieu. 2017. Description and assessment of the new ONERA 2D icing suite IGLOO2D. In *9th AIAA Atmospheric and Space Environments Conference.* AIAA Paper 2017-3417. https://doi.org/10.2514/6.2017-3417

[27] V. Venkatakrishnan. 1995. Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters. *J. Comput. Phys.* 118, 1 (1995). https://doi.org/10.1006/jcph.1995.1084

[28] Holger Wendland. 2004. *Scattered Data Approximation.* Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511617539

[29] William B. Wright. 2005. Validation Results for LEWICE 3.0. In *43rd AIAA Aerospace Sciences Meeting and Exhibit.* AIAA Paper 2005-1243. https://doi.org/10.2514/6.2005-1243

[30] Chengxiang Zhu, Bin Fu, Zhiguo Sun, and Chunling Zhu. 2012. 3D Ice Accretion Simulation for Complex Configuration basing on Improved Messinger Model. In *International Journal of Modern Physics: Conference Series*, Vol. 19. World Scientific, 341–350. https://doi.org/10.1142/S2010194512008938