Journal of Computational Science 62 (2022) 101682

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

An open-source framework for aircraft damage simulation in engine failure events[☆]

Clara Cid [*], Aitor Baldomir, Miguel Rodríguez-Segade, Santiago Hernández

Structural Mechanics Group, School of Civil Engineering, Universidade da Coruña, 15071, Spain

ARTICLE INFO

Keywords:
Fail-safe
Damage
Uncontained engine rotor failure
Propeller blade failure
Debris

ABSTRACT

The aerospace industry demands designs capable of withstanding the simultaneous collapse of several structural elements. Examples of this failure scenario are propeller blade or uncontained engine rotor failures, where the structural integrity of the aircraft may be compromised by flying debris. To assess aircraft survivability, reliable simulation of accidental damage scenarios using physics-based models is required. Due to the complexity of characterizing these events, practitioners and researchers have traditionally assumed conservative damage envelopes, restraining potential design improvements. This research presents an object-oriented framework to automatically generate any number of damaged aircraft meshes in a realistic manner, taking into account the randomness of the event. Due to the flexibility in the software design, both random and deterministic input parameters are allowed, such as debris origin, impact orientation, number of impacts, debris size, debris velocity, spread angles and ballistic penetration equations. The tool is applied to a commercial narrow-body aircraft in which real failure scenarios are simulated.

1. Introduction

The term “damage” has been widely used in the aerospace community to refer to different situations affecting structural integrity. The first can be understood as the progressive loss of structural capacity of the material (usually due to fatigue or corrosion). The second refers to the complete removal of a chunk of material of a given size (usually due to accidental collision). This paper deals with the latter definition of damage, particularly accidental collision caused by engine debris.

After the introduction of turbine engines in commercial aviation in the mid-1950s, there were several accidents related to failures of different engine components. These failures often result in high-energy debris perforating the engine case and striking the aircraft structure, leading to a catastrophic outcome if not accounted for in the design process. Accordingly, aircraft must be designed to withstand such damage, hence ensuring passengers safety. Depending on the location of the rotating element inside or outside the nacelle, the accidental situation can be classified into blade release or uncontained engine rotor failure (UERF). A blade release occurs when bare engine blades, such as those in turboprop aircraft, become detached and pose potential hazard to the aircraft. A UERF occurs when fragments of rotating engine parts perforate the nacelle.

Accidentals collisions can be addressed through the fail-safe design philosophy, which allows aircraft to withstand significant structural damage during flight without incurring a catastrophic event and land safely. This can be accomplished through a design that contemplates redundant load paths, which provide alternative resistant structural schemes to sustain the stresses in case any material is lost, redistributing the internal forces on the remaining structure. Several investigations have been carried out in recent years that combine optimization techniques with the fail-safe strategy. Several investigations have analyzed the local failure in topology optimization for truss structures [1–6], where local failure can be modeled straightforwardly by removing one bar from the truss, since there is a clear definition of structural members. In contrast, other authors [7–12] addressed local failure in continuum topology optimization, where there is no real notion of a structural member and holes of varying shape and size can occur. In these approaches, damage was limited to a small rectangular subset of the design domain where all material was removed. From a different perspective, recent research focused on fail-safe size optimization needs to consider finite element models of the intact structure and a set of possible partial collapses in the optimization process. Fail-safe size optimization was firstly applied to shell structures by Baldomir et al. [13]. This multi-model approach aimed to obtain a minimum penalty weight

☆ Source code available online at https://github.com/claracbengoa/damagecreator.

* Corresponding author.

E-mail addresses: clara.cid.bengoa@udc.es (C. Cid), abaldomir@udc.es (A. Baldomir), miguel.alonso1@udc.es (M. Rodríguez-Segade), hernandez@udc.es (S. Hernández).

https://doi.org/10.1016/j.jocs.2022.101682
Received 1 October 2021; Received in revised form 15 March 2022; Accepted 15 April 2022
Available online 12 May 2022

on the intact structure, while simultaneously fulfilling the limit-states in both the intact and damaged configurations. Subsequently, several methodologies emerged to contemplate several sources of uncertainty into the design process [14–18]. An important aspect to mention that differs from the way damaged scenarios are addressed in fail-safe topology optimization, is that fail-safe size optimization techniques require imposing design constraints on the set of damaged finite element models, which must exist prior to initiating the optimization process. In these works, the damaged models were generated manually, which could be extremely cumbersome and imprecise due to the inherent uncertainty of the event. Therefore, a reliable way to simulate these events is needed in the aerospace community to adequately address the fail-safe optimization problem. This is the genesis of the present research and the main driver of the work.

The methodologies described above are intended to provide a means to meet regulatory requirements for discrete source damage while obtaining lightweight designs. For this reason, it is necessary to apply them to industrial problems with a real-world application, such as complex aircraft finite element models. To perform fail-safe size optimization, a set of damaged configurations needs to be generated from the intact finite element model. In this regard, regulatory agencies state that the aircraft must withstand discrete source damage events, such as bird, blade and engine strikes. Examples include FAR 25.571(e) [19], FAR 25.905(d) [20] and FAR 25.903(d)(1) [21]. Although several Advisory Circulars provide guidelines for the application of these regulations, (AC 25.571-1D [22], AC 25.905-1 [23] and AC 20–128 A [24]) there are no specific instructions on how to deal with these remote but critical damage scenarios. As a first approach, the Federal Aviation Administration developed the Uncontained Engine Debris Damage Assessment Model (UEDDAM) [25], a CAD-based tool for system level hazard assessment in uncontained engine rotor failure analysis. The objective was to determine the probability of catastrophic hazard (P_{HAZ}) given an uncontained engine event, determine the major contributors to the P_{HAZ} and perform trade studies to minimize the contributions of those components. This was done taking into account the debris characteristics and fragment penetration. Although useful, the main drawback of UEDDAM is the absence of physical models to assess the residual strength of the damaged aircraft. Therefore, there is a need in the aircraft industry for a CAD and CAE-based tool that can generate damaged meshes from aircraft FEMs, thus providing a deeper insight into the structural capacity against these events.

Traditionally, damage envelopes were often adopted to represent the accidental scenario due to the difficulties in characterizing the damaged configuration, eliminating entire panels of the aircraft structure. That strategy led to more conservative designs, since a larger-than-necessary area of the structure was removed. An example of a generic damage envelope in an aircraft is presented in Fig. 1, where the disproportionate size of the holes adopted in the industry is illustrated. Regarding fail-safe optimization strategies, the standard approach is to manually generate holes in the FEMs, with the main drawback of being an extremely tedious, subjective and inaccurate task for various reasons. The randomness of the event, including the number of debris hitting the structure, hole location, size, orientation, impact area or velocity, give rise to infinite possibilities of damaged configurations. Thus, the main question arises: How can we generate a representative sample of damaged scenarios that is sufficiently reliable?

In this research, a tool called DamageCreator is proposed to address the need for an adequate characterization of damage scenarios. Subsequently, the damaged models can be used in conjunction with existent fail-safe optimization strategies to meet current regulations on aircraft survivability to discrete source damage, inevitably leading to more competitive designs. It consists of an open-source, object-oriented framework to automatically generate any number of damaged meshes from aircraft models given an uncontained engine rotor or a propeller blade failure, with a minimum computational cost. Due to the flexibility in the software design, both random and deterministic input parameters are allowed. These input parameters (debris origin, impact orientation, number of impacts, debris size, debris velocity, spread angles and ballistic penetration equations) can be based on real accident databases or come from experimental studies or tests of different failed engine components.

The authors presented a preliminary version of the tool as a conference paper [26] to introduce the concept to the aerospace community. In the current work, the complete definition of the software functionalities is included. Among all the improvements, the code has been completely parameterized and the internal processes are explained in detail, illustrated with figures supporting the implementation procedures. In addition, an in-depth explanation of the relationship between the main classes and methods is provided. Several conservative assumptions specified in the AC 20–128 A [24] are considered. A crucial statement is that "*the fragment is considered to possess infinite energy, and therefore to be capable of severing lines, wiring, cables and unprotected structure in its path, and to be undeflected from its original trajectory unless deflection shields are fitted*". Therefore, this work does not attempt to accurately model impact simulations using explicit computational techniques. The strength of the tool is to predict debris trajectories and sizes within the vulnerable impact area, generating holes in the fuselage assuming that the debris has infinite energy, i.e., removing all the material in its path. Nevertheless, the ballistic curves functionality

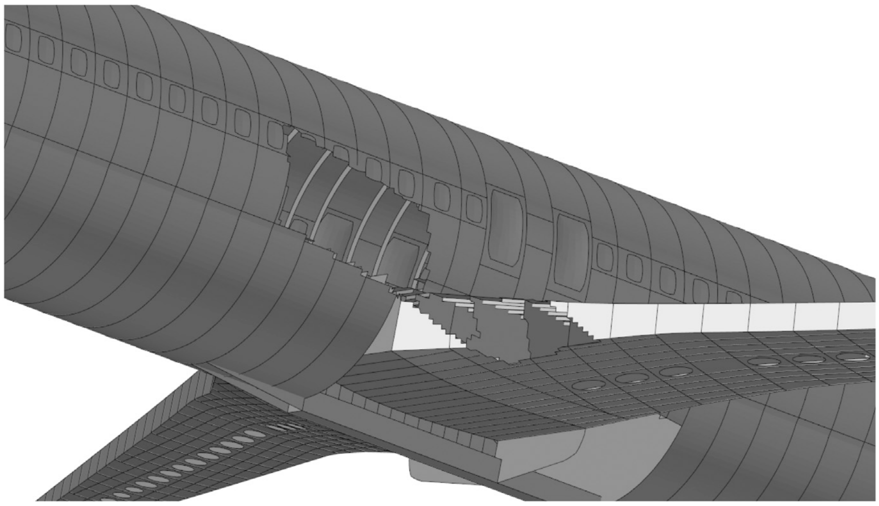


Fig. 1. Example of a damaged model based on the damage envelope approach.

was included in the software architecture, so that further physical knowledge can be incorporated into the simulation process.

This document is structured in the following sections. Section 2 summarizes the engine components that can fail and the types of debris released. In Section 3 an overview of the software is first presented, explaining the main concepts, such as the vulnerable impact area and the spread and translational risk angles. Then, the input parameters of the software are stated, followed by a simple example of the run python script. After that, the detailed description of the software implementation is discussed, including conceptual explanations and the main functions, classes and methods. An example of a narrow-body fuselage-wing assembly is presented in Section 4, where several damaged configurations are generated. Section 5 provides concluding remarks and suggests topics for future study.

2. Events and debris characterization

As stated in the introduction, AC 20–128 A [24] was issued to quantitatively assess the risk of a catastrophic failure in a UERF, defining as a first approximation the general types of fragments to be considered. Released fragments have a wide range of size and shapes, which could result in holes of varying magnitude. As part of the Aircraft Catastrophic Failure Prevention Research Program [27], the Society of Automotive Engineers (SAE) produced Aerospace Information Reports categorizing UERFs between 1976 to 1983 [28] and 1984 to 1989 [29]. Through this documentation, the FAA published reports to define the characteristics of large engine uncontained debris [30] and small engine uncontained debris [31], by gathering historical data from actual accidents and establishing values for the debris sizes, weights, exit velocities, and trajectories associated with each event that can be used to update AC 20–128 A. As a next step, the UEDDAM project [25] improved the fragment characterization conducted so far. In that report, the fragment size is based on the damage (hole) size made in the aircraft. The hole dimensions were normalized to the dimensions of the engine component. For blade or disk fragments, the normalized size is the fragment size divided by the blade length or disk

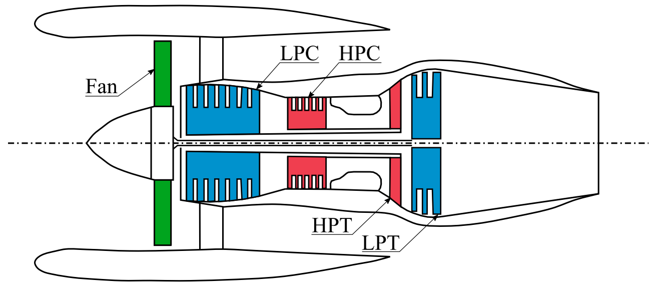


Fig. 2. Engine components (fan, compressor and turbine) whose failure can originate a UERF.

diameter, respectively. This normalization process allows the damage size to be scaled, so it is applicable to other aircraft models when the dimensions of the engine component change.

UERFs can occur due to the failure of different engine components. The main rotating elements are the fan, high and low pressure compressor (HPC and LPC), and high and low pressure turbine (HPT and LPT), as presented in Fig. 2. These rotors consist of a central disk and blades, which can be connected by a rim or spacer. The rupture of any of these elements may cause the engine to fail, releasing debris that could hit the aircraft structure.

Table 1 presents a summary of the failure type and released fragments in a HBPR turbofan engine, created from data in the aforementioned reports. It is important to note that each failure event produces different type of fragments. For instance, if a blade of the fan fails, blades of different sizes are detached. On the other hand, if a disk of the fan fails, both blades and disks are released.

Fig. 3 shows the three types of released fragments. Although each fragment has an irregular 3D shape, it is approximated by the enclosing bounding box, whose dimensions are length (L), width (W) and thickness (T). The impact is simplified to the collision of one of the faces of the bounding box, as presented in Fig. 3. This 3D–2D mapping criterion is based on data provided by the Federal Aviation Administration reports on the analysis of large engine uncontained debris [25,30], containing historical information on accidents in which disks, rims or blades were released. Given the rotational speed of the engine component before being released, the debris projection is chosen according to the most likely striking face, which corresponds to a projection orthogonal to the exit velocity vector. Thus, the projection parallel to the plane of rotation is less probable to occur. The assumption of the most likely rectangular projection is more conservative than alternative shapes, since the debris fragment is fully contained in the bounding box. A graphical representation of the 3D–2D mapping criterion is shown in Fig. 4.

3. Software architecture

3.1. General overview

This section introduces the DamageCreator tool, presenting its overall approach. The aim is to generate a set of damaged FE meshes of an aircraft model, resulting from accidental engine failure events, where debris violently strikes the fuselage. To study the effect of debris collision within the airframe, it is necessary to define the vulnerable impact area. It is that zone of the airplane likely to be impacted by uncontained fragments generated during a rotor failure. This zone is highlighted in red color in Fig. 5. Thus, to fully define the characteristics of the vulnerable impact area, it is necessary to define a set of geometric parameters.

The debris origin is the point on the engine shaft at which the failure occurs and from which the debris fragments break off. This point corresponds to the location of the fan, compressor or turbine,

Table 1
Classification of HBPR turbofan engine failure events.

Engine component	Failure type	Released fragments	Engine component	Failure type	Released fragments
Fan	Blade Event	Blades	HP Turbine	Blade Event	Blades
	Disk Event	Blades		Spacer-Rim Event	Blades
		Disks			Rim
Compressor	Blade Event	Blades			Spacer
	Spacer-Rim Event	Blades		Disk Event	Blades
		Rim			Disk
	Disk Event	Blades	LP Turbine	Blade Event	Blades
		Disk (Large Fragment)		Blade Event Last Stage	Blades
		Disk (Intermediate Fragment)		Spacer-Rim Event	Blades
					Rim
				Disk Event	Blades
					Disk

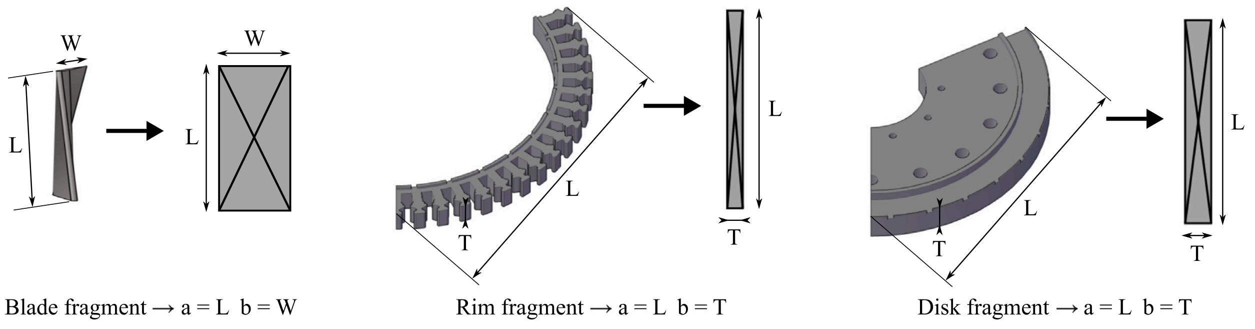

W
L
W
L
L
T
L
T
L
T
L
T
Blade fragment ⟶ a = L b = W
Rim fragment ⟶ a = L b = T
Disk fragment ⟶ a = L b = T

Fig. 3. Debris fragments vs. Debris elements (2D simplification of the debris fragment).

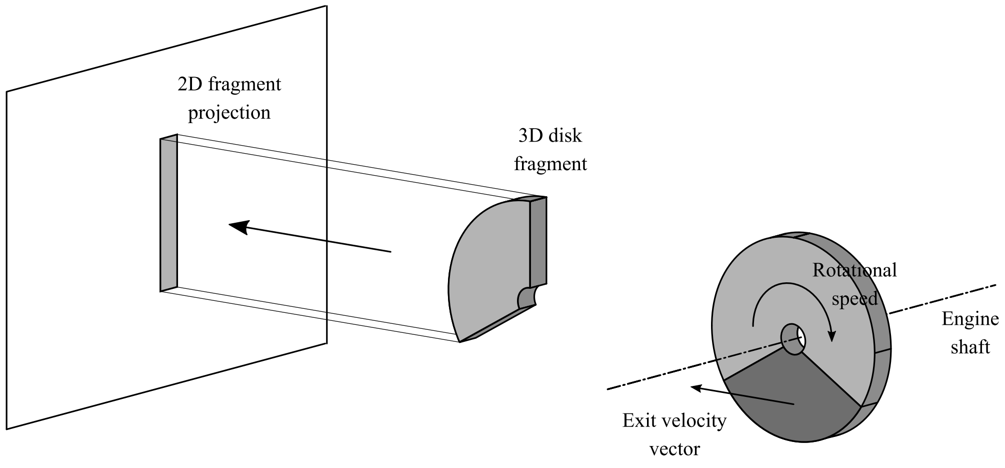


Fig. 4. Graphical representation of the 3D–2D mapping criterion.

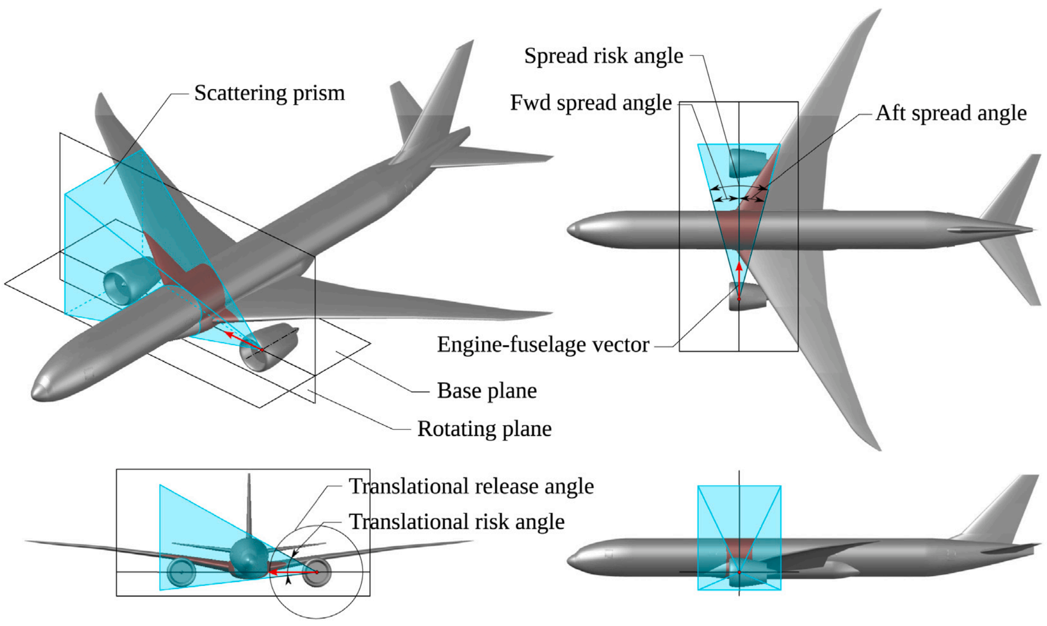


Fig. 5. Example of impact area due to an uncontained engine failure.

depending on the failed component. The plane passing through the debris origin and orthogonal to the longitudinal axis of the aircraft is called the rotating plane. The plane passing through the debris origin and orthogonal to the vertical axis is called the base plane. The intersection of these two planes results in the engine-fuselage vector, which is the reference axis from which the debris scattering angles

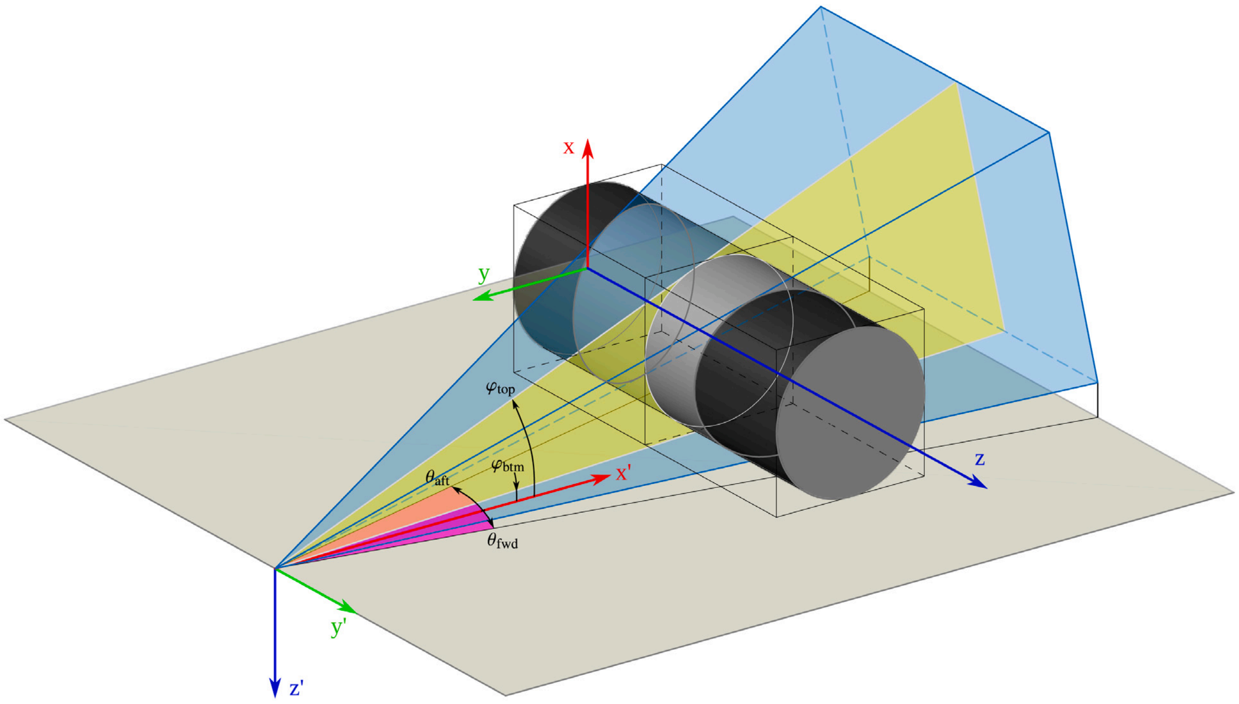


Fig. 6. Definition of the local axis of the scattering prism, spread and translational risk angles.

are measured. Horizontally (contained in the base plane) the debris trajectory is bounded between the spread risk angle. Measured from the engine-fuselage vector, these angles are called forward (fwd) and after (aft) spread angles. Vertically (contained in the rotating plane) the translational release angle is uniformly distributed from 0 to 360 degrees due to the cylindrical symmetry of the rotors. The part of this angle at which the debris may collide is known as the translational risk angle. Measured from the engine-fuselage vector, the bounds are denoted as top and bottom (btm) translational angles. The volume enclosed by these angles gives rise to the scattering prism, which bounds the debris trajectories. The intersection of the scattering prism with the aircraft delimits the impact area, as shown in Fig. 5.

Fig. 6 presents a simple case to explain the references used in the software. The first premise to be met is that the longitudinal axis of the fuselage coincides with one of the three coordinate axes. Thus, the rotating plane will be parallel to a coordinate plane and the engine-fuselage vector will be contained in this plane. In addition, the aircraft nose can point in either the positive or negative direction of the coordinate axis.

This simple graphical example simulates an aircraft fuselage through a cylindrical barrel contained in its bounding box. The z-axis corresponds to the longitudinal axis of the aircraft, with increasing direction towards the aircraft nose, the rotating plane is the xy-plane, and the engine-fuselage vector corresponds to the local x' axis. It can be seen how the spread and translational risk angles are measured taking as reference the engine-fuselage vector as well as the base and rotating plane. Spread risk angles (θ_{fwd}, θ_{aft}) are shown in the base plane in magenta and translational risk angles (φ_{top}, φ_{btm}) are measured in the rotating plane in yellow color.

3.2. Input parameters

The DamageCreator interface requires several input parameters to generate the damaged configurations. The software API allows flexibility in the nature of the input parameters, which can be defined as random or deterministic. Therefore, probability distribution functions can be used to define the debris characteristics, either by using discrete probability distribution functions obtained from databases of real events and experimental tests or by approximating them to a known distribution. In addition, some parameters may depend on others, (e.g. weight and velocity of a specific debris size, the ratio in the debris dimensions, etc.). The list of input parameters is presented below:

- **Mesh file** (`NameFile`): aircraft mesh with elements IDs, node coordinates and connectivity information
- **Number of damaged configurations** (`nDamagedConfigs`): Number of finite element meshes to be generated
- **Aircraft forward direction** (`fwdDirection`): vector parallel to the aircraft's longitudinal axis (origin: tail cone; end: nose of the aircraft)
- **Debris origin** (`origin`): coordinates of the failed engine component (release point)
- **Rotating plane** (`rotatingPlane`): the plane passing through the debris origin and perpendicular to the longitudinal axis of the aircraft (Fig. 5)
- **Engine-fuselage vector** (`orientation3D`): vector defined through the intersection of the base plane and rotating plane, at the debris origin. It is the vector that defines orientation of the scattering prism (Fig. 5)
- **Number of impacts** (`nImpactLines`): number of released fragments that will strike the aircraft structure
- **Fwd and Aft spread angles** (`spreadAngles`): forward and after spread angles (Fig. 6)
- **Debris size** (`a,b`): fragment dimensions specified in Fig. 3
- **Debris velocity** (`velocity`): fragment velocity when released from the engine
- **Ballistic penetration equation** (`ballisticEq`): equation to establish whether the debris element perforates the aircraft structure. DamageCreator assumes that after each impact a new reduced residual velocity has to be considered. To calculate this new velocity value, ballistic penetration data must be supplied as a function of the impact vs. exit velocity. The fragment may not penetrate, produce only one entry hole or generate multiple holes along its trajectory. Ballistic penetration equations simultaneously depend on many factors: debris mass, debris size, aircraft material

```
from damageCreator import FEMesh, ScatteringPrism, DamageCriteria
from scipy.interpolate import interp1d
import numpy as np

nDamagedConfigs = 100
origin = [-1.75, 5.75, 3.0] # (m)
orientation3D = [1, 0, 0] # Engine-fuselage vector
rotatingPlane = 'XY'
nImpactLines = 11
spreadAngles = [15, -30] # Fwd and Aft spread angles
a = 0.406; b = 0.203 # Fan blade debris dimensions (m)
velocity = 246 # (m/s)

# Simple Ballistic equation
ballisticEq = interp1d([175, 200, 225], [120, 150, 175],
                       fill_value='extrapolate')

# Define the damage criteria
vThreshold = 120 # m/s
damageCriteria = DamageCriteria(vThreshold, ballisticEq)

# Mesh to import
NameFile = 'mesh.med'

# Mesh Instance
fuselage = FEMesh(NameFile)

# Get the translational risk angles
translationalRiskAngles = fuselage.getTranslationalRiskAngle(origin,
                          orientation3D,rotatingPlane)

# Define the scattering Prism
scatteringPrism = ScatteringPrism(origin, orientation3D, rotatingPlane,
                  spreadAngles, translationalRiskAngles)

for i in range(nDamagedConfigs):
    # Generate debris list
    debrisList = scatteringPrism.generateDebris(nImpactLines,
                                                [a, b], velocity)
    # Generate damagedMesh instance from a list of debris objects
    damagedConfiguration = fuselage.generateDamagedConfig(debrisList,
                           ballisticEq, selectionMethod='OneNode')
    damagedConfiguration.exportMesh('damagedConfig-'+str(i+1)+'.med')
```

Fig. 7. Run script for a simple example.

and plate thickness of the aircraft structure, as stated by López-Puente et al. [32]. It is an ongoing work to incorporate mass properties and experimental curves into the software to associate a ballistic curve to each hole from a debris element.

A simple example of the run script is presented in Fig. 7. For code simplicity, all input parameters (number of impacts, fwd and aft spread angles, debris size and velocity) were taken in the run script as deterministic values. Nevertheless, there are three internal parameters that are always considered as random variables: the debris trajectory, roll and pitch angles. By default, the debris trajectory is uniformly distributed within the scattering prism, bounded by the translational risk angle and the spread risk angle. However, as the debris trajectory could follow a different probability distribution, the software supports as input parameter any distribution function within this angles. Fragment roll(α) and pitch (β) are uniformly distributed between 0 and 360 degrees and -45 and 45 degrees, respectively. The definition of these angles is further discussed in Fig. 12.

3.3. Software implementation

As mentioned in the previous sections, DamageCreator generates partial collapses in aircraft finite element models as a consequence of uncontained engine rotor or propeller blade failures. For this purpose, the tool is designed based on the interaction of the finite element mesh of an aircraft model with a debris element. Each debris element will be associated with an origin and a trajectory, as well as size and velocity characteristics.

We chose the Python programming language to develop DamageCreator due to the advantages of the object-oriented programming paradigm and an object composition design pattern. We employ the Salome platform API because it allows us to handle complex meshes and geometric operations on those meshes.

The full cycle of DamageCreator comprises interaction between several software environments. To illustrate this idea, the flowchart corresponding for the generation of a single damaged mesh from the entire set of configurations required in the code (`nDamagedConfigs`) is presented in Fig. 8. In addition, a summary of the implementation procedure is presented in Fig. 9, showing the relationship between the main classes and methods.

The blue boxes contain the class name and instance generated, whereas the pink boxes refer to the relevant methods and attributes in the corresponding instance. Nested boxes represent the call-stack hierarchy. In addition, the arrows indicate input/output relationship and the double dashed lines show instance relationship.

A detailed explanation of the software implementation is provided below, focusing on the sequential description of the main classes, methods and attributes presented in the diagram of Fig. 9. The steps

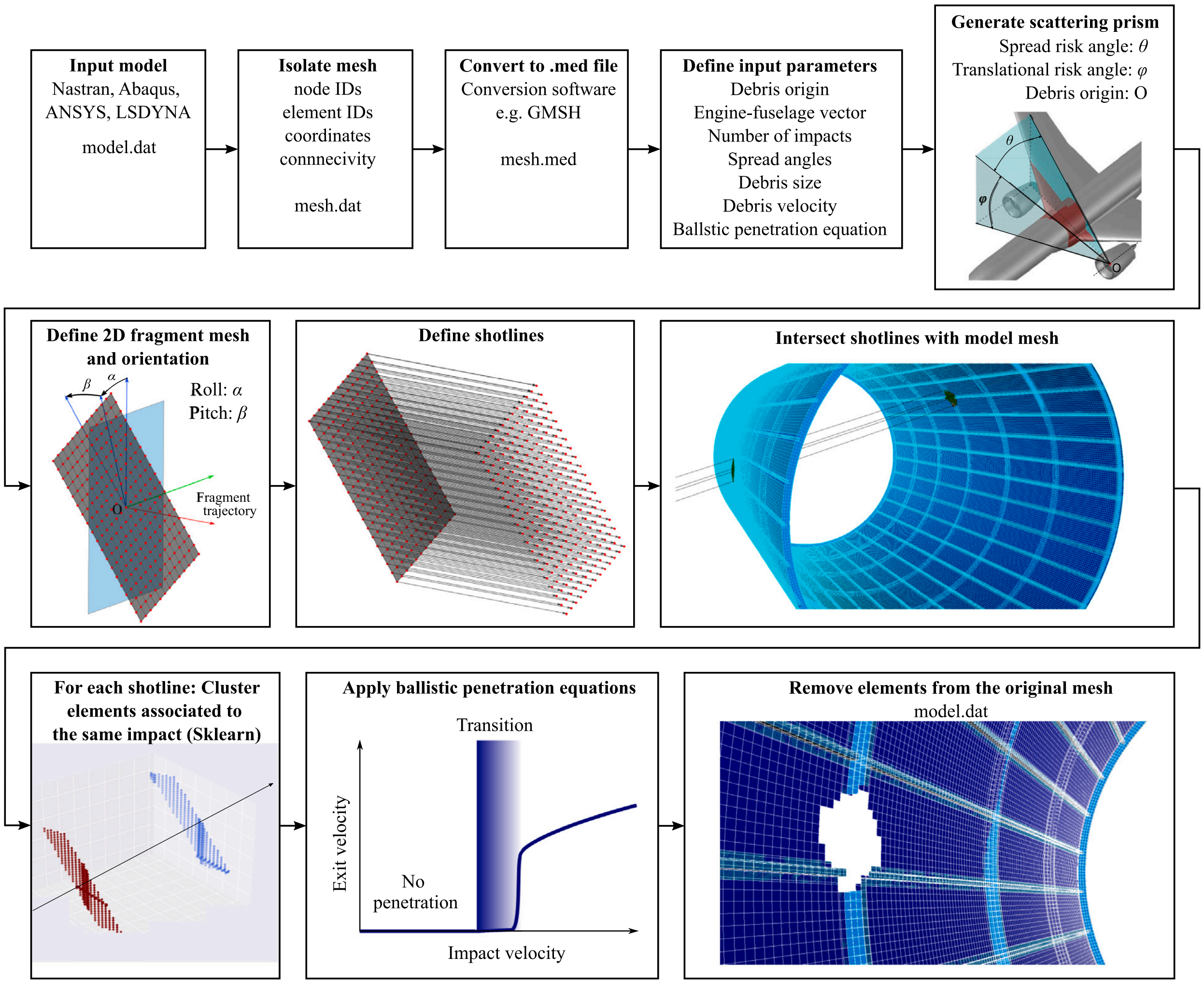


Fig. 8. Flowchart of the DamageCreator.

carried out to generate a damaged configuration can be summarized in the following ideas:

- Aircraft FEM import
- Definition of input parameters
- Determination of vulnerable impact area
- Generation of the geometry of the debris element, impact line and velocity
- Definition of damage criteria (ballistic curves)
- Meshing of the debris element, generation of shotlines and intersection with the aircraft mesh
- Clustering elements and sorting by proximity to the debris origin
- Application of damage criteria to define number of holes
- Deletion of elements from the original mesh

3.3.1. *Aircraft FEM import*

Initially, the mesh must be converted to a readable format (.med) by the Salome platform.

3.3.2. *Definition of input parameters*

Then, the input parameters are defined. They are classified into forward direction, prism parameters (debris origin, engine-fuselage vector and rotating plane) and debris database (forward and spread angle, number of impacts, debris dimensions and debris velocity).

3.3.3. *Determination of vulnerable impact area*

The first step is to define and calculate the spread and translational risk angles. As explained in Fig. 6, these angles are defined relative to the engine-fuselage vector (x'), the base plane and the rotating plane.

The forward and after spread angles (θ_{fwd}, θ_{aft}) are measured from the engine-fuselage vector (`orientation3D`) and are contained in the base plane, considering that the nose of the aircraft points in the upward direction of the aircraft longitudinal axis. Otherwise, the parameter `fwdDirection` has to be entered as a vector parallel to the longitudinal axis and of negative sign to update the sign of the spread angles.

The top and bottom translational risk angles (φ_{top}, φ_{btm}) are measured from the engine-fuselage vector and are contained in the rotating plane, with the tangent lines to the structure passing through the debris origin. The process of calculating the tangent lines is illustrated graphically in Figs. 10 and 11. Using the scattering prism parameters (`origin`, `orientation3D` and `rotatingPlane`), the method `getTranslationalRiskAngle` is applied to the `fuselage` object to calculate the top and bottom translational risk angles. The boundary vertices ($v_{1,\text{near}}$, $v_{1,\text{far}}$, $v_{2,\text{near}}$, $v_{2,\text{far}}$) are computed using the method `getBoundVertices`.

First, the boundary planes of the bounding box are defined. After the intersection the boundary planes with the rotating plane, the coordinates of the boundary points are obtained extracting the vertices of the boundary lines, as illustrated in Fig. 10. Then, the method

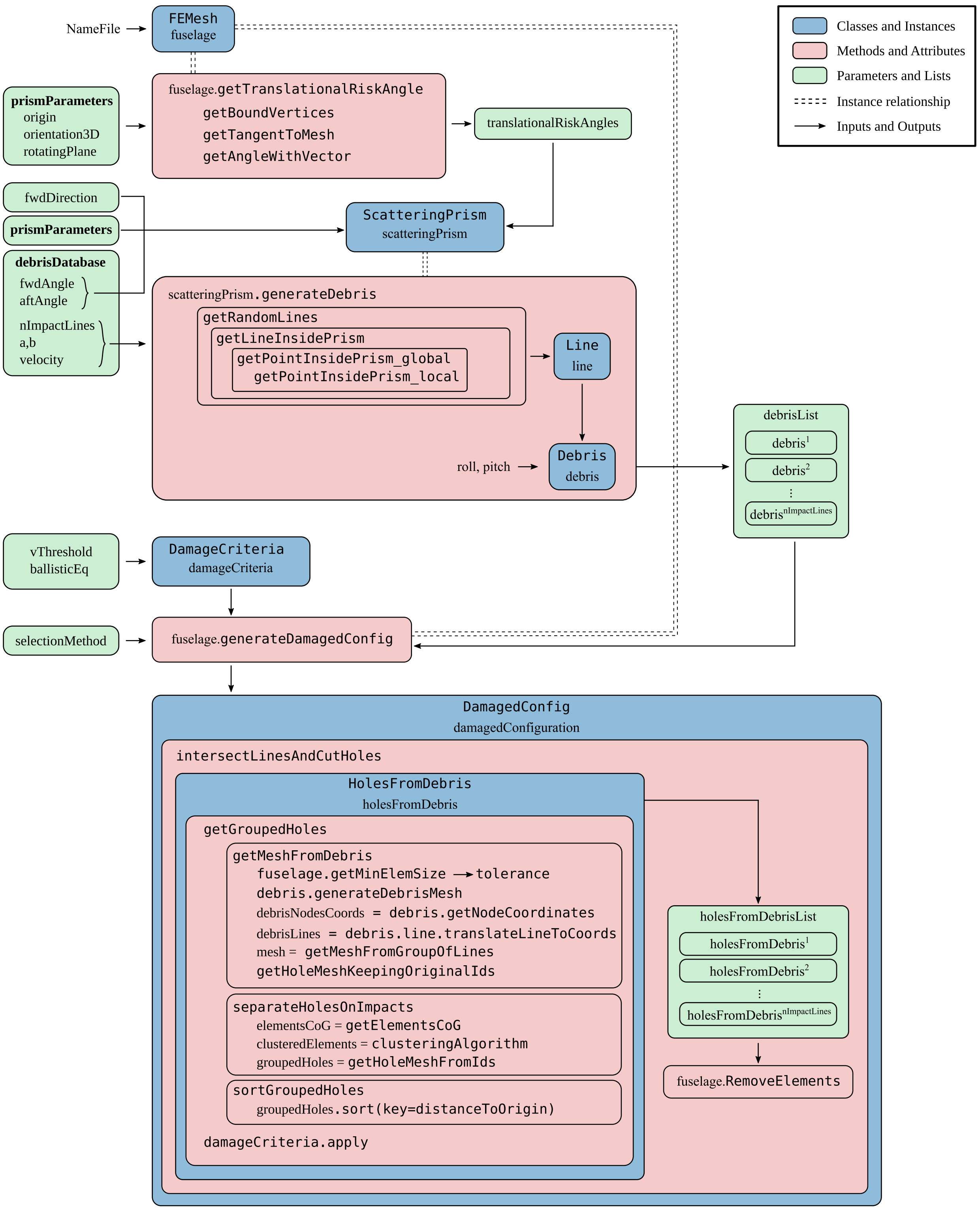

NameFile
FEMesh fuselage
fuselage.getTranslationalRiskAngle
getBoundVertices
getTangentToMesh
getAngleWithVector
translationalRiskAngles
prismParameters origin orientation3D rotatingPlane
fwdDirection
prismParameters
ScatteringPrism scatteringPrism
debrisDatabase
fwdAngle aftAngle
nImpactLines a,b velocity
scatteringPrism.generateDebris
getRandomLines
getLineInsidePrism
getPointInsidePrism_global
getPointInsidePrism_local
Line line
roll, pitch
Debris debris
debrisList
debris¹ debris² debris^nImpactLines
vThreshold ballisticEq
DamageCriteria damageCriteria
selectionMethod
fuselage.generateDamagedConfig
DamagedConfig damagedConfiguration
intersectLinesAndCutHoles
HolesFromDebris holesFromDebris
getGroupedHoles
getMeshFromDebris
fuselage.getMinElemSize → tolerance
debris.generateDebrisMesh
debrisNodesCoords = debris.getNodeCoordinates
debrisLines = debris.line.translateLineToCoords
mesh = getMeshFromGroupOfLines
getHoleMeshKeepingOriginalIds
separateHolesOnImpacts
elementsCoG = getElementsCoG
clusteredElements = clusteringAlgorithm
groupedHoles = getHoleMeshFromIds
sortGroupedHoles
groupedHoles.sort(key=distanceToOrigin)
damageCriteria.apply
holesFromDebrisList
holesFromDebris¹ holesFromDebris² holesFromDebris^nImpactLines
fuselage.RemoveElements
Classes and Instances
Methods and Attributes
Parameters and Lists
Instance relationship
Inputs and Outputs

Fig. 9. Flowchart showing the relationship between main classes and methods of DamageCreator.

`getTangentToMesh` computes the lines tangent to the mesh, using the debris origin and the boundary vertices. This process is based on the bisection method, and it is explained graphically in Fig. 11 for the tangent point T_{top}. A series of intervals $[L_i, U_i]$ are updated recursively depending on whether a line passing from the origin to each boundary intersects the mesh. The interval is divided in half to obtain a new point and check whether the line passing through it intersects the mesh. If it does, the nearest half is taken as the new interval. Otherwise, the farthest part is chosen. The process is repeated until the length of the interval reaches a given tolerance. Finally, the translational risk angles are calculated as the angle between the tangent line and the engine-fuselage vector, using the method `getAngleWithVector`. The translational risk angles $(\varphi_{top}, \varphi_{btm})$ are shown in Fig. 11.

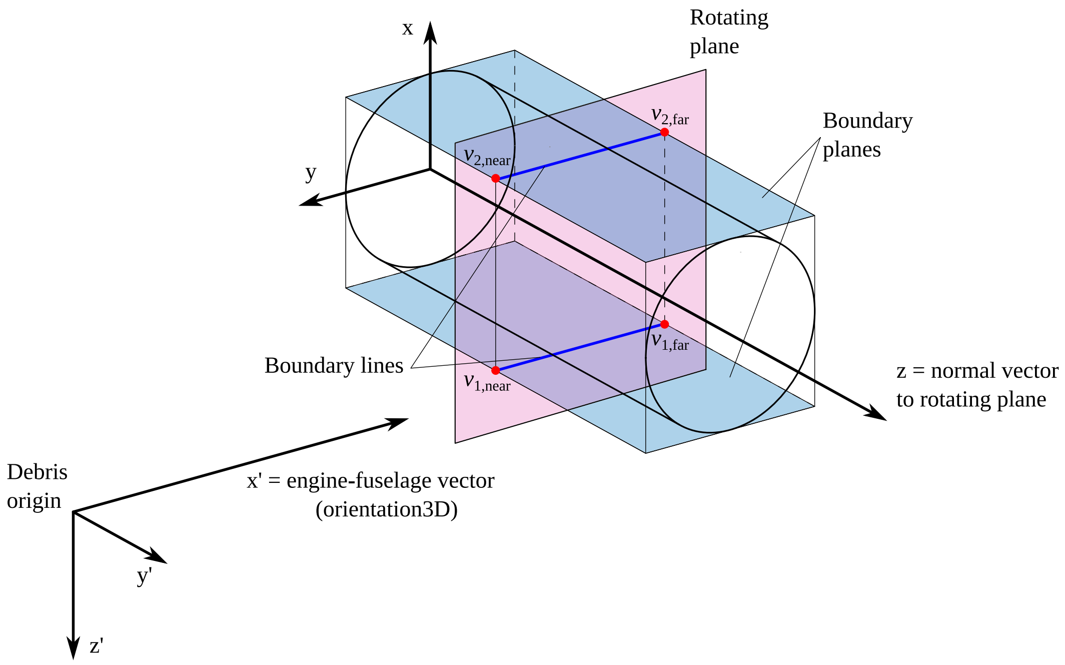


Fig. 10. Determination of boundary vertices to calculate the translational risk angles.

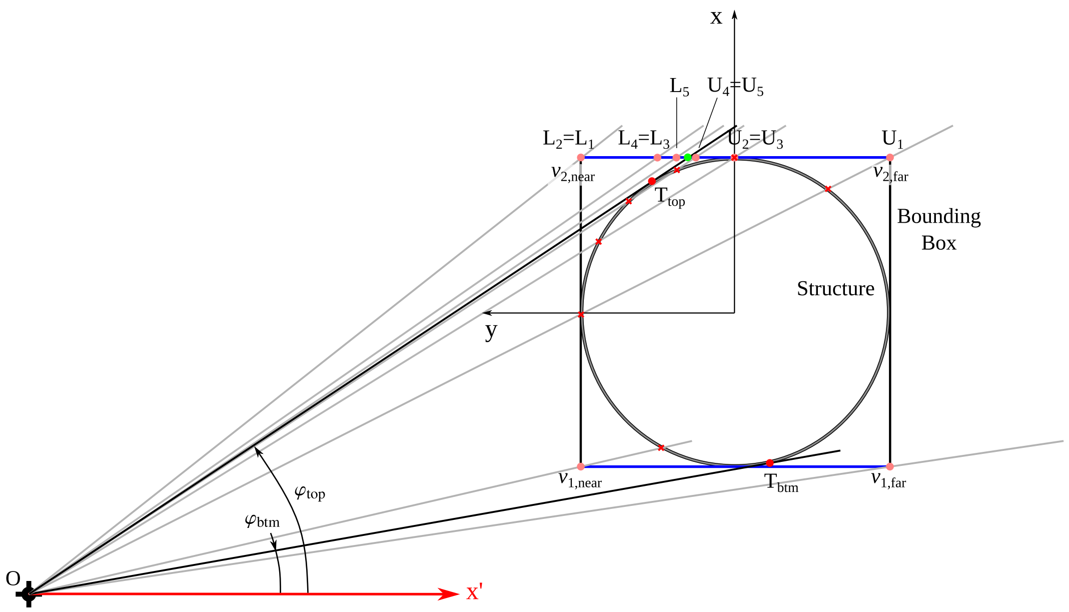


Fig. 11. Determination of tangent points, T_{top} and T_{btm}, for the calculation of translational risk angles.

Using the after and forward spread angles, and top and bottom translational risk angles, the scattering prism of the debris fragment is defined, generating an instance of the class `ScatteringPrism`.

3.3.4. Generation of the geometry of the debris element, impact line and velocity

The next step is to define the `debris` object, which includes the generation of its equivalent geometric surface, its impact line and velocity inside the scattering prism.

To this end, a 2D surface is defined perpendicular to the debris trajectory vector, with length `a` and width `b`, and two angles are considered for orientating the element: fragment roll(α) and pitch (β), defined as random variables uniformly distributed between 0 and 360 degrees and 45 and -45 degrees, respectively. These angles are graphically represented in Fig. 12. In addition, the debris trajectory is also a random variable, uniformly distributed within the scattering prism, bounded by the translational risk angle and the spread risk angle defined in Fig. 5.

For the software implementation, the method `generateDebris` is applied to the `scatteringPrism` object. First, the method `getRandomLines` is used to generate `nImpactLines` inside the scattering prism. For each line, the method `getLineInsidePrism` calculates the vector that defines the impact line in the local axes ($x'y'z'$), as presented in Fig. 6. Then, this vector is transformed into the global reference system (xyz) and the coordinates of the point inside the prism are obtained. This is done through the methods `getPointInsidePrism_local` and `getPointInsidePrism_global`. Subsequently, the line that defines the debris trajectory is generated through the `Line` class.

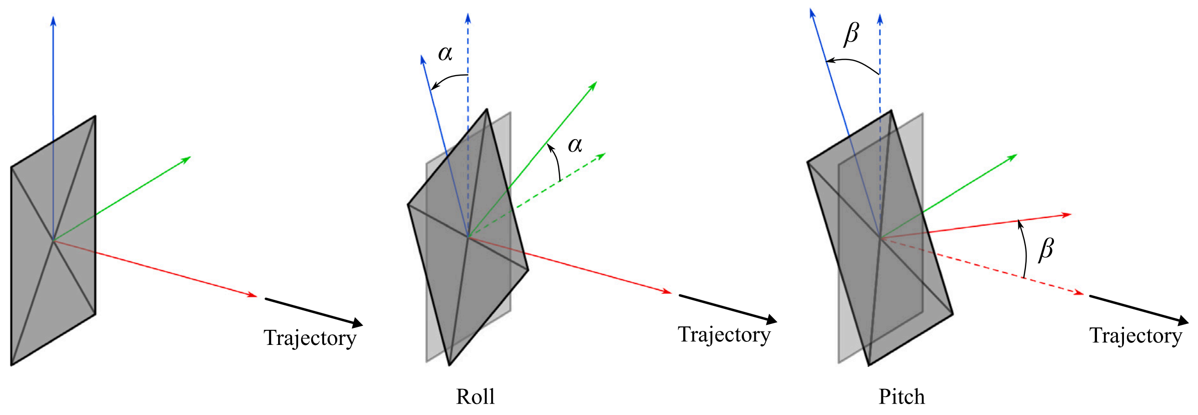


Fig. 12. Orientation of the debris element in the 3D space according to the debris trajectory.

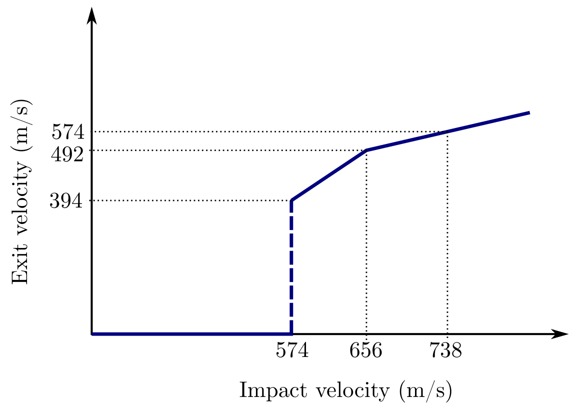


Fig. 13. Simplification of the ballistic penetration curve.

The `debris` object (2D rectangular shape) is defined perpendicularly to the impact line, generating its geometry and orientation according to its length (`a`), width (`b`), `roll` and `pitch` angles. Both the `line` object and the `velocity` become attributes of the `debris` object. By repeating the process for each debris, the list of debris objects is obtained.

3.3.5. *Definition of damage criteria (ballistic equation)*

At this stage, all the ballistic curves must be imported. To do so, the `damageCriteria` object is defined as an instance of the class `DamageCriteria`, which includes the ballistic penetration equations (`ballisticEq`) and the threshold velocity (`vThreshold`). This object will be used to determine whether or not the debris perforates the structural elements reached. A graphical view of a simple ballistic equation is presented in Fig. 13.

The characterization of ballistic penetration equations is a complex issue that depends simultaneously on many factors, such as the mass, size and material of the debris, as well as the characteristics of the aircraft surface hit (only skin, skin and frames, skin and stringers or a complex union). As mentioned in Section 1, a conservative simplification of this phenomenon can be adopted, assuming that the debris possesses infinite energy and all the material in its path is removed.

3.3.6. *Meshing of the debris element, generation of shotlines and intersection with the aircraft mesh*

The first step is to apply the method `getMeshFromDebris` to obtain the mesh of possible finite elements of the structure that the debris could hit in its path. To do so, a discretization of the debris fragment with an appropriate mesh size is needed, which is calculated proportionally to the size of the smallest finite element in the aircraft model. This is implemented by applying the method `generateDebrisMesh` to the `debris` object. After meshing the debris, the coordinates of the nodes of each finite element are calculated and lines parallel to the impact line are generated for each node. We will call these lines shotlines. This is implemented using the methods `getNodeCoordinates` and `translateLineToCoords`. Then, each shotline is intersected with the aircraft mesh and the set of finite elements to be removed from the aircraft mesh is obtained. The process is carried out using the method `getMeshFromGroupOfLines`. This idea is illustrated in Fig. 14.

To select the elements, a radial tolerance is calculated around each line according to the minimum element size previously calculated. To obtain the set of possible finite elements to eliminate, it is necessary to define the selection method (`selectionMethod`). Two possibilities have been defined: If all the nodes of the finite element are within the radial tolerance defined by the user (`AllNodes`) or if only one of the element is within the tolerance to be considered as a damaged element (`OneNode`). The concept is explained in detail in Fig. 15, where the meshes of the debris element and the structural model are presented. It can be seen how when using the `OneNode` option, the elements selected correspond to a larger affected area than with the `AllNodes` option, hence, being `OneNode` more conservative from the structural analysis point of view. However, it is important to mention that the choice of one method or the other is irrelevant as the mesh becomes finer because they will converge to the same selected elements.

Once we have the list of possible finite elements of the aircraft that the debris could strike, the method `getHoleMeshkeepingOriginalIds` is applied so that the IDs of the elements remain unchanged with respect to the IDs of the aircraft mesh.

3.3.7. *Clustering elements and sorting by proximity to the debris origin*

The next step consists in grouping those elements associated to the same impact. To do so, the DBSCAN clustering algorithm from the sklearn package [33] divides the list of elements into groups. The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Thus, using as input points the center of gravity of the mesh elements and a distance threshold equal to the maximum minimum-distance between any two points, the clusters of elements associated with each possible hole are identified. This is done with the method `separateHolesOnImpacts`, which in turn uses the methods `getElementsCoG`, `clusteringAlgorithm` and `getHoleMeshFromIds`.

After that, the clusters must be sorted by the distance to the debris origin in order to apply the `damageCriteria` previously defined.

3.3.8. *Application of damage criteria to define number of holes*

Finally, the ballistic curves are applied to determine which clusters must be removed from the original mesh. The debris element was released at an initial velocity and will reach the nearest cluster of

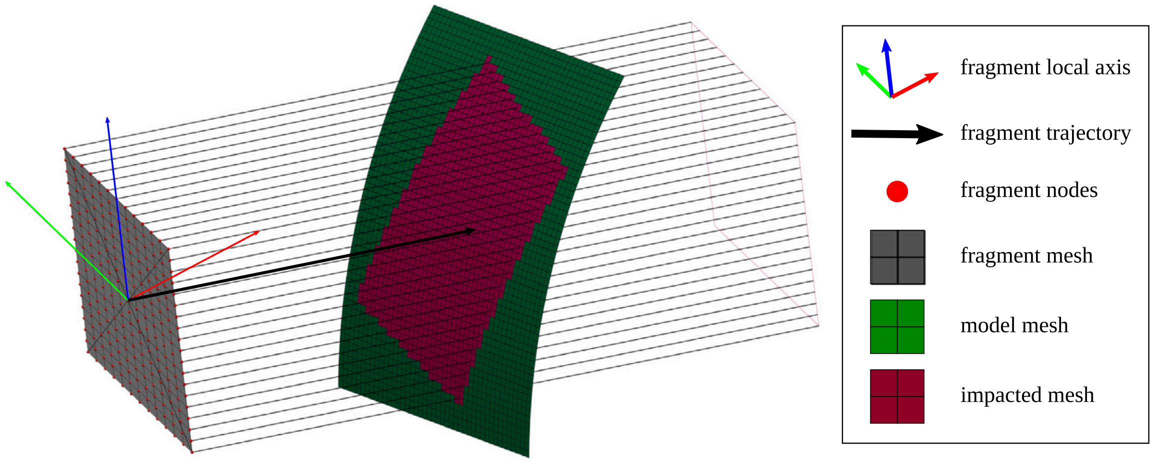


Fig. 14. Finite elements to be removed from the model.

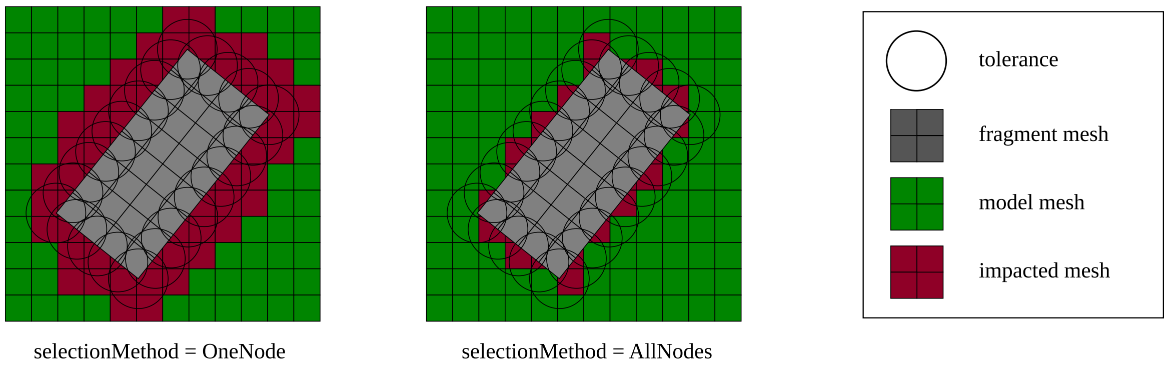


Fig. 15. Selection method that determines if an element must be selected to be removed from the aircraft mesh.

elements first. Using this velocity as input, we use the curve in Fig. 13 that returns the residual velocity as output. If the initial debris velocity is greater than the threshold velocity, the residual velocity is used as the new impact velocity for the second cluster reached, and so on. When the initial debris velocity, or any residual velocities is less than the threshold velocity, the advance of the debris ends and the remaining clusters will not be removed from the mesh. The clusters to be removed are stored in the `holesFromDebris` object.

3.3.9. *Remove mesh elements from the original mesh*

Once the `holesFromDebris` object is obtained for all the impact lines, the list `holesFromDebrisList` is obtained. Then, these meshes are removed by applying to the fuselage object the method `RemoveElements`. As a result, the `damagedConfiguration` object is generated.

4. Application example

This example corresponds to a narrow-body aircraft such as the Boeing 737 or Airbus 320. The FEM model was taken from the database of examples available in the Hypersizer software [34]. Modifications were made on this model to generate the complete barrel section, as well as a mesh refinement to obtain a model suitable for the DamageCreator application. The aircraft dimensions are summarized in the sketch presented in Fig. 16. The fuselage section is 23.80 m long, 3.75 m wide and 4 m high, and the wing span is 33.25 m. Attending to the model mesh, the frames, ribs, spars, keel beam and floor beams are defined as shell elements, while the stringers are defined as beam elements, resulting in 92,576 1D and 428,460 2D elements. The origin of the coordinate system simulates the location of the aircraft nose, 7.5 m from the forward section of the model, as shown in Fig. 17. It is assumed that an uncontained engine failure occurs and rotor fragments are released, striking the fuselage. The engine coordinates of the left wing engine correspond to [17, −5.75, −1.75] m, which is considered to be the debris origin. As a result, the rotating plane corresponds to a plane parallel to the YZ plane containing the debris origin.

As stated above, a ballistic curve is needed to characterize the residual velocity of each fragment, which requires a large number of complex FE simulations. Therefore, as indicated in the AC20.128 A [24], we adopt the conservative estimate that the debris possesses infinite energy and all the material in its path is removed.

The uncontained engine rotor failure presented in this example simulates a fan disk event in a High-Bypass Ratio turbofan engine, using the debris database defined in the UEDDAM report [25]. The fragment characterization table is summarized in Table 2. Thus, this case simulates a real accidental scenario in which debris of different sizes hit the fuselage, based on previous historical accidents. To calculate the debris dimensions related to the normalized sizes, blades 812.8 mm long and 203.2 mm wide were considered, as well as disks 655.32 mm in diameter and 170.18 mm thick. Combining this information with the fragment characterization, the discrete random distribution functions of normalized sizes for blades and disks can be obtained.

For each damaged configuration, the number of blades and disks impacts was randomly defined as N(μ=27, σ = 0.05μ) and N(μ=3, σ = 0.2μ), respectively. Then, using the discrete random distribution function in Table 2, blade and disk impacts were generated accordingly. It is important to note that a different scattering prism needs to be defined associated with each debris size, since they have different after and forward angle. Figs. 18, 19, 20 and 21 show four different damaged configurations generated with DamageCreator software. Two types of

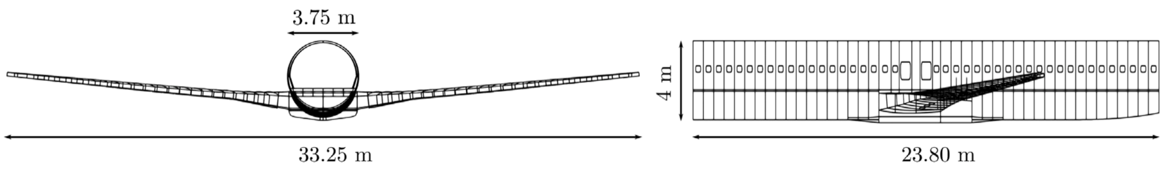


Fig. 16. Fuselage-wing dimensions.

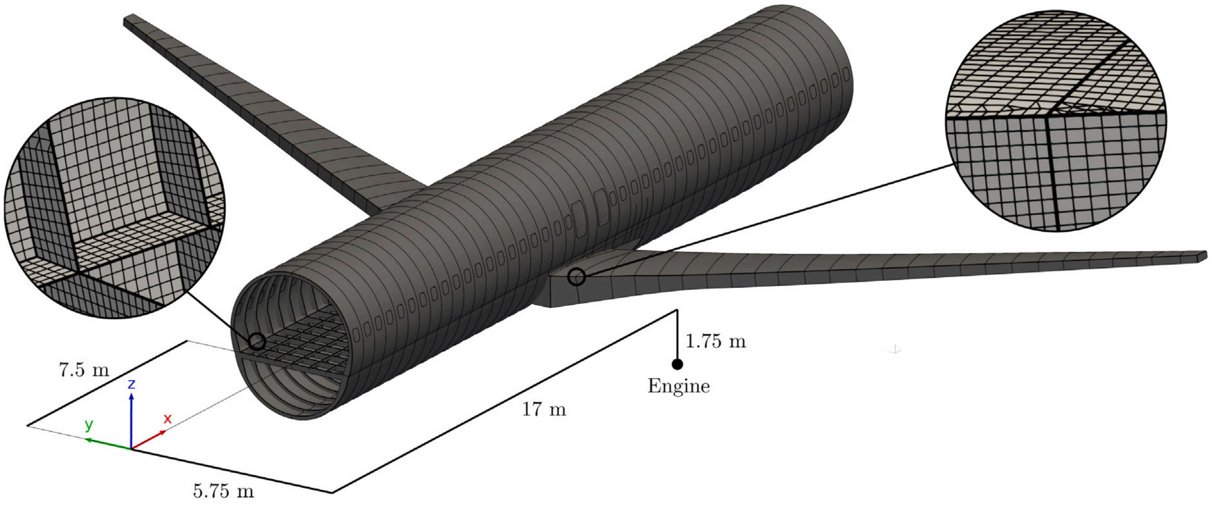


Fig. 17. General layout of the fuselage-wing model. Zoom views of the refined FE mesh.

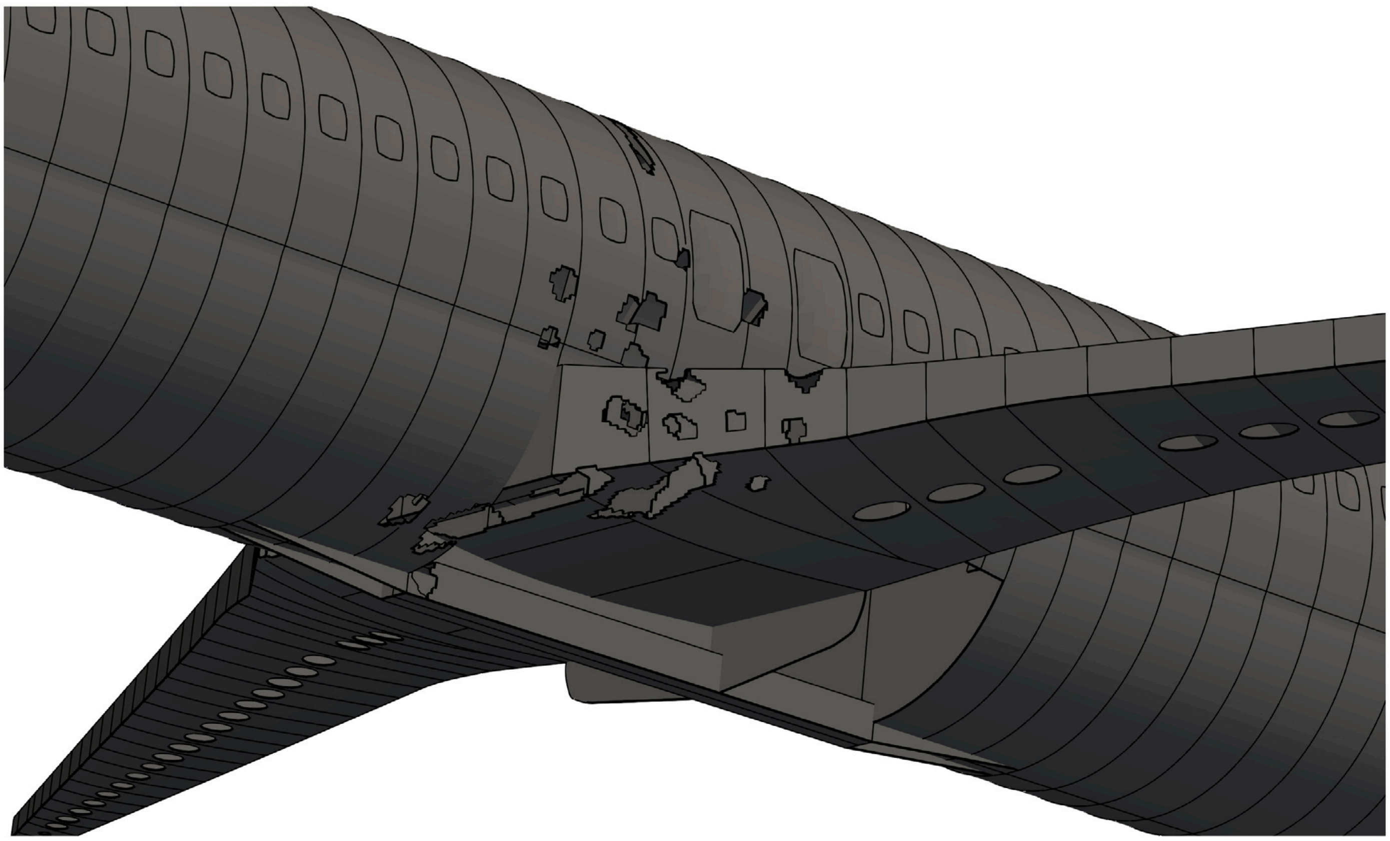
Table 2
Fragment characterization in a fan disk event for a High-Bypass Ratio turbofan engine.

Debris released	Number of fragments (Average/Event)	Normalized size	Velocity (ft/sec)	Spread angle (degrees)	Discrete random distribution a (mm)	b (mm)	Probability
					Length	Width	
Blades	27						
	10	10%	935	+10 to −30	81.28	81.28	10/27
	9	20%	928	+15 to −25	162.56	162.56	9/27
	2	30%	894	+10 to −25	243.84	203.20	2/27
	4	50%	822	+10 to −20	406.40	203.20	4/27
	1	70%	796	+10 to −20	568.96	203.20	1/27
	1	100%	644	+15 to +5	812.80	203.20	1/27
					Length	Thickness	
Disks	3	100%	303	+2 to −3	655.32	170.18	1

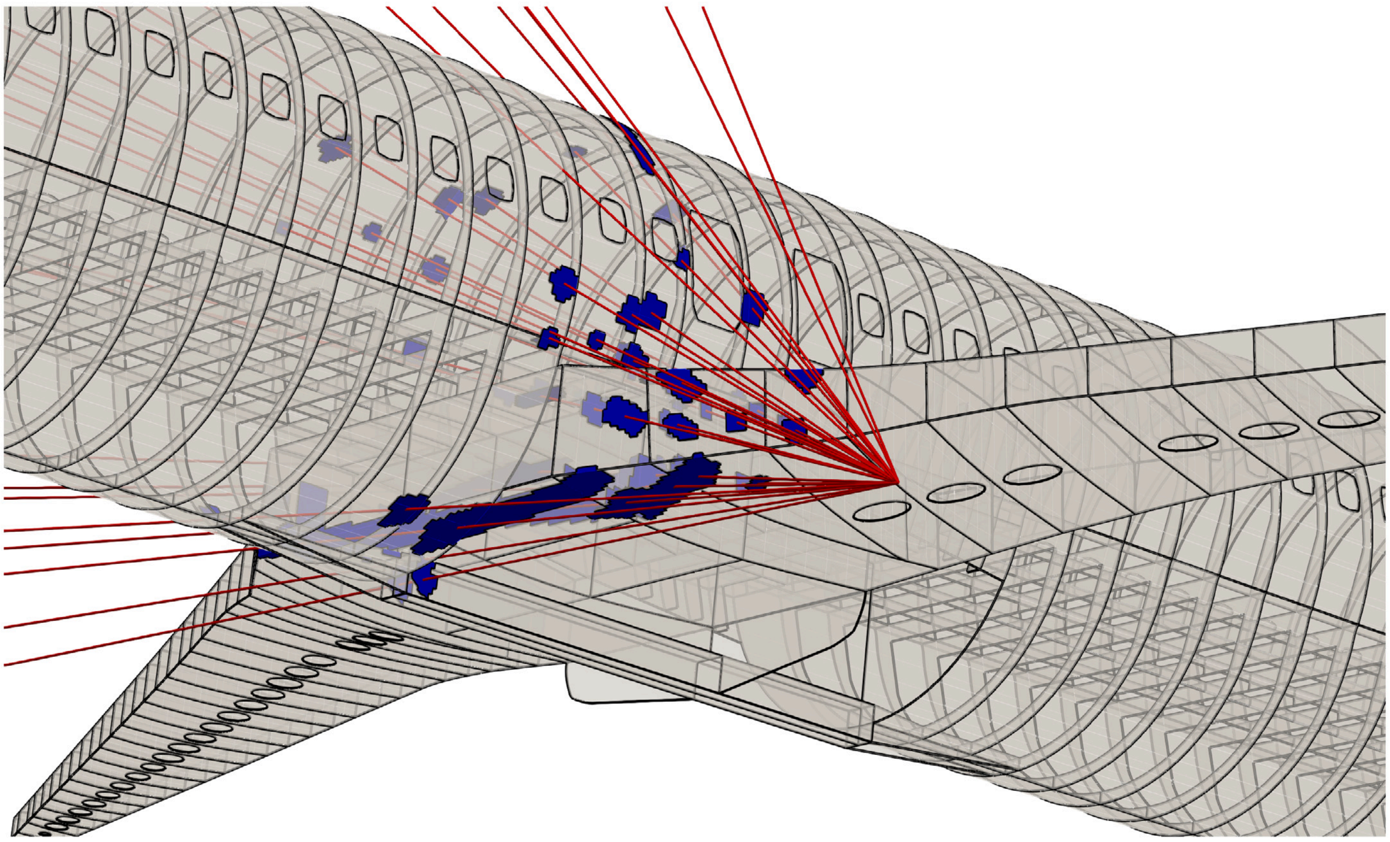
figures are included for each configuration, one of them representing the damaged mesh and the other one highlighting the areas where material is removed as well as debris trajectories. The intrinsic random nature associated with the uncontained rotor failure event is readily appreciated by the different variety of debris trajectories and hole sizes located at diverse areas of the aircraft structure. As can be observed, both skin and stiffeners from the central fuselage and wing root are damaged.

5. Conclusions

This paper presents an open-source framework (DamageCreator) to simulate realistic local damage to aircraft structure due to an engine failure event. Several damaged configurations representing a fan disk event were generated in the application example. By analyzing the set of damaged models obtained, it is possible to observe the complex patterns that emerge in the case of study. To address the randomness of the event, the tool incorporates statistical information on historical databases from real accidents to characterize the location and size of holes in the wing and fuselage. As the infinite energy criterion specified in the regulations is adopted, all material that intersects the debris path is eliminated. This is a more accurate approach than the industry practice of removing entire fuselage panels and, in turn, more conservative than calculating and applying ballistic curves to determine the material to be removed. A major advantage is that manual generation is avoided,



(a) Damaged mesh



(b) Holes and debris trajectories

Fig. 18. Damaged configuration 1.

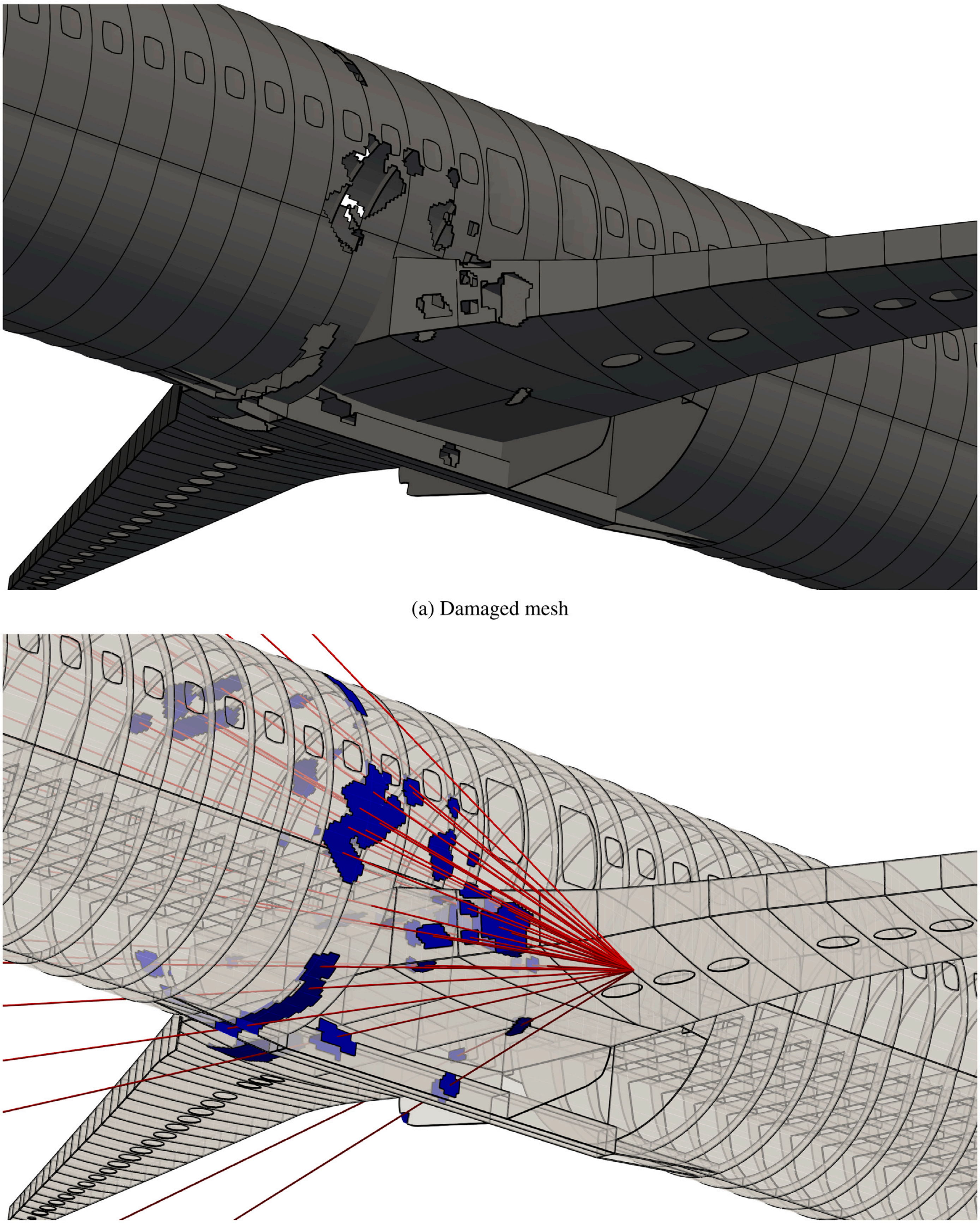
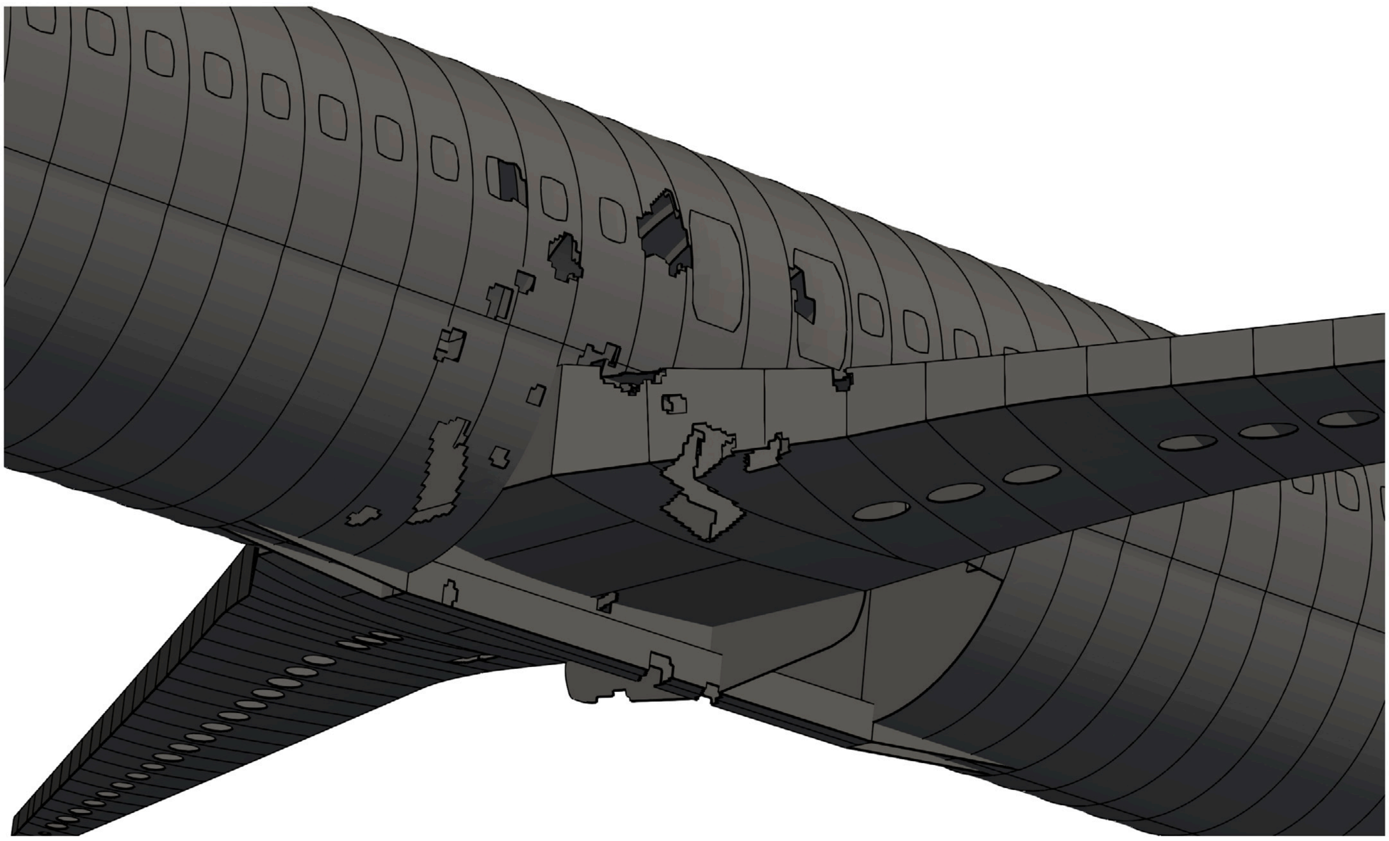


Fig. 19. Damaged configuration 2.


(a) Damaged mesh

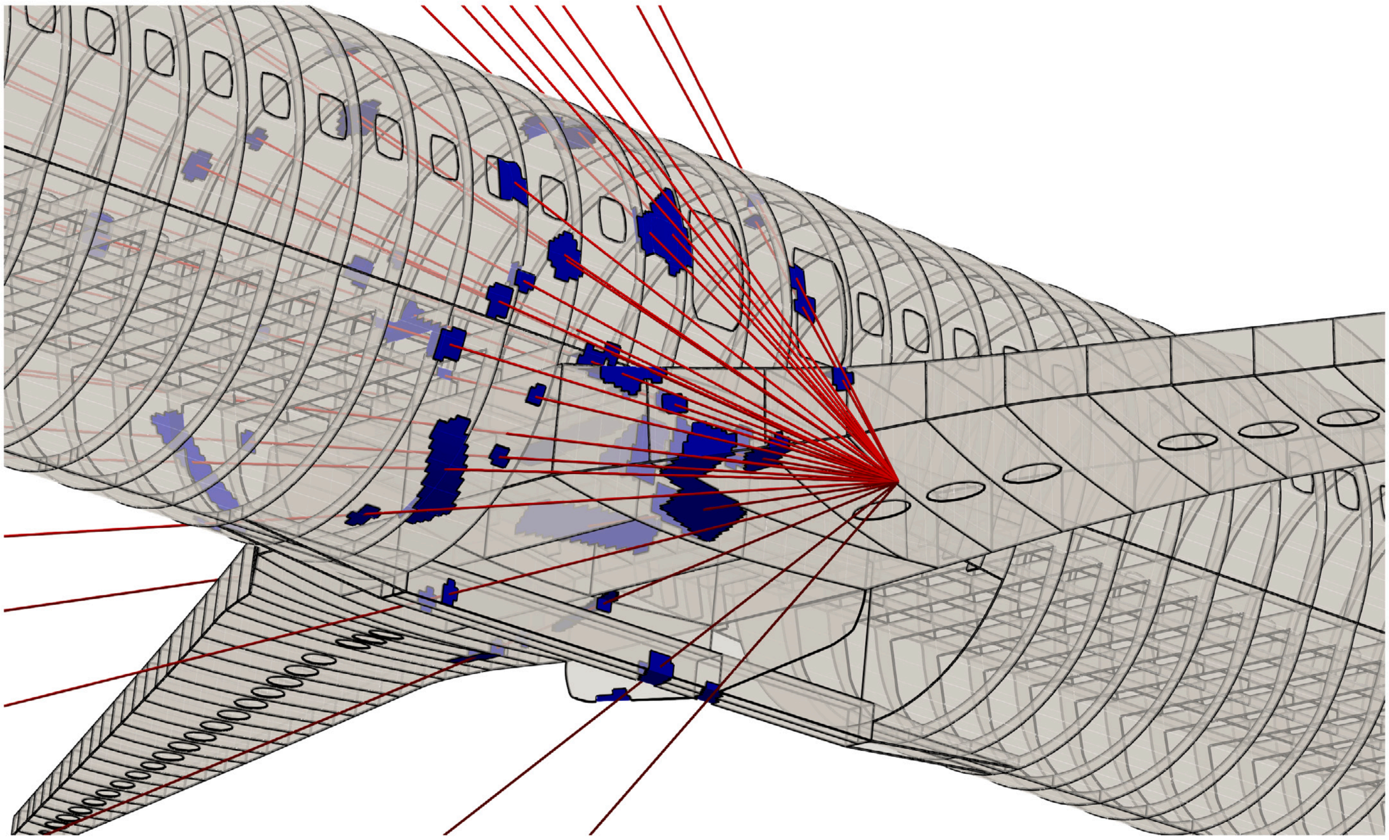
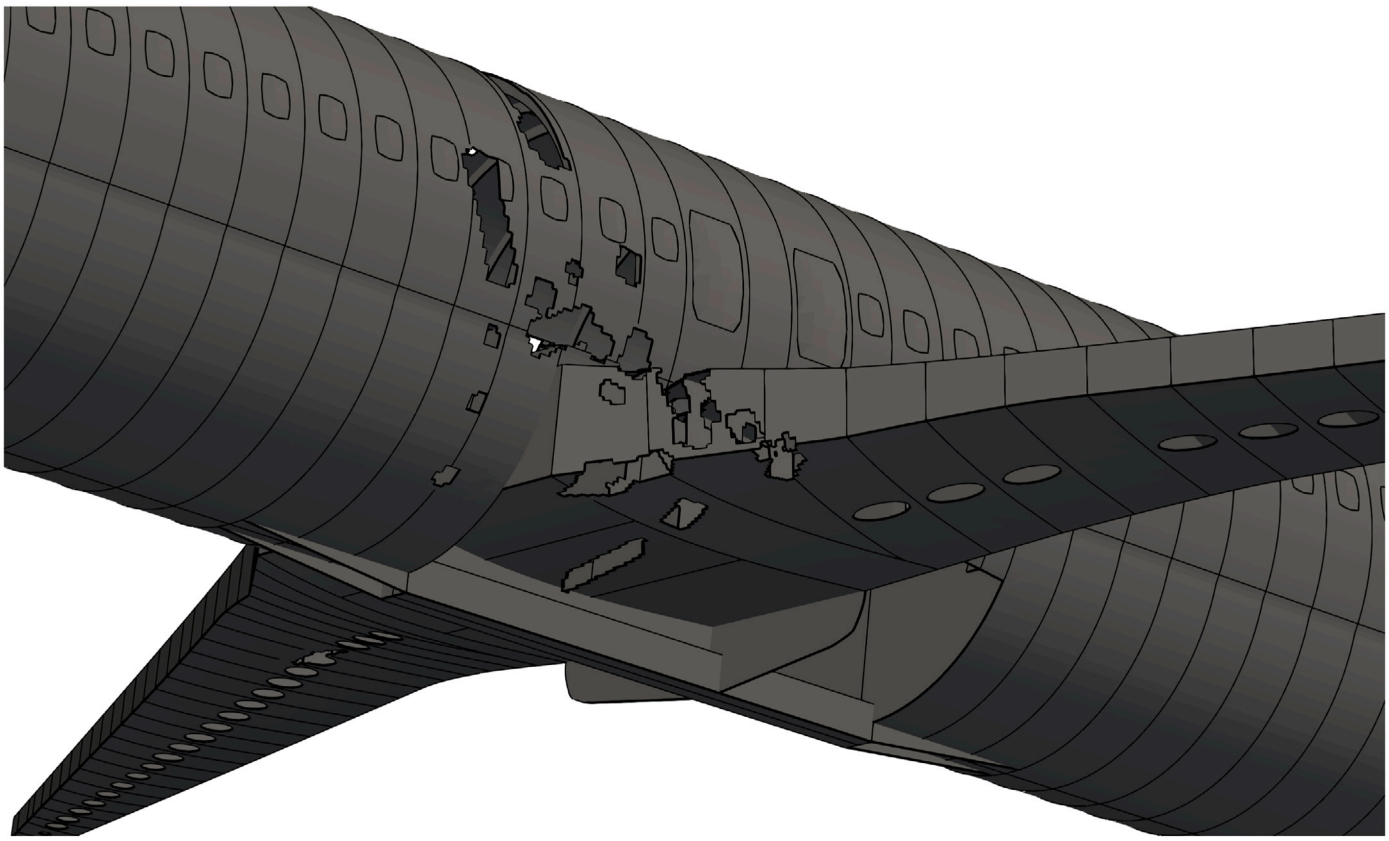

(b) Holes and debris trajectories

Fig. 20. Damaged configuration 3.


(a) Damaged mesh

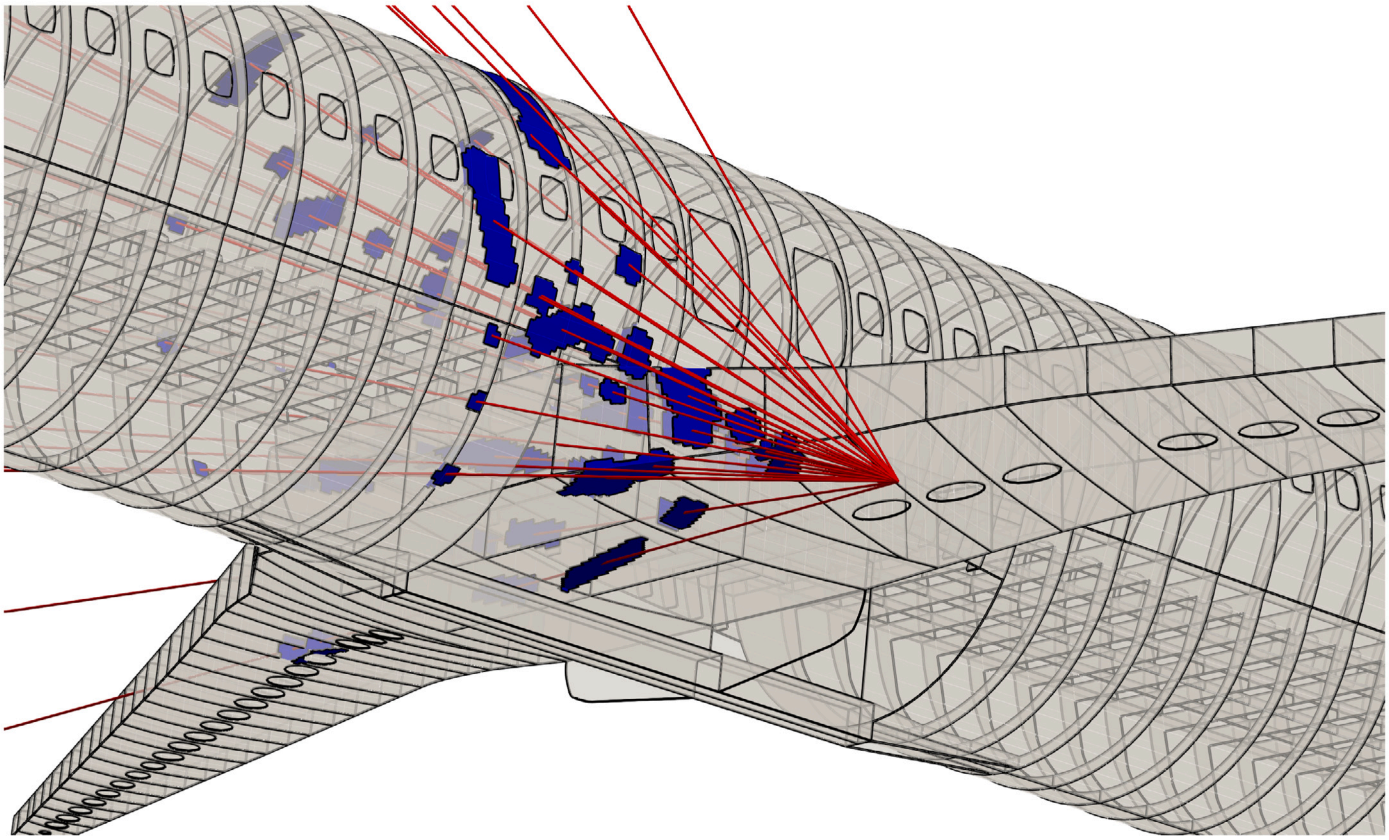

(b) Holes and debris trajectories

Fig. 21. Damaged configuration 4.

which would be extremely cumbersome, too conservative in assuming a larger-than-necessary damage size, and imprecise due to the difficulties in predicting the location and shape of holes in the fuselage.

This is the first time that damage caused by uncontained debris from engine failure is modeled in complex finite element models of assembled aircraft structures, generating a comprehensive set of damaged meshes. As a result, the application of this tool in the aircraft design process would improve the approaches based on fail-safe optimization, leading to lighter structures.

Future work involves incorporating ballistic curves into the software by performing explicit impact simulations, to determine the holes produced by each debris. A possible approach would be to build a physics-informed machine learning model using a sufficiently large set of simulations, in order to predict damaged areas more accurately. Additionally, another impact sources, such as an Uncontained APU Rotor Failure (UARF), bird strikes or drone collisions, could be easily implemented due to the modular software architecture.

CRediT authorship contribution statement

Clara Cid: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing, Visualization. **Aitor Baldomir:** Conceptualization, Methodology, Supervision, Writing – review & editing. **Miguel Rodríguez-Segade:** Conceptualization, Methodology, Software, Writing – review & editing, Visualization. **Santiago Hernández:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research leading to these results has been conducted under Grant PID2019-108307RB-I00 funded by MCIN/AEI/10.13039/50110 0011033. The authors also acknowledge funding received from the Galician Government through research grant ED431C 2017/72. The first author also acknowledges the sponsorship of the Galician Government through the grant "axudas de apoio á etapa predoutoral cofinanciadas parcialmente polo programa operativo FSE Galicia 2014-2020" under identification number ED481A-2018/193. Funding for open access charge: Universidade da Coruña/CISUG.

References

[1] J.S. Arora, D.F. Haskell, A.K. Govil, Optimal design of large structures for damage tolerance, AIAA J. 18 (5) (1980) 563–570, http://dx.doi.org/10.2514/3.7669.

[2] Y.Y.S.F. Feng, F. Moses, Optimum design, redundancy and reliability of structural systems, Comput. Struct. 24 (2) (1986) 239–251, http://dx.doi.org/10.1016/0045-7949(86)90283-X.

[3] K.S. Marhadi, S. Venkataraman, S.A. Wong, Load redistribution mechanism in damage tolerant and redundant truss structure, Struct. Multidiscip. Optim. 44 (2) (2011) 213–233, http://dx.doi.org/10.1007/s00158-011-0623-1.

[4] M. Stolpe, Fail-safe truss topology optimization, Struct. Multidisc. Optim. 60 (2019) 1605–1618, http://dx.doi.org/10.1007/s00158-019-02295-7.

[5] S. Dou, M. Stolpe, On stress-constrained fail-safe structural optimization considering partial damage, Struct. Multidiscip. Optim. 63 (2) (2021) 929–933, http://dx.doi.org/10.1007/s00158-020-02782-2.

[6] S. Dou, M. Stolpe, Fail-safe optimization of tubular frame structures under stress and eigenfrequency requirements, Comput. Struct. 258 (2022) 106684, http://dx.doi.org/10.1016/j.compstruc.2021.106684.

[7] M. Jansen, G. Lombaert, M. Schevenels, O. Sigmund, Topology optimization of fail-safe structures using a simplified local damage model, Struct. Multidisc. Optim. 49 (4) (2014) 657–666, http://dx.doi.org/10.1007/s00158-013-1001-y.

[8] M. Zhou, R. Fleury, Fail-safe topology optimization, Struct. Multidisc. Optim. 54 (2016) 1225–1243, http://dx.doi.org/10.1007/s00158-016-1507-1.

[9] O. Ambrozkiewicz, B. Kriegesmann, Density-based shape optimization for fail-safe design, J. Comput. Des. Eng. 7 (2020) 615–629, http://dx.doi.org/10.1093/jcde/qwaa044.

[10] H. Wang, J. Liu, G. Wen, Y.M. Xie, The robust fail-safe topological designs based on the von mises stress, Finite Elem. Anal. Des. 171 (2020) 103376, http://dx.doi.org/10.1016/j.finel.2019.103376.

[11] J. Martínez-Frutos, R. Ortigosa, Risk-averse approach for topology optimization of fail-safe structures using the level-set method., Comput. Mech. (2021) http://dx.doi.org/10.1007/s00466-021-02058-6.

[12] J. Martínez-Frutos, R. Ortigosa, Robust topology optimization of continuum structures under uncertain partial collapses, Comput. Struct. 257 (2021) 106677, http://dx.doi.org/10.1016/j.compstruc.2021.106677.

[13] A. Baldomir, S. Hernández, L. Romera, J. Díaz, Size optimization of shell structures considering several incomplete configurations, in: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Honolulu, Hawaii, 2012, http://dx.doi.org/10.2514/6.2012-1752.

[14] C. Cid, A. Baldomir, S. Hernández, L. Romera, Reliability based design optimization of structures considering several incomplete configurations, in: 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2016, http://dx.doi.org/10.2514/6.2016-4290.

[15] C. Cid, A. Baldomir, S. Hernandez, L. Romera, Multi-model reliability-based design optimization of structures considering the intact configuration and several partial collapses, Struct. Multidiscip. Optim. 57 (3) (2018) 977–994, http://dx.doi.org/10.1007/s00158-017-1789-y.

[16] C. Cid, A. Baldomir, S. Hernández, Probability-damage approach for fail-safe design optimization (PDFSO), Struct. Multidiscip. Optim. 62 (6) (2020) 3149–3163, http://dx.doi.org/10.1007/s00158-020-02660-x.

[17] C. Cid, A. Baldomir, S. Hernández, Probability-damage approach for fail-safe design optimization under aleatory uncertainty (β-PDFSO), in: AIAA SciTech 2021 Forum, 2021, http://dx.doi.org/10.2514/6.2021-1480.

[18] C. Cid, A. Baldomir, S. Hernández, Reliability index based strategy for the probability damage approach in fail safe design optimization (β-PDFSO), Eng. Comput. (2022) http://dx.doi.org/10.1007/s00366-022-01611-y.

[19] Federal Aviation Administration, FAR Final Rule, Federal Register, 1978, (Volume 43, Number 194), 14 CFR Part 25, (Docket No. 16280; Amendment No. 25–45).

[20] Federal Aviation Administration, FAR Final Rule, Federal Register, 2008, 14 CFR Part 25, (Docket No. 2007-27310; Amendment No. 25–126).

[21] Federal Aviation Administration, FAR Final Rule, Federal Register, 1970, (Volume 35, Number 68), 14 CFR Part 25, (Docket No. 9079; Amendment No. 25–23).

[22] Federal Aviation Administration, Damage Tolerance and Fatigue Evaluation of Structure , AC 25.571-1D, U.S. Department of Transportation, 2011.

[23] Federal Aviation Administration, Minimizing the Hazards from Propeller Blade and Hub Failures , AC 25-905-1, U.S. Department of Transportation, 2000.

[24] Federal Aviation Administration, Design Considerations for Minimizing Hazards caused by Uncontained Turbine Engine and Auxiliary Power Unit Rotor Failure , AC 20-128A, U.S. Department of Transportation, 1997.

[25] Federal Aviation Administration, Uncontained Engine Debris Analysis Using the Uncontained Engine Debris Damage Assessment Model, AR-04/16, U.S. Department of Transportation, 2004.

[26] C. Cid, A. Baldomir, M. Rodríguez-Segade, S. Hernández, DamageCreator: A global tool for generating finite element models of damaged aircraft due to blade release or uncontained engine rotor failure, in: AIAA Aviation 2021 Forum, 2021, http://dx.doi.org/10.2514/6.2021-2436.

[27] Federal Aviation Administration, Aircraft Catastrophic Failure Prevention Research Program Plan, CT-94/26, U.S. Department of Transportation, 1994.

[28] Federal Aviation Administration, Committee on Uncontained Turbine Engine Rotor Events Data Period 1976 through 1983, 1991, FAA/SAE, Aerospace Information Report, Report No. AIR4003.

[29] Federal Aviation Administration, Committee on Uncontained Turbine Engine Rotor Events Data Period 1984 through 1989, 1994, FAA/SAE, Aerospace Information Report, Report No. AIR4770.

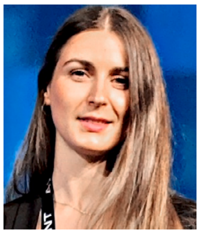
[30] Federal Aviation Administration, Large Engine Uncontained Debris Analysis, AR-99/11, U.S. Department of Transportation, 1999.

[31] Federal Aviation Administration, Small Engine Uncontained Debris Analysis, AR-99/7, U.S. Department of Transportation, 1999.

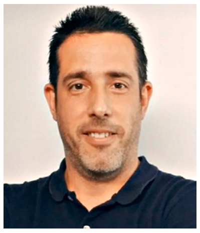
[32] J. López-Puente, D. Varas, J. Loya, R. Zaera, Analytical modelling of high velocity impacts of cylindrical projectiles on carbon/epoxy laminates, Composites A 40 (8) (2009) 1223–1230, http://dx.doi.org/10.1016/j.compositesa.2009.05.008.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (Oct) (2011) 2825–2830.

[34] Collier Research Corporation, Hypersizer 8.0 example in documentation: Data/training/FEA/fuse-wing, 2020, URL https://hypersizer.com.



Clara Cid completed her Ph.D. in Aerospace and Civil Engineering at the University of A Coruña (Spain) in January 2022. Upon graduation, she continued working on the Structural Mechanics Group (GME) as a posdoctoral researcher. She won the 2021 SwRI Student Paper Award in NonDeterministic Approaches and the 2022 Lockheed Martin Student Paper Award in Structures, in the AIAA SciTech Forum and Exposition. Cid's research interests include fail-safe size optimization and optimization under aleatory and epistemic uncertainty.



Aitor Baldomir is an Associate Professor of Structural Mechanics at the University of A Coruña since 2008. He has collaborated with Airbus in structural analysis and optimization of aeronautical structures in the A380, A350, A30x and A320neo programs. He also collaborated in two European research projects: MAAXIMUS (Seventh Framework Programme) and STRATOFLY (Horizon 2020). He is an AIAA member since 2011. Currently, his research lines are fail-safe size optimization and optimization under aleatory and epistemic uncertainty.

Miguel Rodríguez-Segade is a Ph.D. student in Aerospace and Civil Engineering at the University of A Coruña. His research interests includes hypersonic vehicles and optimization. He collaborated in the European research project STRATOFLY (Horizon 2020).

Santiago Hernández is an Emeritus Professor of Structural Mechanics. Professor Hernández founded the Structural Mechanics Group (GME) at the University of A Coruña. He received the Distinguished Service Award as a member of the AIAA Multidisciplinary Design Optimization Technical Committee (2013). He is an Associate Fellow of AIAA (2009) Fellow of ASCE (2007), IABSE (2016), and Chief Academic Officer at WIT (2018). He has participated in a large number of research projects at national and international level. He has almost three hundred scientific publications, has authored several books and edited books written by leading scientists. He is well known for his expertise in aeroelasticity and for the application of numerical optimization techniques to engineering design.