Elementary Implementation of VTK files in C
A FEMDEM Application

JOHN-PAUL LATHAM*†, ADO FARSI*†, and MICHAEL TRAPP*

A recent state-of the arts review by EU decision makers revealed that current building guidelines are not taking into account threats such as natural disasters and acts of terrorism. Updating these guidelines requires accurate numerical models to quantify the resilience of building elements against explosive and projectile impact.

The Applied Modelling and Computation Group AMCG at Imperial College London has recently developed a novel coupled dynamic gas/solid FEMDEM code.

In this paper, this Y code is applied to simulate 2D projectile impact on laminated glass. Obsolete external VTK libraries were required to compile the code. A novel implementation is presented to prepare and write VTK output files without the need of these dependencies.

The simulation results are not realistic, indicating errors in the pre-processing software or in the underlying code. The implementation of VTK files is successful and valid output files are produced. The files are visualisable using Paraview. The results are useful for all further applications of this code.

CCS Concepts: • **Applied computing** → *Engineering*.

*Imperial College London.
†Supervisor

Authors' address: John-Paul Latham, j.p.latham@imperial.ac.uk; Ado Farsi, ado.farsi@imperial.ac.uk; Michael Trapp, mt5918@ic.ac.uk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
XXXX-XXXX/2019/1-ART1 $15.00
https://doi.org/123456789

Additional Key Words and Phrases: FEMDEM, VTK, Legacy, XML, base64, encoding

ACM Reference Format:
John-Paul Latham, Ado Farsi, and Michael Trapp. 2019. Elementary Implementation of VTK files in C: A FEMDEM Application. *ACM Trans. Model. Comput. Simul.* 1, 1, Article 1 (January 2019), 25 pages. https://doi.org/123456789

1 PHYSICS THEORY

1.1 Laminated Glass Model

Laminated glass is a sandwich structure of two brittle glass plies and an adhered polymer inter-layer (or inter-face) in between. The bonding between the glass and the inter-layer is without physical adhesive [1]. Secondary laminated glass consists of two such laminates, separated by a layer of air.

Advantageous properties of laminated glass include a relatively high penetration resistance [2], low weight [3] and the adherence of fractured glass fragments to the structure to reduce the risk of injuries[2, 4, 5, 6]. Breakage of the inner ply significantly reduces strength and facilitates a full collapse of the glass [5]. An optional back layer (usually poly-carbonate (PC) [7, 8]) improves structural stability and additional energy absorption [7, 9].

The prediction of crack initiation and propagation poses a significant challenge and requires ongoing research effort. Local stress intensification is caused by pre-existing micro-structural material flaws[1] such as micro-cracks and voids [10].

[1]inhomogeneities, discontinuities

Many fracture models have been proposed. The Y code uses a local combined single and smeared crack model approach [11, 12], in which a single crack is replaced by a blunt crack band [13].

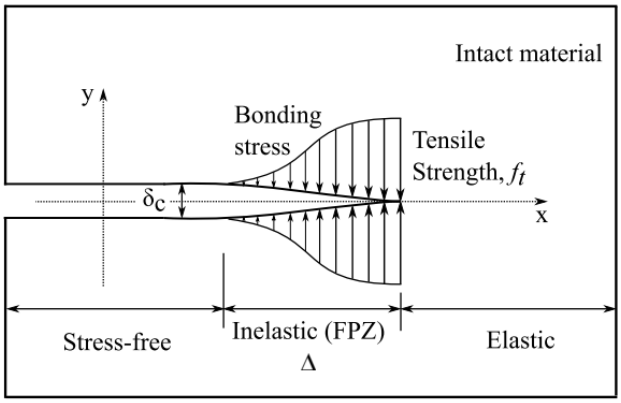

Fig. 1: Single Crack (Dugdale) Model. [14]

The single crack model in Fig. 1 assumes a crack tip located in a fracture process zone (FPZ). Plastic bonding stress $\sigma \leq f_t$ causes the crack tip to open up, up to a critical separation length $\delta = \delta_c$ [14].

To determine the bonding stress, the Y code adopts the Mohr-Coulomb constitutive model [12], described by

$$\tau = c + \sigma \tan\phi\,, \tag{1}$$

with shear stress τ, normal stress σ, internal cohesion c and internal friction angle ϕ. The internal friction coefficient is given by

$$\eta = \tan\phi \tag{2}$$

The stresses σ and τ correspond to normal and shear displacements δ_n and δ_s and tensile and shear strengths f_t and f_s. These material strengths are defined as the maximum strength in the stress-displacement diagram (Fig. 2), with maximum elastic displacement δ_p and critical displacement δ_s.

The fracturing in the strain-softening part is modelled using joint elements which are generated between two neighbouring shell elements. Cracks coincide with the element edges [14].

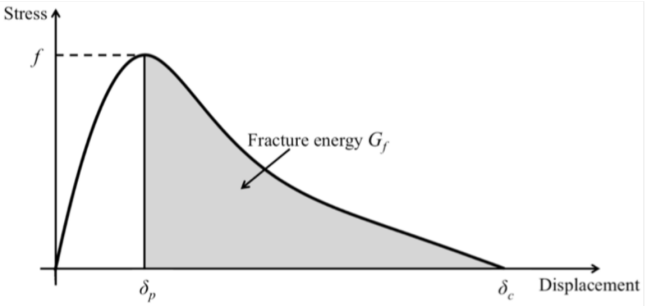

Fig. 2: Stress-displacement curve according to the single and smeared crack model. [12]

1.2 Inter-layer Model

The task of the inter-layer is the absorption of impact energy and the maintenance of adhesion to the plies [3]. The inter-layer consists of one or more sheets. Common inter-layer materials include polymers such as polyvinyl butyral (PVB), thermoplastic polyurethane (TPU), and most recently SentryGlas®Plus (SGP) [15, 16].

Mechanical properties of the inter-layer are dependent on the fracture state of the laminated glass [17]. Prior to fracture, straining of the inter-layer is limited, permitting the application of linear visco-elastic laws. On failure of the glass, the inter-layer is subject to large strains and linear visco-elasticity is no longer applicable. Instead, the inter-layer is modelled as a hyper-elastic material [18, 19]. Work done by stresses onto such materials only depends on the reference state X and the current state x, but not on the load path (Fig. 3). Deformation from X to x is described by a deformation gradient [20]

$$F = \frac{\mathrm{d}\varphi}{\mathrm{d}X} \tag{3}$$

with mapping function φ mapping from X to x.

Hyper-elastics are mathematically described by a characteristic strain energy density function W. One of the simplest hyper-elastic models is the Neo-Hookean model [18] whose characteristic function is given by

$$W = \frac{\mu_0}{2}(I_1 - 3) - \mu_0 \ln J + \frac{\lambda_0}{2}\ln^2 J\,, \tag{4}$$

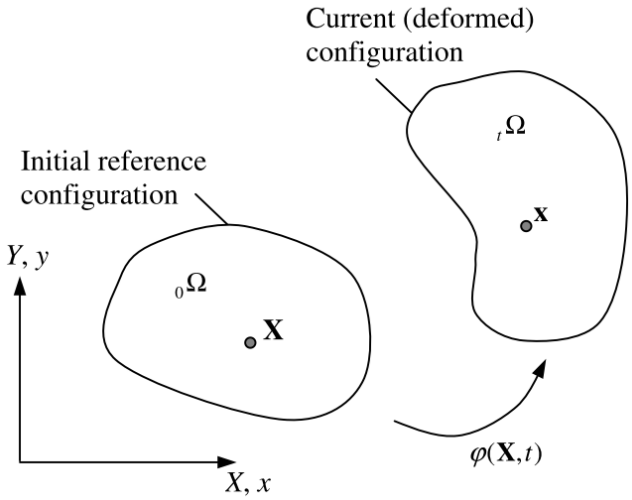

Fig. 3: Deformation from reference to current state. [20]

with Lamé constants λ_0 and μ_0 from the linearised theory, $J = |F|$ and first invariant $I_1 = C_{\mathrm{II}}$ with right Cauchy stress tensor [18]

$$C_{\mathrm{ij}} = F_{\mathrm{Ii}}\, F_{\mathrm{Ij}}\,. \tag{5}$$

Upper case indices refer to the reference configuration, while lower case indices refer to the current configuration.

Another common, simple hyper-elastic model is the `Mooney - Rivlin` model [21, 22]. The characteristic strain energy function for the compressible 2-Parameter `Mooney - Rivlin` model [22] is given by

$$W_2 = C_{10}\left(\bar{I}_1 - 1\right) + C_{01}\left(\bar{I}_2 - 1\right) + \frac{1}{d}\left(J - 1\right), \tag{6}$$

where C_{10} and C_{01} are adjustable parameters, $d = 2\,/\,K$ with bulk modulus K and $\bar{I}_1 = J^{-\frac{1}{3}}\, I_1$ and $\bar{I}_2 = J^{-\frac{1}{3}}\, I_2$ are deviatoric invariants [21]. The second invariant is given by

$$I_2 = \frac{1}{2}\left(C_{JJ}^2 - C_{IK}C_{KI}\right). \tag{7}$$

Other hyper-elastic models [21] include a more general polynomial model, `Arruda-Boyce`, `Ogden` and `Yeoh`. For the polynomial model, customized coefficients [1] already exist in the literature, specifically for laminated glass.

Modelling the occurence of fracture needs to be permitted for the inter-layer as well, as fracturing of the inter-layer is possible. This consideration necessitates the extension of the smeared single and combined fracture model to the inter-layer.

2 NUMERICAL MODELLING THEORY

2.1 FEMDEM Model

The combined finite-discrete element method, `FEMDEM` or `FDEM`[2, 11, 13, 16, 23, 24, 25, 26, 27, 28], contains discrete element that interact with neighbouring discrete elements. In addition, each discrete element is discretised into finite elements. Each finite element mesh captures the deformation of the corresponding discrete element.

Fracture and fragmentation is modelled as a transition from continua to discontinua. It is also in principle possible to imagine an inverse process of particles merging together.[13]

Based on the `3D FDEM` code `Y3D` by Munjiza et al [11, 13, 23], Xiang et al [29] added many detailed improvements and features to the code. The new model, `Solidity`, is deemed capable of creating realistic coupled multi-physics simulations. In contrast to conventional models, Solidity is not reliant on element deformability restriction constraints (due to the locking problem) and uses simpler triangular, quadratic and tetrahedral elements [12].

2.2 Governing Equations

The governing equations for the finite element calculations in the `FEMDEM` method are the equations of motion. The equations are given by

$$M\ddot{x} + \mu\dot{x} + f_{\mathrm{int}} = f_{\mathrm{ext}} = f_{\mathrm{l}} + f_{\mathrm{b}} + f_{\mathrm{c}}\,, \tag{8}$$

with lumped nodal mass matrix M, nodal displacements x, viscosity μ, internal nodal forces f_{int} and external nodal forces f_{ext}. External forces contain consist of external loads f_{l}, bonding forces f_{b} and contact forces f_{c}. Internal forces f_{int} are generated by element deformation. `FEMDEM` systems solve these equations via explicit time integration using the forward `Euler` method [30].

2.3 Contact Detection

The combination of discrete and finite elements is established via an interaction algorithm [30]. This algorithm consists of contact detection [31] and contact interaction [25]. Contact detection is carried out using search algorithms.

The contact detection algorithm detects couples of discrete elements close to each other by eliminating those that are too far from each other to be in contact. In other words, contact detection avoids processing contact interaction when there is no contact. This reduces `CPU` requirements and processing run time [13].

Contact interaction is thus only considered for discrete elements which are within a buffer size b of each other, given by

$$b = 0.1\,\Delta_{\text{min}}\,, \tag{9}$$

where Δ_{min} is the minimum edge[2].

2.4 Contact Interaction

Penetration of a contractor[3] body into a target[4] body is implemented in the `Y` code by the `potential contact force method`, see Fig. 4 [13]. Assumption of a conservative force field enables a description of the contact forces using potentials,

$$f_{\text{c}}^{\text{2D}} = -f_{\text{t}}^{\text{2D}} = \oint_{\Gamma_{\beta\text{t}\cap\beta\text{c}}} n_\Gamma\,(\varphi_{\text{c}} - \varphi_{\text{t}})\;\text{d}\Gamma\,, \tag{10}$$

$$f_{\text{c}}^{\text{3D}} = -f_{\text{t}}^{\text{3D}} = \int_{S_{\beta\text{t}\cap\beta\text{c}}} n_S\,(\varphi_{\text{c}} - \varphi_{\text{t}})\;\text{d}S\,, \tag{11}$$

with force potentials φ and outward unit normal n to the penetration boundary Γ or surface area S.

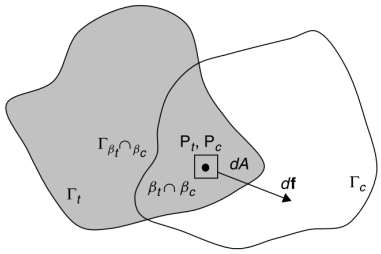

Fig. 4: Infinitesimal overlap of contractor and target body. [13]

The penetration distance d for each case is given by [13]

$$d = \frac{\sigma\, h}{p}\,, \tag{12}$$

with applied pressure σ, element height h and penalty factor p.

The penalty factor p is given by

$$p = \alpha\, E\,, \tag{13}$$

with Young's Modulus E and a constant α (in this paper: $\alpha = 10$) to regulate penetration.

Contact damping[5] due to plastic deformation, breakage of surface asperities, etc. is given by [13]

$$\sigma_{\text{C}} = 4\,\xi\frac{\sqrt{p/\rho}}{h}\dot{d}\,, \tag{14}$$

with damping ratio $0 \leq \xi^{6} \leq 1$ and finite element density ρ.

[2] minimum element side length, minimum size
[3] index c
[4] index t
[5] Rayleigh damping, viscous damping
[6] describes energy dissipation

The mass damping coefficient is given by

$$\zeta = \sqrt{E\,\rho} \tag{15}$$

3 NUMERICAL MODEL APPROACH

3.1 Workflow

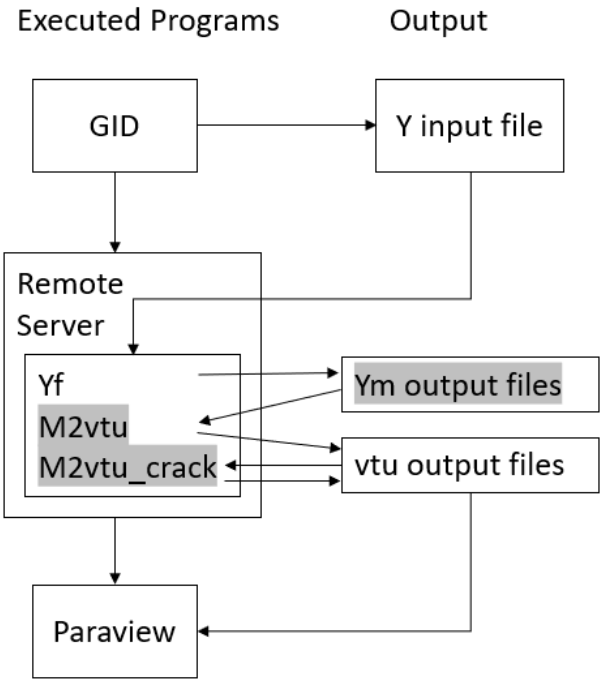

Fig. 5: Workflow of generating a simulation

The applied workflow to generate simulations is illustrated in Fig. 5. The grey shaded items are ready to be removed using the results from this paper such that the `Yf` code generates the `VTK` output files directly.

The required operating system presumably is `Ubuntu 14.4`, although this could not be confirmed using `Travis CI` automatic testing (url: https://travis-ci.org/). Therefore, the exact requirements are not known to the author at this point. The workflow was tested on `Ubuntu 18.04.2 LTS` subsystem for `windows 10`.

Input `Y` files are generated using the open source pre-processor `CAD` software `GiD` [32]. The software was tested on `Windows 10`. It is possible to import the geometry to `GiD` from other `CAD` software, such as `AutoCAD`, although this approach is suspected to potentially cause errors.

With the input `Y` file as an argument, the program is compiled using three binary executables `Yf`, `m2vtk` and `m2vtk_crack`. The executables are generated using a make file, along with the `Y2D` code files, written in the programming language `C`. The code may be obtained from the `AMCG` at Imperial College London.

Code execution requires obsolete `VTK 5.8` libraries. Tests to produce simulations were thus conducted exclusively on the high performance computing (`HPC`) system at Imperial College. Another private remote server was available for testing, but could not be employed due to extremely high latency.

The code outputs `.vtu` files which are combined to a time series simulation. The output is visualised using the open source software `Paraview` (url: https://www.paraview.org/).

3.2 Geometry Setup

The input `Y` file contains the model including geometry, constraints, materials and simulation parameters. The majority of modelling parameters are adduced from the literature. A list of material parameters is given in Table 1. A list of input file simulation parameters is provided in Table 2. Auxiliary parameters are provided in Table 3.

Fig. 6: `2D` geometry of the laminated glass structure and projectile [28]

Fig. 6 illustrates the `2D` geometric setup. The figure shows the initial state of the projectile (yellow) immediately before impact on the laminated glass

(impactor glass plies in red, inner glass ply in blue, inter-layer in green). The glass is fixed via a support system (in brown). The undersurface (the bottom lines) of the support structure contains no-velocity boundary conditions.

The dimensions are given in Fig. 7. The mesh for all bodies consists of triangular elements. The element size is 1 mm.

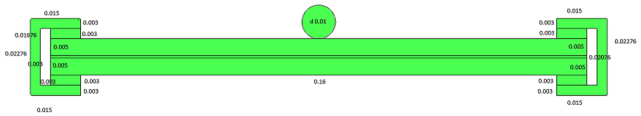

Fig. 7: Dimensions of the test geometry

3.3 Material Parameters

Table 1 lists the material parameters for glass, PVB, steel and rock. The literature provides values for most quantities, although values vary to a great extend in some cases. The values used in this paper are marked bold.

3.4 Problem Parameters

The radius of the steel circle is set to 10 mm. The critical time step [40][11] is set accordingly to

$$\Delta t_{\text{crit}} = \sqrt{\frac{{V_e}^{12}\,\rho^{13}}{p}} = \sqrt{\frac{4.19 \cdot 10^{-6} \cdot 7.8 \cdot 10^{3}}{4 \cdot 10^{11}}}\,\text{s}\,. \tag{16}$$

The acceptable time step Δt is given by

$$\Delta t = \lfloor \Delta t_{\text{crit}} \rfloor = \text{1e-8s}\,, \tag{17}$$

where $\lfloor\ \rfloor$ denotes the floor operator. To simulate real time $t = 1\,\text{s}$, the maximum number of time steps required [40][11] is given by

$$n_{\text{t}} = \frac{t}{\Delta t} = \frac{1\,\text{s}}{1\,\text{e-8 s}} = \text{1e8}\,. \tag{18}$$

3.5 Verification

Meeting specified accuracy standards [41] requires verification of the numerical model by use of data from previous numerical experiments.

Physical experiments involving the breakage of glass require special safety arrangements. Impact experiments are being conducted outdoors by service company Jabisupra [42] in cooperation with Imperial College London. The company is active in the field of envelope security and specialises in protecting infrastructure from certain threats. Data for from this company was not ready and could not be used for this project.

Many other researchers have already conducted impact experiments in the past. A majority of the findings from these experiments are applicable to verify the model for this project.

Dynamic impact on laminated glass comprises hard and soft body impact [43]. Hard body impact such as ballistic impact [7] causes minimal deformation to the projectile, while soft body impact such as bird impact [43] causes the projectile to undergo extensive deformation.

Relevant parameters of the impact projectile include the normal velocity [3, 44, 45, 46], the mass [45, 46], the angle [44, 45, 46], the shape [46] and the size [3]. Relevant parameters for the outer glass ply include its dimensions [16], its mass, the support conditions [16] and the make-up [16]. For the inter-layer, the material [8, 15, 16], thickness [6, 16, 45] and temperature [15, 47] are relevant.

,

[7] for reference
[8] Rayleigh damping, viscous damping
[9] not relevant for glass, PVB and steel
[10] at laboratory conditions
[11] for inquiries concerning this reference please contact the AMCG at Imperial College London
[12] volume (in m^3) of the smallest finite element, without units
[13] density (in kg/m^3) of the material of the smallest finite element, without units

Table 1: List of Y2 input material parameter values.

Property	Glass	PVB	Steel Projectile	Rock[7]
Density ρ $\left[\mathrm{kg/m^3}\right]$	2500 [21, 27, 28, 33, 34]	870 [27] **1100** [28, 33, 34] 1105e3 [28]	7800 [28]	2700
Young's Modulus E [Pa]	**7e10** [28, 33, 34] 7.409e10 [27] 7.41e10 [28] 7.5e10 [21]	**1e8** [28] 2.2e2 [34] 5e10 [27]	2e11 [28]	3e10
Poisson's Ratio ν	0.2 [27, 28] 0.21 [21] 0.3 [28] 0.22 [33] **0.23** [34] 0.25 [28]	0.42 [28] 0.45 [21] 0.48 [27] 0.49 [31] **0.495** [34]	0.29 [28]	0.205
Mass damping coefficient μ[8] (see Eq. 15)	$1.32e7$	$3.31e5$	$1.25e8$	$2.85e7$
Elastic penalty term (see Eq. 13)	7e11	1e9	2.0e12	3.0e11
Contact penalty (see Eq. 13)	7e11	1e9	2.0e12	3.0e11
Mode I energy rate G_{I} $\left[\mathrm{J/m^2}\right]$	**10** [33] 3.9 [28] 4.0 [28]	2.8e3 [35] **20** [28]	1.9e5 [36]	20.0
Mode II energy rate G_{II} $\left[\mathrm{J/m^2}\right]$	50 [33]	2.8e3	1.9e5	100.0
Mode III energy rate G_{III} $\left[\mathrm{J/m^2}\right]$	50 [33]	2.8e3	1.9e5	100.0
Tensile Strength σ [Pa]	6e7 [31] 3e7 [28] **3.46e7** [28]	**2e7** [37] 1.862e7 [28]	1e7 [3]	4e6
Shear Strength τ [Pa]	17.9 [28]	17.9 [28]		
Internal friction coefficient η (Eq. 2)	0.1 [31]	0.7 [17]	0.15 [38]	0.6
Internal cohesion c [Pa] (Eq. 1)	7e10	2.2e2	2e11	8e6
Pore fluid pressure[9] [Pa]	0	0	0	0
Joint friction coefficient[9]	0.6	0.6	0.6	0.6
Joint roughness coefficient $\mathrm{JRC_0}$ [9, 10]	15	15	15	15
Joint compressive strength $\mathrm{JCS_0}$ [9, 10] [MPa]	120	120	120	120
Joint sample size [m][9]	0.2	0.2	0.2	0.2
Interface friction	0.1 [31]	0.62 [39]	0.44 [39]	0.6
2D Problems	plane strain			

Table 2: List of Y2D input simulation parameter values.

Parameter	Value
Maximum number of timesteps n_t (Eq. 18)	3.33e7
Current number of timesteps	0
Restart file saving frequency	3.33e7
Gravity in X Direction (m/s2)	0
Gravity in Y Direction (m/s2)	0
Gravity in Z Direction (m/s2)	−9.8
Timestep (s) (Eq. 17)	3e − 8
Output frequency	3.33e5
Current number of iterations	0
Gravity setting stage (s)	0
Load ramping stage (s)	0
Maximum dimension (m)	10
Maximum force (N)	1e6
Maximum velocity (m/s)	100
Maximum stress (Pa)	1e8
Maximum displacement (m)	2
Minimum joint aperture (m)	1e − 7
Maximum contacting couples	1e7
Buffer Size b for NBS (m) (Eq. 9)	4.13e − 5
Accuracy (bit)	32
Joint friction model	Coulomb
Initial aperture correlation	roughness

Table 3: List of auxiliary parameter values (to calculate simulation parameter values).

Parameter	Value
Total minimum element volume V_e (m^3)	1.00e − 07
Total minimum edge Δ_{min} (m)	4.13e − 04
Total real simulation time t (s)	1
Critical time step glass (s) (Eq. 16)	1.89e − 8
Critical time step steel (s) (Eq. 16)	1.97e − 8
Critical time step PVB (s) (Eq. 16)	3.32e − 7
Overall Mesh size (m)	1e − 3
Mesh size projectile (m)	1e − 3
Mesh size plies (m)	1e − 3
Mesh size inter-layer (m)	1e − 3

Low velocity ($\approx$ 20 m/s) hard impact experiments include the use of projectiles in form of road construction chippings [44], ballistics [8], drop-down weights [16, 31, 48], aluminum projectiles [48] and steel balls [5, 16, 49]. High velocity (around 180 m/s) soft impact experiments include the use of silicon rubber projectiles [43] and gas guns [15].

Wang et al [16] found that the panel size had an inferior effect on the breakage resistance [16]. Similarly, Monteleone et al [8] found that only a local area of the ply around the impact absorbed the impact energy for high velocities.

Karunarathna [45] found that impact velocity and plate thickness contributed significantly towards the impact resistance, compared to impact mass and inter-layer thickness. Wang et al [16] found an increased inter-layer thickness to have a negative effect on energy absorption. Liu et al [50] established that the inter-layer thickness did not contribute towards energy absorption. Behr and Kremer [49] found an increased inter-layer thickness to better protect the inner ply. Kim et al [51] numerically optimised the PVB inter-layer constitution to prevent all damage to the inner glass ply.

Liu et al [50] numerically investigated the optimisability of the inter-layer in terms of energy absorption by simulating the impact of a human head. Zhang et al [47] investigated the influence of temperature on the inter-layer and found that a hybrid TPU/SGP/TPU inter-layer performed best over the entire range of tested temperatures.

4 COMPUTER SCIENCE APPROACH

Lst. 1 lists data type macros which are used in this section.

```
#define FLT float
#define INS int
#define DBL double
#define INT long
#define UCHR unsigned char
#define CHR char
#define UINT unsigned int
#define ULON unsigned long
```

Listing 1: Data type definitions

4.1 VTK format

The Visualisation Toolkit VTK [52] is an extremely popular open source software for graphic visualisation of scientific data. VTK APIs are available in many different programming languages and facilitate the generation of VTK files. Until now, the Y code included VTK API for programming language C to generate output files.

The required VTK version 5.8 library dependencies revealed themselves to be outdated and unobtainable online. All attempts by the author to obtain such a version for testing were not successful. Rewriting the code using a newer VTK version of the API would only temporarily solve this problem.

To diversify application to a wider range of operating systems, it was deemed necessary to hard code the output without using VTK libaries. Aside from the ill-documented offical website [52], one of the only few reliable resources available was deemed to be the Earth Models website by Bunge [53].

A complete implementation of the output algorithm can be found in file Yod.c. In this section, an in-depth description of the different VTK formats and the algorithm is given.

Fig. 8: Snapshot of two rectangles moving apart from each other horizontally, at initial (top) and final time step (bottom).

For testing this VTK algorithm, a simple test setup was used consisting of two rectangular surfaces slowly moving apart from each other in the horizontal direction, as shown in Fig. 8.

4.2 VTK legacy file format

The VTK legacy file format [52] is relatively easy to implement. On the downside, it only supports a minimal amount of features and is thus relatively inflexible.

Section 5 shows examples of VTK legacy output files. The header defines VTK version, file name, data encoding (ascii or binary) and dataset type [14] [52].

The data is divided into sections including points, cells and cell types. The section header contains the keyword along with the total amount of entities of the section. The data is written continuously using spaces as separators. [52]

Every section has additional unique formatting rules. The points sections requires an additional data type identifier (here: `float`) to correctly identify the data of the coordinates [52]. The data in the cells section lists the node numbers of each element, appended to a number identifying the total amount of nodes of the element [52]. The cell type section identifies the geometric shape of each element, given by an index. Each index is written on a separate line. [52]

According to the specified data encoding option, the data may be encoded in decimal ascii or in binary.

```
1  void i2i64(FILE *fout, INS ib10)
2  {
3    putc((ib10 >> 56) & 0xFF, fout);
4    putc((ib10 >> 48) & 0xFF, fout);
5    putc((ib10 >> 40) & 0xFF, fout);
6    putc((ib10 >> 32) & 0xFF, fout);
7    putc((ib10 >> 24) & 0xFF, fout);
```

[14] here: unstructured grid

```
8    putc((ib10 >> 16) & 0xFF, fout);
9    putc((ib10 >> 8) & 0xFF, fout);
10   putc(ib10 & 0xFF, fout);
11 }
```

Listing 2: Function for binary encoding and outputting an `int64`

An example function which was used to encode the data to binary is given in List. 2. Decimals of different sizes are encoded accordingly. The `int64` input decimal `ib10` (size: 8 bytes) is encoded byte for byte, starting with the most significant byte. The byte order[15] was assumed to be little endian, as operating systems with big endian were not available for testing.

For the algorithm, the fact was employed that the decimal is stored as a binary internally and the decimal can thus be treated as binary. First, the bytes are shifted to the beginning of the memory location of the decimal. A bit-wise comparison with $0xFF = 0b11111111 = 255$ then yields the value of the byte. This value is then encoded to a single character and printed to the output file via `putc`.

4.3 VTK XML file format

Compared to `legacy` files, `XML` files are much more difficult to implement. An examples are given in section 5.

The file is structured using nested keyword headers which are enclosed in angle brackets ("<" and ">"), according to `XML` language. Text indentation is not required but enhances readability.

Keyword headers are opened using <keyword> and closed using </keyword>. Keywords which do not contain any sub-keyword headers are opened and closed in one line using <keyword\>. Keyword headers usually contain additional mandatory and optional keywords in the format of `option="value"` [52].

The file header specifies `xml` and VTK versions, VTK file type (`VTKFile type`) and byte order (`byte_order`) [52]. Depending on the specified file type, a unique set of sub-keyword headers is used. For the `Unstructured_Grid` VTK file type chosen here, the file content is structured using <Piece>. Within this header, it is required to specify the number of points and cells (elements) using `NumberOfPoints` and `NumberOfCells` [52].

<Piece> contains definition of <Points> and <Cells>, among other optional keyword sub-headers. <Points> expects exactly one <DataArray> containing the nodal coordinates. <Cells> expects one <DataArray> each for the cell connectivity (node configuration of each cell), the cell offsets (offsets in the cell connectivity list), as well as the cell type (cell shape index, see [52]) [53].

The data, listed after <DataArray> includes `type` (data type), `Name`, `NumberOfComponents`, `format`, `RangeMin` and `RangeMax`.

`RangeMin` and `RangeMax` indicate the minimal and maximal values of the data and are optional[16]. Simple array search functions were implemented in order to find the individual values.

`Name` specifies a unique name, enclosed by "", for the data set. The data array contained in <Points> requires no name. The data arrays enclosed by <Cells> require fixed names `Name="connectivity"`, `Name="offsets"` and `Name="types"` [52].

`format` specifies the encoding of the data.

In the case `format="ascii"`, the data is provided in decimal form, delimited by spaces, similar to section 4.2 [53]. Specifying an inappropriate data `type` for this format will not leave the data completely unusable, although induced type casting may yield unexpected results. The dependence on spaces as delimiters is highly impractical when transferring

[15] endianess

[16] confirmed by tests

files between different operating systems and applications.

A more robust option is given by `format="binary "`. Characters such as "<" (binary encoding of 0b00111100=d060) may potentially appear in binary encoded data. As these characters compose XML keyword headers, binary encoding is not realizable [53]. Thus, the data is encoded to base 64 (b64) (see section 4.4).

For b64 encoding, a data header needs to be prepended to the data. This data header is always of type `int32` (data size: 4 bytes) and specifies the data size, i.e. the amount of binary bytes of subsequent data. The data size is not to be confused with the length of the printed b64 encoded data string in the output file, which is insignificant for now. The data size rather refers to the size of the binary data array in `bytes`, stored as specified by the datatype in the keyword header. Specifying a data `type` of a different byte size for this format is likely to produce errors and to render the data useless.

As an example, consider a `float32` array which contains 100 data values. The corresponding header is given by:

$$h = 100 \frac{32\,\text{bits}}{8\,\text{bits/byte}} = 400\,\text{bytes} \qquad (19)$$

The header[17] then needs to be encoded to b64, separately from the data. The encoded header string is immediately followed by the encoded data string, without any delimiters. Considering the fact that the data header type is fixed (`int32`), the b64 encoded data header string length is always the same. Thus, no additional offset is necessary to specify the positional onset of the data after the data header [53].

[17] omitting the unit (bytes)

Instead of supplying the data inline, it is possible to append data once, to an `<AppendedData>` section at the end of the file. This is achieved by specifying option `format="appended"` for each data array keyword header and providing an `offset` each time.

In doing so, the entire data is provided in a single long b64 encoded string, appended to an underscore "_". Like in the inline format, each encoded data string must be prepended by a b64 encoded header, which specifies the size of the following unencoded data section in bytes. The headers and the data sections are encoded separately each, without any delimiters between headers and data sections.

The offset value provided in each `<DataArray>` specifies the amount of bytes of unencoded data after the underscore to the beginning of the corresponding header. The method of evaluation of header offsets is illustrated in Tab. 4. The values in Tab. 4 are taken from an example file (see [53]), which is visualisable in Paraview.

In Tab. 4, the header offset of the first header is always 0. As the unencoded header is always an `int32` of size 4 bytes, the first data offset is 4. For a data array with 1926 values and data type size 8 bytes (e.g. `float64`), the data size in `bytes` amounts to $1926 \cdot 8$ bytes = 15408bytes. The number of values is comprised by the number of components and the number of points, i.e. $3 \cdot 642 = 1926$. The next header offset is evaluated by adding the data size to the data offset, i.e. 4 + 15408 = 15412. The remaining offset values are evaluated accordingly.

4.4 Base 64 encoding

The output data consists of one- and two-dimensional **`int`** and **`float`** arrays containing nodal properties, elemental properties and flags. The computer stores the data in binary form, according to byte order of the operating system at hand.

`float` numbers are usually stored in IEEE-754 format [54]. As the numbers are stored internally,

Table 4: Breakdown of offset values for an output file of a working example [53] with 642 points and 1280 cells. The desired (header) offsets are given in column 2.

Name	header offset	data offset	data size	data type size	values	components	points/cells
Points	0	4	15408	8	1926	3	642
Connectivity	15412	15416	15360	4	3840	3	1280
Offsets	30776	30780	5120	4	1280	1	1280
Types	35900	35904	1280	1	1280	1	1280
Points s	37184	37188	5136	8	642	1	642
Points v	42324	42328	15408	8	1926	3	642
Points n	57736	57740	15408	8	1926	3	642
Density	73148	73152	5136	8	642	1	642

the exact manner of storage is theoretically not significant as long as the positions[18] of the bits in the binary-stored `float` are not being violated.

As the base64 encoding function takes in a contiguous `CHR*` array, the data needs to be segmented into bytes. For the `INS` data header, this simply involves conversion from a 4-byte `int32` to 4 `CHR` bytes. The segmentation process for arrays is illustrated in Fig. 9. The array is segmented into single bytes, rearranged in groups of 3 bytes and again rearranged in groups of 6 bits. The 6-bit groups are encoded to characters and compose the encoded string.

As an example, the implementation approach is discussed by use of the 2-dimensional contact force array. In a first step, 2-dimensional arrays are reduced to contiguous one-dimensional arrays (List. 3).

```
1  DBL *cf; //contact force
2  INS i, j, k;
3  INS header;
4  header = sizeof(DBL) * ydn->nnopo * ydn->
       ↪ nnodim;
5  cf = malloc(header);
6  k = 0;
7  for (i = 0; i != ydn->nnopo; i++){
8    for (j = 0; j != ydn->nnodim; j++){
9      cf[k] = ydn->d2nfcon[j][i];
10     k++;}
11   cf[k] = ydn->d2nfcon[j][i];
12   k++;}
```

Listing 3: Converting 2-dimensional array to 1-dimensional array

In List. 3, `ydn->nnopo` denotes, in the class "**Y** **d**atabase of **n**odes", the current **n**umber of **no**dal **po**ints. `ydn->nnodim` denotes the current **n**umber of **no**dal **dim**ensions[19] and `ydn->d2nfcon` denotes the **d**ouble **2**-dimensional **n**odal **f**orce of **con**tact[55].

As a result, the data is now stored in a contiguous one-dimensional array of `float64` and now needs to be segmented into bytes.

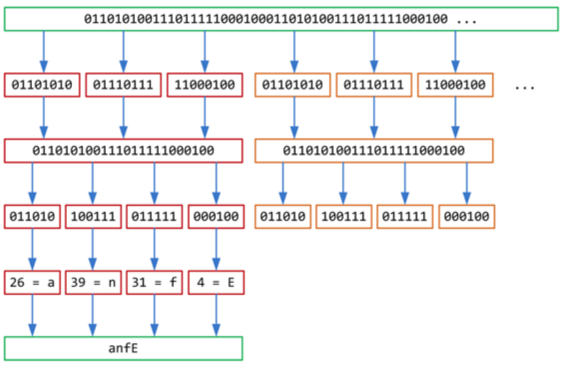

Fig. 9: Subsequent conversion of binary data to bytes, bit sextets, and encoded characters [56].

A first possible strategy is to apply simple data type casting, see List. 4.

[18] the significance

[19] here: 2

```
FLT floats[4] = {1, 2, 3, 4}
//actual array is of variable size;
CHR *bytes = (CHR *) floats;
```

Listing 4: Segmentation approach via type casting with pointers

The `CHR* bytes` array could then be passed on to the encoding function. Test showed that this strategy seems to only work if the array to type cast does not contain any data before type casting. Thus, this approach is not appropriate here.

A second possible strategy is to declare a **union**, see List. 5.

```
union INTToCHR {
  INS i;
  CHR *c[sizeof(INS)];};
```

Listing 5: Segmentation approach via a union

Here, both `i` and `c` refer to the same location in memory. This approach potentially works for a single **float**. As the `CHR*` arrays would need to be concatenated dynamically, this approach is (according to the author's opinion) not appropriate either. Test which were carried out using this strategy showed inconsistent outputs, indicating that the prompted data may not be contiguously stored.

A third strategy for dividing the data arrays into bytes uses a combination of `malloc` and `memcpy`, see List. 6.

```
CHR *bytes;
bytes = malloc(header);
memcpy(&bytes[0], cf, header);
```

Listing 6: Segmentation approach via memcpy

Testing confirmed that this method is successful. The `byte` array now contains the floats in contiguous order in binary form, stored as bytes.

The prepared data is passed to the base 64 encoding function `b64enc` in file `Yod.c`. The function takes some `CHR*` array `src` of length `len` as input, encodes it byte for byte and outputs character for character into the VTK output file `fout`.

The core of the algorithm (List. 7) was adopted from Malinen [57].

```
while (end - in > 2){
  putc(enc[in[0] >> 2], fout);
  putc(enc[((in[0] & 0x03) << 4) } (in[1] >>
      ↪ 4)], fout);
  putc(enc[((in[1] & 0x0f) << 2) } (in[2] >>
      ↪ 6)], fout);
  putc(enc[in[2] & 0x3f], fout);
  in += 3;}

if (end - in){
  putc(enc[in[0] >> 2], fout);
  if (end - in == 1){
    putc(enc[(in[0] & 0x03) << 4], fout);
    putc('=', fout);}
  else{
    putc(enc[((in[0] & 0x03) << 4) } (in[1]
        ↪ >> 4)], fout);
    putc(enc[(in[1] & 0x0f) << 2], fout);}
  putc('=', fout);}
return;}
```

Listing 7: b64 algorithm [57]

In List. 7, `in` and `end` are `CHR*` pointers which point to the beginning and end of the input array. Converting to base 64 requires rearranging the input in chunks of 6 bits. Each of these 6-bit chunks produces a value $v < 64 = 0b1000000$ and is able to be converted to one of 64 unique characters from the encoding array

```
CHR* enc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklmnopqrstuvwxyz0123456789+/" .
```

For every 3 bytes (or 24 bits), the algorithm produces 4 groups of 6 bits, i.e. 4 characters. The bits are extracted continuously via bit shifting. As mentioned before, the position or significance of the bits in the byte must not be violated. In other words,

the value of the bit must not get lost through bit shifting. The algorithm is now explained in detail:

In line 2, the first 6 bits of the current character `in[0]` are extracted by bit shifting to produce the first 6-bit chunk. The value of this chunk is used to produce the index of the corresponding character in the encoding array `enc`.

The last two remaining bits of the current input character `in[0]` are evaluated and bit shifted in line 3 to account for the first 2 significant bits of the next 6-bit chunk. The 4 missing bits to complete the 6 bits are extracted from the first 4 bits of the second character `in[1]`, again by bit shifting.

In line 4, the 4 remaining bits from the second input character `in[1]` make up the first four bits of the next 6-bit chunk. The missing 2 bits of the 6-bit chunk are extracted from the third input character `in[2]`. This leaves 6 bits of the third input character remaining, making up another 6-bit chunk, as implemented in line 5.

If the input array length `len = end - in` is not divisible by 3, padding characters are required. The amount of padding padding depends on the amount of remaining input byes to complete a group of three input bytes. If $\text{len} \,\%\, 3 == 1$, two padding characters '==' are appended to the encoded string. If $\text{len} \,\%\, 3 == 2$, one padding character '=' is required. The algorithm is given in List. 7, lines 8-17.

Writing the output characters to a buffer instead of using `putc` reduces the amount of times of invoking the `I/O` buffer and could save computational time. This alternative was discarded as it did not provide any significant advantage in terms of computational costs in this minimal example. The advantage of reducing computational time faces the disadvantage of potentially introducing memory allocation errors.

Table 5: List of conducted tests.

Test	Setup	Status	Remark
1	initial	success	rock
2	initial	no geometry	glass/steel
3	final	success	glass/PVB/steel
4	initial	no geometry	glass
5	initial	success	steel
6	initial	no geometry	test 4, high penalty
7	initial	failed	test 5, high penalty
8	final	failed	glass
9	final	failed	steel
10	final	failed	PVB
11	final	failed	rock, coarser mesh
12	initial	failed	test 4, smaller int. coh.
13	initial	failed	test 4, no `jrc`
14	initial	failed	test 4, smaller timestep

5 RESULTS

5.1 Simulation

A series of 14 tests was conducted, see Table 5. For comparison, some of the tests were conducted using a simple geometry shown in Fig. 10. Of these 14 tests, 8 tests failed during run time and 3 tests did not display any geometries.

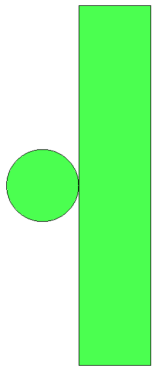

Fig. 10: Setup of an initial simple test geometry for comparison.

Fig. 11 shows snapshots of the simulation of the simple test geometry. Fig. 12 shows a snapshot of simulation test 3 using the final geometry.

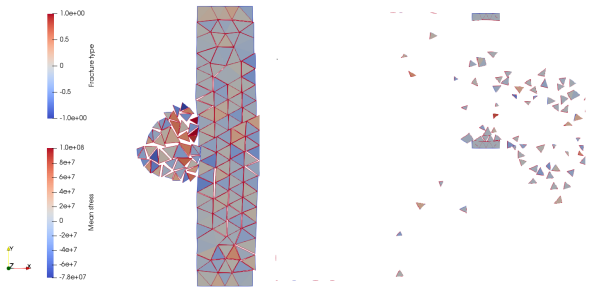

Fig. 11: Snapshots of simulation test 1 at initial (left) and final time steps (right).

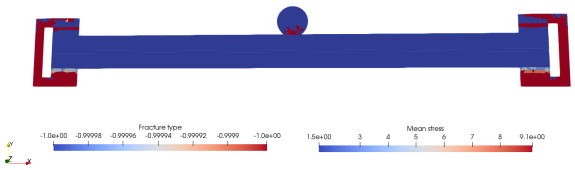

Fig. 12: Snapshot of simulation test 3 at final time step

5.2 VTK implementation

Flags to switch between different output formats must be changed the code. As the output format for simulations will be fixed, input flags were not considered. Flag `ym==1` generates the original `.ym` files, `vtu_leg==1` generates VTK `.vtu` legacy files, and `vtu_xml==1` generates VTK `.vtu` XML files.

For VTK legacy files, `b10==1` generates decimal ascii output, and `b10==0` generates binary output. For VTK XML files, `app==1` generates appended output, while `app=0` generates inline output. For inline output, `b10==1` produces decimal ascii output, and `b10==0` produces base 64 encoded output. For appended output, `b64==1` generates base 64 encoded output, whereas `b64==0` creates raw binary encoded output.

A working example VTK legacy output file is given in List. 1. Working examples VTK XML b64 inline encoded output files are given in List. 2 and 3.

Table 6: Flags for producing different output in file `Yod.c`

flags	app	b10	b64	output
ym				Ym (original)
vtu_leg		1		legacy ascii
		0		legacy binary
vtu_xml	0	1		XML inline ascii
	0	0		XML inline base 64
	1		0	XML appended raw
	1		1	XML appended base

Some file formats are not visualisable for now. Fig. 13 shows an example of a VTK legacy file with binary encoding. Fig. 14 and 4 show examples of raw and base 64 encoded appended VTK XML output files. These examples are not visualisable in Paraview, presumably due to a bug in the algorithm.

```
1  # vtk DataFile Version 5.8
2  CODED
3  ASCII
4  DATASET UNSTRUCTURED_GRID
5  POINTS 20 float
6  -1.000000 1.000000  0.000000 1.000000
   ↪  0.000000 1.000000  -1.000000 0.000000
   ↪  0.000000 0.000000  0.000000 0.000000
7  1.000000 1.000000  1.000000 0.000000  0.000000
   ↪  -0.000000  0.000000 1.000000  -1.000001
   ↪  0.000000  0.000000 1.000000
8  -1.000001 1.000000  -1.000001 0.000000
   ↪  1.000001 1.000000  0.000000 1.000000
   ↪  1.000001 0.000000  0.000000 -0.000000
9  1.000001 0.000000  0.000000 1.000000
10 CELLS 56
11 3 14 15 16
12 3 17 18 19
13 3 8 9 10
14 3 11 12 13
15 3 8 9 9
16 3 9 10 13
17 3 10 8 8
18 3 11 12 12
19 3 12 13 13
20 3 14 15 15
21 3 15 16 18
```

```
3 16 14 14
3 17 18 18
3 19 17 17
CELLTYPE 14
8
8
8
8
8
8
8
8
8
8
8
8
8
```

Listing 1. Example VTK legacy ascii encoded file

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid"
  version="0.1" byte_order =
  "LittleEndian">
  <UnstructuredGrid>
    <Piece NumberOfPoints="20"
      NumberOfCells="14">
      <Points>
        <DataArray type="Float32"
          NumberOfComponents="3"
          format="ascii"
          RangeMin="-1.000001"
          RangeMax="1.000001">
          -1.000000 1.000000 0 0.000000
            1.000000 0 0.000000 1.000000 0
            -1.000000 0.000000 0 0.000000
            0.000000 0 0.000000 0.000000 0
            1.000000 1.000000 0 1.000000
            0.000000 0 0.000000 -0.000000 0
            0.000000 1.000000 0 -1.000001
            0.000000 0 0.000000 1.000000 0
            -1.000001 1.000000 0 -1.000001
            0.000000 0 1.000001 1.000000 0
            0.000000 1.000000 0 1.000001
            0.000000 0 0.000000 -0.000000 0
            1.000001 0.000000 0 0.000000
            1.000000 0
        </DataArray>
      </Points>
      <Cells>
        <DataArray type="Int64"
          Name="connectivity"
          NumberOfComponents="1"
          RangeMin="" RangeMax=""
          format="ascii">
          14 15 16 17 18 19 8 9 10 11 12 13 8 9
            9 9 10 13 10 8 8 11 12 12 12 13
            13 14 15 15 15 16 18 16 14 14 17
            18 18 19 17 17
        </DataArray>
        <DataArray type="UInt32" Name="offsets"
          NumberOfComponents="1"
          RangeMin="" RangeMax=""
          format="ascii">
          3 6 9 12 15 18 21 24 27 30 33 36 39 42
        </DataArray>
        <DataArray type="UInt32" Name="types"
          NumberOfComponents="1"
          RangeMin="" RangeMax=""
          format="ascii">
          5 5 5 5 5 5 5 5 5 5 5 5 5 5
        </DataArray>
      </Cells>
    </Piece>
  </UnstructuredGrid>
</VTKFile>
```

Listing 2. Example VTK XML file with decimal ascii inline encoding

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid"
  version="0.1" byte_order =
  "LittleEndian">
  <UnstructuredGrid>
    <Piece NumberOfPoints="20"
      NumberOfCells="14">
      <Points>
        <DataArray type="Float32"
          NumberOfComponents="3"
          format="binary"
          RangeMin="-1.000001"
          RangeMax="1.000001">
```

Fig. 13: Example VTK legacy binary encoded file

```
7          8AAAAA==AACAvwAAgD8AAAAAAAAAAAAAgD8
           AAAAAAAAAAAAAgD8AAAAAACAvwAAA
           AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
           AAAAAAAAAAACAPwAAgD8AAAAAACAP
           wAAAAAAAAAAAAAAAEdnLK0AAAAAAAA
           AAAAAgD8AAAAACACAvwAAAAAAAAAA
           AAAAAAAgD8AAAAACACAvwAAgD8AAA
           ACACAvwAAAAAAAAACACAPwAAgD8AA
           AAAAAAAAAAAgD8AAAAACACAPwAAAA
           AAAAAAAAAAEdnLK0AAAAACACAPwAA
           AAAAAAAAAAAAAAAgD8AAAA
        </DataArray>
      </Points>
      <Cells>
        <DataArray type="Int64"
           Name="connectivity"
           NumberOfComponents="1"
           RangeMin="" RangeMax=""
           format="binary">
```

```
12         UAEAAA==DgAAAAAAAAPAAAAAAAAABAAAAA
           AAAAAEQAAAAAAAAASAAAAAAAABMAA
           AAAAAAACAAAAAAAAAJAAAAAAAAAAo
           AAAAAAAAACwAAAAAAAAMAAAAAAAAA
           A0AAAAAAAACAAAAAAAAAJAAAAAAA
           AAAkAAAAAAAACQAAAAAAAAKAAAAA
           AAAAA0AAAAAAAACgAAAAAAAAIAAA
           AAAAAAAgAAAAAAAACwAAAAAAAAMA
           AAAAAAAAAwAAAAAAAADAAAAAAAAA
           NAAAAAAAAA0AAAAAAAADgAAAAAA
           AAPAAAAAAAAA8AAAAAAAADwAAAA
           AAAAQAAAAAAAABIAAAAAAAAEAAAA
           AAAAAAOAAAAAAAAA4AAAAAAAAEQA
           AAAAAAAASAAAAAAAABIAAAAAAAAE
           wAAAAAAAARAAAAAAAABEAAAAAAAA
        </DataArray>
        <DataArray type="UInt32" Name="offsets"
           NumberOfComponents="1"
           RangeMin="" RangeMax=""
           format="binary">
           0AAAAA==AwAAAAYAAAAJAAAADAAAA8AAAA
           SAAAAFQAAABgAAAAbAAAAHgAAACEAA
           AAkAAAAJwAAACoAAAA=
        </DataArray>
        <DataArray type="UInt32" Name="types"
           NumberOfComponents="1"
           RangeMin="" RangeMax=""
           format="binary">
```

```
            0AAAAA==BQAAAAUAAAAFAAAABQAAAAUAAAA
            ↪   FAAAABQAAAAUAAAAFAAAABQAAAAUAA
            ↪   AAFAAAABQAAAAUAAAA=
          </DataArray>
        </Cells>
      </Piece>
    </UnstructuredGrid>
</VTKFile>
```

Listing 3. Example VTK XML file with b64 inline encoding

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid"
↪  version="0.1" byte_order =
↪  "LittleEndian">
  <UnstructuredGrid>
    <Piece NumberOfPoints="20"
    ↪  NumberOfCells="14">
      <Points>
        <DataArray type="Float32"
        ↪  NumberOfComponents="3"
        ↪  format="appended"
        ↪  RangeMin="-1.000001"
        ↪  RangeMax="1.000001" offset="0"/>
      </Points>
      <Cells>
        <DataArray type="Int64"
        ↪  Name="connectivity" RangeMin=""
        ↪  RangeMax="" format="appended"
        ↪  offset="244"/>
        <DataArray type="UInt32" Name="offsets"
        ↪  RangeMin="" RangeMax=""
        ↪  format="appended" offset="584"/>
        <DataArray type="UInt32" Name="types"
        ↪  RangeMin="" RangeMax=""
        ↪  format="appended" offset="644"/>
      </Cells>
    </Piece>
  </UnstructuredGrid>
  <AppendedData encoding = "base64">
    _8AAAAA==AACAvwAAgD8AAAAAAAAAAAAgD8AAAAAA
    ↪  AAAAAAAgD8AAAAAACAvwAAAAAAAAAAAAAAAA
    ↪  AAAAAAAAAAAAAAAAAAAAAAAAAAAACAPwAAgD8
    ↪  AAAAAAACAPwAAAAAAAAAAAAAAEdnLK0AAAAA
    ↪  AAAAAAAAgD8AAAACACAvwAAAAAAAAAAAAAAA
    ↪  AAAgD8AAAAACACAvwAAgD8AAAAACACAvwAAAA
    ↪  AAAAAACACAPwAAgD8AAAAAAAAAAAAAgD8AAAA
    ↪  ACACAPwAAAAAAAAAAAAAAEdnLK0AAAAACACA
    ↪  PwAAAAAAAAAAAAAAAAAgD8AAAAAUAEAAA==D
    ↪  gAAAAAAAAAPAAAAAAAAABAAAAAAAAAAEQAAAA
    ↪  AAAAASAAAAAAAABMAAAAAAAAACAAAAAAAAAA
    ↪  JAAAAAAAAAAoAAAAAAAAACwAAAAAAAAAMAAAA
    ↪  AAAAAA0AAAAAAAACAAAAAAAAAAJAAAAAAAAA
    ↪  AkAAAAAAAAACQAAAAAAAAAKAAAAAAAAAA0AAA
    ↪  AAAAAACgAAAAAAAAAIAAAAAAAAAAgAAAAAAAA
    ↪  ACwAAAAAAAAAMAAAAAAAAAAwAAAAAAAADAAA
    ↪  AAAAAAANAAAAAAAAAA0AAAAAAAADgAAAAAAA
    ↪  AAPAAAAAAAAAA8AAAAAAAAADwAAAAAAAAQAA
    ↪  AAAAAAABIAAAAAAAAAEAAAAAAAAAOAAAAAAA
    ↪  AAA4AAAAAAAAAEQAAAAAAAAASAAAAAAAAABIA
    ↪  AAAAAAAAEwAAAAAAAAARAAAAAAAABEAAAAAA
    ↪  AAAOAAAAA==AwAAAAYAAAAJAAAADAAAAA8AAA
    ↪  ASAAAAFQAAABgAAAAbAAAAHgAAACEAAAAkAAA
    ↪  AJwAAACoAAAA=OAAAAA==BQAAAAUAAAAFAAAA
    ↪  BQAAAAUAAAAFAAAABQAAAAUAAAAFAAAABQAAA
    ↪  AUAAAAFAAAABQAAAAUAAAA=
  </AppendedData>
</VTKFile>
```

Listing 4. Example VTK XML file with b64 appended encoding

6 DISCUSSION

6.1 Simulation

The simulation clearly only produces realistic results for simple geometry setups.

Simulation results for more complex geometries are not realistic. Here, the projectile of prescribed material steel locally deforms on impact on the glass, while the glass ply does not show any reaction to the impact. Due to these unrealistic results, none of the findings from previous research could be confirmed.

The simulation might have failed due to errors in the pre-processing software.

Fig. 14: Example VTK XML file with raw appended encoding

The pre-processor GiD appeared to be faulty. The prepared input Y file presumably contained definitions which are contradicted in the Y2D code. For example, the nodes which make up each element may have been defined in the opposite rotational direction. It may be possible that this error may have been caused by importing geometries from AutoCAD. This error may be connected to a potential inconsistency in applied operating systems, i.e. the preprocessor may have produced different outputs on different operating systems. As code for these programs is not publicly available, which makes further investigation difficult to imossible.

As GiD seems to be unpredictable, it is of significant importance to replace this software with more reliable alternatives such as Gmesh [58]. Alternatively, an algorithm could be implemented as part of the Y2D code which could check the validity of the input Y file before any further computations.

The problem may also be located in the original Y2D code. Running the code in a clean environment using for instance Travis CI results in errors. The failing Travis build strongly indicates that errors in the code exist and are a likely cause for the simulation to fail.

```
1 JCS0=d1pjcs[propID];
```

Listing 8: Memory allocation error for variable JCS0 in file Yrd.c

One such error occurs in line 676 in file Yad.c, see List. 8. The variable JCS0 is reserved for rock materials, which are not applied in this paper. There seems to be a memory allocation error, although the circumstances are not quite clear to the author.

```
1 #define SETLINEBUF(fcheck) setvbuf((fcheck),
    ↪  NULL, _IONBF, 0); //definition
2 SETLINEBUF(ydc->fcheck); //line 733
```

Listing 9: SETLINEBUF Error in file Yrd.c

Another error occurs at the beginning of the program in line 733 in file Yrd.c, see List. 9. The command setvbuf((fcheck), NULL, _IONBF, 0) presumably assigns an empty buffer NULL to some file ydc-> ↪ fcheck. However, the specifics and the purpose of this command are again unclear to the author.

6.2 VTK Implementation

The results have a significant impact on further development of the FEMDEM code. Before, the Y2D code needed to be run on remote servers. Specifically, this involved submitting jobs each time via the HPC system. Debugging proved to be extremely inconvenient and inefficient, as error files from failed jobs generally do not specify the exact position or kind of error.

Code using VTK libraries to produce the output was replaced by an elementary implementation. VTK output files are now able to be generated and simulated without additional dependencies. Removing the VTK module allows the code to be able to be run locally.

The current algorithm needs to be further improved and adjusted. As tests were conducted using a simple setup without fracturing, joint elements need to be visualised accordingly. The following output file types are still faulty:

(1) binary output for VTK legacy files
(2) appended output for VTK XML files

The output algorithm may be further improved by the implementation of a compression algorithm to compress to the VTK XML files. This is indicated in the file by adding the option

```
compressor="vtkZibDataCompressor"
```

to the header [52]. Bunge [53] provides a promising algorithm, but it remains to be revised, adjusted and implemented.

Most significantly, the modified code paves the way for program debugging and testing. Possible software for debugging on Linux Ubuntu operating system includes the GNU Project Debugger gdb (url: https://www.gnu.org/software/gdb/). Debugging is necessary until the code runs in a clean environment.

Once the code is debugged, additional features may be implemented in the code. One such feature includes inserting joint elements between elements of different materials. This would be particularly of interest for glass, considering the interface interaction between glass and the PVB inter-layer. Another future task is to implement a model for this inter-layer based on hyper-elastic laws.

The results can also be applied to the Y3D code to simulate three-dimensional output. The algorithm to generate three-dimensional output code is estimated to be of similar nature as for the Y2D code. It is expected that only minor adjustments to the algorithm are necessary.

It should only be necessary to involve the HPC system once the debugging and testing is finished. Sensible applications involving the HPC system include simulations on a larger scale.

ACKNOWLEDGMENTS

Special gratitude pertains to the Research Computing Service RCS at Imperial College for their constant support throughout the project. The author would also like to thank course director Dr Gerard Gorman[20] and course administrator Ying Ashton [21] for support during this research. Finally, the author would like to thank the project coordinators, and the supervisors for enabling this project.

[20] g.gorman@imperial.ac.uk, Imperial College London, Department of Earth and Science
[21] y.ashton@imperial.ac.uk, Imperial College London, Department of Earth and Science

REFERENCES

[1] M. A. Samieian, D. Cormie, D. Smith, W. Wholey, B. R. Blackman, J. P. Dear, and P. A. Hooper. 2019. On the bonding between glass and PVB in laminated glass. *Engineering Fracture Mechanics*, 214, April, 504–519. ISSN: 00137944. DOI: https://doi.org/10.1016/j.engfracmech.2019.04.006. https://doi.org/10.1016/j.engfracmech.2019.04.006 (cited on pages 1, 3).

[2] W. Xu and M. Zang. 2014. Four-point combined DE/FE algorithm for brittle fracture analysis of laminated glass. *International Journal of Solids and Structures*, 51, 10, (May 2014), 1890–1900. ISSN: 00207683. DOI: https://doi.org/10.1016/j.ijsolstr.2014.01.026 (cited on pages 1, 3).

[3] C. D. Wu, X. Q. Yan, and L. M. Shen. 2014. A numerical study on dynamic failure of nanomaterial enhanced laminated glass under impact. In *IOP Conference Series: Materials Science and Engineering* number 1. Volume 10. Institute of Physics Publishing. DOI: https://doi.org/10.1088/1757-899X/10/1/012176 (cited on pages 1, 2, 6, 7).

[4] S. Chen, M. Zang, D. Wang, S. Yoshimura, and T. Yamada. 2017. Numerical analysis of impact failure of automotive laminated glass: A review. (August 2017). DOI: https://doi.org/10.1016/j.compositesb.2017.04.007 (cited on page 1).

[5] F. W. Flocker and L. R. Dharani. 1998. Modeling interply debonding in laminated architectural glass subject to low velocity impact. *Structural Engineering and Mechanics*, 6, 5, 485–496. ISSN: 12254568. DOI: https://doi.org/10.12989/sem.1998.6.5.485 (cited on pages 1, 8).

[6] F. S. Ji, L. R. Dharani, and R. A. Behr. 1998. Damage probability in laminated glass subjected to low velocity small missile impacts. *Journal of Materials Science*, 33, 19, 4775–4782. ISSN: 00222461. DOI: https://doi.org/10.1023/A:1004457624817 (cited on pages 1, 6).

[7] X. Brajer, F. Hild, and S. Roux. 2010. On the dynamic fragmentation of glass: A meso-damage model. *International Journal of Fracture*, 163, 1-2, (May 2010), 121–131. ISSN: 03769429. DOI: https://doi.org/10.1007/s10704-009-9421-9 (cited on pages 1, 6).

[8] L. Monteleone, R. Steindler, and G. Kajon. 2004. Monitored ballistic tests on shockproof sandwich glasses in different conditions. *Experimental Techniques*, 28, 5, 28–32. ISSN: 07328818. DOI: https://doi.org/10.1111/j.1747-1567.2004.tb00183.x (cited on pages 1, 6, 8).

[9] L. Biolzi, S. Cattaneo, and G. Rosati. 2010. Progressive damage and fracture of laminated glass beams. *Construction and Building Materials*, 24, 4, 577–584. ISSN: 09500618. DOI: https://doi.org/10.1016/j.conbuildmat.2009.09.007 (cited on page 1).

[10] P. J. G. Schreurs. 2012. Fracture Mechanics. Eindhoven, (September 2012). http://www.mate.tue.nl/~piet/edu/frm/pdf/frmsyl1213.pdf (cited on page 1).

[11] A. Munjiza, K. R. F. Andrews, and J. K. White. 1999. Combined single and smeared crack model in combined finite-discrete element analysis. *International Journal for Numerical Methods in Engineering*, 44, 1, (January 1999), 41–57. ISSN: 00295981. DOI: https://doi.org/10.1002/(SICI)1097-0207(19990110)44:1<41::AID-NME487>3.0.CO;2-A (cited on pages 2, 3).

[12] J. P. Latham, J. Xiang, A. Farsi, C. Joulin, N. Karantzoulis, A. Obeysekara, and P. Yang. 2019. Solidity Website. (2019). http://solidityproject.com/ (cited on pages 2, 3).

[13] A. Munjiza. 2004. *The Combined Finite-Discrete Element Method*. English. John Wiley & Sons, Ltd, Hoboken, 350. ISBN: 0470841990. DOI: https://doi.org/10.1002/0470020180. https://onlinelibrary.wiley.com/doi/book/10.1002/0470020180 (cited on pages 2–4).

[14] M. Abuaisha, D. W. Eaton, J. Priest, and R. Wong. 2015. Finite / Discrete Element Method (FDEM) by Y- Geo : An overview. *Microseismic Industry Consortium*, 5, February, 13. https://www.researchgate.net/publication/295705226_FiniteDiscrete_Element_Method_FDEM_by_Y-Geo_An_overview (cited on page 2).

[15] I. Mohagheghian, M. N. Charalambides, Y. Wang, L. Jiang, X. Zhang, Y. Yan, A. J. Kinloch, and J. P. Dear. 2018. Effect of the polymer interlayer on the high-velocity soft impact response of laminated glass plates. *International Journal of Impact Engineering*, 120, (October 2018), 150–170. ISSN: 0734743X. DOI: https://doi.org/10.1016/j.ijimpeng.2018.06.002 (cited on pages 2, 6, 8).

[16] X. e. Wang, J. Yang, Q. Liu, and C. Zhao. 2018. Experimental investigations into SGP laminated glass under low velocity impact. *International Journal of Impact Engineering*, 122, (December 2018), 91–108. ISSN: 0734743X. DOI: https://doi.org/10.1016/j.ijimpeng.2018.06.010 (cited on pages 2, 3, 6, 8).

[17] J. K. Kuntsche. 2015. *Mechanical behaviour of laminated glass under time-dependent and explosion loading*. German. Springer-Verlag Berlin Heidelberg, Darmstadt, 265. ISBN: 978-3-662-48830-0. DOI: https://doi.org/10.1007/978-3-662-48831-7 (cited on pages 2, 7).

[18] M. H. Ghadiri Rad, F. Shahabian, and S. M. Hosseini. 2015. A meshless local Petrov–Galerkin method for nonlinear dynamic analyses of hyper-elastic FG thick hollow cylinder with Rayleigh damping. *Acta Mechanica*, 226, 5, (May 2015), 1497–1513. ISSN: 0001-5970. DOI: https://doi.org/10.1007/s00707-014-1266-2. http://link.springer.com/10.1007/s00707-014-1266-2 (cited on pages 2, 3).

[19] N. H. Kim. 2015. *Introduction to nonlinear finite element analysis*. (1st edition). Springer, New York, NY, XIV, 430. ISBN: 9781441917461. DOI: https://doi.org/10.1007/978-1-4419-1746-1. https://www.springer.com/gp/book/9781441917454 (cited on page 2).

[20] Y. T. Gu, Q. X. Wang, and K. Y. Lam. 2007. A meshless local Kriging method for large deformation analyses. *Computer Methods in Applied Mechanics and Engineering*, 196, 9-12, 1673–1684. ISSN: 00457825. DOI: https://doi.org/10.1016/j.cma.2006.09.017 (cited on pages 2, 3).

[21] Abaqus. 2013. Abaqus Analysis User's Guide. (2013). http://ivt-abaqusdoc.ivt.ntnu.no:2080/v6.14/books/usb/default.htm (cited on pages 3, 7).

[22] N. Kumar and V. V. Rao. 2016. Hyperelastic Mooney-Rivlin Model : Determination and Physical Interpretation of Material Constants. *MIT International Journal of Mechanical Engineering*, 6, 1, 43–46. ISSN: 2230-7680. https://www.mitpublications.org/yellow_images/75618-me-book.43-46.pdf (cited on page 3).

[23] A. Munjiza, D. R. Owen, and N. Bicanic. 1995. A combined finite-discrete element method in transient dynamics of fracturing solids. *Engineering Computations*, 12, 2, (February 1995), 145–174. ISSN: 02644401. DOI: https://doi.org/10.1108/02644409510799532 (cited on page 3).

[24] A. Munjiza, E. E. Knight, and E. Rougier. 2012. *Computational mechanics of discontinua. Wiley series in computational mechanics*. Wiley, Chichester, West Sussex, U.K. DOI: https://doi.org/10.1002/9781119971160 (cited on page 3).

[25] A. Munjiza, Z. Lei, V. Divic, and B. Peros. 2013. Fracture and fragmentation of thin shells using the combined finite-discrete element method. *International Journal for Numerical Methods in Engineering*, 95, 6, 478–498. ISSN: 1097-0207. DOI: https://doi.org/10.1002/nme.4511. https://doi.org/10.1002/nme.4511 (cited on pages 3, 4).

[26] L. Guo, J. Xiang, J. P. Latham, and B. Izzuddin. 2016. A numerical investigation of mesh sensitivity for a new three-dimensional fracture model within the combined finite-discrete element method. *Engineering Fracture Mechanics*, 151, (January 2016), 70–91. ISSN: 00137944. DOI: https://doi.org/10.1016/j.engfracmech.2015.11.006 (cited on page 3).

[27] W. Gao and M. Zang. 2014. The simulation of laminated glass beam impact problem by developing fracture model of spherical DEM. *Engineering Analysis with Boundary Elements*, 42, 2–7. ISSN: 09557997. DOI: https://doi.org/10.1016/j.enganabound.2013.11.011 (cited on pages 3, 7).

[28] X. Chen and A. H. Chan. 2018. Modelling impact fracture and fragmentation of laminated glass using the combined finite-discrete element method. *International Journal of Impact Engineering*, 112, (February 2018), 15–29. ISSN: 0734743X. DOI: https://doi.org/10.1016/j.ijimpeng.2017.10.007 (cited on pages 3, 5, 7).

[29] J. Xiang, A. Munjiza, and J. P. Latham. 2009. Finite strain, finite rotation quadratic tetrahedral element for the combined finite-discrete element method. *International Journal for Numerical Methods in Engineering*, 79, 8, (August 2009), 946–978. ISSN: 00295981. DOI: https://doi.org/10.1002/nme.2599 (cited on page 3).

[30] Q. Lei, J. P. Latham, and J. Xiang. 2016. Implementation of an Empirical Joint Constitutive Model into Finite-Discrete Element Analysis of the Geomechanical Behaviour of Fractured Rocks. *Rock Mechanics and Rock Engineering*, 49, 12, (December 2016), 4799–4816. ISSN: 0723-2632. DOI: https://doi.org/10.1007/s00603-016-1064-3. http://link.springer.com/10.1007/s00603-016-1064-3 (cited on page 4).

[31] S. Chen, M. Zang, and W. Xu. 2015. A three-dimensional computational framework for impact fracture analysis of automotive laminated glass. *Computer Methods in Applied Mechanics and Engineering*, 294, (September 2015), 72–99. ISSN: 00457825. DOI: https://doi.org/10.1016/j.cma.2015.06.005. https://www.sciencedirect.com/science/article/pii/S0045782515001978 (cited on pages 4, 7, 8).

[32] GID. 2011. GID, The universal, adaptative and user friendly pre and post processing system for computer analysis in science and engineering. Reference Manual. (2011). http://www.compassis.com/downloads/Manuals/GiD_User_Manual.pdf (cited on page 5).

[33] J. Xu, Y. Li, X. Chen, Y. Yan, D. Ge, M. Zhu, and B. Liu. 2010. Numerical study of PVB laminated windshield cracking upon human head impact. *Computers, Materials and Continua*, 18, 2, 183–211. ISSN: 1546-2218. DOI: https://doi.org/10.3970/cmc.2010.018.183. (cited on page 7).

[34] A. Vedrtnam and S. J. Pawar. 2017. Laminated plate theories and fracture of laminated glass plate - A review. *Engineering Fracture Mechanics*, 186, 316–330. ISSN: 00137944. DOI: https://doi.org/10.1016/j.engfracmech.2017.10.020 (cited on page 7).

[35] M. A. Samieian, D. Cormie, D. Smith, W. Wholey, B. R. Blackman, P. A. Hooper, and J. P. Dear. 2017. Delamination in Blast Resistant Laminated Glass. *21st International Conference on Composite Materials Xi'an, 20-25th August 2017*, August, 20–25. DOI: https://doi.org/10.1016/j.ijimpeng.2017.09.001 (cited on page 7).

[36] J. Stampfl and O. Koldenik. 2000. The separation of the fracture energy in metamaterials. *International Journal of Fracture*, 101, 321–345. DOI: https://doi.org/10.1023/A:1007500325074 (cited on page 7).

[37] M. Zang and S. Chen. 2012. Laminated glass. In (Second Edi edition). L. Nicolais and A. Borzacchiello, editors. Joh Wiley &, Guangzhou. DOI: https://doi.org/10.1002/9781118097298.weoc121 (cited on page 7).

[38] M. Sahin, C. S. Çetinarslan, and H. E. Akata. 2007. Effect of surface roughness on friction

coefficients during upsetting processes for different materials. *Materials and Design*, 28, 2, 633–640. ISSN: 18734197. DOI: https://doi.org/10.1016/j.matdes.2005.07.019 (cited on page 7).

[39] M. Fahlbusch. 2007. Zur Ermittlung der Resttragfähigkeit von Verbundsicherheitsglas am Beispiel eines Glasbogens mit Zugstab. German, (March 2007), 1–123. http://elib.tu-darmstadt.de/diss/000962 (cited on page 7).

[40] A. Farsi and J. Xiang. 2019. DEM Plus Manual. (2019) (cited on page 6).

[41] A. Stolz, O. Millon, C. Bedon, C. Kevin, A. V. Doormaal, C. Haberacker, M. Larcher, E. Commission, O. Millon, A. Saarenheimo, G. Solomos, E. Commission, and A. Stolz. 2015. *Recommendations for the Improvement of Existing European Norms for Testing the Resistance of Windows and Glazed Façades to Explosive Effects*. ISBN: 9789279533945. DOI: https://doi.org/10.2788/319252. https://ec.europa.eu/jrc/en/publication/recommendations-improvement-existing-european-norms-testing-resistance-windows-and-glazed-facades (cited on page 6).

[42] J. Zucker, J. Brades, A. New, and A. Toon. 2016. Jabisupra Company Website. (2016). https://www.jabisupra.co.uk/ (cited on page 6).

[43] I. Mohagheghian, Y. Wang, J. Zhou, L. Yu, X. Guo, Y. Yan, M. N. Charalambides, and J. P. Dear. 2017. Deformation and damage mechanisms of laminated glass windows subjected to high velocity soft impact. *International Journal of Solids and Structures*, 109, (March 2017), 46–62. ISSN: 00207683. DOI: https://doi.org/10.1016/j.ijsolstr.2017.01.006 (cited on pages 6, 8).

[44] P. Grant, W. Cantwell, H. McKenzie, and P. Corkhill. 1998. The Damage Threshold Of Laminated Glass Structures. *International Journal of Impact Engineering*, 21, 9, (October 1998), 737–746. ISSN: 0734743X. DOI: https://doi.org/10.1016/s0734-743x(98)00027-x (cited on pages 6, 8).

[45] K. A. D. L. P. Karunarathna. 2013. Low-Velocity Impact Analysis Of Monolithic And Laminated Glass Using Finite Element Method (FEM). March. https://etheses.bham.ac.uk/id/eprint/5067/2/Kuruvita14MPhil.pdf (cited on pages 6, 8).

[46] U. A. Dar, W. Zhang, and Y. Xu. 2013. FE Analysis of Dynamic Response of Aircraft Windshield against Bird Impact. *International Journal of Aerospace Engineering*, 2013, (May 2013), 1–12. ISSN: 1687-5966. DOI: https://doi.org/10.1155/2013/171768 (cited on page 6).

[47] X. Zhang, I. K. Mohammed, M. Zheng, N. Wu, I. Mohagheghian, G. Zhang, Y. Yan, and J. P. Dear. 2019. Temperature effects on the low velocity impact response of laminated glass with different types of interlayer materials. *International Journal of Impact Engineering*, 124, (February 2019), 9–22. ISSN: 0734743X. DOI: https://doi.org/10.1016/j.ijimpeng.2018.09.004 (cited on pages 6, 8).

[48] F. Mili. 2012. Effect of the Impact Velocity on the Dynamic Response of E-Glass/Epoxy Laminated Composites. *Arabian Journal for Science and Engineering*, 37, 2, (March 2012), 413–419. ISSN: 13198025. DOI: https://doi.org/10.1007/s13369-012-0174-9 (cited on page 8).

[49] R. A. Behr, P. A. Kremer, L. R. Dharani, F. S. Ji, and N. D. Kaiser. 1999. Dynamic strains in architectural laminated glass subjected to low velocity impacts from small projectiles. *Journal of Materials Science*, 34, 23, (December 1999), 5749–5756. ISSN: 00222461. DOI: https://doi.org/10.1023/A:1004702100357 (cited on page 8).

[50] B. Liu, T. Xu, X. Xu, Y. Wang, Y. Sun, and Y. Li. 2016. Energy absorption mechanism of polyvinyl butyral laminated windshield subjected to head impact: Experiment and numerical simulations. *International Journal of Impact Engineering*, 90, (April 2016), 26–36. ISSN: 0734743X. DOI: https://doi.org/10.1016/j.ijimpeng.2015.11.010 (cited on page 8).

[51] B. H. Kim, K. C. Ahn, and C. W. Lee. 2016. Low Velocity Impact Behaviors of a Laminated Glass. *Smart Science*, 2, 4, (June 2016), 209–213. DOI: https://doi.org/10.1080/23080477.2014.11665628 (cited on page 8).

[52] Kitware. [n. d.] Visualisation Toolkit. (). vtk.org (cited on pages 9, 10, 20).

[53] H.-P. Bunge. 2009. Earth Models. Munich, (March 2009). http://www.earthmodels.org (cited on pages 9–12, 20).

[54] J. Aspnes. 2014. C/FloatingPoint. New Haven, Conneticut, (2014). http://www.cs.yale.edu/homes/aspnes/pinewiki/C(2f)FloatingPoint.html (cited on page 11).

[55] A. Munjiza. 2000. Manual for the " Y " Fem / Dem Computer Program. C, 1–19 (cited on page 12).

[56] D. Apps. [n. d.] Simple online tools. https://www.pinterest.co.uk/davinapps/simple-online-tools/ (cited on page 12).

[57] J. Malinen. 2005. B64 encoding function. (2005). http://web.mit.edu/freebsd/head/contrib/wpa/src/utils/base64.c (cited on page 13).

[58] C. Geuzaine and J.-F. Remacle. 2009. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. http://gmsh.info/ (cited on page 19).