Massachusetts Institute of Technology
Institute for Soldier Nanotechnologies

Gmsh as a toolbox for Summit

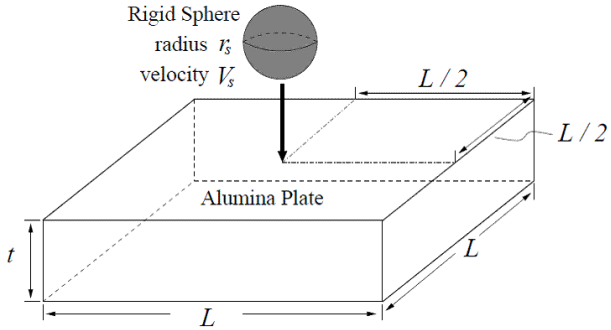
G. BECKER

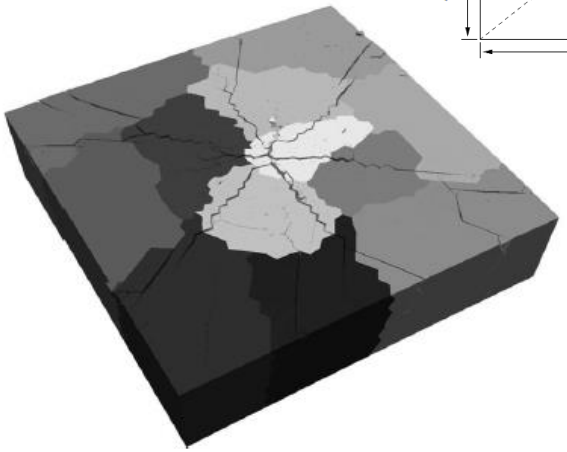
2nd Gmsh Workshop

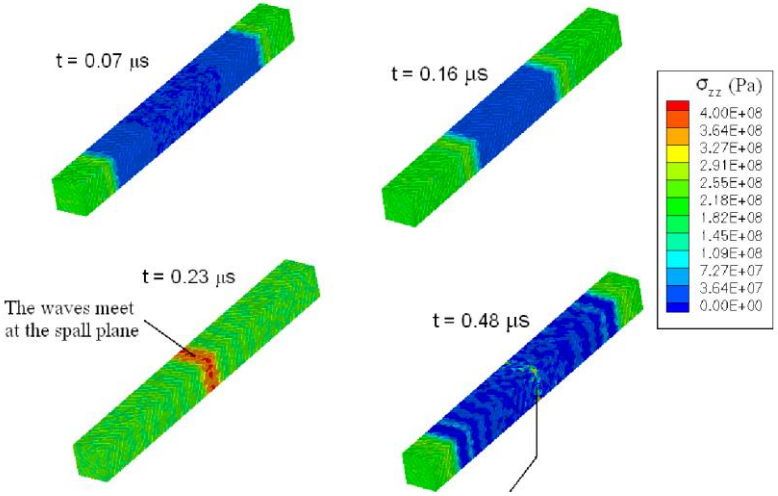
May 2013

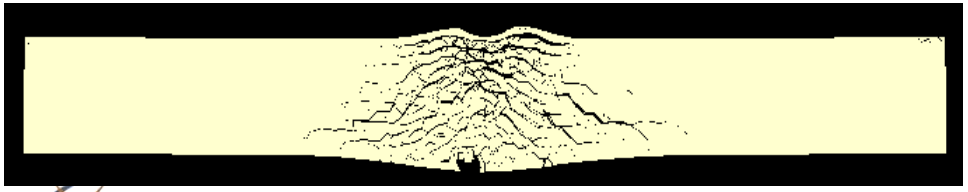
Summit

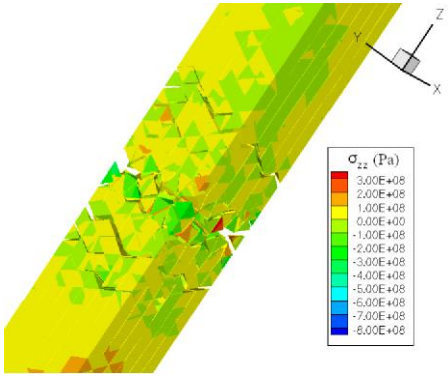
- Summit is a DG-based FE software that mainly focuses on the dynamics of large structures including fracture
  - Up to 23 billions dofs [Radovitzky 2012, Seagraves 2013]









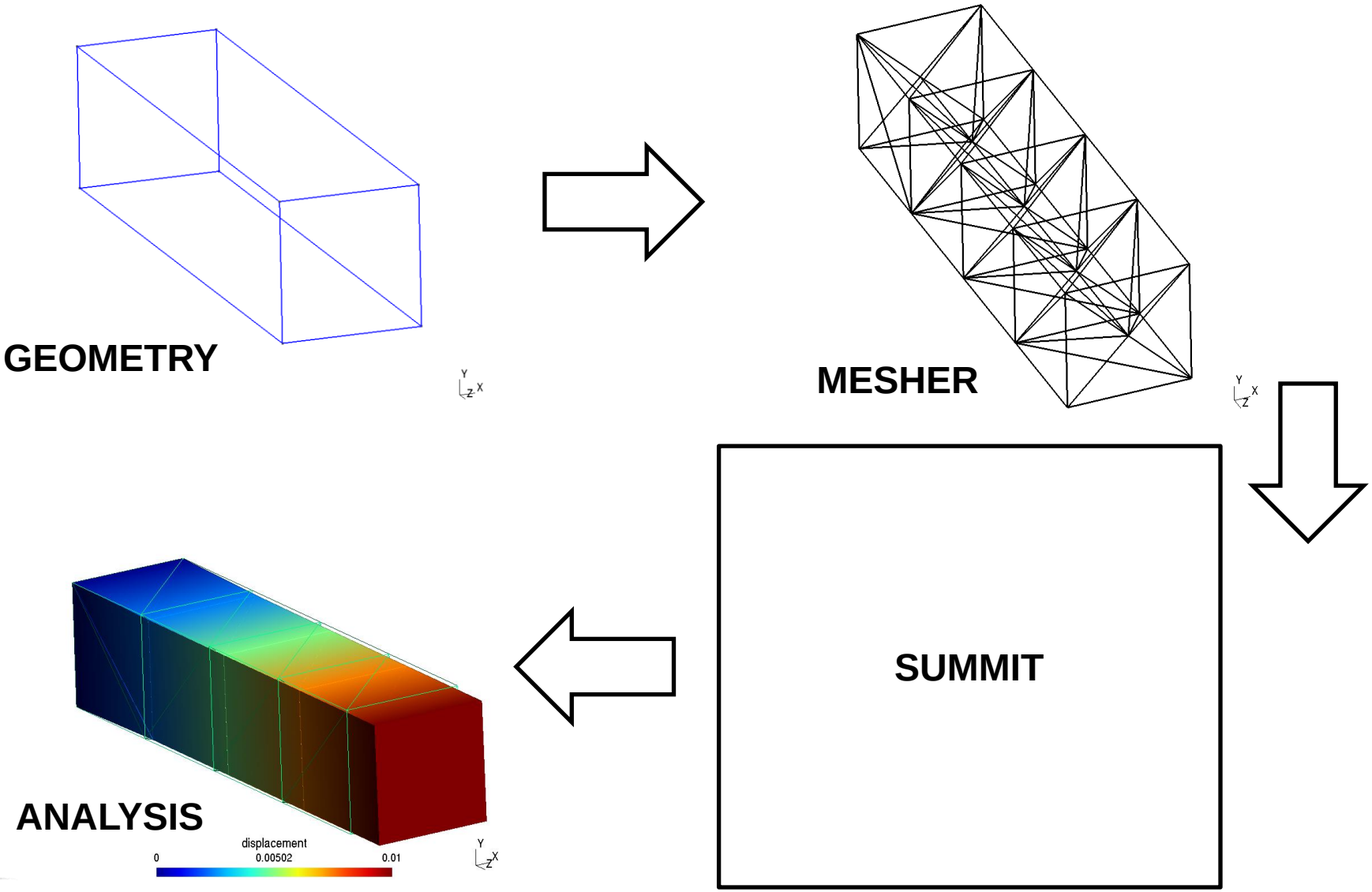


Summit

- There is no standard way to generate a mesh but the (too heavy) common procedure includes
  - (Reboot to Windows)
  - Use a mesher (Abaqus, Gmsh, …)
  - (Reboot to Linux)
  - Launch (user) conversion script (1st order triangle and tetrahedron only) to generate a summit mesh format
  - Launch computation

Summit

- This procedure can be simplify by supporting a mesher (Gmsh) format



Plan

- Gmsh mesh support for Summit
- 3D/Shell DG formulation
- Quad and hexa elements support through Gmsh
  - Debugging
  - Peridynamics
- Benchmarks with the 3D/shell DG formulation

Gmsh mesh support for Summit

- An interface to the Gmsh objects is more efficient and appealing than developing a summit ".msh" reader
  - Change in the (future) ".msh" formats
  - Support of all the Gmsh element library (even only triangle and tetrahedron are used)

Gmsh mesh support for Summit

- MeshGmsh interface class uses the GModel to build the summit::Mesh
  - Summit is linked to a libgmsh.a

```
MeshGmsh(const std::string& filename, const int elemOrder, const int spdim) : _pModel(NULL)
{
  _pModel = new GModel();
  _pModel->readMSH(filename.c_str()); // read the ".msh" file
  int dim = _pModel->getDim();
  // Allow to give a spatial dimension != from the geometric one (eg for shell dim=2 but spatialDim=3)
  int spatialDim = (spdim!=-1) ? spdim : dim;

  // allocate storage for the nodes
  std::vector<real> coordinates(spatialDim * _pModel->getNumMeshVertices() );

  int nodecount,j;
  int lastver = _pModel->getMaxVertexNumber()+1;
  for(j=1,nodecount=0;j!=lastver;++j,nodecount+=spatialDim)
  {
      MVertex* ver = _pModel->getMeshVertexByTag(j);
      coordinates[nodecount] = ver->x(); coordinates[nodecount+1] = ver->y();
      if(spatialDim>2) coordinates[nodecount+2] = ver->z(); // summit could be 2D or 3D

  }
  // create the geometry of the mesh
  _geometry = geometry_t(new MeshGeometry(spatialDim, _pModel->getNumMeshVertices(), coordinates));

  ... // some operations

  // Mesh_Topology
  this->readOneDim(dim,spatialDim,elemOrder,total1stOrderNodes,physgroups,mapVertexNum);
  // in case of element of different dim in the mesh
  if(dim==3)
    this->readOneDim(2,spatialDim,elemOrder,total1stOrderNodes,physgroups,mapVertexNum);

  // end of constructor
  return;
}
```

Create a GModel

Extract the needed information from the GModel

Create the required summit objects

Gmsh mesh support for Summit

- MeshWriterGmsh class put the results in a suitable form for the PViewData such a way that Summit does not write the output files itself

```
void
MeshWriterGmsh::InsertNodalField(const NodalField<real>& nodal_field,const std::string &viewname)
{
    PViewDataGModel nodalView(PViewDataGModel::ElementNodeData); // Gmsh view
    std::map<int,std::vector<real> > data; // Gmsh output format

    if(physnum<100) // If physnum < 100 CG formulation --> field is stored by vertices
    {
        ...
    }
    else // DG formulation --> field is stored by element
    {
        groupOfElements::elementContainer::iterator ite;
        int indexVertex=0;
        for(ite = mygroup.begin() ; ite!=mygroup.end();++ite){
            MElement *ele = *ite;
            int elenum = ele->getNum();
            int nvertices = ele->getNumVertices();
            nodedata.resize(3*nvertices);
            for(int j=0;j<nvertices;++j,++indexVertex){
                for(int comp=0;comp<meshdim;++comp)
                   nodedata[3*j+comp] = (double)nodal_field(indexVertex,comp);
                if(meshdim==2) nodedata[3*j+2] = 0.; // 2D --> 3D
            }
            data.insert(std::pair<int,std::vector<real> >(elenum,nodedata));
        }
    }
    nodalView.addData(_mesh,data,step,(double) time,Msg::GetCommRank(),ncomp);
}

void
MeshWriterGmsh::Write(int step, real time) const {
    nodalView.writeMSH(_filename, 2.2,false, wmel,true);
}
```

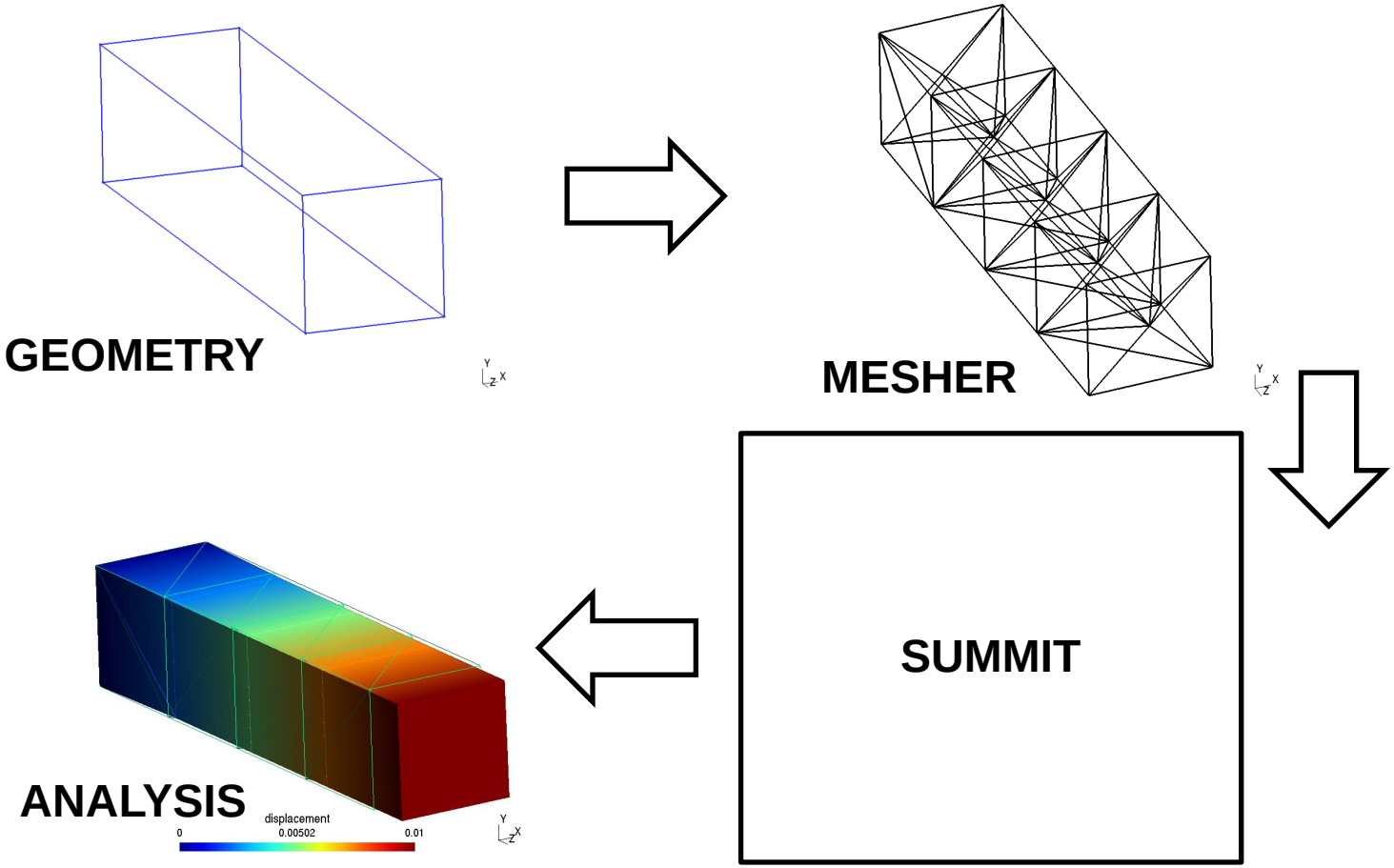
Create a Gmsh view

Put the Summit data into the required form

Write via Gmsh

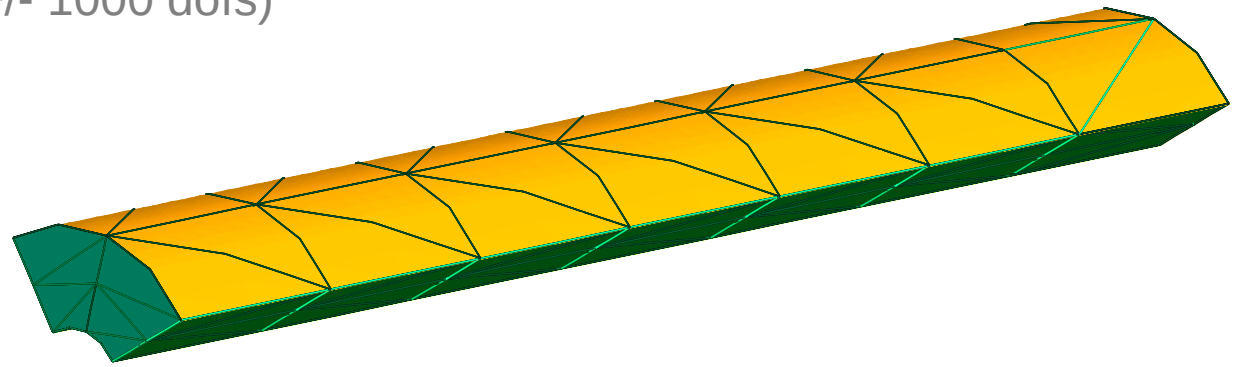
Gmsh mesh support for Summit

- 2 interfaces class (< 900 code lines) allows to support all the Gmsh elements library (even if only triangle and tetrahedron are used) by linking a “light” Gmsh lib (Mesh and Post modules)

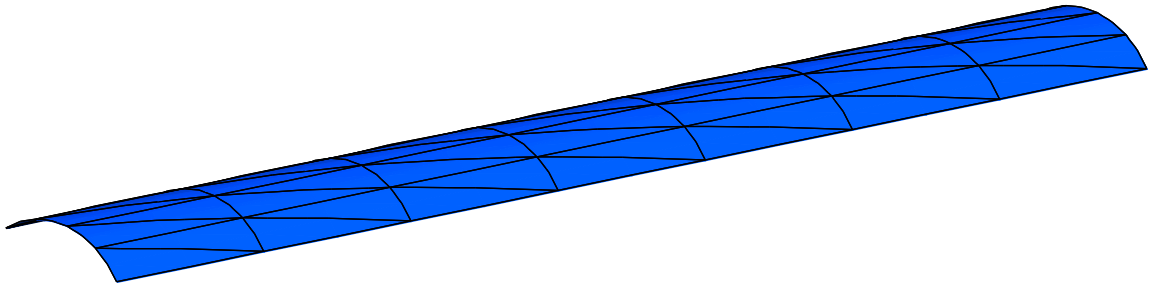


3D/shell motivation

- Shell formulation reduces the computational time
  - 3D (+/- 1000 dofs)

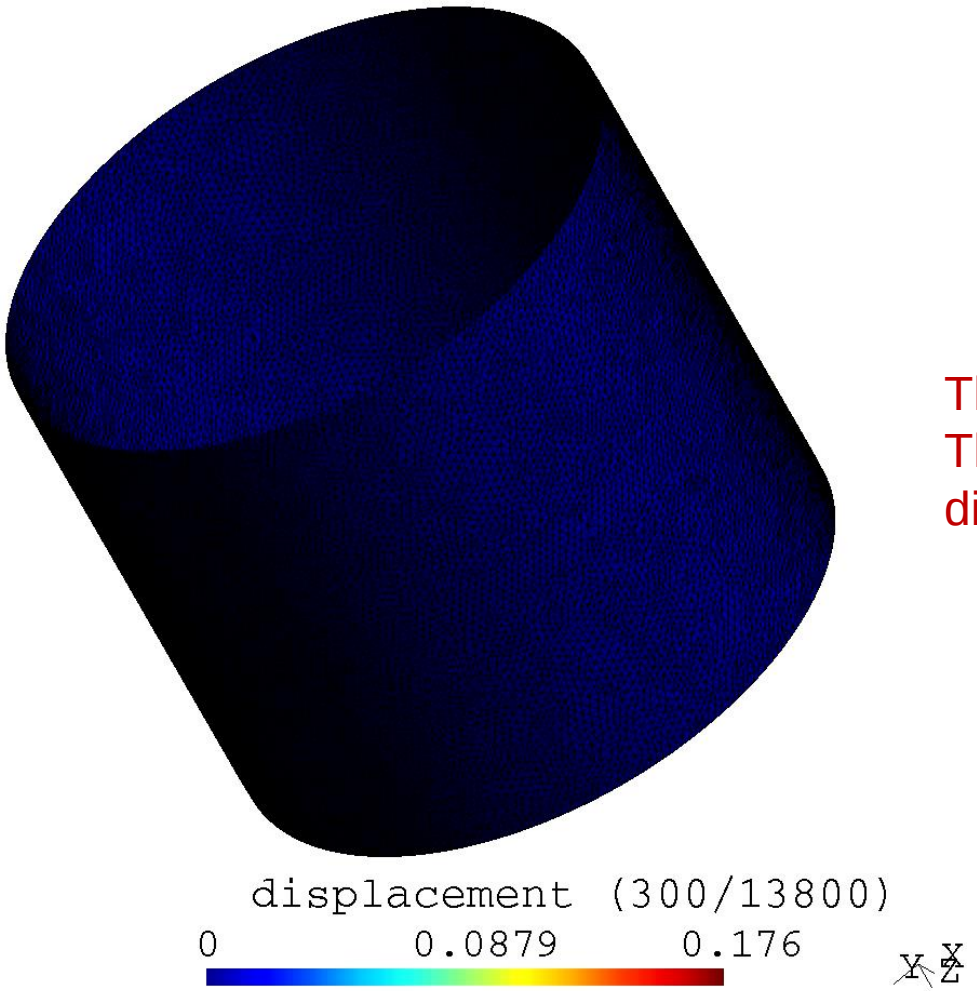


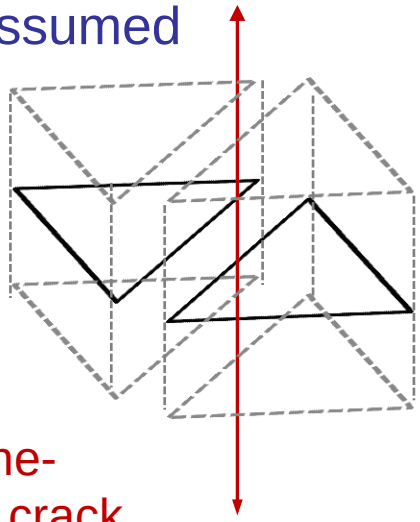
  - Shell (+/-500 dofs)



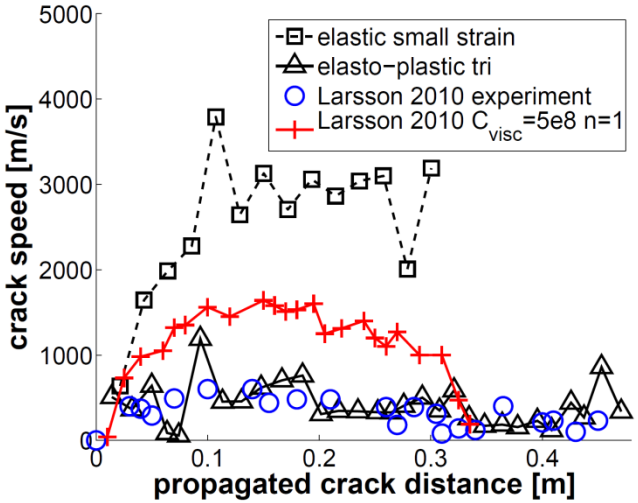
3D/shell motivation

- The through-the-thickness crack path cannot be captured by shell elements as normal propagation to mid-plane is assumed



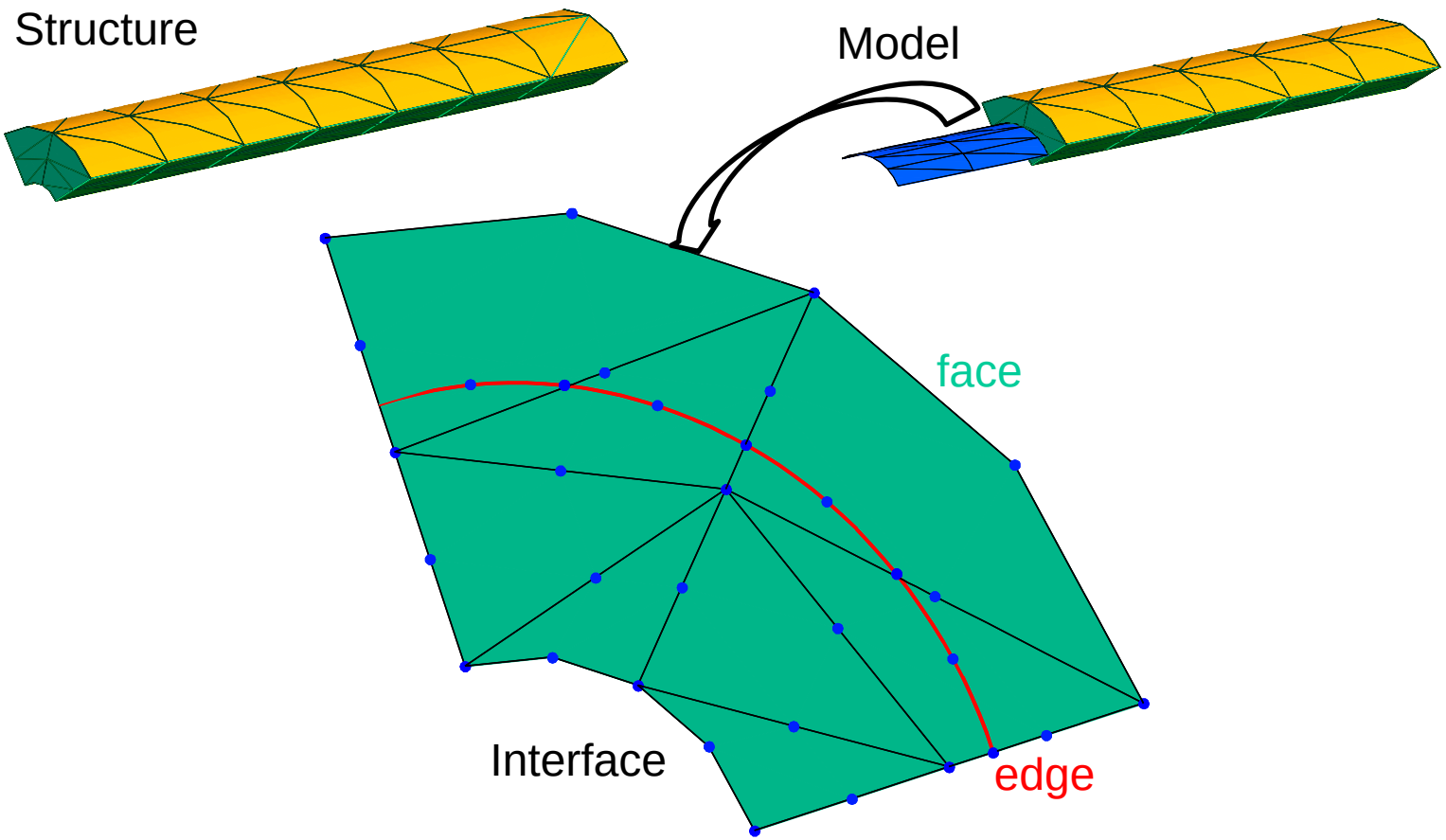


Through-the-Thickness crack direction



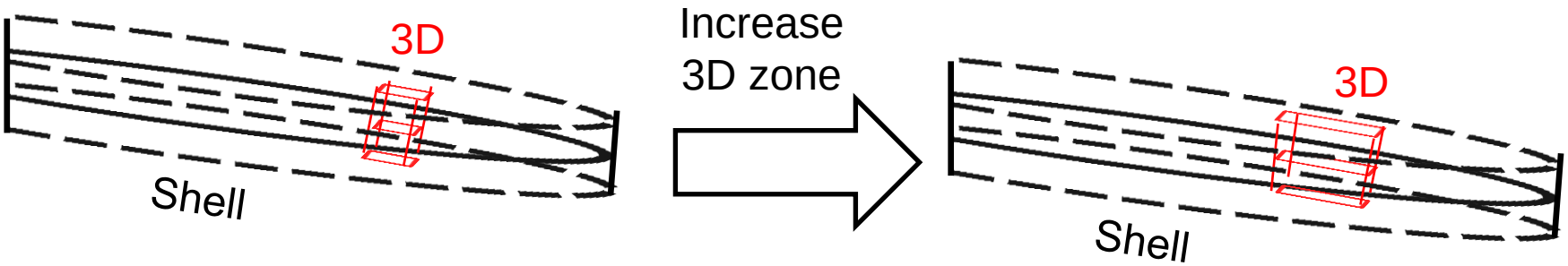
3D/shell motivation

- The continuity (displacements & efforts) at the interface between shell & 3D elements is the main issue
  - An edge (Shell) “must be linked” to a face (3D)



3D/shell motivation

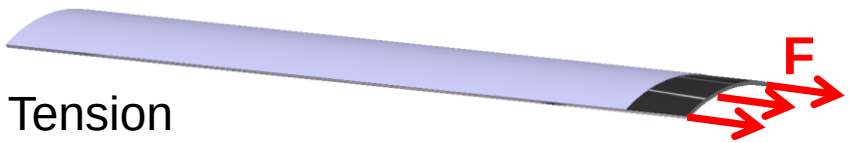
- The technique developed in literature (Rigid body, works equivalence, transition elements, displacement compatibility) are limited to the linear range or leads to perturbation at the interface
  - The interface is generally put far for the “3D effects zone” to minimize its effects on the solution



  - Increase the Computational cost

3D/shell DG formulation

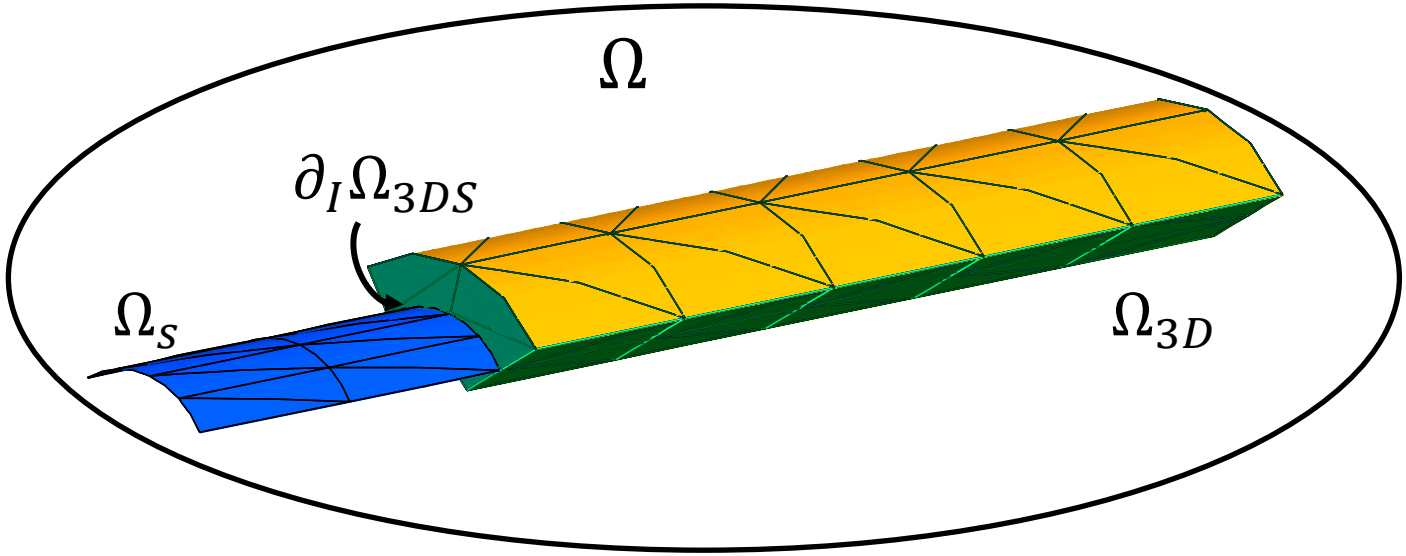
- Hereafter, the development of the method is limited to the (linear) membrane deformation mode to verify the wave stress propagation at interface
  - Membrane mode: extension or compression



  - Aim: Verify that the method allows to propagate a stress wave through a 3D/Shell interface without spoiled it

3D/shell DG formulation

- Using DG method the interface between 3D and shell appears naturally in the equation and therefore can be accounted easily

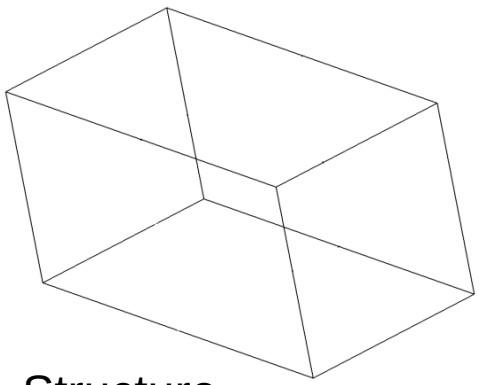


$$\int_{\Omega_s} \delta\nabla\varphi : \sigma dV - \int_{\Gamma_s} \delta\varphi \cdot \sigma \cdot N_s dS + \int_{\Omega_{3D}} \delta\nabla\varphi : \sigma dV - \int_{\Gamma_{3D}} \delta\varphi \cdot \sigma \cdot N_V dS = 0$$

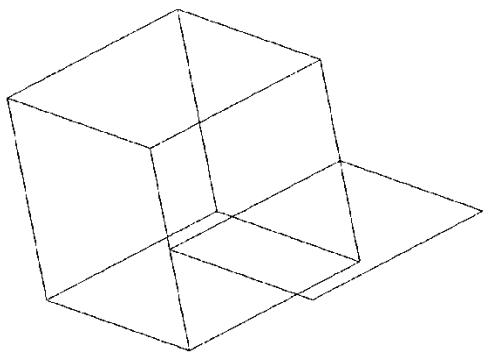
$$\int_{\mathcal{A}} \left(\bar{j} n^{\alpha}\right)_{,\alpha} \cdot \delta\varphi d\mathcal{S} + \int_{\Omega_{3D}} \sigma : \delta\epsilon dV + \int_{\partial_I \Omega_{3DS}} [\![\delta\varphi \cdot \sigma]\!] \cdot N_S dS = 0$$

3D/shell DG formulation

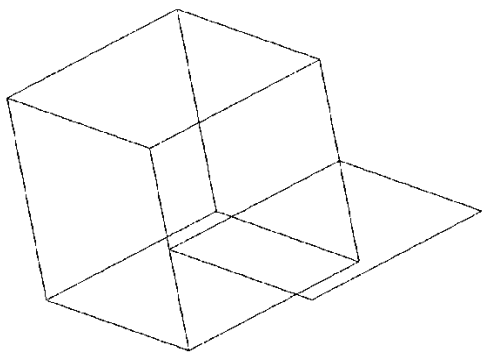
- The interface terms are developed with the idea that a side see the other one in its kinematics
  - Works as “weak BCs” are applied at the interface



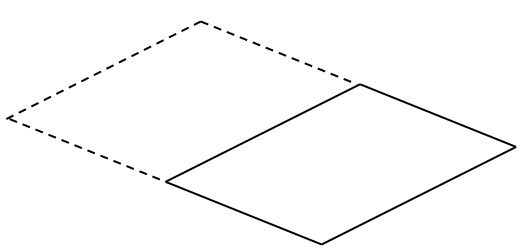
Structure

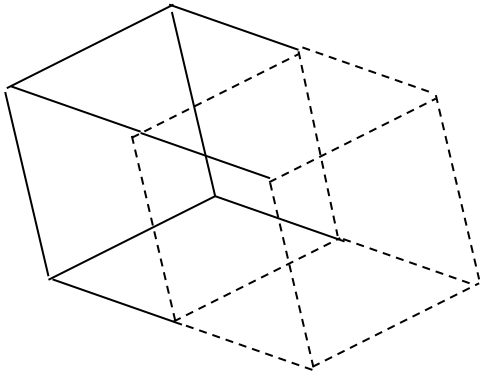


Hybrid model
Shell interface



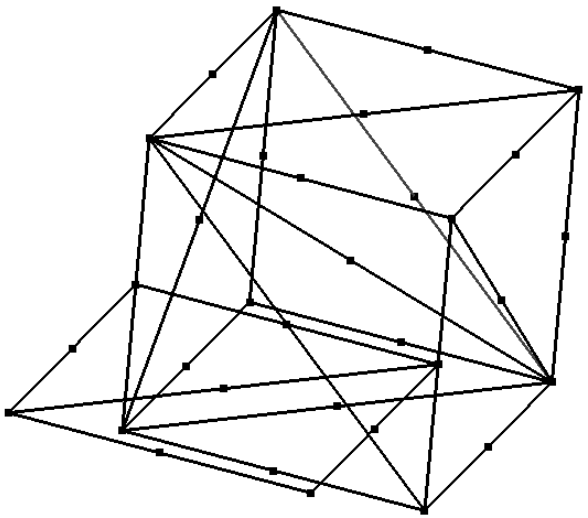
Hybrid model
3D interface

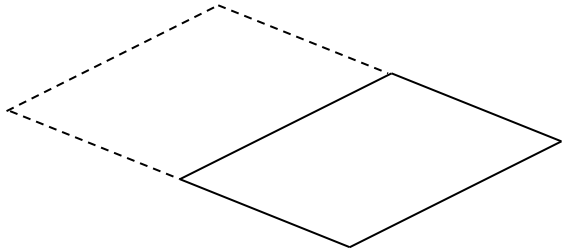




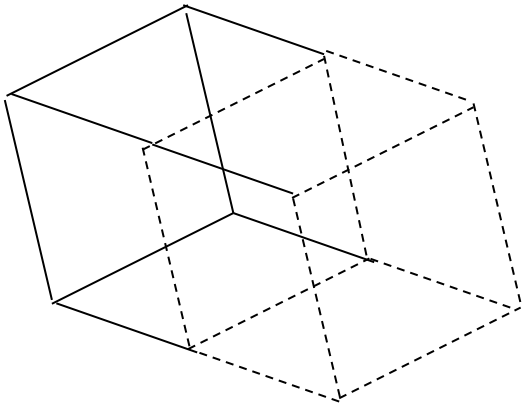
3D/shell DG formulation

- Unstructured meshes (triangle and tetrahedron) are the best choice to propagate crack at inter element boundaries but they are very complex to debug





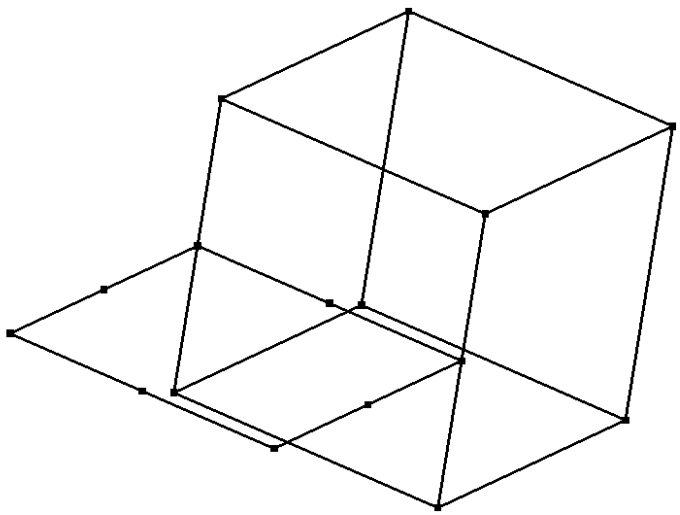
“Ideal” Shell interface debugging

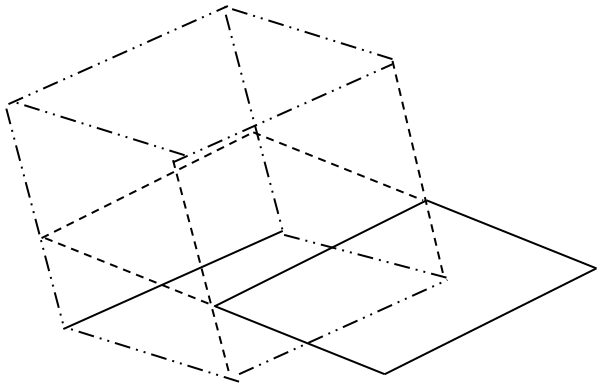


“Ideal” 3D interface debugging

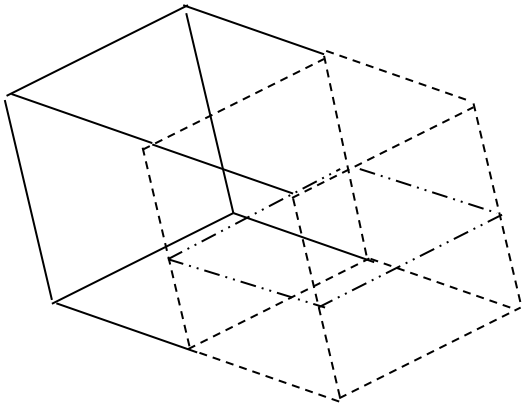
3D/shell DG formulation

- A structured mesh is very appealing to debug but the quadrangle and hexahedron are not supported in Summit.





“Ideal” Shell interface debugging



“Ideal” 3D interface debugging

Quad and hexa elements support through Gmsh

- Use the Gmsh ability to compute the Gauss points and the shapes (with their derivatives) in a Summit class ElementSet3DBulkGmsh

```
summit::ElementSet3DBulkGmsh::
ElementSet3DBulkGmsh(size_t topo, const int pOrder,const int nen,
                     const std::vector<int>& connectivity,
                     const int nelements,
                     const std::vector<int>& globalElementId,
                     const std::vector<MElement*> &velement,
                     const int materialLabel) :
  ElementSetBulk(3, topo, nelements, nen, pOrder, materialLabel, globalElementId),
  _gmsh_type(velement[0]->getType()), _groupElem(velement), _SampleWeight(0)
{
    MElement* ele = _groupElem[0];
    Get the quadrature points
    IntPt *GP;
    int nquad;
    ele->getIntegrationPoints(2*_pOrder,&nquad,&GP);
    real ival[256];
    _shape.resize(1, _nquad, _nen); // shape summit container
    real * shape = _shape.local(elem_t(0), quad_t(0));
    real* shp = &ival[0];
    for(size_t q(0); q<_nquad;++q)
    {
        _SampleWeight[q] = GP[q].weight; // weight summit container
        ele->getShapeFunctions(GP[q].pt[0],GP[q].pt[1],GP[q].pt[2],ival);
        for (size_t a=0; a<_nen; ++a,++shp) {
            shape[a] = *shp;
        }
        shape += _nen;
    }
    ...
    // idem Grad and Hess

    return;
}
```

Derive from the Summit class containing the discretization

Compute Gauss point and shape functions through Gmsh

Store them in the summit containers

Quad and hexa elements support through Gmsh

- The vtk output for quad and hexa is easily implemented in Summit as it is the default output format

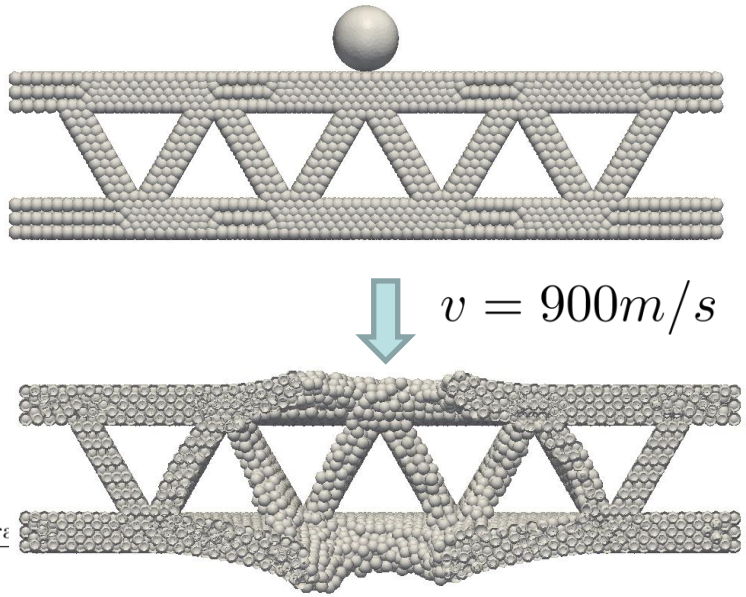
```
void
summit::MeshWriterVTK::
LoadMeshData(const FunctionSpace& fs)
{   ... // init operation

    // ElementSetBulk3DGmsh::vtkType(){return _groupElem[0]->getTypeForVTK();}
    switch(element_sets->vtkType()){
       case VTK_TRIANGLE :
         cell = vtkTriangle::New();
         break;
       case VTK_HEXAHEDRON:
         cell = vtkHexahedron::New();
         break;
       case VTK_TETRA:
         cell = vtkTetra::New();
         break;
       case VTK_QUAD:
         cell = vtkQuad::New();
         break;
    }
    for (size_t e = 0; e < element_sets->elements(); ++e) {
        int npel = element_sets->dof_map()->NodesElement(e);
        const int* lconn = element_sets->dof_map()->Connectivity(e);
        for (int j = 0; j < npel; ++j)
            cell->GetPointIds()->SetId(j, lconn[j]);
        _mesh->InsertNextCell(cell->GetCellType(), cell->GetPointIds());
        vtk_array->InsertNextTupleValue(&label);}
    cell->Delete();
    // insert array into output mesh
    _mesh->GetCellData()->AddArray(vtk_array);
    // clean up array
    vtk_array->Delete();
    return;
}
```

Quad and hexa elements support through Gmsh

- The peridynamics is a particle based method that allow to account for very large deformations that is developed in Summit [M. Tupek]

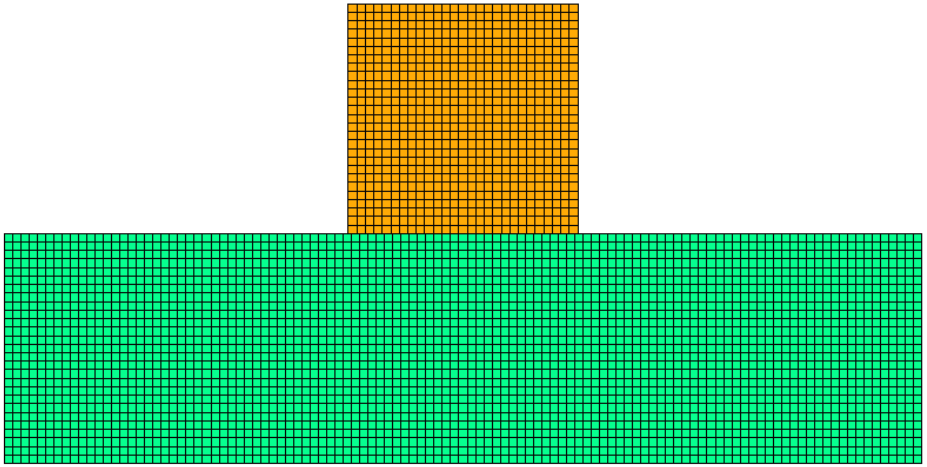
2394 particles 16929 particles 130563 partilces

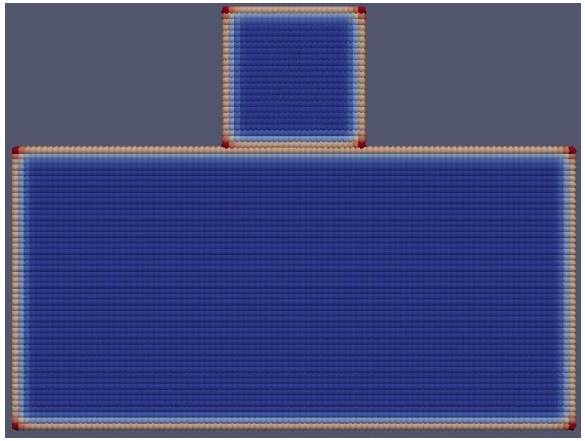
	Final length (mm)	Max. radius (mm)	Max. eff. plastic stra
Kamoulakis, 1990	21.47-21.66	7.02-7.12	2.47-3.24
Zhu and Cescotto, 1995	21.26-21.49	6.89-7.18	2.47-3.24
Camacho and Ortiz, 1997	21.42-21.44	7.21-7.24	2.97-3.25
Li, Habbal and Ortiz, 2010	21.43	6.8	3.0
Peridynamics, mesh 1	21.5	7.1	2.69
Peridynamics, mesh 2	21.4	7.5	2.88
Peridynamics, mesh 3	21.4	7.4	3.29

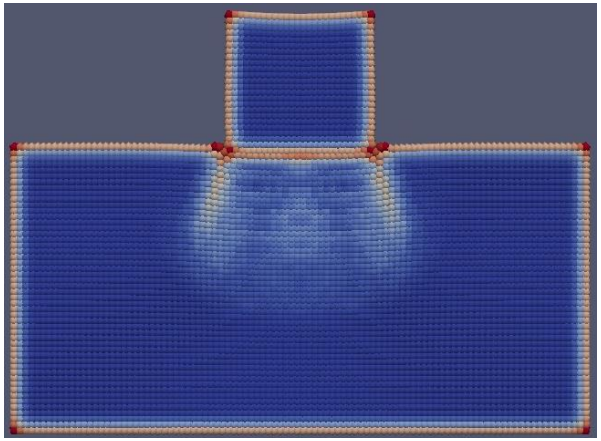


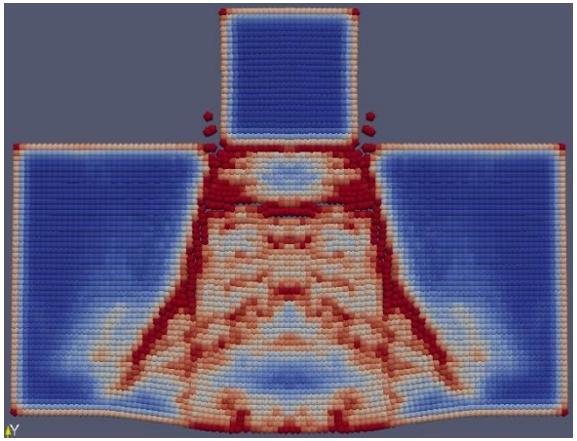
Quad and hexa elements support through Gmsh

- Structured meshes are available for peridynamics using the MeshGmsh class (each node becomes a particle)



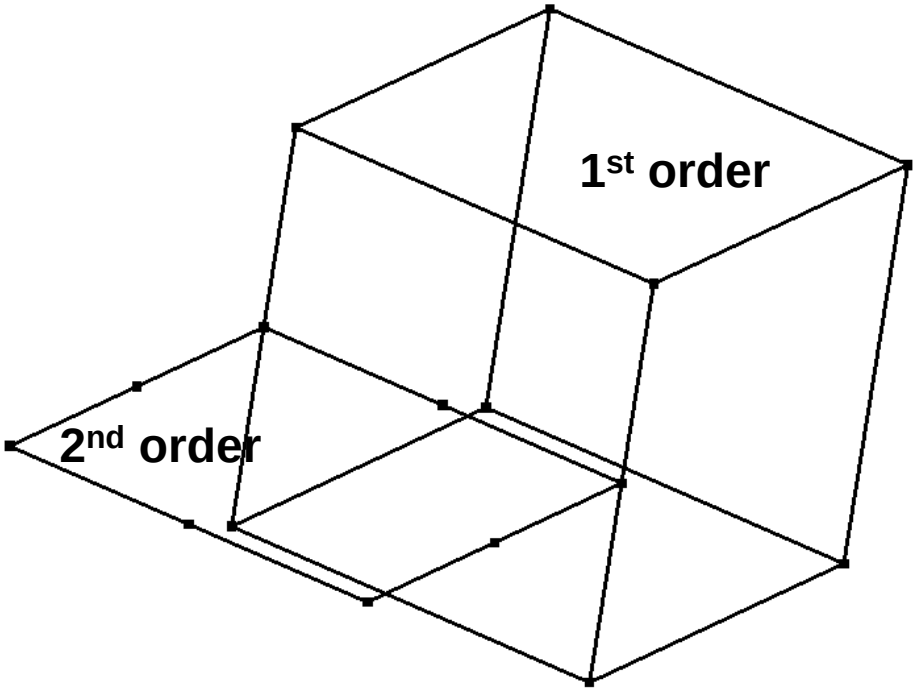






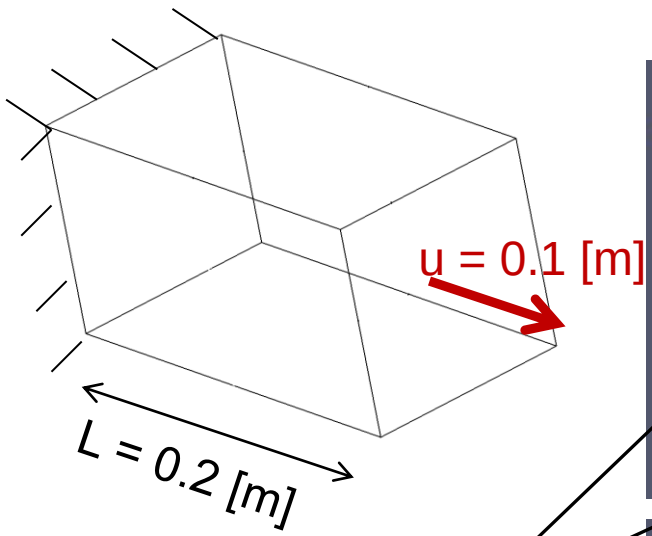
Benchmarks with the 3D/shell DG formulation

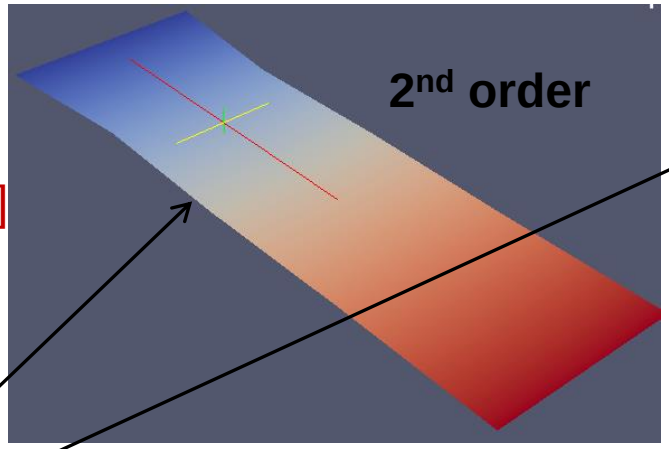
- A mesh with 1st order 3D and 2nd order shell elements can be easily generated in Gmsh
  - The polynomial order are not the same in general for 3D and shell
  - No nodal connectivity between the shell and the 3D element

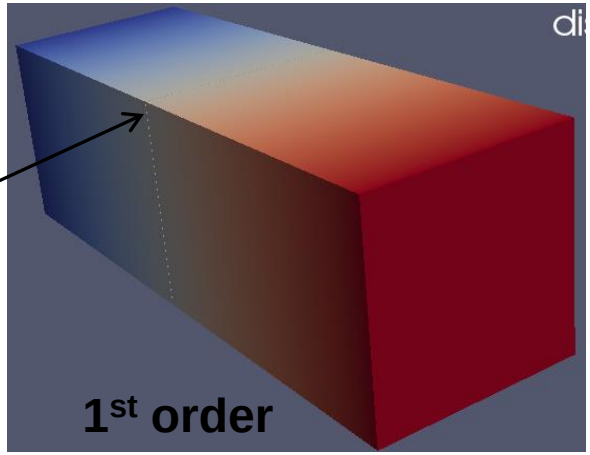


Benchmarks with the 3D/shell DG formulation

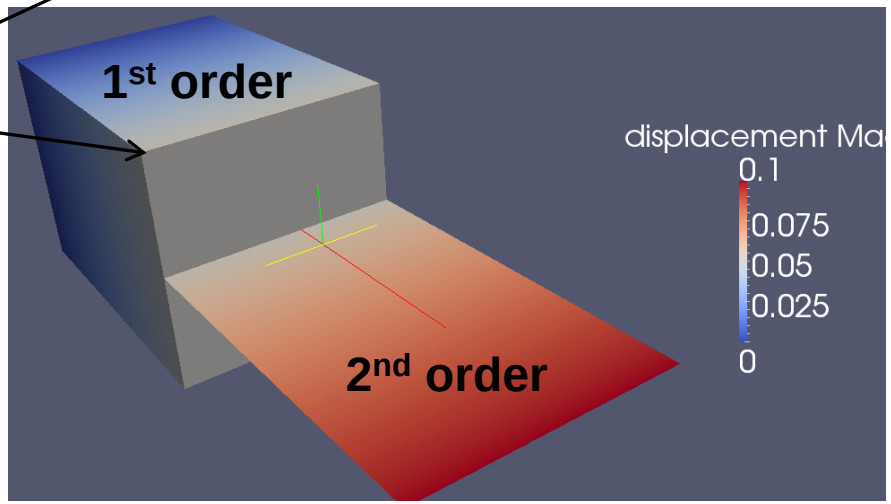
- For a simply clamped square beam loaded statically as benchmark, with 2 elements (1 cube and 1 quad-shell), the 3D/shell interface provides the same solution as 3D or shell formulation





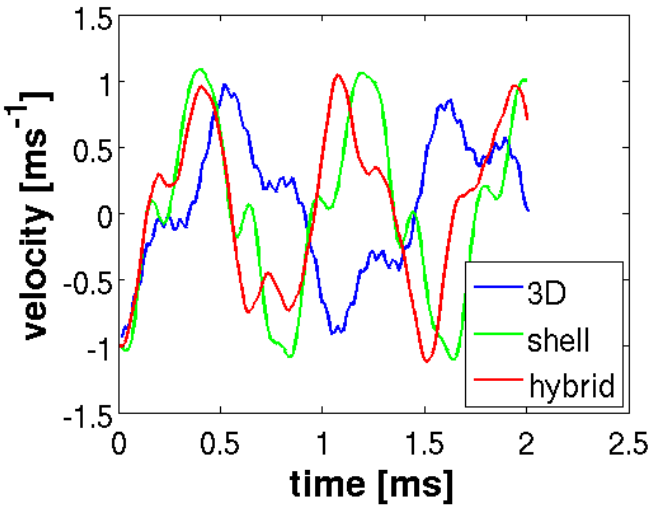


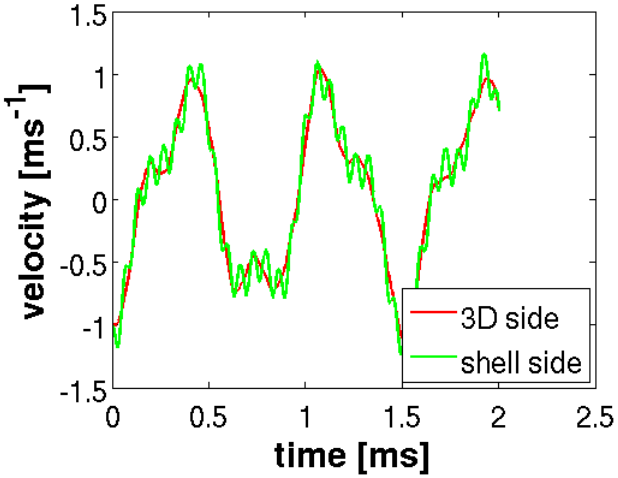
Displacement at middle = 0.05 [m]

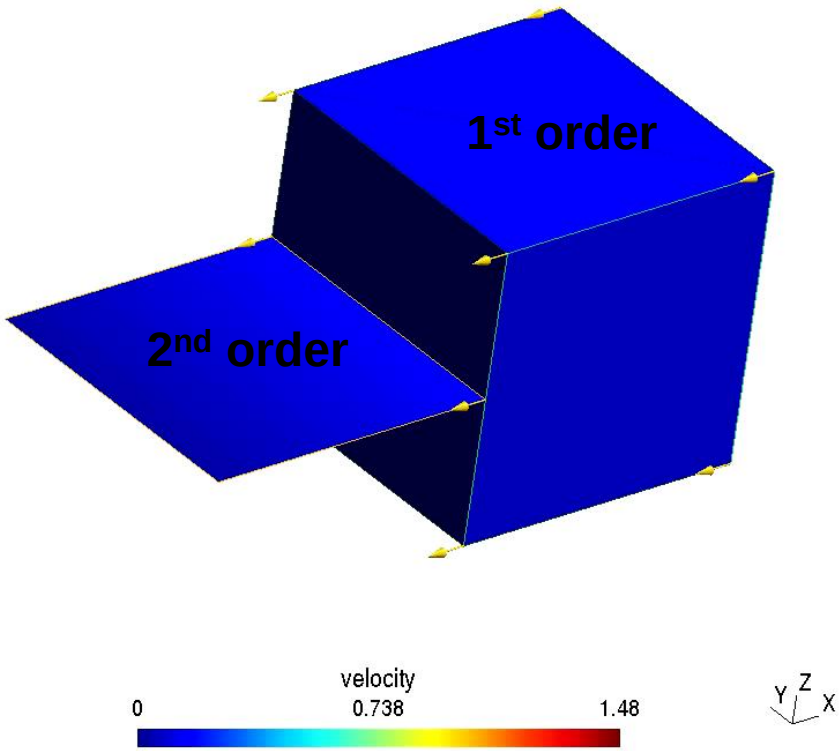


Benchmarks with the 3D/shell DG formulation

- A stress wave can propagate through the 3D/shell interface without being spoiled







Conclusions

- The recourse to the Gmsh library allows to extend the capabilities of Summit (or other solver) without significant effort
  - Avoid fastidious conversion script
  - Structured mesh support (peridynamics, debug)
  - All Gmsh elements can (in principle) be used including pyramid, prism, …
  - Direct mesh with different orders (shell 2nd order, 3D 1st order)
  - Light library including only Mesh and Post modules (no external dependency)
  - +/-1500 code lines