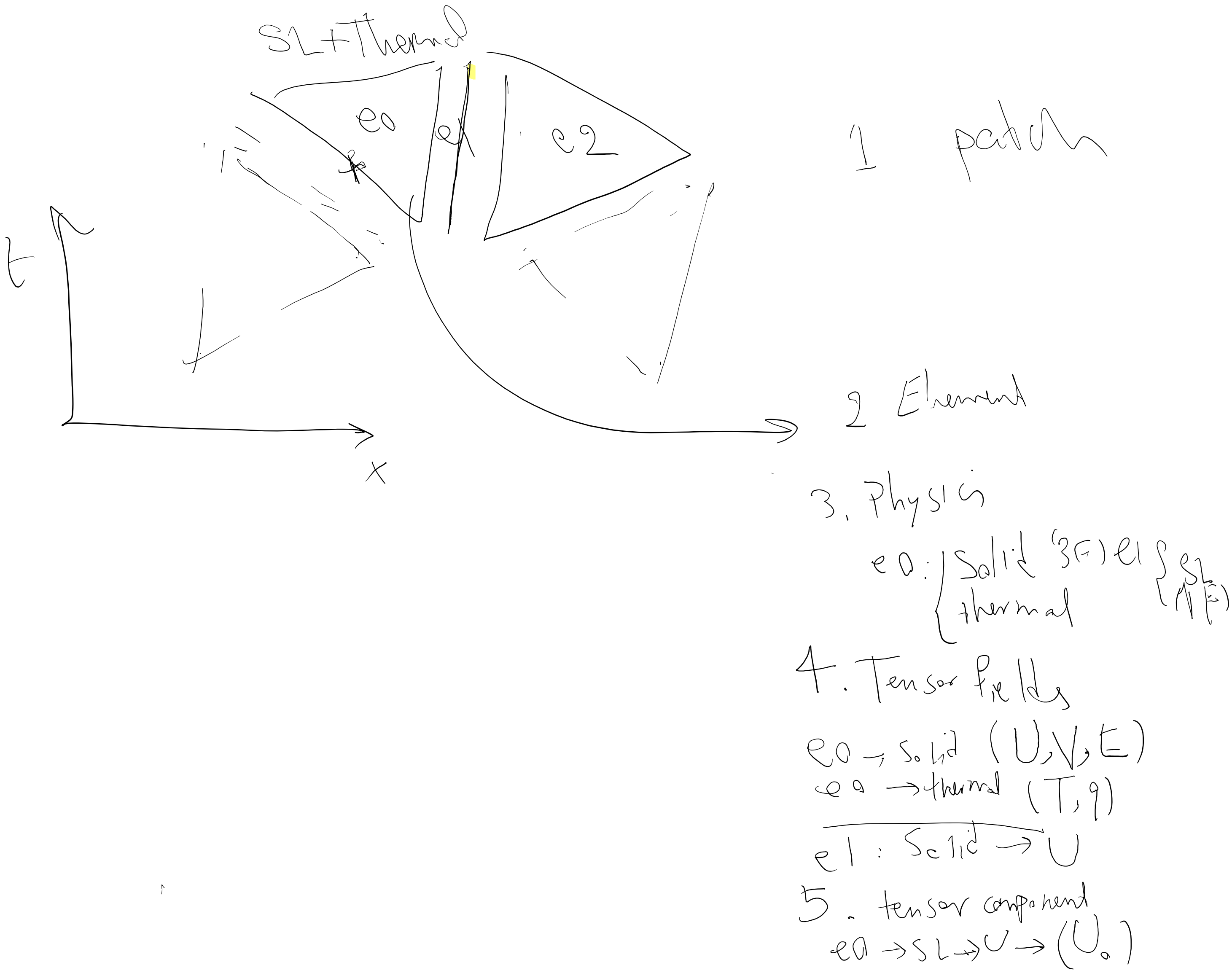
DG2020/04/29

Wednesday, April 29, 2020 11:40 AM

Second hierarchy: PhyPatch -> PhyElement -> PhyPhysics -> tensor fields -> component



Level 1: PhyElement

In PhyPatch

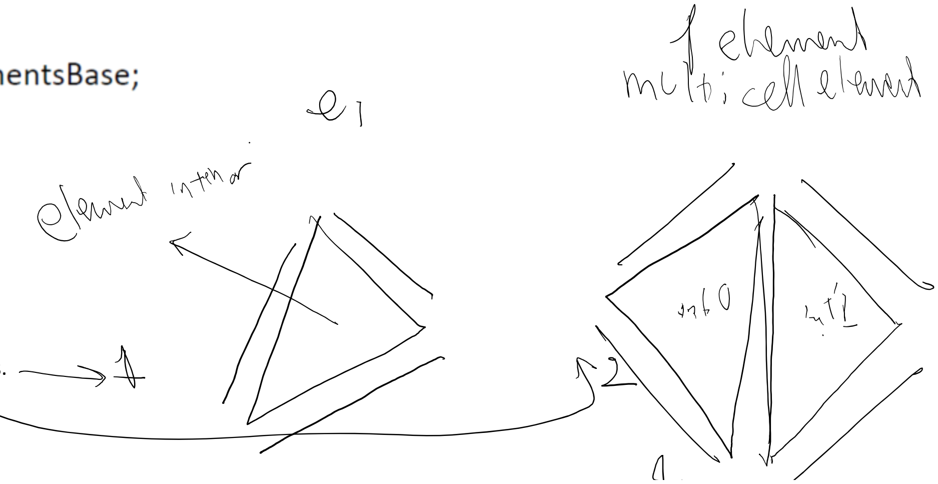
vector<PhyElementBase*> phyElementsBase;

This is a list of elements in the patch.

class PhyElementBase
{
....

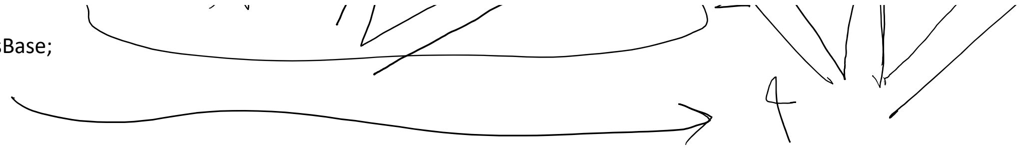
vector<PhyElementBaseInterior*> interiorCellsBase;
int numInteriorCellsBase; ///< number of Interior Cells.

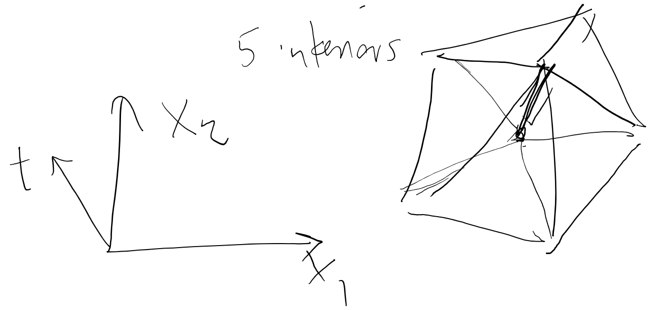
vector<PhyElementBaseFacet*> facetCellsBase;
int numFacetCellsBase;



```
vector<PhyElementBaseFacet*> facetCellsBase;
int numFacetCellsBase;
```

→ 3

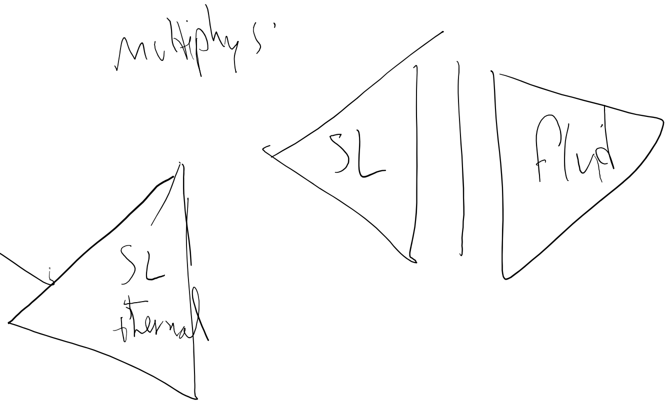




SL, thermal

```
vector<PhyPhysics*> physics;      // the physics in the element
int     num_physics;
```

2



we have a list of pointers

→ Polymorphism & having virtual functions for different physics (e.g diff WR, ...)

Members of PhyPhysics and creation of PhyPhysics

For specific physics we need to derive them from a base PhyPhysics class.

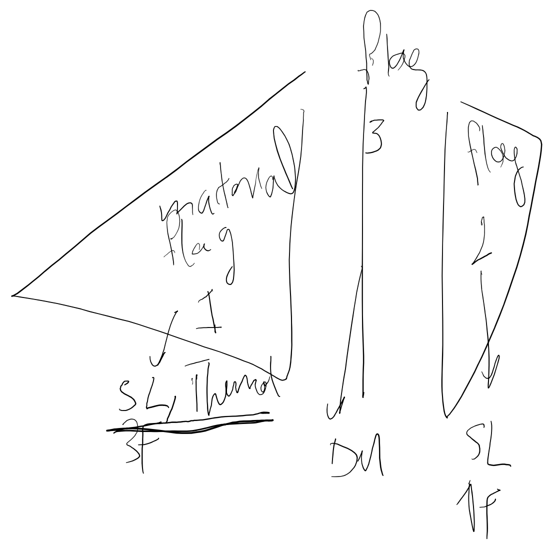
There are many specific physics implementations. We use the notation of factory to create them.

PhyPhysics are created by a factory:

Physics/PhysicsFactory.h

```
PhyPhysics* createPhysics(subConfigRef subConRef)
{
    PhyPhysics* pp;
    int subConfigIndex = subConRef.subConfigIndex;
    int option;
    switch(subConRef.formulationT)
    {
        case CL:
            option = phyConf->subConf[subConfigIndex]->
            physics_options(0);
            pp = createCLInstance(option);
            break;
        case SL:
            pp = new SLPhysics();
            break;


// the use of the factory in PhyElement
void PhyElementBase::setPhysics()
{
    num_physics = descProp.subConfigRefs.size();
    physics.resize(num_physics);
    for(int i = 0; i < num_physics; i++)
    {
        physics[i] =
        createPhysics(descProp.subConfigRefs[i]);
        physics[i]->phyLocInElement = i;
        physics[i]->peParent = this;
//          physics[i]->patch = patch;
    }
}
```



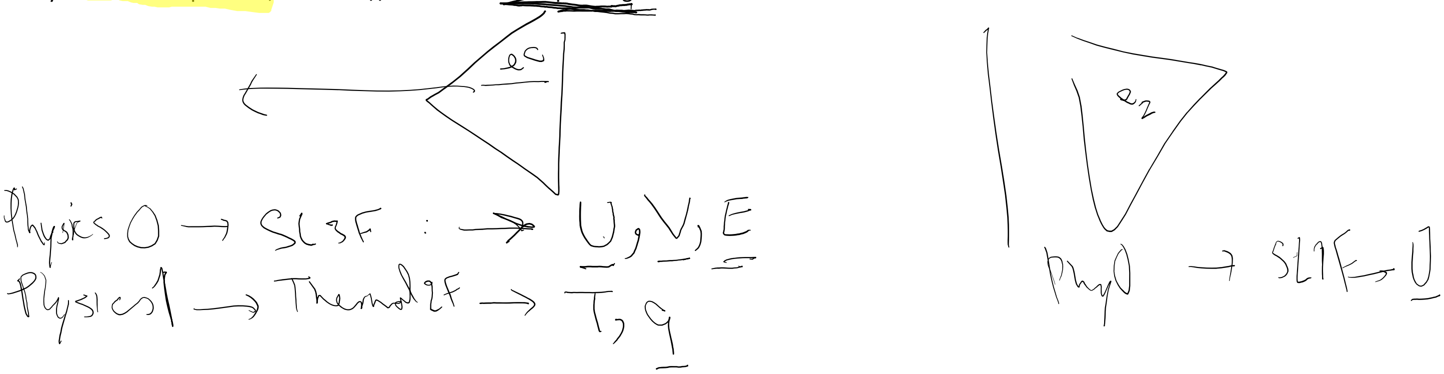
By using this function, we have created the vector of PhyPhysics inside the element.

Level 2: PhyPhysics
What is inside PhyPhysics?

```
class PhyPhysics
{
virtual bool IntegrandFacet_intSAssembly_inflowT(double factor, int e_Index, ptCoords& crds, PhyIntCellBase* pic, int quadPN, PhyFieldVals& fldVals);
virtual bool IntegrandFacet_intSAssembly_outflowT(double factor, int e_Index, ptCoords& crds, PhyIntCellBase* pic, int quadPN, PhyFieldVals& fldVals);
virtual bool IntegrandFacet_intSAssembly_interiorT(double factor, int e_Index, ptCoords& crds, PhyIntCellBase* pic, int quadPN, PhyFieldVals& fldVals);
virtual bool IntegrandFacet_intSAssembly_boundaryT(double factor, int e_Index, ptCoords& crds, PhyIntCellBase* pic, int quadPN, PhyFieldVals& fldVals);
posDof                              physicsDof;         // the dof for PhyPhysics


vector<PhyTensorField>  pTFields;             //tensor fields interpolating the element
```

```
vector<PhyTensorField>  pTFields;            //tensor fields interpolating the element
```



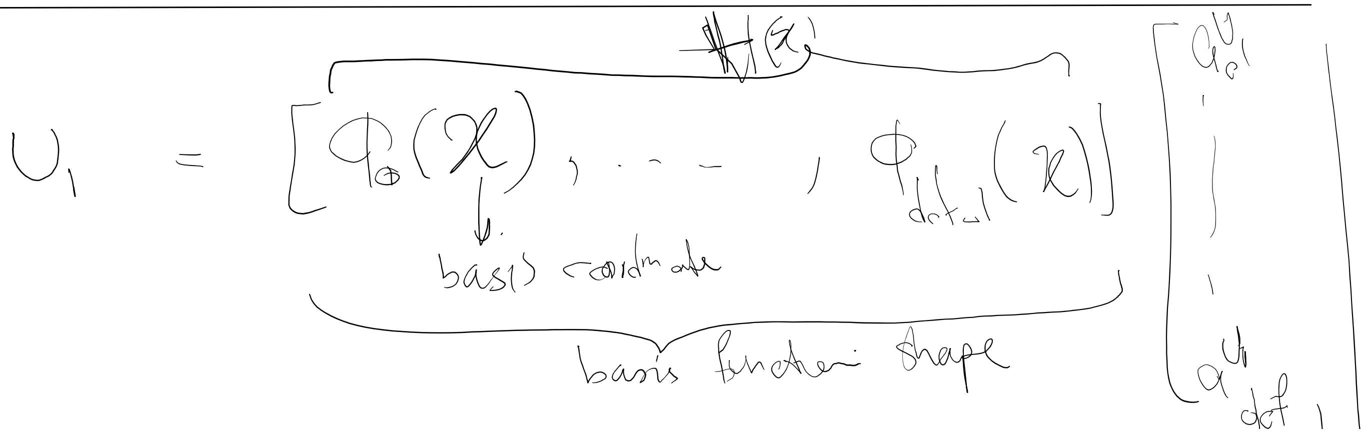
Level 3: PhyTensorField

```
class PhyTensorField
{
void ComputeDivHDX(ptCoords& crds, PhyFieldVals& fldVals, IntHStorage& basisShapes, int e_Index, vsT cVH, rotT rT);
void ComputeCurl(ptCoords& crds, PhyFieldVals& fldVals, IntHStorage& basisShapes, int e_Index, vsT cVH, rotT rT);

vTensor<phyField>      physicsFs;
int num_physicsFs

posDof                  tenFDof;        // the dof and pos for the tensor field;
}
```

Level 4:
PhyField



basis function shape

$[a^{U_1}_{dof\_1}]$

coefficients

class phyField
{
PhyBasisElement pBasis; // Basis for the field

Basis functions

pCoef

$x \longrightarrow H(x)$

$$\frac{\partial H}{\partial x_i}$$

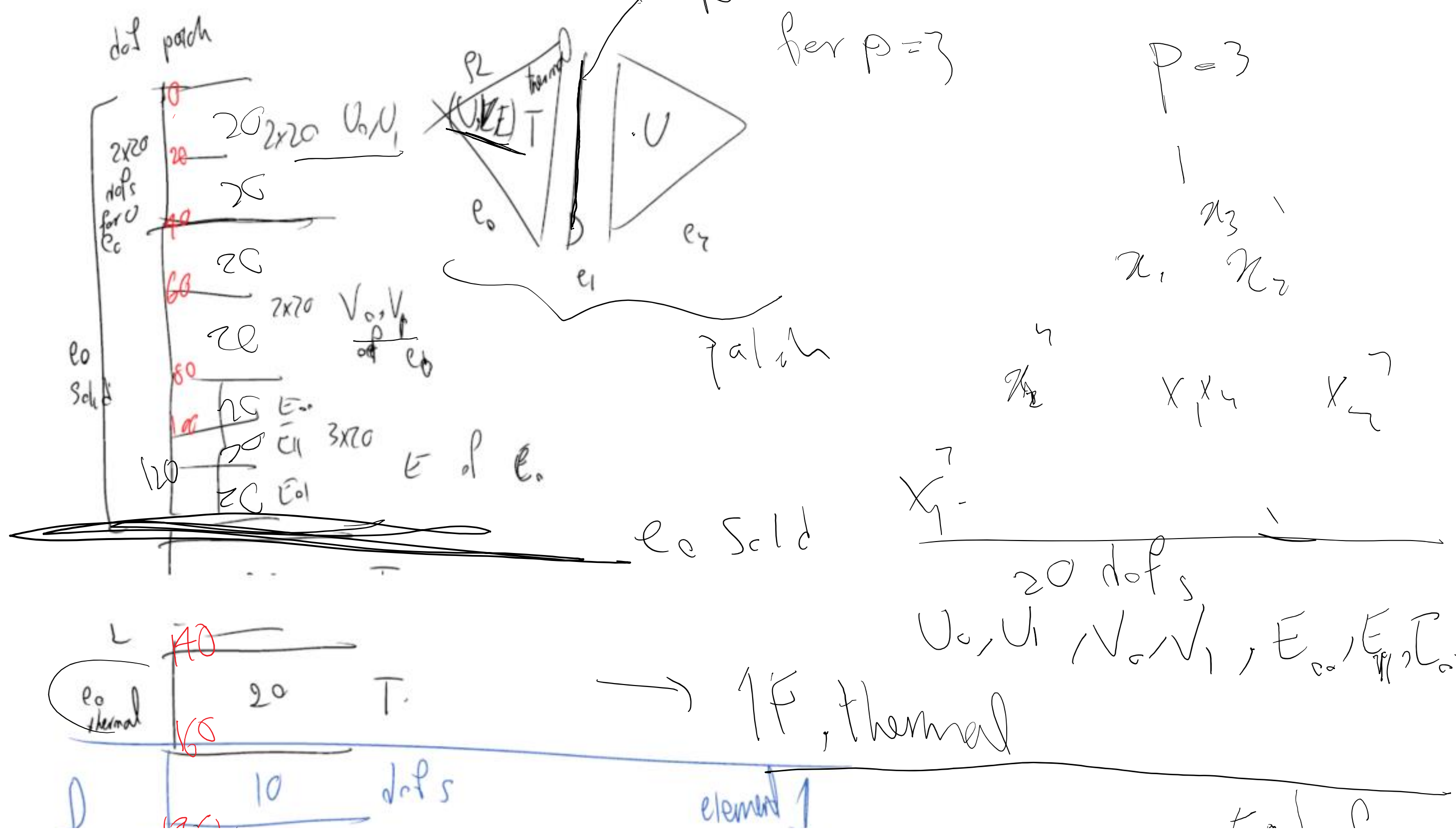
$$\frac{\partial^2 H}{\partial x_i \partial x_j}$$

VECTOR pCoef; // Coefficients for the given phyiscs interpolant

vector of unknowns for this coefficient

posDof pDof; // for storing the position and dof info
of the given physics interpolant

10 unknowns for p=3



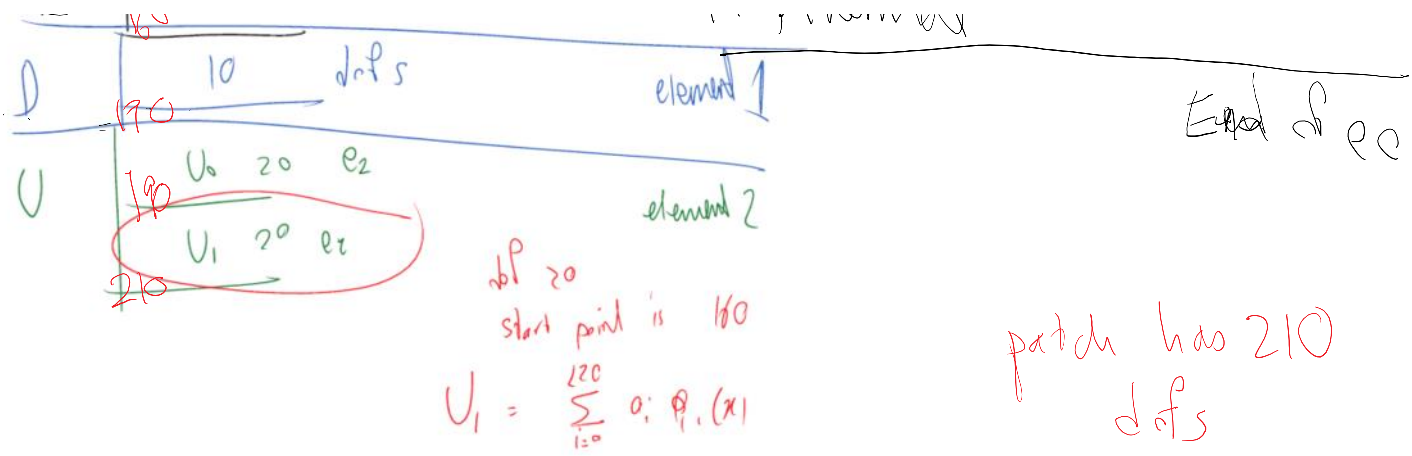
e0 Solid

20 dofs

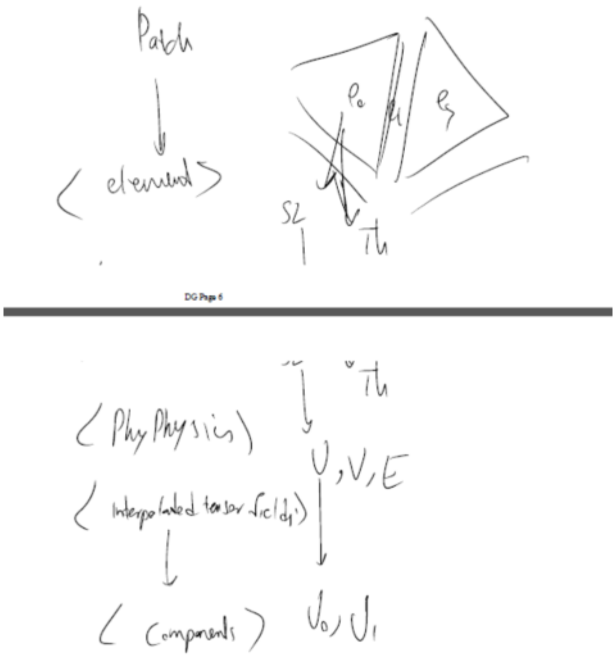
$U_0, U_1, V_0, V_1, E_{00}, E_{11}, E_{01}$

→ 1F, thermal

element 1



posDof is a class that stores dof of a thing (component of a tensor field, tensor field, physics, element) and its position w.r.t. to all objects owning it (tensor field, physics, element, patch)



class PhyIntCellBase

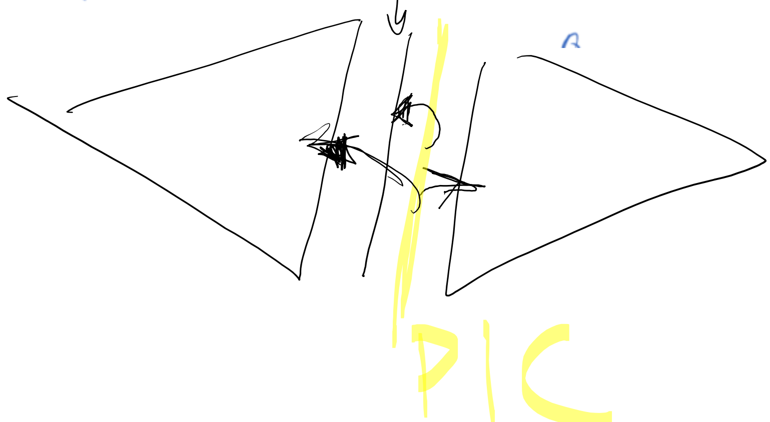
PhyInt2EBasePtr interiorIntBase;
bool bInteriorInt;

if the cell is integrating any interior

vector<PhyInt2EBasePtr> facetIntBase;

size of this — *nF*

how many faces it integrates



PhyElements and PICs are stored in PhyPatch

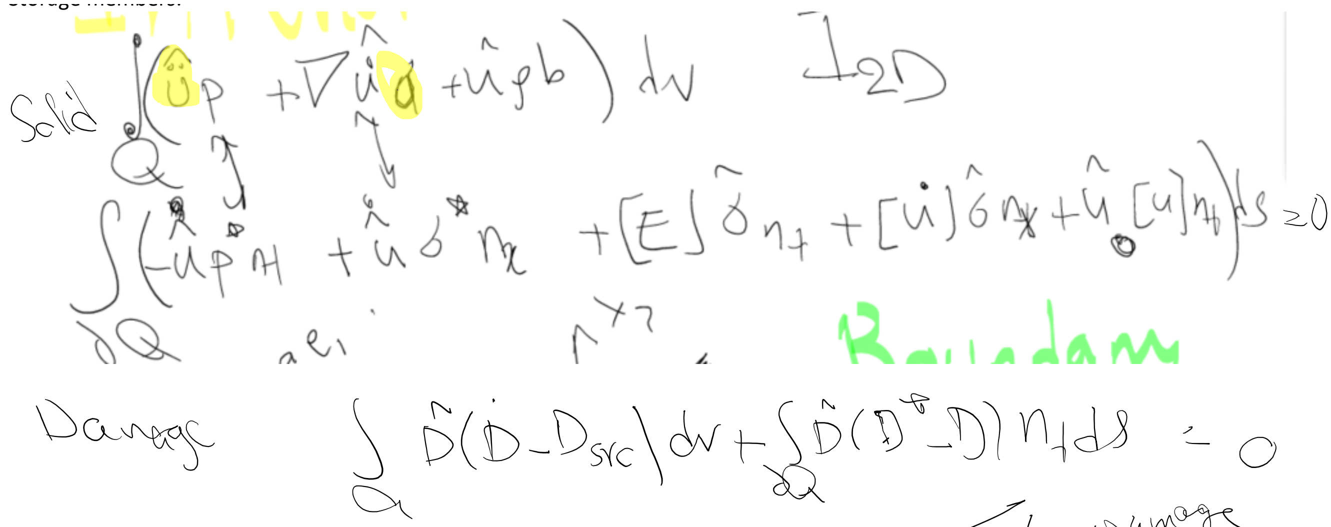
Class PhyPatch
....
vector<PhyElementBase*> phyElementsBase;
int num_elementsBase;

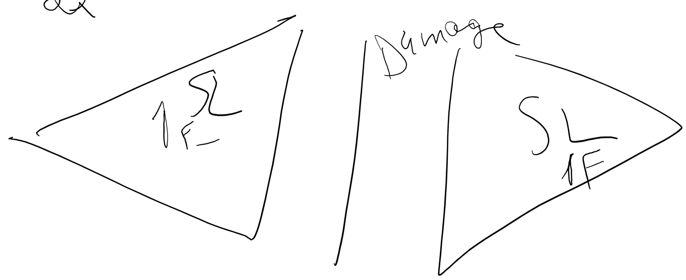
} *elements*

vector<PhyIntCellBase*> phyIntsBase;
int num_phyIntsBase;

} *integration cells*

Storage members:



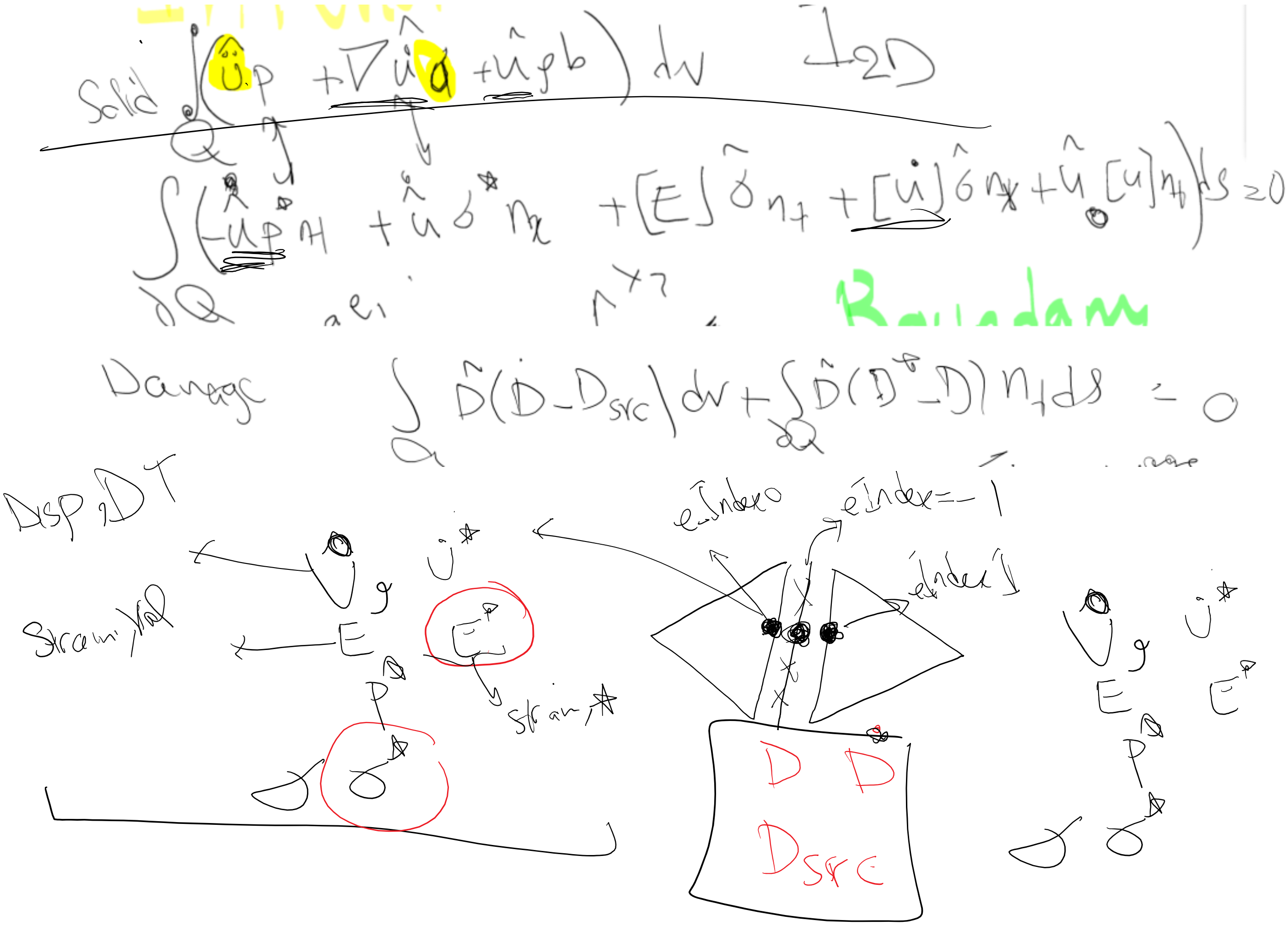


```
class PhyFldC
...
phyFld phyF;
compT cT;
```

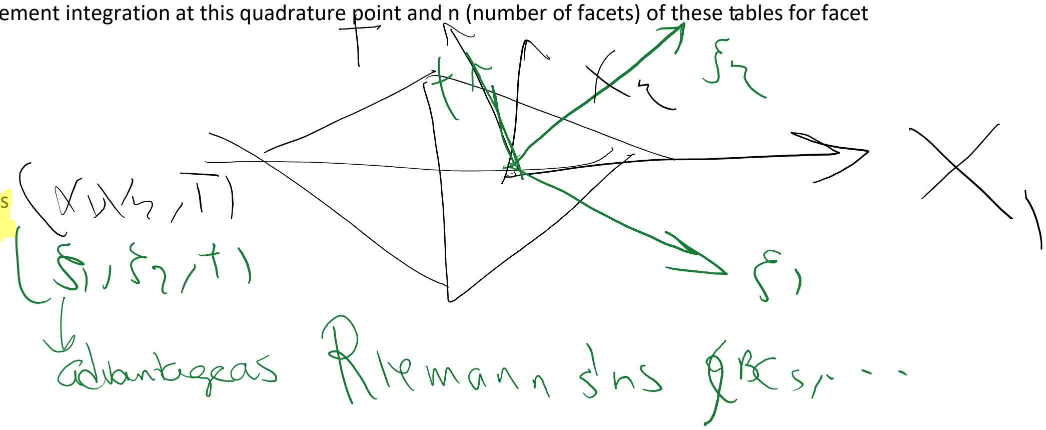
```
typedef enum {
	pfNONE = -1,
	pfVoid = 0, pfDisp, pfVel, pfStrnL, pfplm, pfStrsL, pfStrsElasL, pfRho,
	pfID,
	pfT, pfHeatFlx, pfUEnergy, pfKEnergy, pfTEnergy, pfDampingF, pfConc, pfConcFlx, pfMaxWaveSpeed, pfBDM, pfBDKappa, pfRTEI,
	/// EM fields, FEM is the field solved (e.g. scatter), inc is incident, and tot is total = inc + FEM
	pfEM_EFEM = 40, pfEM_Einc = 41, pfEM_Etot = 42, // E field
	} phyFld;

	typedef enum {
		ctVal,			//Value
	//	// crDT = ctDangle0 for RTE, ctDS2 = ctDAngle1 for RTE
		ctDT, /* ctDS2,*/  ctDX, //First order derivatives BASE ones
		ctDT2, ctDXDT, ctDXY, //Second Order Derivatives BASE ones
		ctDXSym,
		ctSource,   // source term
		ctTFlux, ctTFluxStar, ctTFluxRiemann, ctTFluxExact,	// conservation law Time flux
		ctXFlux, ctXFluxStar, ctXFluxRiemann, ctXFluxExact,	// conservation law Space flux
		ctStar, ctStarDT, ctRiemann, ctRiemannSlip, ctExact,		// general value Riemann, Star values
		ctTFluxDT, ctXFluxDiv,
		ctStarMVal, ctExactMVal, ctStarMExact,
		//	Derived values from others
		ctVisual,
		ctProj, ctFilter01,
		ctDiv, ctCurl,	//First order derivatives derived ones
		ctDXSymDT, ctDivDT, ctCurlDT, ctLaplacian, ctdS, // ctdS directional dirivative for solid angle RTE //Second Order Derivatives BASE ones
		ctDXDiv,
		// computation types (mostly related to the J term in EM problem)
		// eps = E -> D, H -> B factors; (not including total part)  sigma = conductivity (not including PML part)
		// wl = dispersive linear,		(not including total part)
		// pml = PML layer, (including all PML related)
		//	inc related to incident wave
```

```
    // tot = total = inc + sig + wl + pml
     ctEMTot = 40, ctEMInc = 41, ctEMEps = 42, ctEMSig = 43, ctEMwl = 44, ctEMPML = 45
    // ctInc is used to store incident value in new incident / scatter formulation
    , ctInc = 50, ctIncDT = 51,
} compT;
```



We need 0 or 1 table for the interior of the element integration at this quadrature point and n (number of facets) of these tables for facet integrations.



```
class PhyFieldVals
{
PhyElementFields cVal;      // Cartesian values
PhyElementFields rVal;      // rotated values
}
```

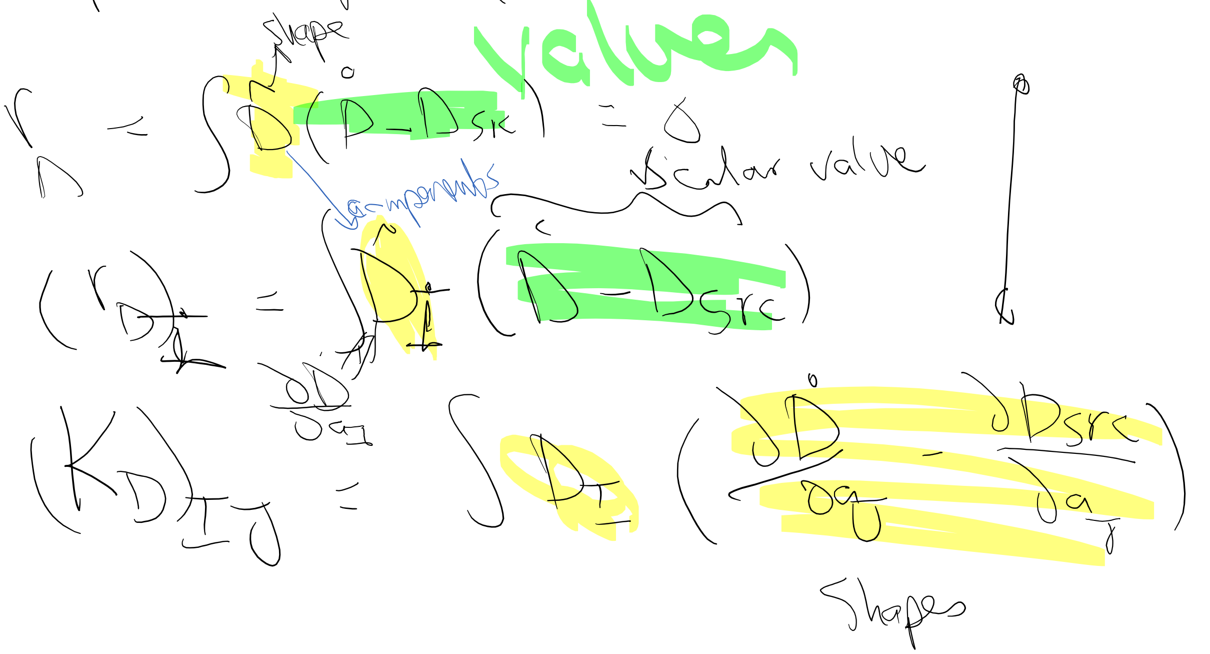
```
class PhyElementFields
{
PhyFieldElement eI;
vector<PhyFieldElement> eF;                  //size = numFTotal
bool bInterior;    // if there is an interior for field values
```

```
{
PhyFieldElement eI;
vector<PhyFieldElement> eF;                    //size = numFTotal
bool bInterior;    // if there is an interior for field values
mapPfc2Td delValF;
vector< mapPfc2DTd > DdelValF;
mapPfc2Td aveValF;
vector< mapPfc2DTd > DaveValF;
}
```

values → tables for storing facet values(u, v, s, ... for e_Index above)

shapes (der. w.r.t element unknowns)

Shapes are weights & are needed for stiffness calculation

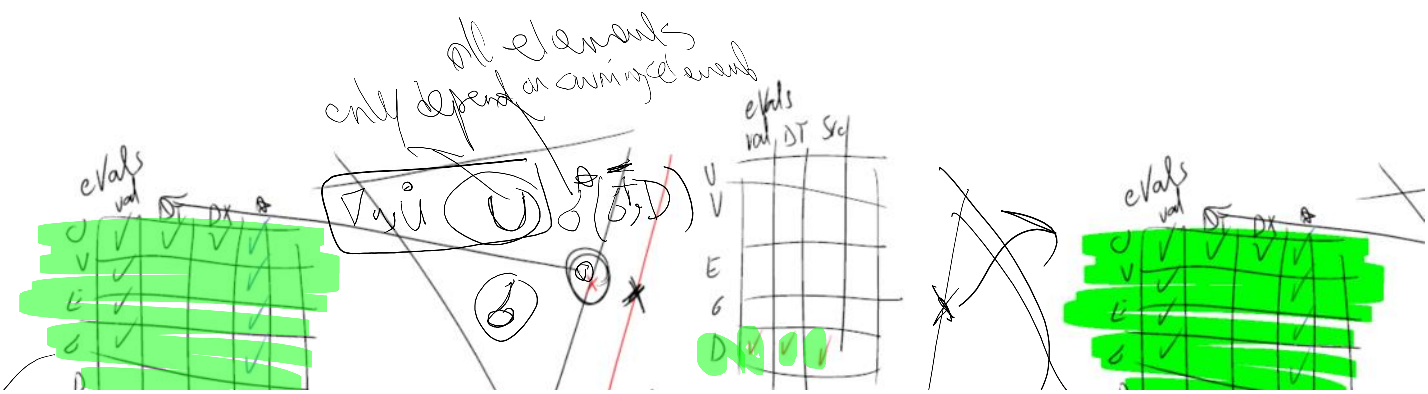


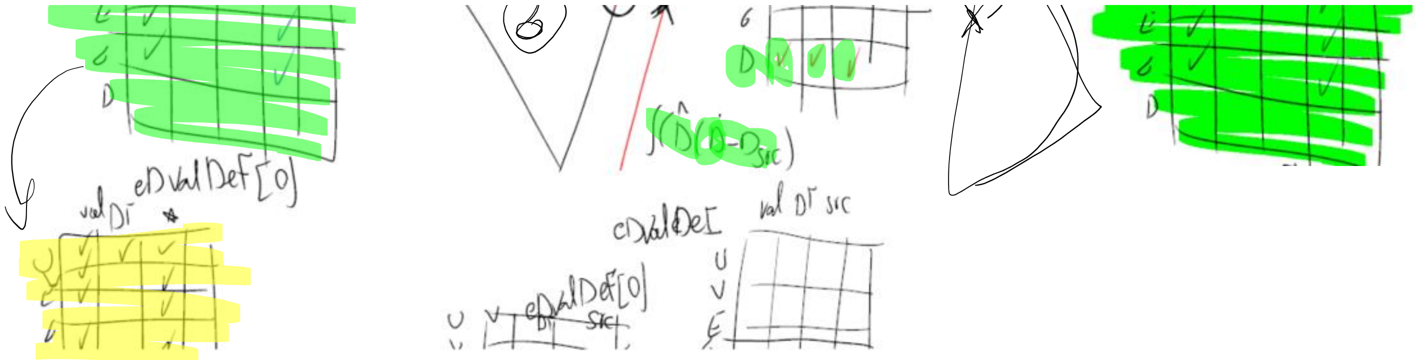
class PhyFieldElement -> class for storing Values (in green above) and shapes (in yellow above) for interior cell (eI above) of facet cells (eF above) at a quadrature points.

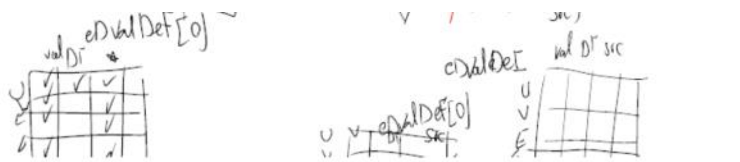
```
{
mapPfc2Td eVals;
mapPfc2DTd eDValDeI;       // derivatives wof the vals with respect to the element coefficients on the interior part
vector< mapPfc2DTd > eDValDeF;        // derivatives wof the vals with respect to the element coefficients of the elements that share a facet on the intCell
}
```

Automatic differentiation: automatically calculates all shapes as calculations go (product rule ….

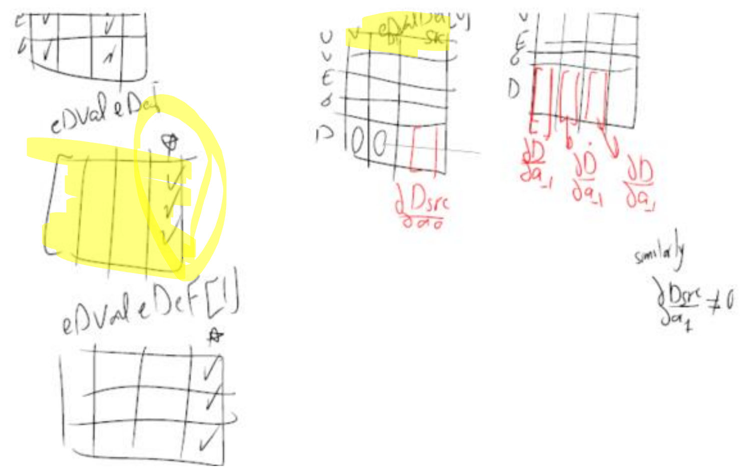
$$\frac{\partial (UV)}{\partial a} = U\frac{\partial V}{\partial a} + \frac{\partial U}{\partial a} V$$




eDValDef[0]
val Di
eDValDeI
val Di src


eDValDef[0]
eDValDeI

DG Page 2


eDValeDef[1]
similarly