

# An Introduction to the Finite Element Method

---

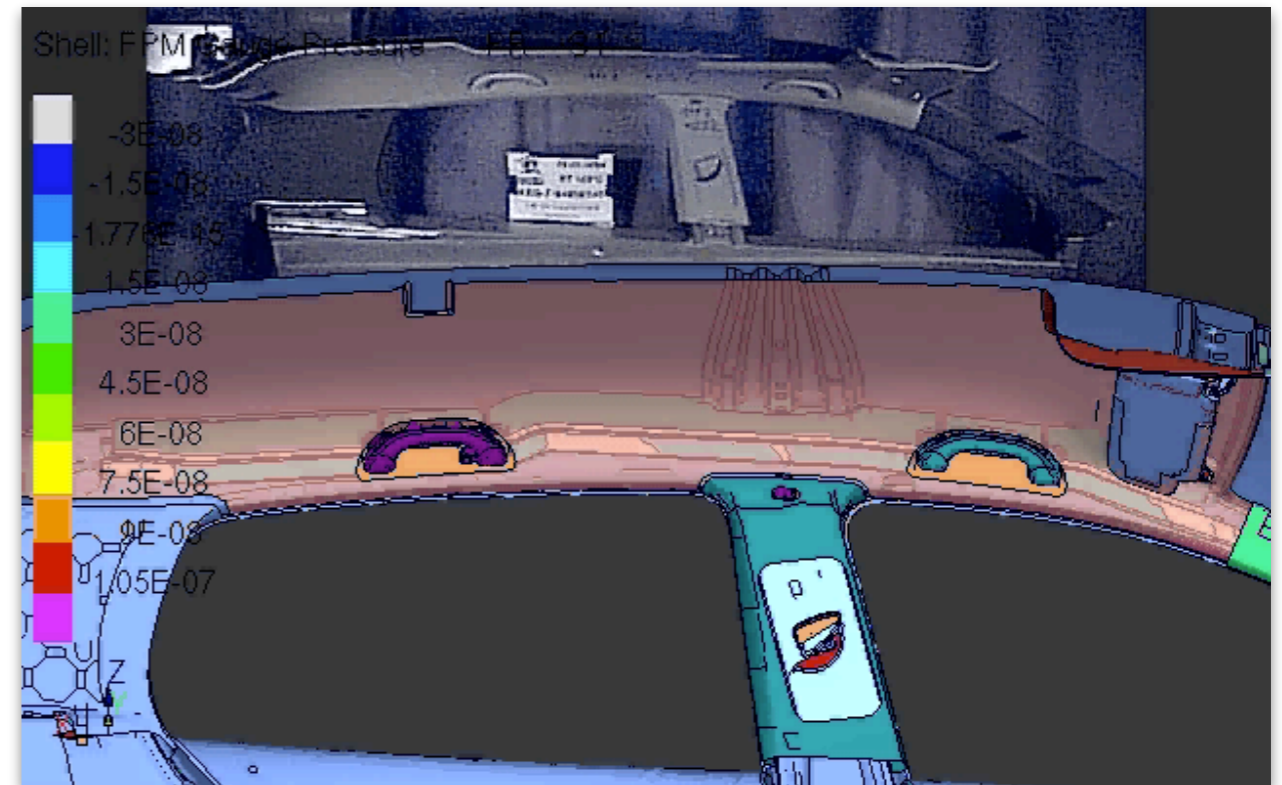
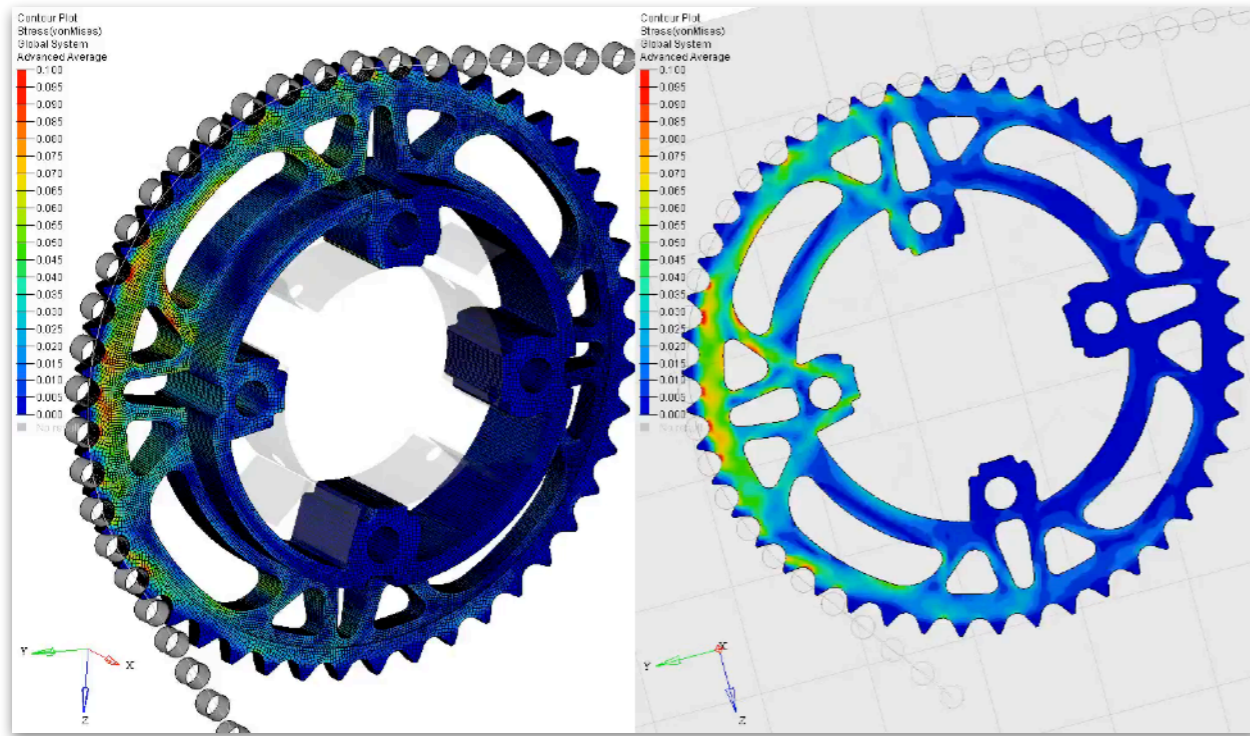
Dr. Erich Gaertig  
BU Central / Simulation  
Hilite Germany GmbH

# Outline

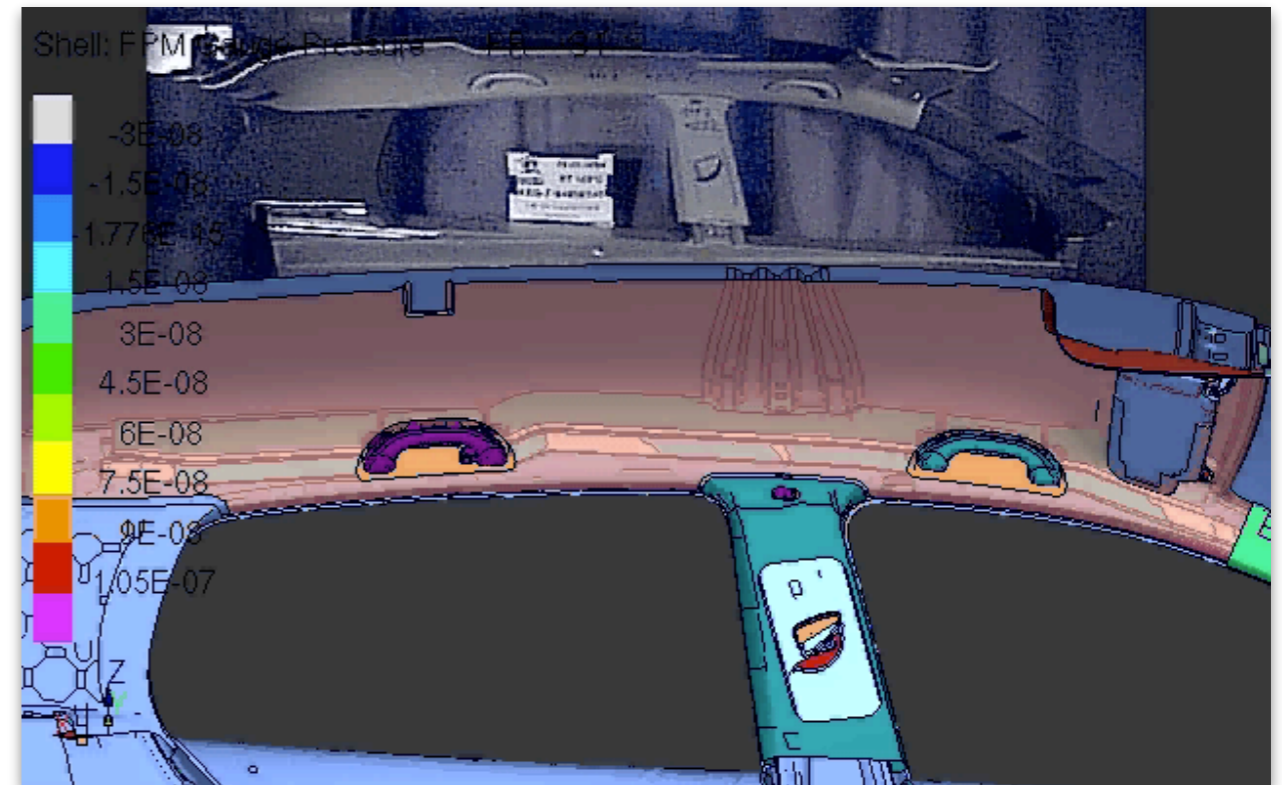
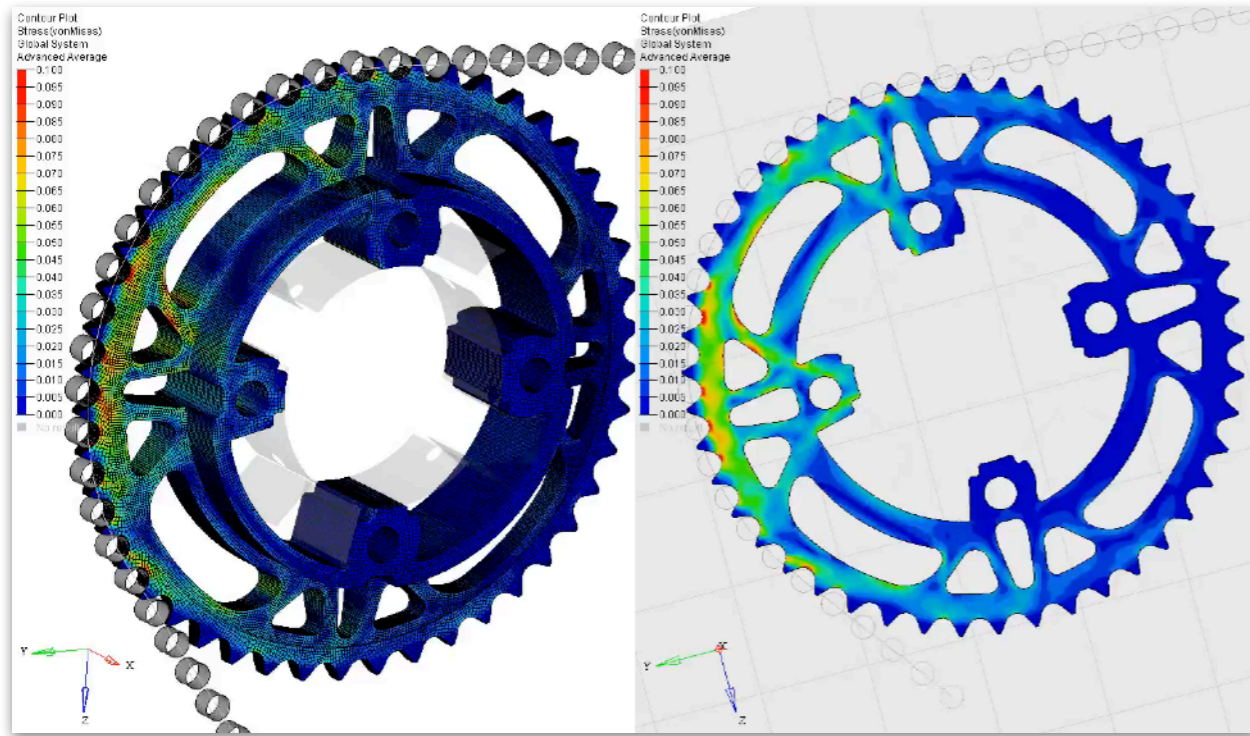
---

- Introduction to FEM - Motivation
- Mathematical Description/Numerical Implementation for Linear Elements on Triangular Meshes
- Solving a Simple PDE on an Irregular Domain in 2D/3D - Demo

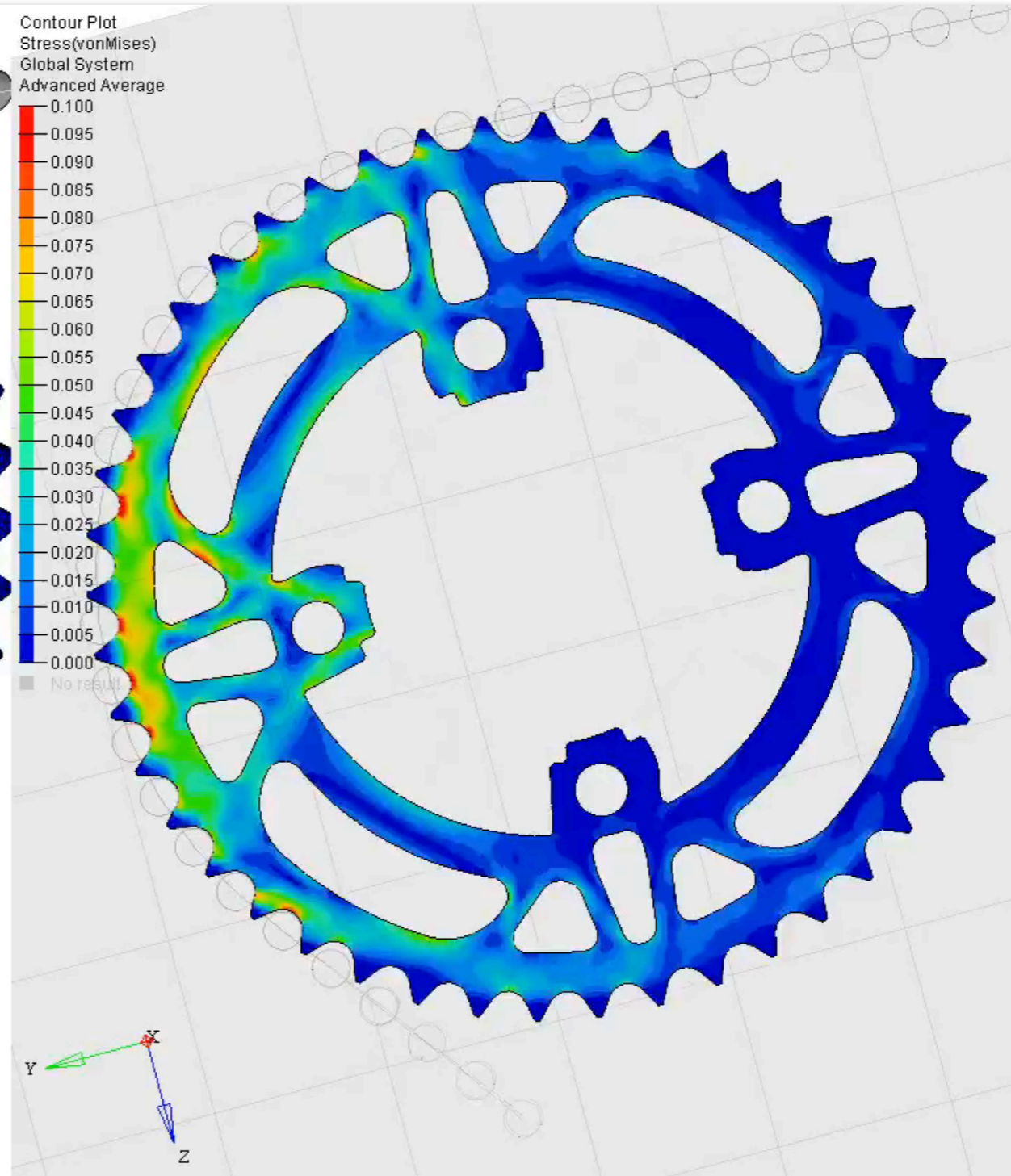
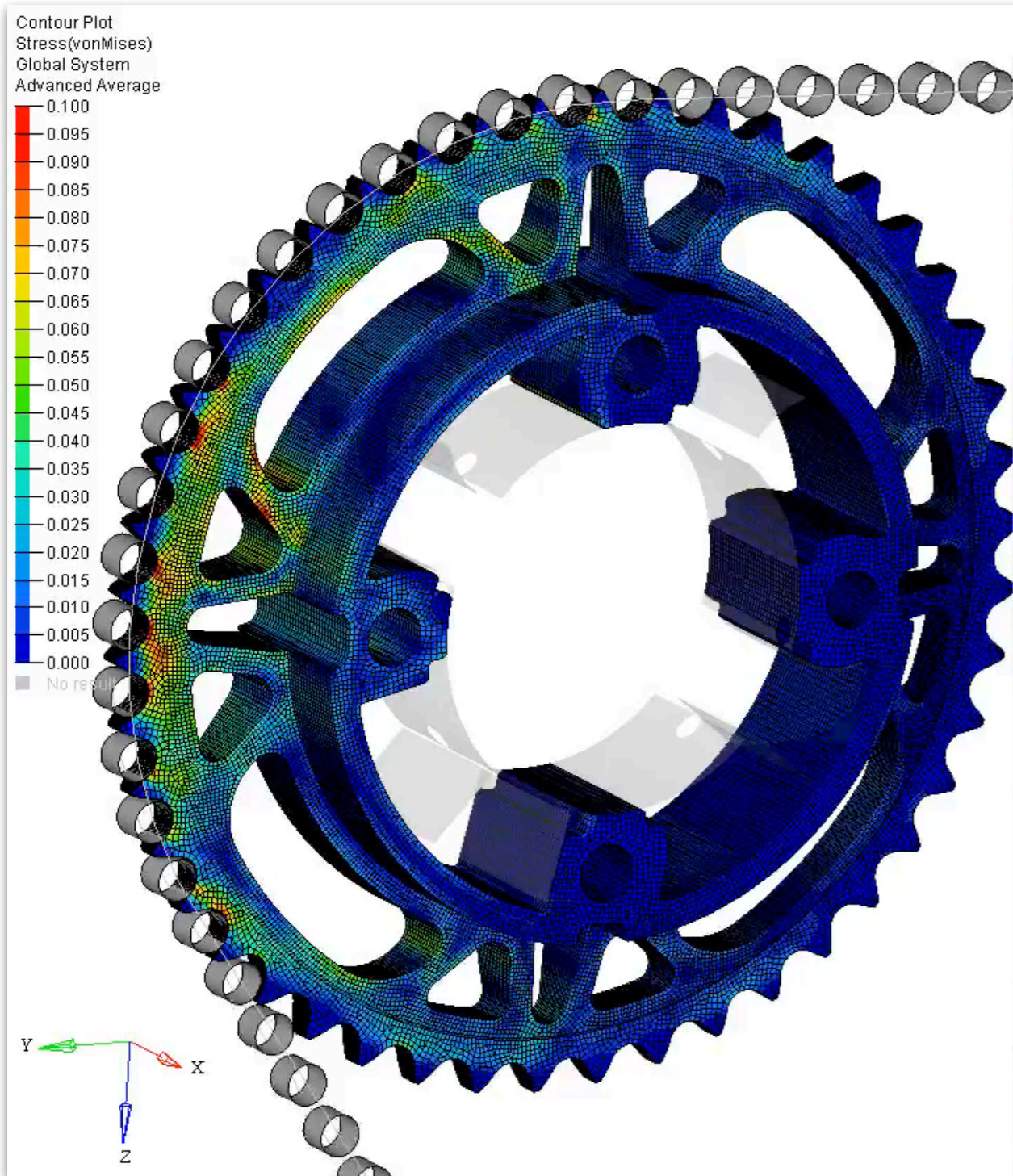
- Complex problems necessitate the need for a flexible and powerful computational technique
- Most real problems are defined on domains that are geometrically complex and may have different boundary conditions on different portions of the boundary
- It is therefore impossible to find an analytical solution; also Finite Differences (FD) are difficult to adapt for arbitrary geometries
- Solution of problems in structural mechanics by known functions with unknown coefficients (Ritz [1909]), later refined by using nodal shape functions (Courant [1943])
- FEM originally developed mainly by engineers in the aerospace environment (The finite element method in plane stress analysis (1960), *Turner, Clough, Martin, Topp [1950 - 1960]*)
- Thorough mathematical investigations and extensions during the 1960s (variational formulation, *Argyris, Zienkiewicz, Turner, Reissner*)
- First commercial codes showed up in the 1970s (NASAs Nastran commercialized in 1971)



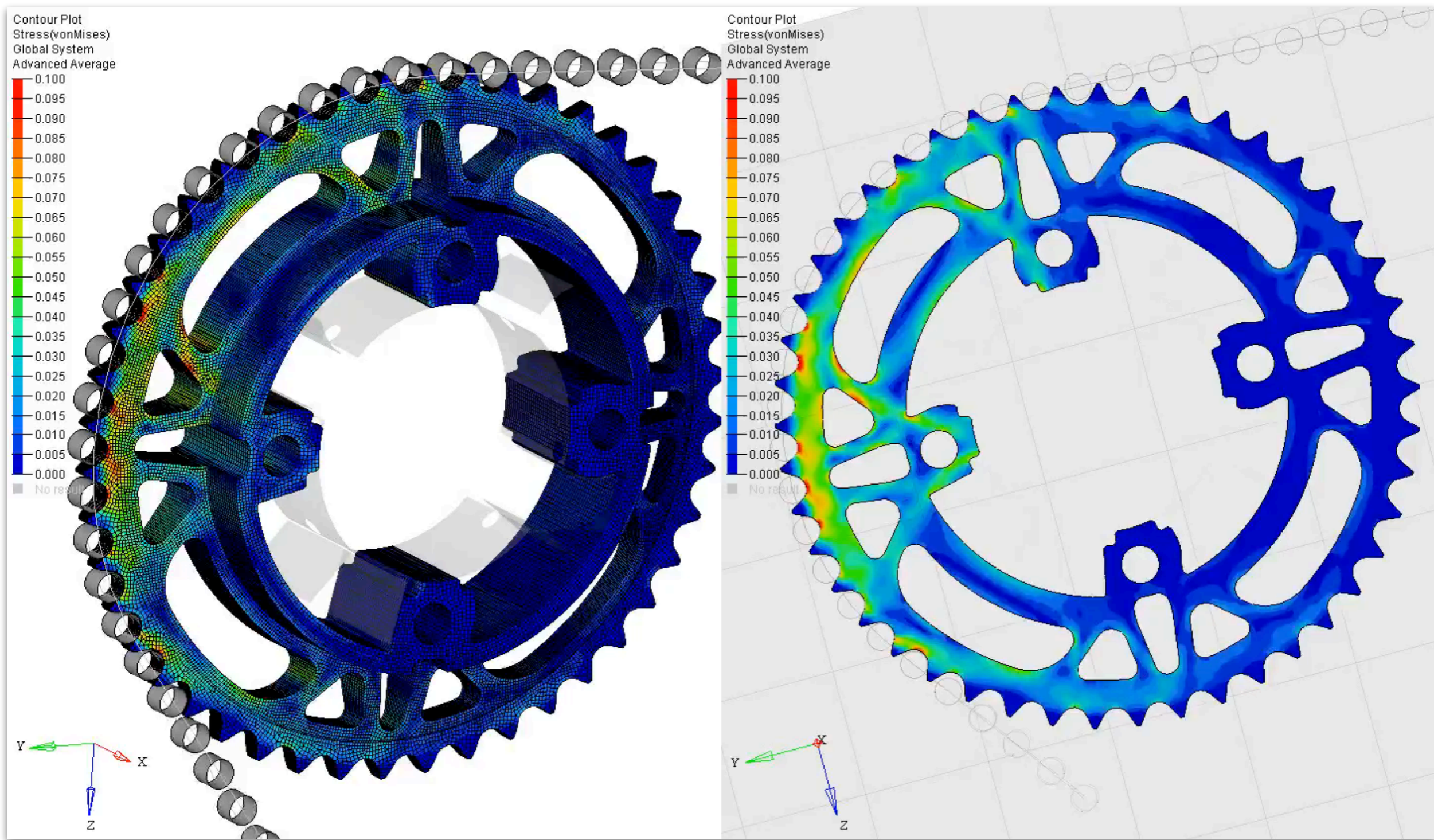




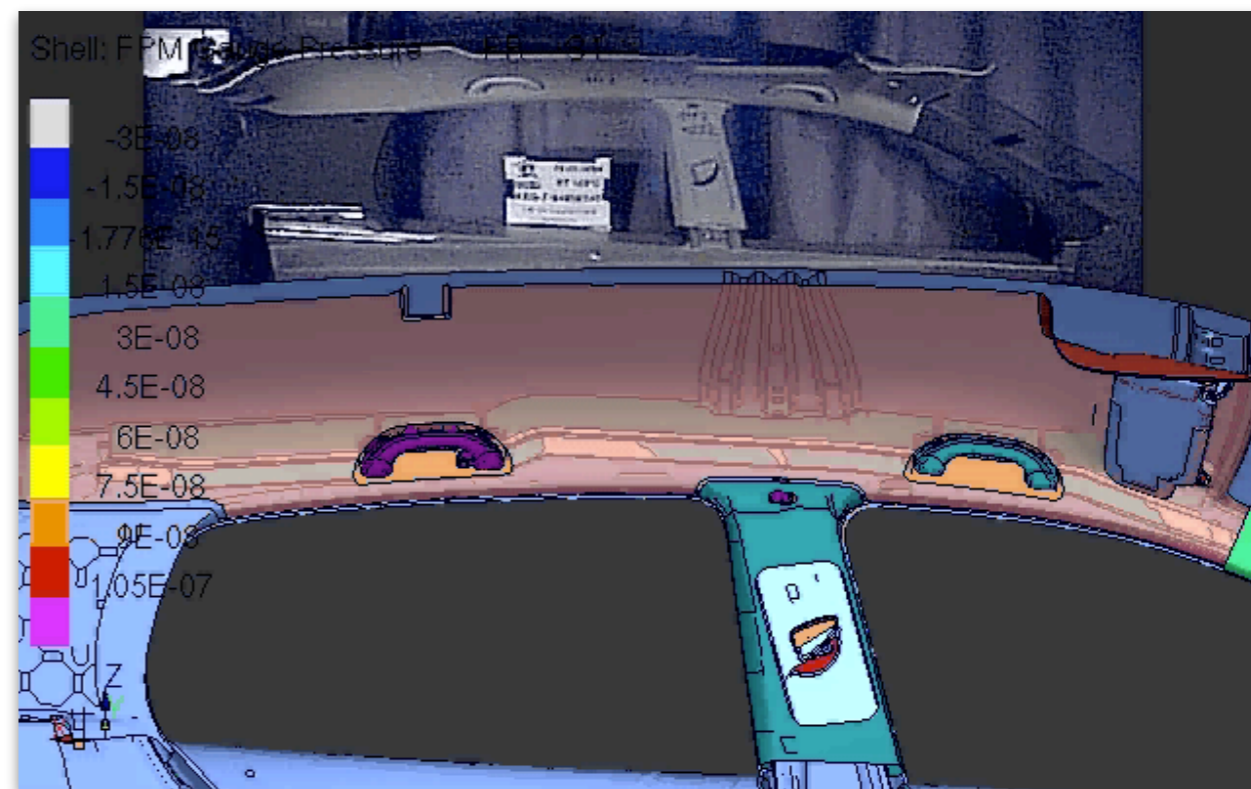
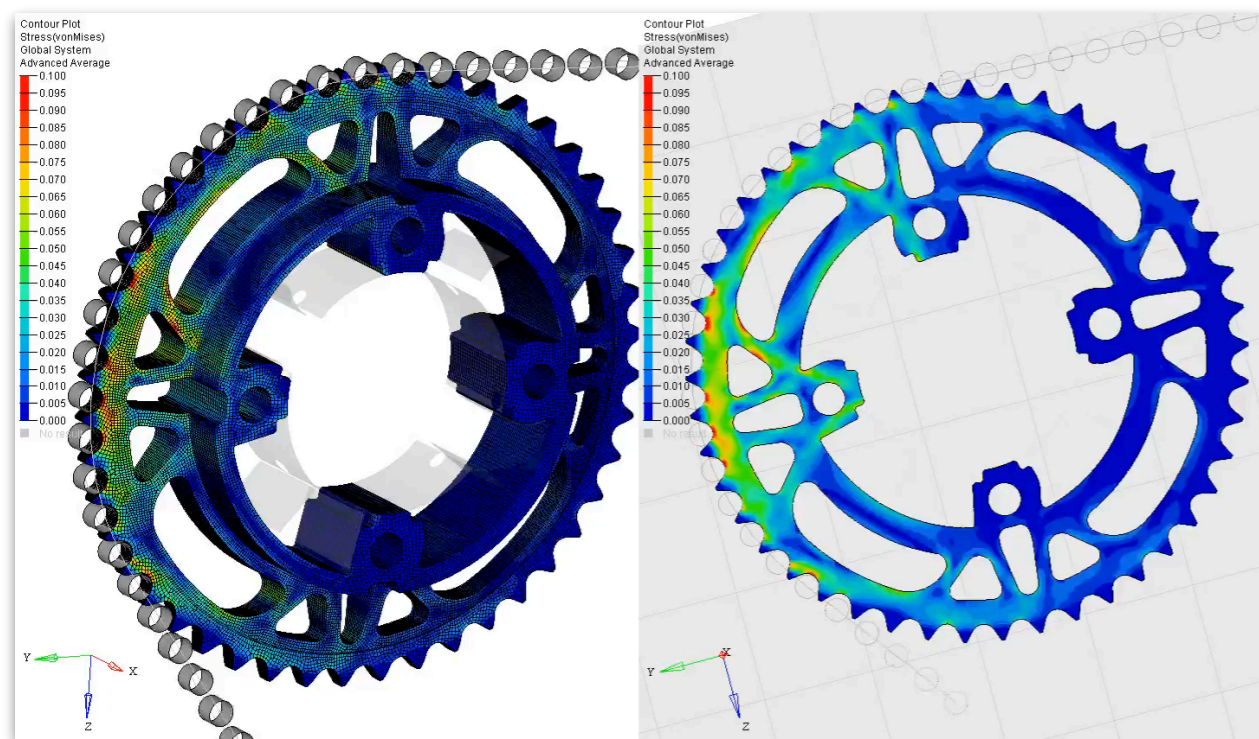


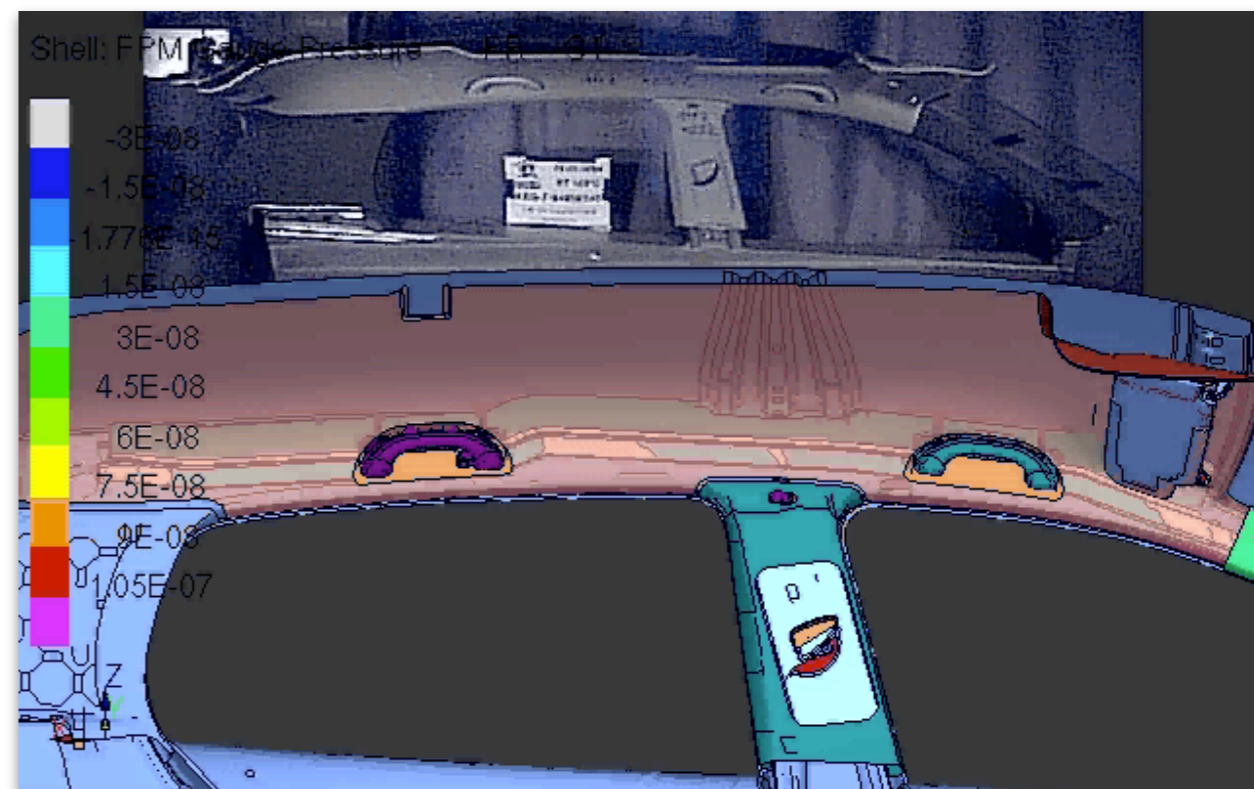
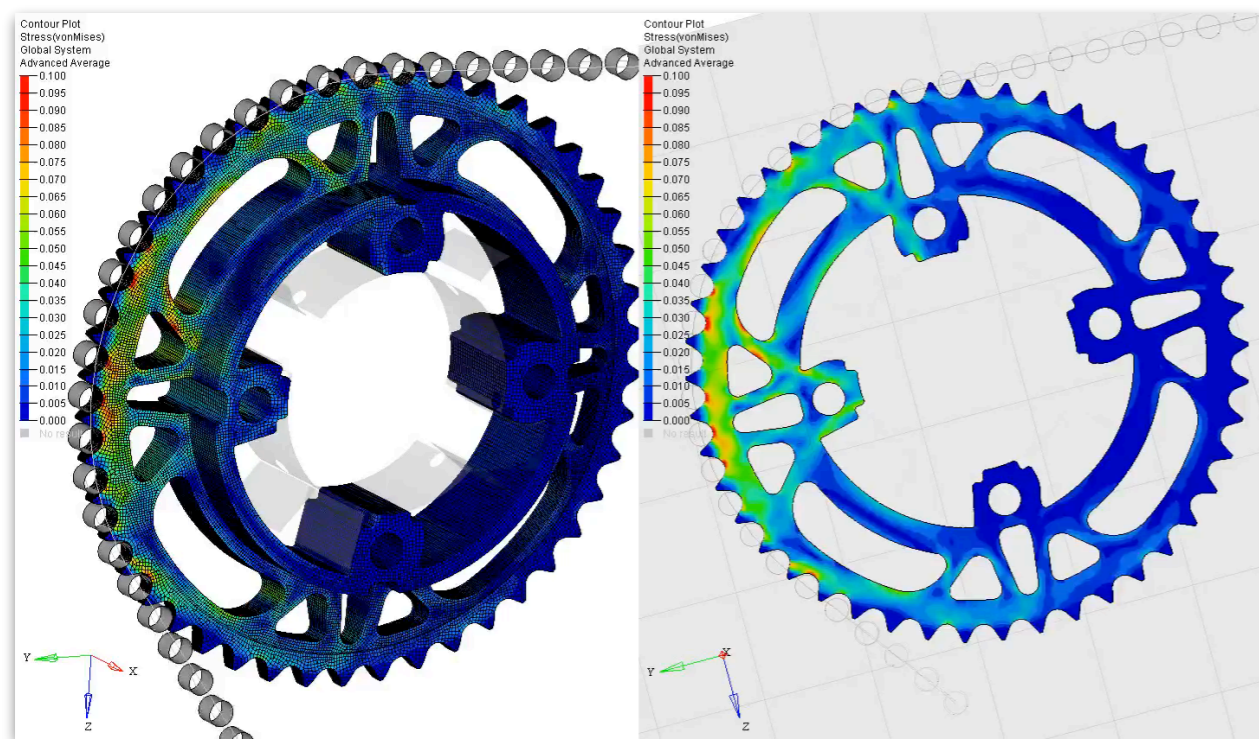




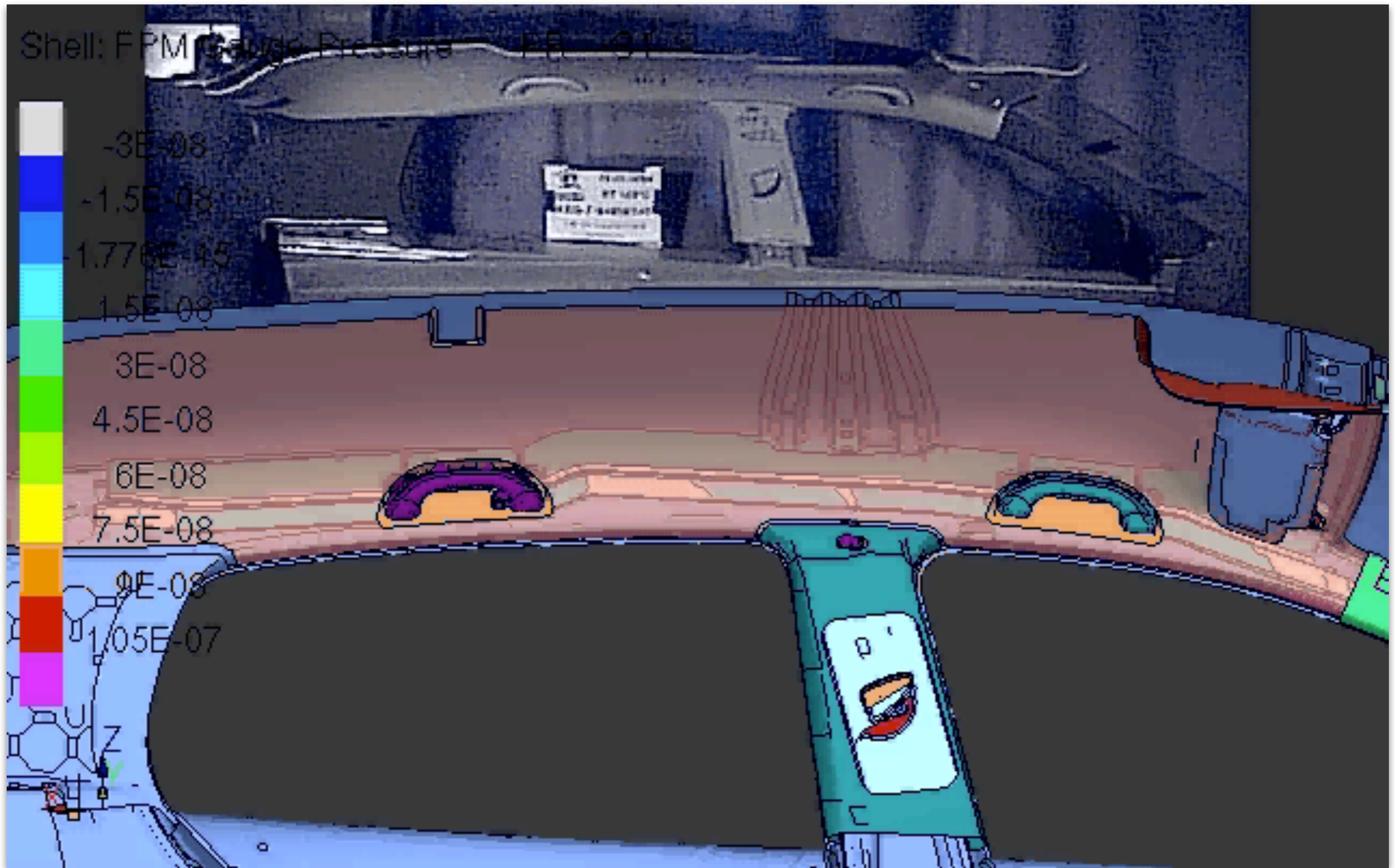




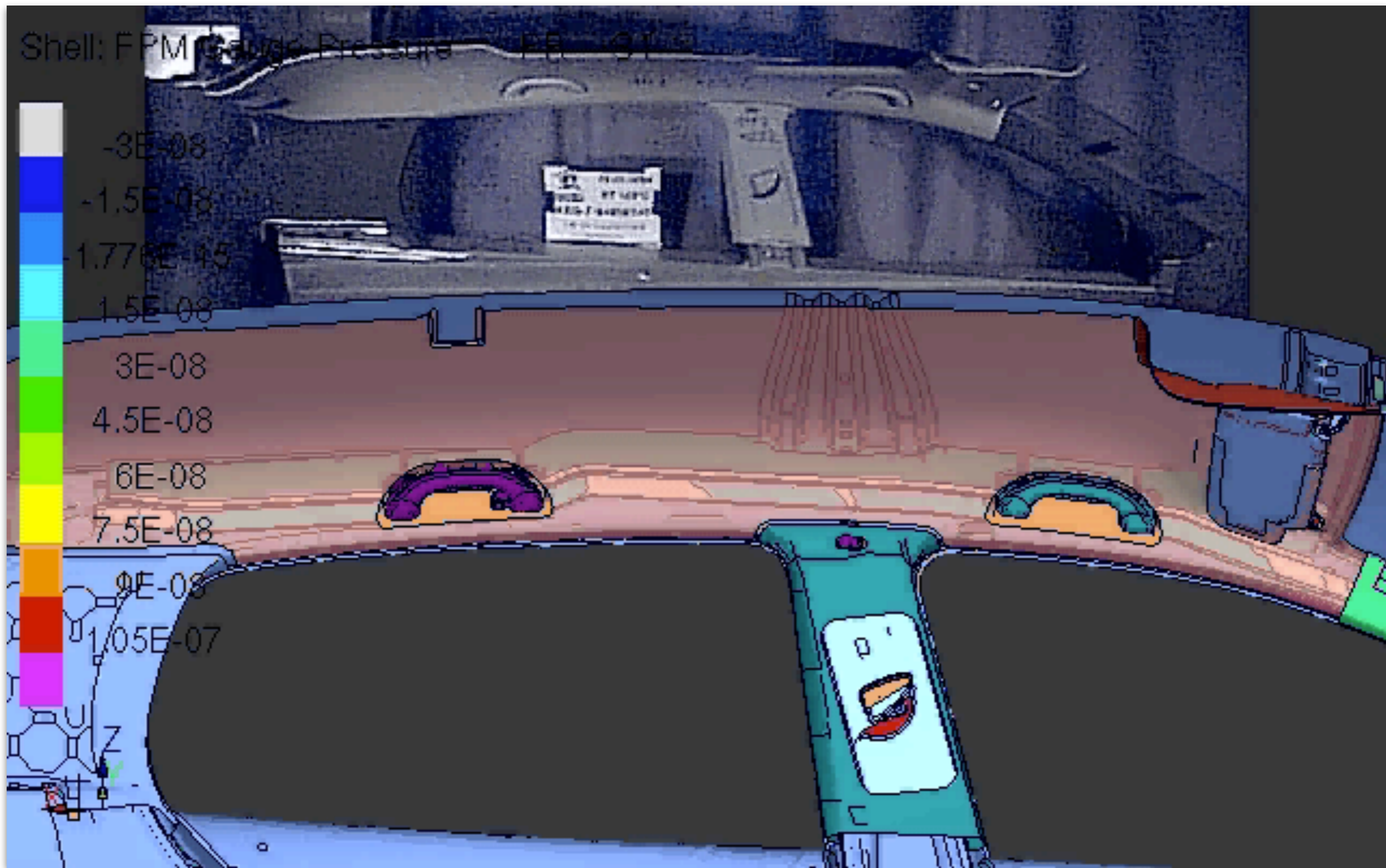












- Instead of satisfying the PDE point-by-point in the computational domain (FD approach), FEM (and other methods as well) uses an weighted integral formulation with test functions, that average the PDE globally

- Consider as an example 
$$-\frac{d}{dx} \left( x \frac{du}{dx} \right) + u = 0, \quad 0 < x < 1$$
  
$$u(0) = 1, \quad \left( x \frac{du}{dx} \right)_{x=1} = 0$$

- We seek a solution of the form

$$U_N = \sum_{j=1}^2 c_j \phi_j(x) + \phi_0(x)$$

- Here, all three functions satisfy the homogeneous boundary conditions UNLESS there are non-homogeneous ones; then  $\phi_0(x)$  has to obey them as well
- Since in this case, there's only one non-homogeneous BC for the left boundary point, we can make the ansatz

$$U_2 = c_1 \phi_1 + c_2 \phi_2 + \phi_0 \quad \text{with} \quad \phi_0 = 1, \quad \phi_1(x) = x^2 - 2x, \quad \phi_2(x) = x^3 - 3x$$

- For  $U_2(x)$  to satisfy the PDE, we need to have

$$-\frac{dU_2}{dx} - x\frac{d^2U_2}{dx^2} + U_2 = -2c_1(x-1) - 3c_2(x^2-1) - 2c_1x - 6c_2x^2 + c_1(x^2-2x) + c_2(x^3-3x) + 1 = 0$$

- This relation has to hold for all  $x$ , comparing powers leads to

$$\begin{aligned} 1 + 2c_1 + 3c_2 &= 0 \\ -6c_1 - 3c_2 &= 0 \\ c_1 - 9c_2 &= 0 \\ c_2 &= 0 \end{aligned}$$

- These relations are inconsistent; there is no solution to this linear system at all!!
- Instead, multiply both sides of the PDE with a weight function and integrate over the domain

$$\int_0^1 w(x)R(x)dx = 0 \quad \text{with} \quad R(x) = -\frac{dU_2}{dx} - x\frac{d^2U_2}{dx^2} + U_2$$

- From this, we obtain as many linearly independent equations as there are linear independent weight functions. Since here we have  $N = 2$ , let's choose  $w = 1$  and  $w = x$

- This leads to

$$0 = \int_0^1 1 \cdot R(x) dx = (1 + 2c_1 + 3c_2) + \frac{1}{2}(-6c_1 - 3c_2) + \frac{1}{3}(c_1 - 9c_2) + \frac{1}{4}c_2$$

$$0 = \int_0^1 x \cdot R(x) dx = \frac{1}{2}(1 + 2c_1 + 3c_2) + \frac{1}{3}(-6c_1 - 3c_2) + \frac{1}{4}(c_1 - 9c_2) + \frac{1}{5}c_2$$

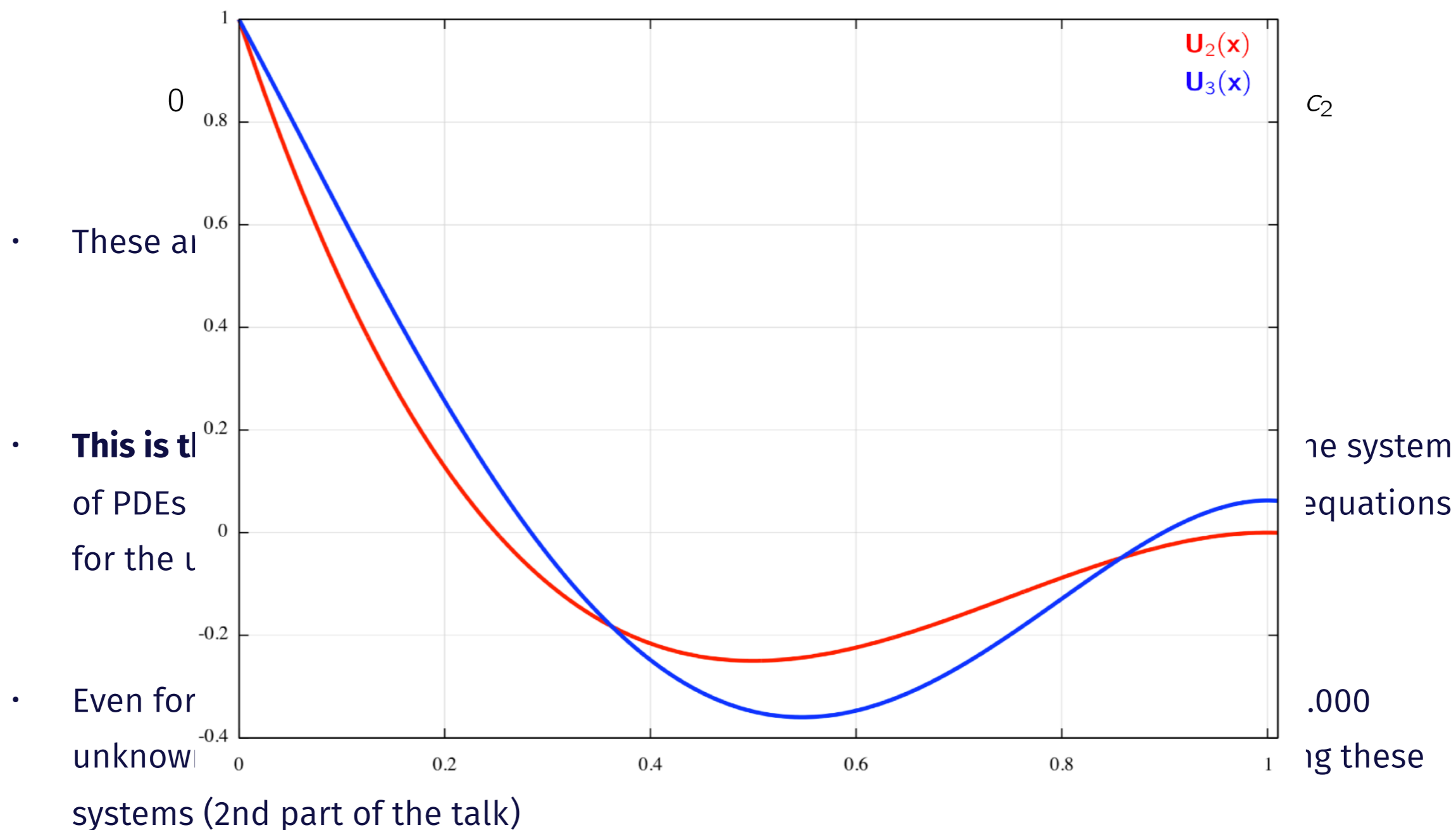
- These are two linear equations for two unknowns with the solution

$$c_1 = \frac{222}{23}, \quad c_2 = -\frac{100}{23}$$

- **This is the heart of FEM; if anything, take this from the lecture:** By transforming the system of PDEs into an averaged, weighted integral statement, derive a linear system of equations for the unknown expansion coefficients of your global solution
- Even for moderately sized problems in 2D, it's very easy to get systems with ~ 100.000 unknowns; need for accessible and efficient framework for assembling and solving these systems (2nd part of the talk)

- This leads to

$$0 = \int_0^1 1 \cdot R(x) dx = (1 + 2c_1 + 3c_2) + \frac{1}{2}(-6c_1 - 3c_2) + \frac{1}{3}(c_1 - 9c_2) + \frac{1}{4}c_2$$



- This leads to

$$0 = \int_0^1 1 \cdot R(x) dx = (1 + 2c_1 + 3c_2) + \frac{1}{2}(-6c_1 - 3c_2) + \frac{1}{3}(c_1 - 9c_2) + \frac{1}{4}c_2$$

$$0 = \int_0^1 x \cdot R(x) dx = \frac{1}{2}(1 + 2c_1 + 3c_2) + \frac{1}{3}(-6c_1 - 3c_2) + \frac{1}{4}(c_1 - 9c_2) + \frac{1}{5}c_2$$

- These are two linear equations for two unknowns with the solution

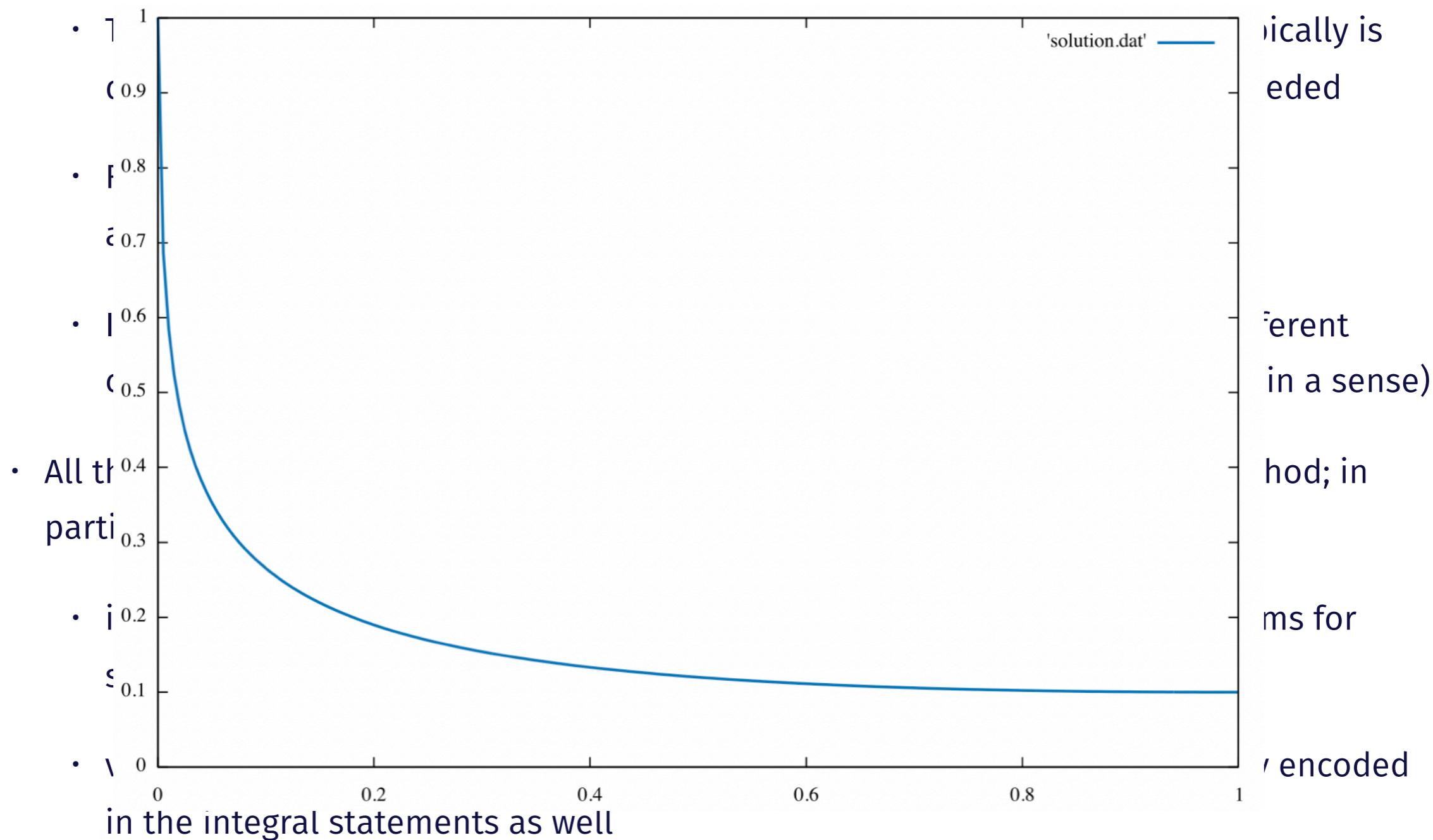
$$c_1 = \frac{222}{23}, \quad c_2 = -\frac{100}{23}$$

- **This is the heart of FEM; if anything, take this from the lecture:** By transforming the system of PDEs into an averaged, weighted integral statement, derive a linear system of equations for the unknown expansion coefficients of your global solution
- Even for moderately sized problems in 2D, it's very easy to get systems with ~ 100.000 unknowns; need for accessible and efficient framework for assembling and solving these systems (2nd part of the talk)



- Note:
  - There are (at least) three problems with this approach (*Ritz method, 1908*)
    - The matrix one needs to solve for the unknown expansion coefficients  $c_i$  typically is dense, so computations will be slow; even more so, if high accuracies are needed
    - For complicated PDEs, it might be very difficult to find basis functions that automatically satisfy the imposed boundary conditions
    - It is not clear, how to choose the weighting functions for the integration; different choices will naturally lead to different coefficients  $c_i$  (you need ALL of them in a sense)
  - All these issues are remedied by a proper formulation of the Finite Element Method; in particular
    - it will lead to a linear system with sparse matrices (so very efficient algorithms for solving these type of systems can be employed)
    - very simple basis functions are used; boundary conditions are transparently encoded in the integral statements as well
    - the very same functions are used as basis for the weights in the FE-formulation

- Note:
  - There are (at least) three problems with this approach (*Ritz method, 1908*)

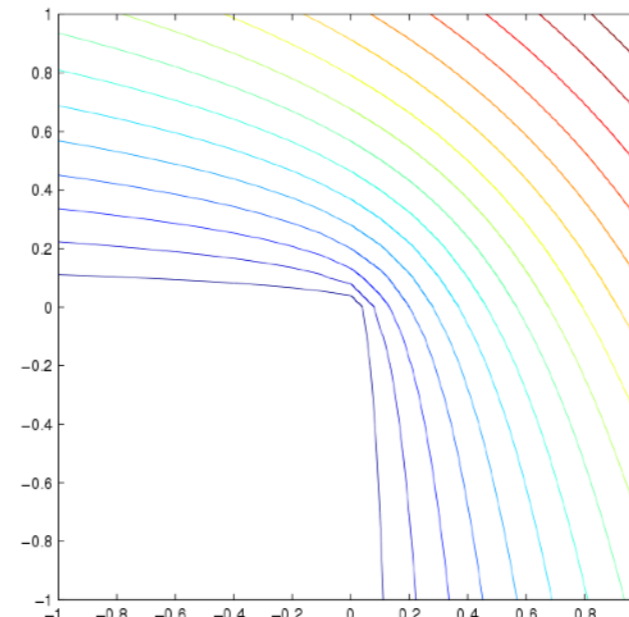
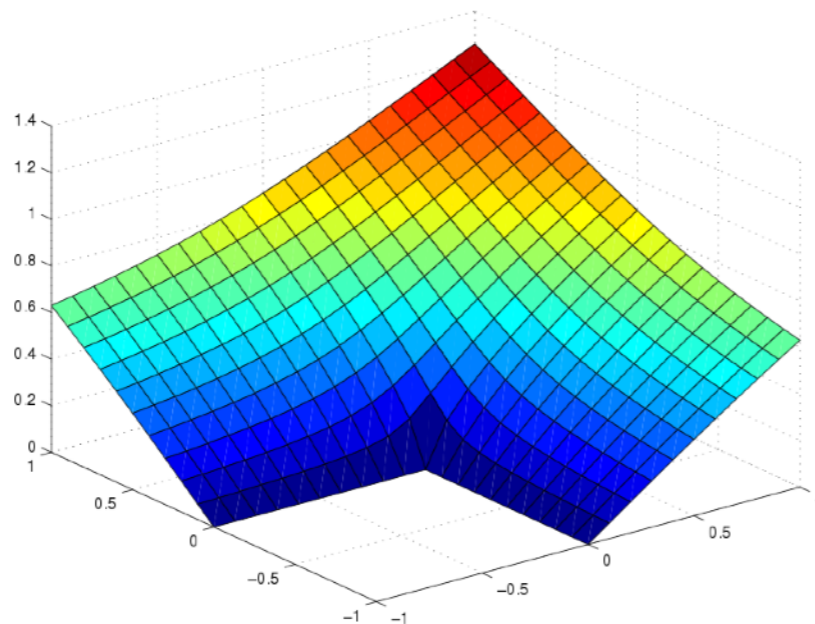


- the very same functions are used as basis for the weights in the FE-formulation

- Note:
  - There are (at least) three problems with this approach (*Ritz method, 1908*)
    - The matrix one needs to solve for the unknown expansion coefficients  $c_i$  typically is dense, so computations will be slow; even more so, if high accuracies are needed
    - For complicated PDEs, it might be very difficult to find basis functions that automatically satisfy the imposed boundary conditions
    - It is not clear, how to choose the weighting functions for the integration; different choices will naturally lead to different coefficients  $c_i$  (you need ALL of them in a sense)
  - All these issues are remedied by a proper formulation of the Finite Element Method; in particular
    - it will lead to a linear system with sparse matrices (so very efficient algorithms for solving these type of systems can be employed)
    - very simple basis functions are used; boundary conditions are transparently encoded in the integral statements as well
    - the very same functions are used as basis for the weights in the FE-formulation

- There are various reasons for using an integral version of the PDE
  - even for smooth initial data, PDEs may exhibit discontinuous behaviour
  - discontinuous source functions or irregular domains also lead to non-classical solutions
- Consider as prototype example the Poisson equation  $\nabla^2 u = f$  in an open domain  $\Omega$ 
  - a classical solution has continuous second derivatives in the interior ( $u \in C^2(\Omega)$ ) and is continuous up to the boundary ( $u \in C^0(\bar{\Omega})$ )
- If the source function is discontinuous (e.g. weight placed on parts of a membrane), then  $u \notin C^2(\Omega)$
- The same happens for so-called *re-entrant corners* in a non-convex domain

$$\Omega = [-1, 1]^2 \setminus [-1, 0]^2, \quad \Gamma_D = \Gamma, \quad f = 0, \quad g = u|_{\Gamma}, \quad u = r^{2/3} \sin\left(\frac{2\theta + \pi}{3}\right)$$

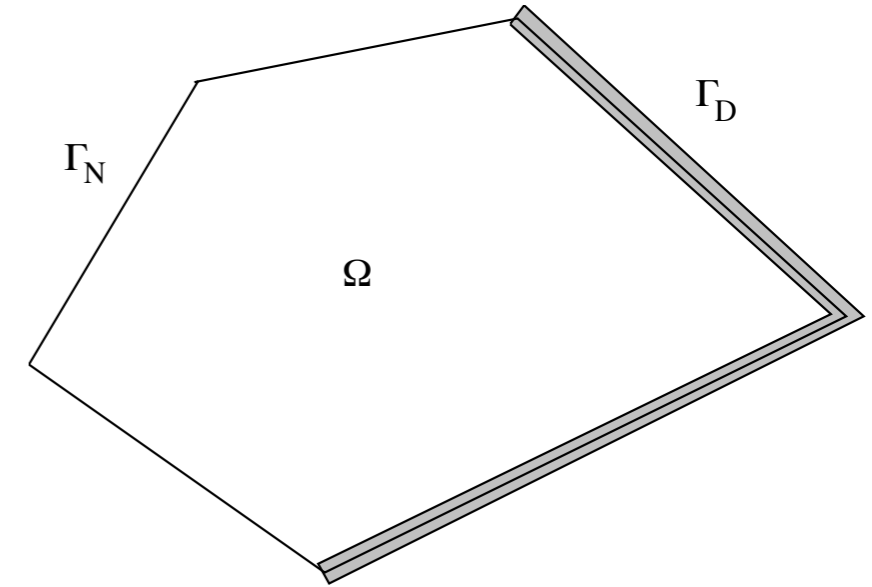


- Our model problem is

$$-\Delta u + cu = f \text{ in } \Omega$$

$$u = g_0 \text{ on } \Gamma_D$$

$$\partial_n u = g_1 \text{ on } \Gamma_N$$



- Multiply with test function, integrate and use Green's identity

$$\int_{\Omega} (\Delta u)v + \int_{\Omega} \nabla u \nabla v = \int_{\Gamma} (\partial_n u)v = \int_{\Gamma_D} (\partial_n u)v + \int_{\Gamma_N} (\partial_n u)v$$

- Impose  $v = 0$  on  $\Gamma_D$  and search for solutions  $u \in H^1(\Omega)$  such that

$$u = g_0 \text{ on } \Gamma_D$$

$$\int_{\Omega} \nabla u \nabla v + c \int_{\Omega} uv = \int_{\Omega} fv + \int_{\Gamma_N} g_1 v \quad \forall v \in H_{\Gamma_D}^1(\Omega)$$

**(weak formulation)**

$$H^1(\Omega) = \left\{ u \in L^2(\Omega) \left| \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2} \in L^2(\Omega) \right. \right\}$$

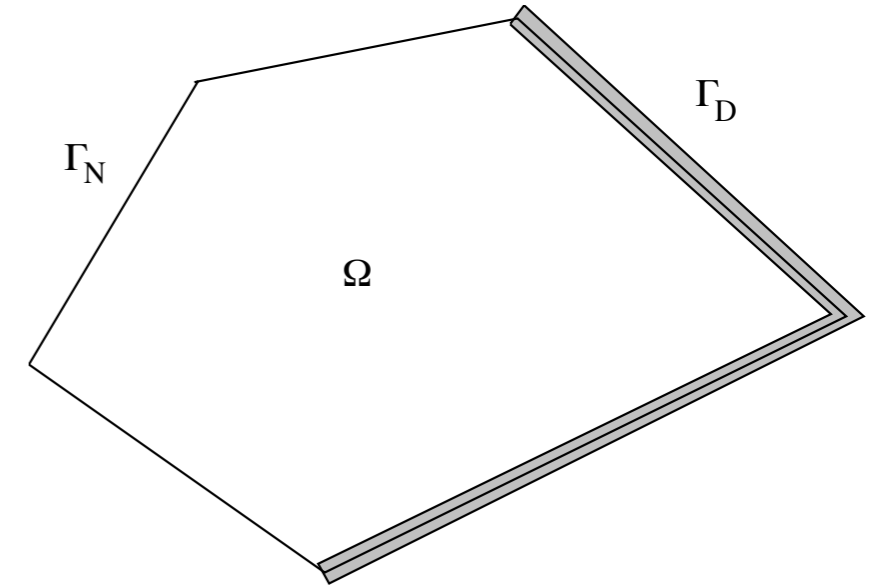
$$H_{\Gamma_D}^1(\Omega) = \{ v \in H^1(\Omega) \mid v = 0 \text{ on } \Gamma_D \}$$

- Our model problem is

$$-\Delta u + cu = f \quad \text{in } \Omega$$

$$u = g_0 \quad \text{on } \Gamma_D \quad \text{essential BC}$$

$$\partial_n u = g_1 \quad \text{on } \Gamma_N \quad \text{natural BC}$$



- Multiply with test function, integrate and use Green's identity

$$\int_{\Omega} (\Delta u)v + \int_{\Omega} \nabla u \nabla v = \int_{\Gamma} (\partial_n u)v = \int_{\Gamma_D} (\partial_n u)v + \int_{\Gamma_N} (\partial_n u)v$$

- Impose  $v = 0$  on  $\Gamma_D$  and search for solutions  $u \in H^1(\Omega)$  such that

$$u = g_0 \quad \text{on } \Gamma_D$$

$$\int_{\Omega} \nabla u \nabla v + c \int_{\Omega} uv = \int_{\Omega} fv + \int_{\Gamma_N} g_1 v \quad \forall v \in H_{\Gamma_D}^1(\Omega)$$

**(weak formulation)**

$$H^1(\Omega) = \left\{ u \in L^2(\Omega) \mid \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2} \in L^2(\Omega) \right\}$$

$$H_{\Gamma_D}^1(\Omega) = \{ v \in H^1(\Omega) \mid v = 0 \text{ on } \Gamma_D \}$$



- By reducing the smoothness of the solution variable required by the weak formulation (i.e.  $C^1(\Omega)$  instead of  $C^2(\Omega)$ ), we can get a weak solution, that is not smooth enough to be a classical solution
- One even extends the admissible function space to include  $C^0(\Omega)$  - functions as well via so-called *weak derivatives*:

- for  $v \in C_0^\infty(\Omega)$ , the partial integration relation in 1D reads

$$\int_{\Omega} u'v = - \int_{\Omega} uv'$$

- idea: use this as the definition for the weak derivative of  $u$
- we call  $\partial_x u|_w$  the *weak derivative* of  $u$ , if

$$\int_{\Omega} \partial_x u|_w v = - \int_{\Omega} uv' \quad \forall v \in C_0^\infty(\Omega)$$

- This means, differentiation is also defined via integration over test functions
- Advantage: functions, that are classically not differentiable nevertheless are weakly differentiable, e.g.  $u(x) = |x|$

- Similar definitions for weak derivatives exist for all differential operators, e.g. div, rot etc.; these are used for computational electrodynamics for example
- example: a function  $\nabla \cdot \mathbf{u}|_w \in L^2(\Omega)$  is called *weak divergence* of  $\mathbf{u} \in \mathbf{L}^2(\Omega)$ , if for any function  $v \in C_0^\infty(\Omega)$

$$\int_{\Omega} \nabla \cdot \mathbf{u}|_w v = - \int_{\Omega} \mathbf{u} \cdot \nabla v$$

(hint: use Gauss' theorem)

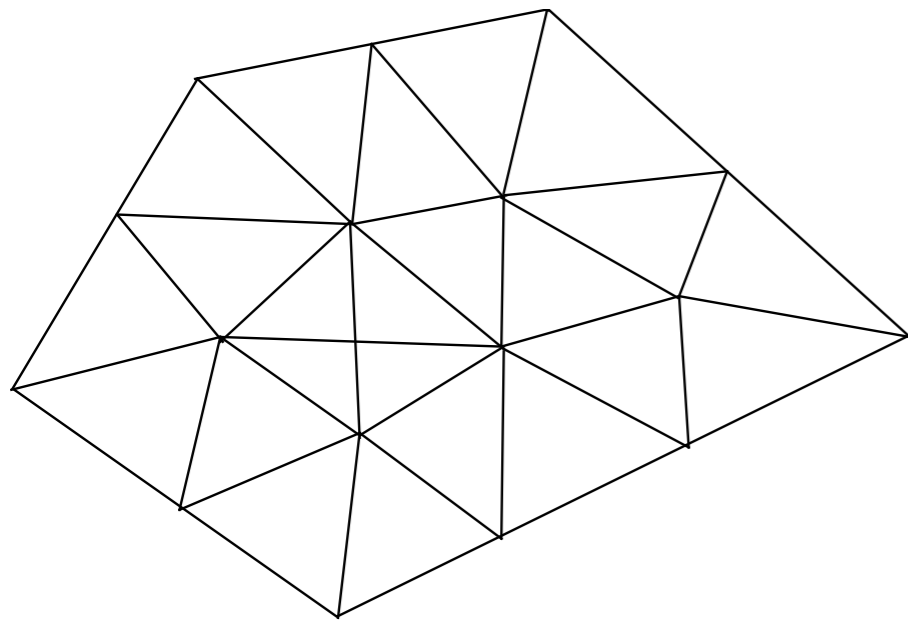
- The test space  $H_{\Gamma_D}^1(\Omega) = \{v \in H^1(\Omega) | v = 0 \text{ on } \Gamma_D\}$  with weak derivatives and equipped with

$$\langle u, v \rangle := \int_{\Omega} (uv + \nabla u \cdot \nabla v) dx$$

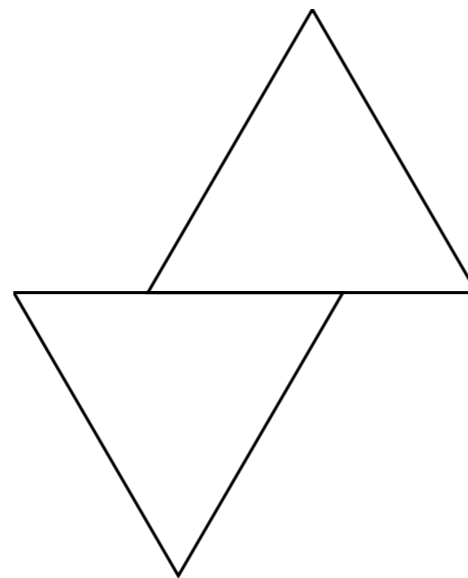
is a special Hilbert space, a so-called *Sobolev space*

- Now keep in mind that the so defined *function-* and *test-spaces*  $H^1(\Omega)$  and  $H_{\Gamma_D}^1(\Omega)$  are infinite-dimensional vector spaces (the latter being a proper sub-vectorspace of the former)
- This makes it difficult for a direct numerical implementation of the weak formulation (computers can't handle infinities very well...)
- We need to restrict ourselves (and set up) a convenient, finite-dimensional version of the two different Sobolev-spaces involved here
- These surrogates, being finite-dimensional vector-spaces, therefore feature a finite basis as well
  - The weak formulation can then be recast in terms of a cleverly chosen basis („*hat-functions*“)
  - This leads to a linear system of equations for the approximate solution of the PDE within the particular function space
- An obvious question then is: How well does this approximation represent the true solution of the original problem (existence and uniqueness, convergence behavior, refinement, etc.)
  - We're not going to delve into this; see literature list at the end of this talk

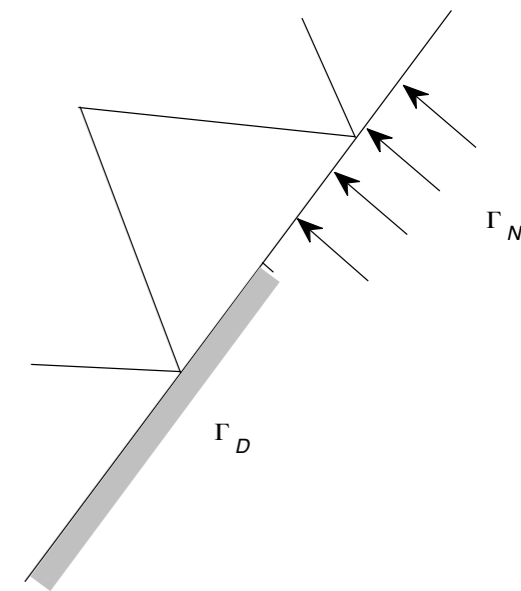
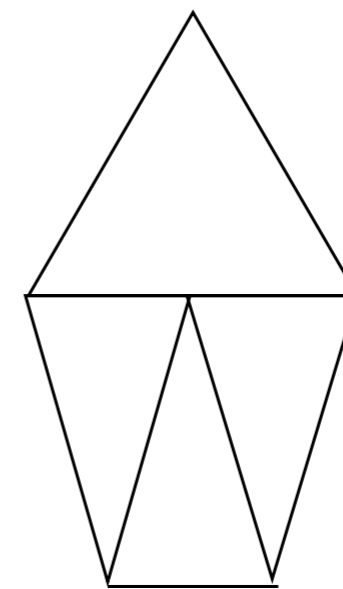
- In order to solve our test problem numerically, we need to discretize the physical domain (**triangulation**), the function space (**finite elements**) and the weak formulation (**assembly**)
- A triangulation  $\mathcal{T}_h$  of  $\Omega$  is a subdivision of this domain into triangles, so that
  - if two triangles have some intersection, it is either on a common vertex or a common full edge. In particular, two triangles do not overlap
  - the triangulation has to respect the partition of the boundary into Dirichlet and Neumann boundary



valid triangulation



hanging nodes



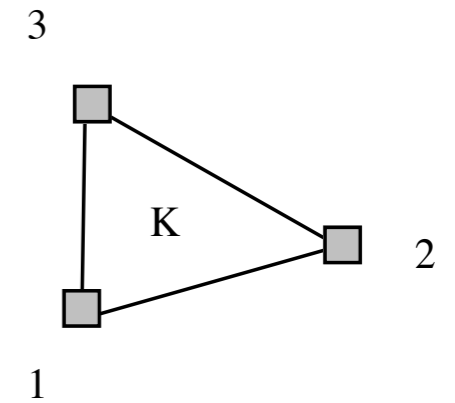
improper partitioning

- Consider bi-linear functions on one of these triangles

$$p \in \mathbb{P}_1 = \{a_0 + a_1x_1 + a_2x_2 \mid a_0, a_1, a_2 \in \mathbb{R}\}$$

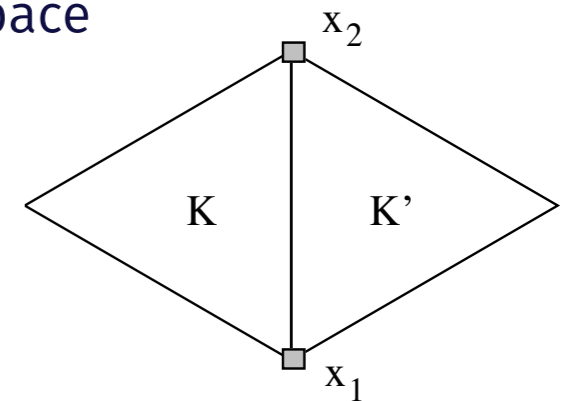
- They are uniquely determined by

- either from its three defining coefficients
- or from its values on the three vertices of a triangle  $K$
- in particular, the value of  $p \in \mathbb{P}_1$  on any edge of the triangle depends only on the values of  $p$  at the two attached vertices



- Two piecewise linear functions on triangles sharing a common edge can be continuously glued together. Do this globally and arrive at the  $\mathbb{P}_1$  - finite element space

$$V_h = \{u_h \in C^0(\bar{\Omega}) \mid u_h|_K \in \mathbb{P}_1 \forall K \in \mathcal{T}_h\}$$

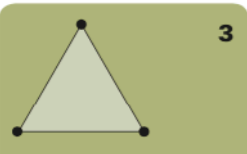


- Any element of this space is uniquely determined by its values on the nodes of the triangulation (here, nodes = vertices)
  - This space will serve as finite-dimensional surrogate for the function space  $H^1(\Omega)$

- Remark: There is a whole 'zoo' of tailor-made elements (Periodic Table of Finite Elements)

**n=2**

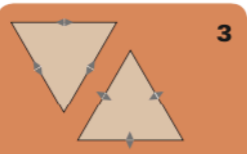
**r=1**



**P<sub>1</sub>**  $\mathcal{P}_1 \Lambda^0(\Delta_2)$

$3 \times \mathcal{P}_1 \Lambda^0(\Delta_2) = 3$


⚡ ("P", triangle, 1)



**RT<sub>1</sub><sup>[e/f]</sup>**  $\mathcal{P}_1 \Lambda^1(\Delta_2)$

$3 \times \mathcal{P}_1 \Lambda^0(\Delta_2) = 3$

⚡ ("RT[E,F]", triangle, 1)



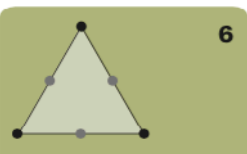
**dP<sub>0</sub>**  $\mathcal{P}_1 \Lambda^2(\Delta_2)$

$1 \times \mathcal{P}_1 \Lambda^2(\Delta_2) = 1$

⚡ ("DP", triangle, 0)

[e/f]

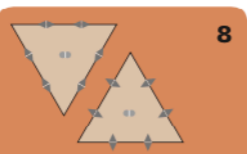
**r=2**



**P<sub>2</sub>**  $\mathcal{P}_2 \Lambda^0(\Delta_2)$

$3 \times \mathcal{P}_2 \Lambda^0(\Delta_2) + 3 \times \mathcal{P}_2 \Lambda^0(\Delta_2) = 6$


⚡ ("P", triangle, 2)



**RT<sub>2</sub><sup>[e/f]</sup>**  $\mathcal{P}_2 \Lambda^1(\Delta_2)$

$3 \times \mathcal{P}_2 \Lambda^0(\Delta_2) + 1 \times \mathcal{P}_2 \Lambda^1(\Delta_2) = 8$

⚡ ("RT[E,F]", triangle, 2)

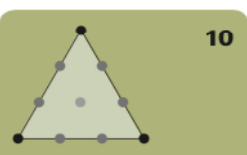


**dP<sub>1</sub>**  $\mathcal{P}_2 \Lambda^2(\Delta_2)$

$1 \times \mathcal{P}_2 \Lambda^2(\Delta_2) = 3$

⚡ ("DP", triangle, 1)

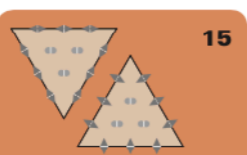
**r=3**



**P<sub>3</sub>**  $\mathcal{P}_3 \Lambda^0(\Delta_2)$

$3 \times \mathcal{P}_3 \Lambda^0(\Delta_2) + 3 \times \mathcal{P}_3 \Lambda^0(\Delta_2) + 1 \times \mathcal{P}_3 \Lambda^0(\Delta_2) = 10$


⚡ ("P", triangle, 3)



**RT<sub>3</sub><sup>[e/f]</sup>**  $\mathcal{P}_3 \Lambda^1(\Delta_2)$

$3 \times \mathcal{P}_3 \Lambda^0(\Delta_2) + 1 \times \mathcal{P}_3 \Lambda^1(\Delta_2) = 15$

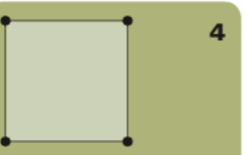
⚡ ("RT[E,F]", triangle, 3)



**dP<sub>2</sub>**  $\mathcal{P}_3 \Lambda^2(\Delta_2)$

$1 \times \mathcal{P}_3 \Lambda^2(\Delta_2) = 6$

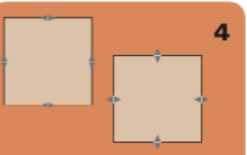
⚡ ("DP", triangle, 2)



**Q<sub>1</sub>**  $\mathcal{Q}_1 \Lambda^0(\square_2)$

$4 \times \mathcal{Q}_1 \Lambda^0(\square_2) = 4$

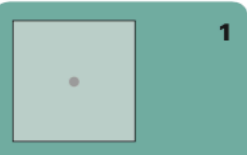
⚡ ("Q", quadrilateral, 1)



**RTc<sub>1</sub><sup>[e/f]</sup>**  $\mathcal{Q}_1 \Lambda^1(\square_2)$

$4 \times \mathcal{Q}_1 \Lambda^0(\square_2) = 4$

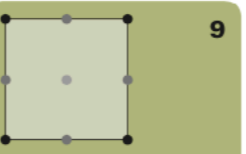
⚡ ("RT[C,E,F]", quadrilateral, 1)



**dQ<sub>0</sub>**  $\mathcal{Q}_1 \Lambda^2(\square_2)$

$1 \times \mathcal{Q}_1 \Lambda^2(\square_2) = 1$

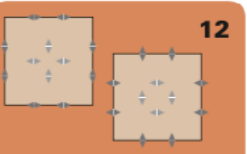
⚡ ("DQ", quadrilateral, 0)



**Q<sub>2</sub>**  $\mathcal{Q}_2 \Lambda^0(\square_2)$

$4 \times \mathcal{Q}_2 \Lambda^0(\square_2) + 4 \times \mathcal{Q}_2 \Lambda^0(\square_2) + 1 \times \mathcal{Q}_2 \Lambda^0(\square_2) = 9$

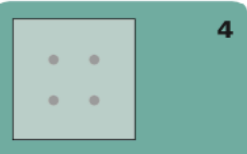
⚡ ("Q", quadrilateral, 2)



**RTc<sub>2</sub><sup>[e/f]</sup>**  $\mathcal{Q}_2 \Lambda^1(\square_2)$

$4 \times \mathcal{Q}_2 \Lambda^0(\square_2) + 1 \times \mathcal{Q}_2 \Lambda^1(\square_2) = 12$

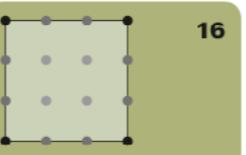
⚡ ("RT[C,E,F]", quadrilateral, 2)



**dQ<sub>1</sub>**  $\mathcal{Q}_2 \Lambda^2(\square_2)$

$1 \times \mathcal{Q}_2 \Lambda^2(\square_2) = 4$

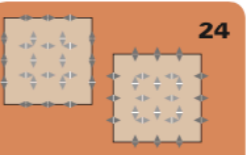
⚡ ("DQ", quadrilateral, 1)



**Q<sub>3</sub>**  $\mathcal{Q}_3 \Lambda^0(\square_2)$

$4 \times \mathcal{Q}_3 \Lambda^0(\square_2) + 4 \times \mathcal{Q}_3 \Lambda^0(\square_2) + 1 \times \mathcal{Q}_3 \Lambda^0(\square_2) = 16$

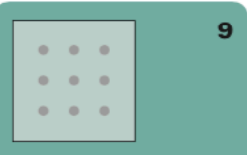
⚡ ("Q", quadrilateral, 3)



**RTc<sub>3</sub><sup>[e/f]</sup>**  $\mathcal{Q}_3 \Lambda^1(\square_2)$

$4 \times \mathcal{Q}_3 \Lambda^0(\square_2) + 1 \times \mathcal{Q}_3 \Lambda^1(\square_2) = 24$

⚡ ("RT[C,E,F]", quadrilateral, 3)




**dQ<sub>2</sub>**  $\mathcal{Q}_3 \Lambda^2(\square_2)$

$1 \times \mathcal{Q}_3 \Lambda^2(\square_2) = 9$

⚡ ("DQ", quadrilateral, 2)

**n=3**

**r=1**




**P<sub>1</sub>**  $\mathcal{P}_1 \Lambda^0(\Delta_3)$

$4 \times \mathcal{P}_1 \Lambda^0(\Delta_3) = 4$

⚡ ("P", tetrahedron, 1)

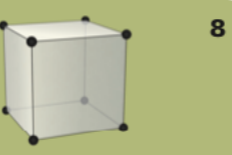
**r=2**



**P<sub>2</sub>**  $\mathcal{P}_2 \Lambda^0(\Delta_3)$

$4 \times \mathcal{P}_2 \Lambda^0(\Delta_3) + 6 \times \mathcal{P}_2 \Lambda^0(\Delta_3) = 10$

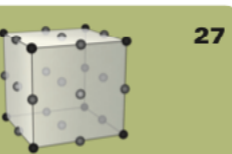
⚡ ("P", tetrahedron, 2)



**Q<sub>1</sub>**  $\mathcal{Q}_1 \Lambda^0(\square_3)$

$8 \times \mathcal{Q}_1 \Lambda^0(\square_3) = 8$

⚡ ("Q", hexahedron, 1)



**Q<sub>2</sub>**  $\mathcal{Q}_2 \Lambda^0(\square_3)$

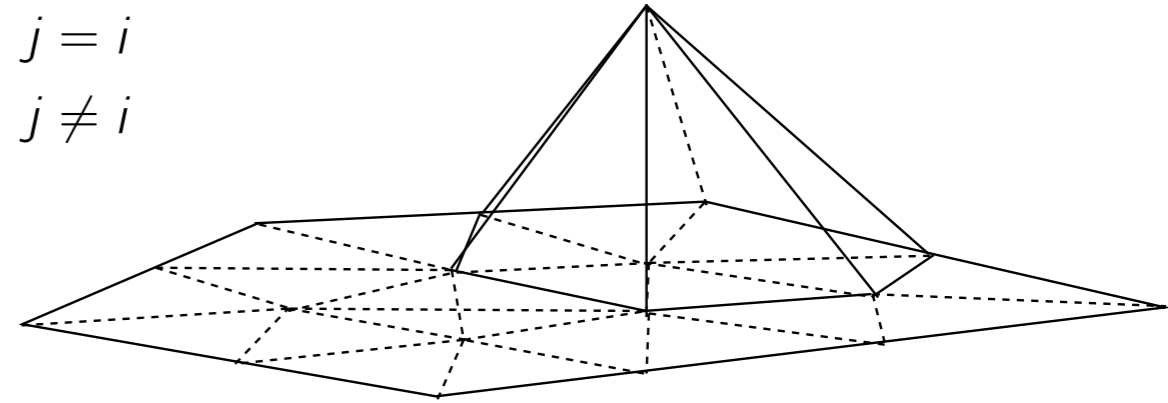
$8 \times \mathcal{Q}_2 \Lambda^0(\square_3) + 12 \times \mathcal{Q}_2 \Lambda^0(\square_3) + 6 \times \mathcal{Q}_2 \Lambda^0(\square_3) + 1 \times \mathcal{Q}_2 \Lambda^0(\square_3) = 27$

⚡ ("Q", hexahedron, 2)



- Let's denote the nodes with  $\mathbf{p}_j$ ,  $j = 1 \dots N = \#\{\text{vertices}\}$ . For a fixed node, consider the unique function

$$\varphi_i(\mathbf{p}_j) = \delta_{ij} = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases}$$



- Now take any  $u_h \in V_h$ ; it is easy to see (why?) that

$$u_h = \sum_{j=1}^N u_h(\mathbf{p}_j) \varphi_j$$

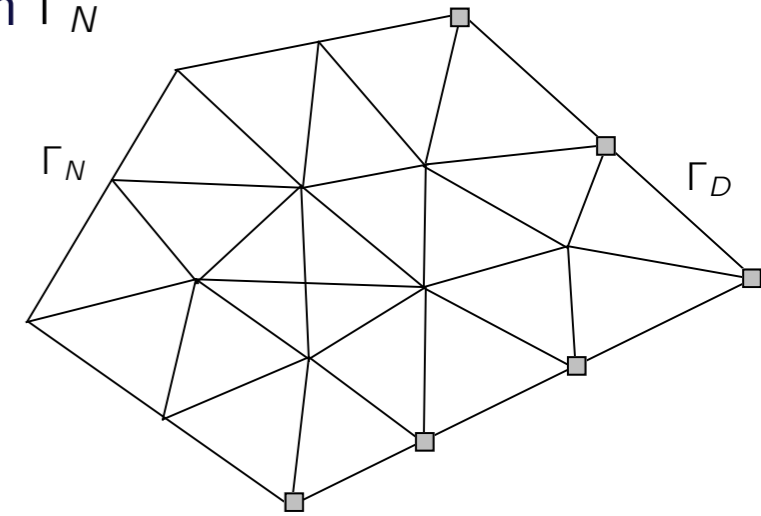
- Therefore  $\{\varphi_i \mid i = 1 \dots N\}$  is a basis of  $V_h$  and

$$\dim V_h = N = \#\{\text{vertices}\}$$

- It's even a very special basis, since the expansion coefficients are values of the function on nodes

$$u_h = \sum_{j=1}^N c_j \varphi_j \text{ with } c_j = u_h(\mathbf{p}_j)$$

- The boundary needs a separate treatment
  - a Dirichlet edge is an edge of a triangle that lies on  $\Gamma_D$ ; its vertices are the Dirichlet nodes
  - a Neumann edge is an edge of a triangle that is contained in  $\Gamma_N$
  - if a node belongs to  $\Gamma_D$  and  $\Gamma_N$ , its a Dirichlet node



- The finite element representation of  $H_{\Gamma_D}^1(\Omega)$  is

$$V_h^{\Gamma_D} = V_h \cap H_{\Gamma_D}^1 = \{v_h \in V_h \mid v_h = 0 \text{ on } \Gamma_D\}$$

- $v_h \in V_h^{\Gamma_D}$  if and only if it vanishes on all Dirichlet nodes (why?). Can we find a basis for  $V_h^{\Gamma_D}$ ?
- If we separate the number of nodes into free/independent nodes and Dirichlet nodes, a basis of  $V_h^{\Gamma_D}$  is given by (why?)

$$v_h = \sum_{j \in \text{Ind}} v_j \varphi_j, \text{ with } v_j = v_h(\mathbf{p}_j)$$

- This proves that

$$\dim V_h^{\Gamma_D} = \#\{\text{Ind}\} = \#\{\text{nodes}\} - \#\{\text{Dirichlet nodes}\}$$

- After all these preliminaries, we're finally able to derive the FEM-version of our model problem. Recall, that our objective is to

- find  $u \in H^1(\Omega)$ , such that

$$u = g_0 \text{ on } \Gamma_D$$

$$\int_{\Omega} \nabla u \nabla v + c \int_{\Omega} uv = \int_{\Omega} f v + \int_{\Gamma_N} g_1 v \quad \forall v \in H_{\Gamma_D}^1(\Omega)$$

- The associated discrete problem is then given by

- find  $u_h \in V_h$ , such that

$$u_h(\mathbf{p}_j) = g_0(\mathbf{p}_j) \quad \forall j \in \text{Dir}$$

$$\int_{\Omega} \nabla u_h \nabla v_h + c \int_{\Omega} u_h v_h = \int_{\Omega} f v_h + \int_{\Gamma_N} g_1 v_h \quad \forall v_h \in V_h^{\Gamma_D}$$

- This means especially

- we look for solutions in the finite-dimensional FE-space instead of the whole (infinite-dimensional) Sobolev space ( $\dim V_h = N = \#\{\text{vertices}\}$ )
- the values on Dirichlet nodes are already fixed (only  $\#\{\text{Ind}\}$  unknowns)
- testing space is reduced to  $V_h^{\Gamma_D}$  (and  $\dim V_h^{\Gamma_D} = \#\{\text{Ind}\}$ )

- Since we know a basis for  $V_h^{\Gamma^D}$ , this is equivalent to

$$\int_{\Omega} \nabla u_h \nabla \varphi_i + c \int_{\Omega} u_h \varphi_i = \int_{\Omega} f \varphi_i + \int_{\Gamma_N} g_1 \varphi_i \quad \forall i \in \text{Ind}$$

- Next, let's write

$$\begin{aligned} u_h &= \sum_{j \in \text{Ind}} u_j \varphi_j + \sum_{j \in \text{Dir}} u_j \varphi_j \\ &= \sum_{j \in \text{Ind}} u_j \varphi_j + \sum_{j \in \text{Dir}} g_0(\mathbf{p}_j) \varphi_j \end{aligned}$$

- Taking the gradient, inserting and rearranging the Dirichlet-data to the right leads to

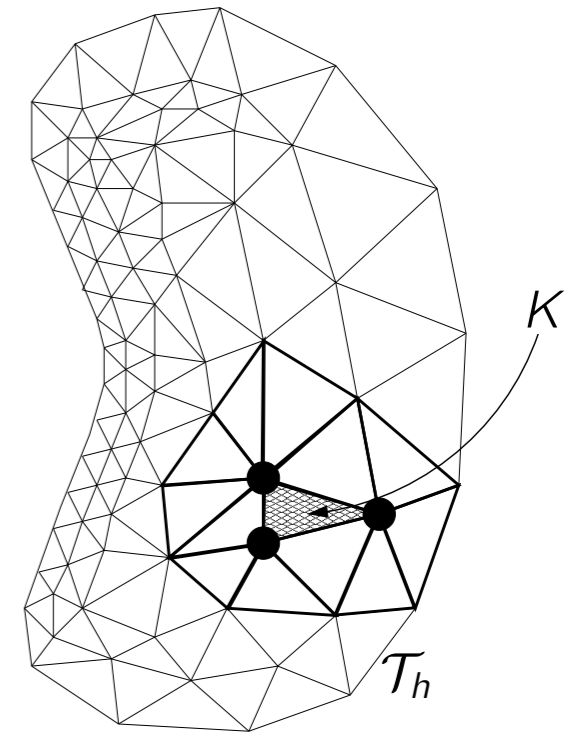
$$\begin{aligned} \sum_{j \in \text{Ind}} \left( \int_{\Omega} \nabla \varphi_j \nabla \varphi_i + c \int_{\Omega} \varphi_j \varphi_i \right) u_j &= \int_{\Omega} f \varphi_i + \int_{\Gamma_N} g_1 \varphi_i \\ &\quad - \sum_{j \in \text{Dir}} \left( \int_{\Omega} \nabla \varphi_j \nabla \varphi_i + c \int_{\Omega} \varphi_j \varphi_i \right) g_0(\mathbf{p}_j) \end{aligned}$$

- This represents a matrix-vector equation which can be inverted to solve for the  $\#\{\text{Ind}\}$  unknowns  $u_j$
- Of course, the matrix- and righthand side-entries need to be computed numerically (**assembly**)

- Consider for example the matrix contributions (RHS works similar)

$$w_{ij} = \int_{\Omega} \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} \int_K \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} w_{ij}^K$$

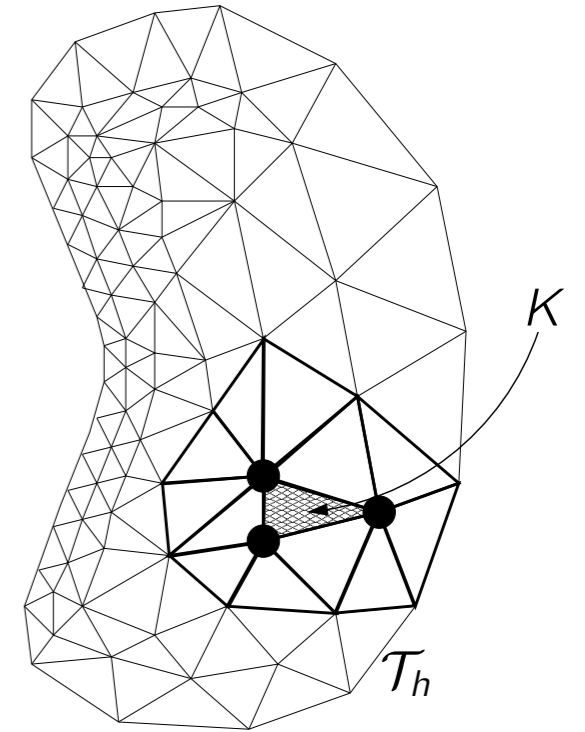
$$m_{ij} = \int_{\Omega} \varphi_i \varphi_j = \sum_{K \in \mathcal{T}_h} \int_K \varphi_i \varphi_j = \sum_{K \in \mathcal{T}_h} m_{ij}^K$$



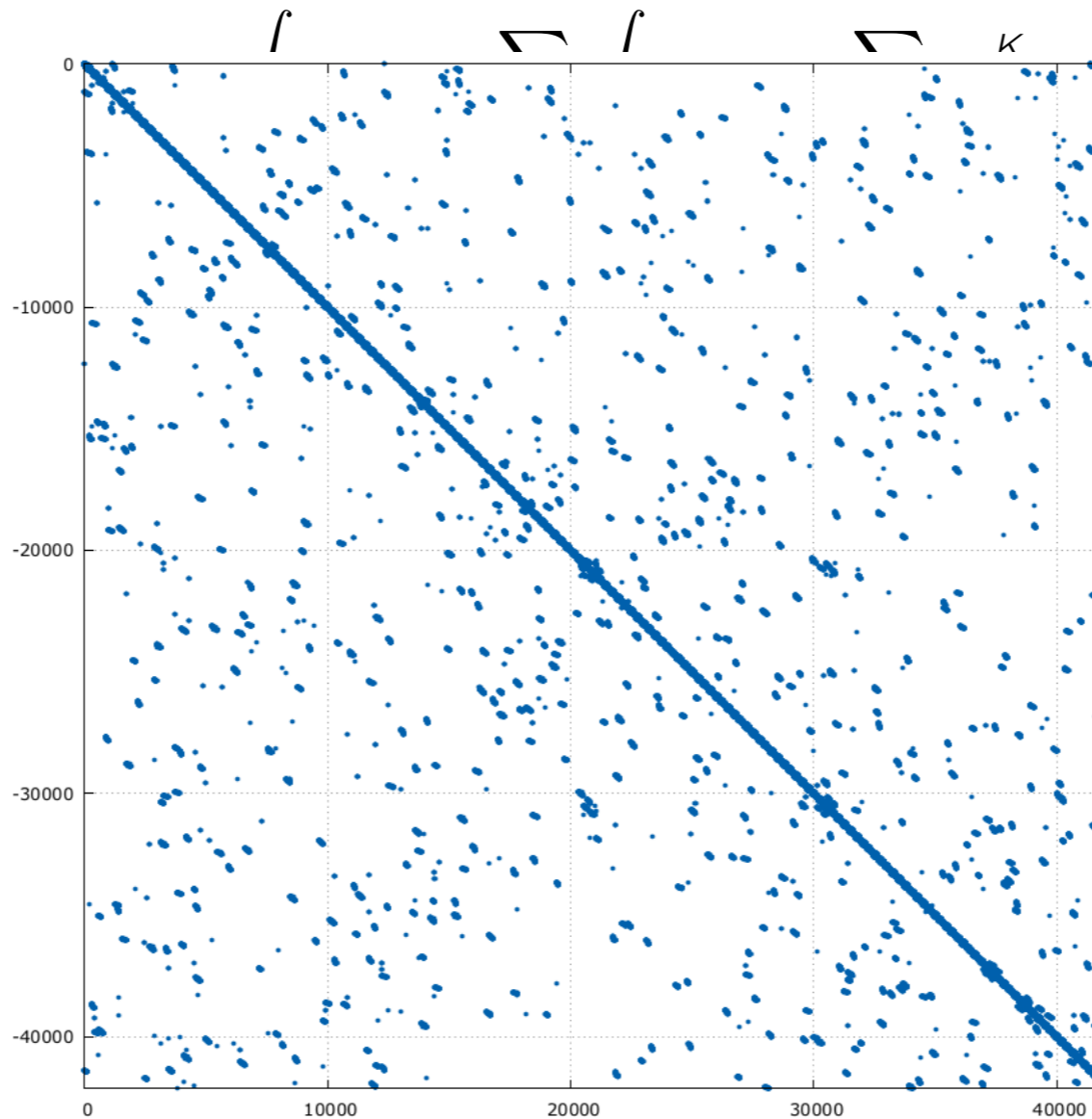
- In theory, the indices  $i, j$  run over all independent nodes, **but**
  - due to the special form of our nodal basis functions, only basis functions corresponding to the nodes of the triangle contribute (why?)
  - the matrices  $w_{ij}^K, m_{ij}^K$  therefore only have 9 entries in total, on positions depending on the global numbering of the nodes
  - all these local contributions from the various cells need to be assembled to the global system matrix and righthand-side
- This leads to a sparse linear system, which is the reason why it is possible to solve problems with millions/billions of unknowns (use CG for example)

- Consider for example the matrix contributions (RHS works similar)

$$w_{ij} = \int_{\Omega} \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} \int_K \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} w_{ij}^K$$



- In theory, the indices
  - due to the sparse structure
  - corresponding to the local support of the basis functions
  - the matrices are sparse and their sparsity pattern depends on the positions of the nodes
  - on the global system
  - all these local matrices are assembled to the global system

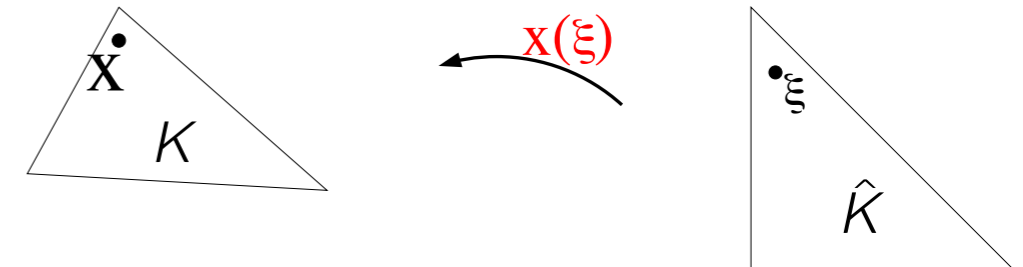


basis functions  
 positions depending  
 are assembled to the

- This leads to a sparse linear system, which is the reason why it is possible to solve problems with millions/billions of unknowns (use CG for example)

- For efficient evaluation of the nodal basis functions and their gradients (and also to increase performance and accuracy), all calculations are performed on a **reference cell**

- The Jacobian of the map  $x(\xi)$  is  $J$



- We then have for example

$$\int_K f \varphi_i dx = \int_{\hat{K}} \hat{f}(\xi) \hat{\varphi}_i(\xi) |\det J| d\xi$$

$$\int_K \varphi_i \varphi_j dx = \int_{\hat{K}} \hat{\varphi}_i(\xi) \hat{\varphi}_j(\xi) |\det J| d\xi$$

- The gradients are more tricky...

$$\int_K \nabla \varphi_i \nabla \varphi_j dx = \int_{\hat{K}} J^{-1} \hat{\nabla} \hat{\varphi}_i(\xi) J^{-1} \hat{\nabla} \hat{\varphi}_j(\xi) |\det J| d\xi$$

- The integrals are numerically evaluated using Gauss quadrature with quadrature points  $\xi_e$  and weights  $w_e$ , e.g.

$$\int_{\hat{K}} \hat{f}(\xi) \hat{\varphi}_i(\xi) |\det J| d\xi = \sum_{e=1}^N \hat{f}(\xi_e) \hat{\varphi}_i(\xi_e) |\det J(\xi_e)| w_e$$



- Want to do it from scratch? How does the assembly process might look like for  $\mathbb{P}_1$  - elements?

- Take as an example  $w_{ij} = \int_{\Omega} \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} \int_K \nabla \varphi_i \nabla \varphi_j = \sum_{K \in \mathcal{T}_h} w_{ij}^K$

- For each triangle, assign a number to its vertices  $\mathbf{p}_1^K$ ,  $\mathbf{p}_2^K$ ,  $\mathbf{p}_3^K$

- Consider the three functions  $N_1^K$ ,  $N_2^K$ ,  $N_3^K \in \mathbb{P}_1$  with

$$N_{\alpha}^K(\mathbf{p}_{\beta}^K) = \delta_{\alpha\beta} \quad \alpha, \beta = 1, 2, 3$$

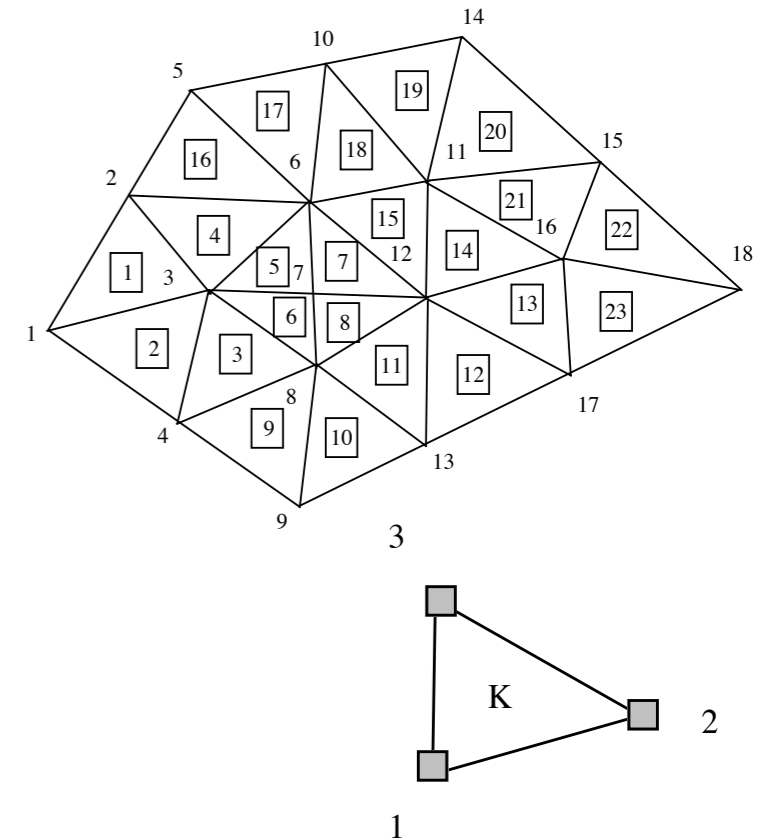
- Let  $n_{\alpha}$  be the global node number of the local node with number  $\alpha$  of the triangle  $K$ . Then

$$N_{\alpha}^K = \varphi_{n_{\alpha}} \text{ on } K$$

- This allows us to compute

$$k_{\alpha\beta}^K = \int_K \nabla N_{\alpha}^K \cdot \nabla N_{\beta}^K = w_{n_{\alpha}n_{\beta}}^K \quad \alpha, \beta = 1, 2, 3$$

- The global matrix is then assembled by all sub-matrices  $\mathbf{W} = \sum_{K \in \mathcal{T}_h} \mathbf{W}^K$

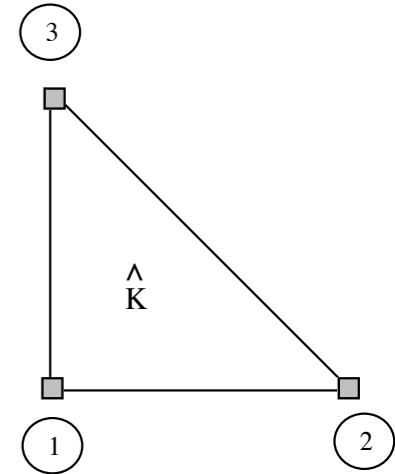


- As previously described, use reference elements to do the calculations. For triangles, it is

$$\hat{\mathbf{p}}_1 = (0, 0), \quad \hat{\mathbf{p}}_2 = (1, 0), \quad \hat{\mathbf{p}}_3 = (0, 1)$$

- The functions  $\hat{N}_\alpha(\hat{\mathbf{p}}_\beta) = \delta_{\alpha\beta}$  are given by

$$\hat{N}_1 = 1 - \xi - \eta, \quad \hat{N}_2 = \xi, \quad \hat{N}_3 = \eta$$



- For the triangle  $K$  with  $\mathbf{p}_1^K = (x_1, y_1)$ ,  $\mathbf{p}_2^K = (x_2, y_2)$ ,  $\mathbf{p}_3^K = (x_3, y_3)$ , we have the bijective affine transformation  $F_K : (\xi, \eta) \rightarrow (x, y)$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix}}_{B_K} \begin{pmatrix} \xi \\ \eta \end{pmatrix} + \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

- It is easy to show, that

$$F_K(\hat{\mathbf{p}}_\alpha) = \mathbf{p}_\alpha, \quad \alpha = 1, 2, 3$$

- Furthermore

$$\hat{N}_\alpha = N_\alpha^K \circ F_K, \quad \alpha = 1, 2, 3$$

$$N_\alpha^K = \hat{N}_\alpha \circ F_K^{-1}, \quad \alpha = 1, 2, 3 \quad \text{i.e.} \quad N_\alpha^K(x, y) = \hat{N}_\alpha(F_K^{-1}(x, y))$$

- Since the inverse affine transformation is straightforward to compute, this is a simple way of evaluating the functions  $N_\alpha^K$  needed for the FE-integrals

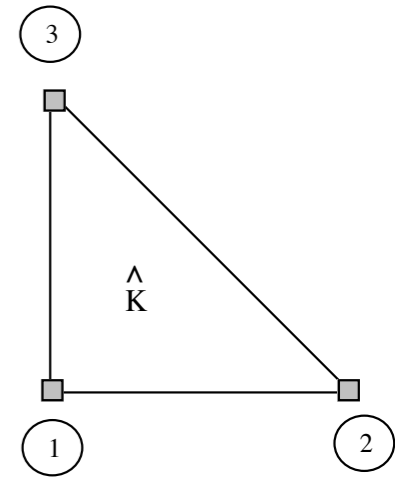
- Evaluating gradients needs more care, since one has to use the chain rule

- For  $\nabla = \begin{pmatrix} \partial_x \\ \partial_y \end{pmatrix}$  and  $\hat{\nabla} = \begin{pmatrix} \partial_\xi \\ \partial_\eta \end{pmatrix}$ , it is for an arbitrary function  $\phi$

$$B_K^T (\nabla \phi \circ F_K) = \hat{\nabla} (\phi \circ F_K)$$

or (with  $\phi = N_\alpha^K$ )

$$\nabla N_\alpha^K = B_K^{-T} ((\hat{\nabla} \hat{N}_\alpha) \circ F_K^{-1})$$



- Here, we have

$$B_K^{-T} = \frac{1}{\det B_K} \begin{bmatrix} y_3 - y_1 & -(y_2 - y_1) \\ -(x_2 - x_1) & x_2 - x_1 \end{bmatrix}$$

with

$$\det B_K = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

- Luckily, for this elementary method here, the gradients are constant vectors

$$\hat{\nabla} \hat{N}_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \quad \hat{\nabla} \hat{N}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \hat{\nabla} \hat{N}_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

- Integration is also performed on the reference element; we therefore have

$$\int_K N_\beta^K N_\alpha^K = |\det B_K| \int_{\hat{K}} \hat{N}_\beta \hat{N}_\alpha$$

- We're done here since only the determinant is dependent on the current triangle and

$$\hat{\mathbf{K}}_0 = \left[ \int_{\hat{K}} \hat{N}_\beta \hat{N}_\alpha \right]_{\alpha, \beta} = \frac{1}{24} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

- For the gradients, it is

$$\begin{aligned} \int_K \nabla N_\beta^K \cdot \nabla N_\alpha^K &= |\det B_K| \int_{\hat{K}} (B_K^{-T} \hat{\nabla} \hat{N}_\beta) \cdot (B_K^{-T} \hat{\nabla} \hat{N}_\alpha) \\ &= |\det B_K| \int_{\hat{K}} C_K \hat{\nabla} \hat{N}_\beta \cdot \hat{\nabla} \hat{N}_\alpha \end{aligned}$$

where

$$C_K = B_K^{-1} B_K^{-T} = \begin{bmatrix} c_{11}^K & c_{12}^K \\ c_{12}^K & c_{22}^K \end{bmatrix}$$

is a symmetric 2x2 matrix that depends only on the triangle

- If we write

$$\hat{\mathbf{K}}_{\xi\xi} = \left[ \int_{\hat{K}} \partial_{\xi} \hat{N}_{\beta} \partial_{\xi} \hat{N}_{\alpha} \right]_{\alpha,\beta} = \frac{1}{2} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\hat{\mathbf{K}}_{\eta\eta} = \left[ \int_{\hat{K}} \partial_{\eta} \hat{N}_{\beta} \partial_{\eta} \hat{N}_{\alpha} \right]_{\alpha,\beta} = \frac{1}{2} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

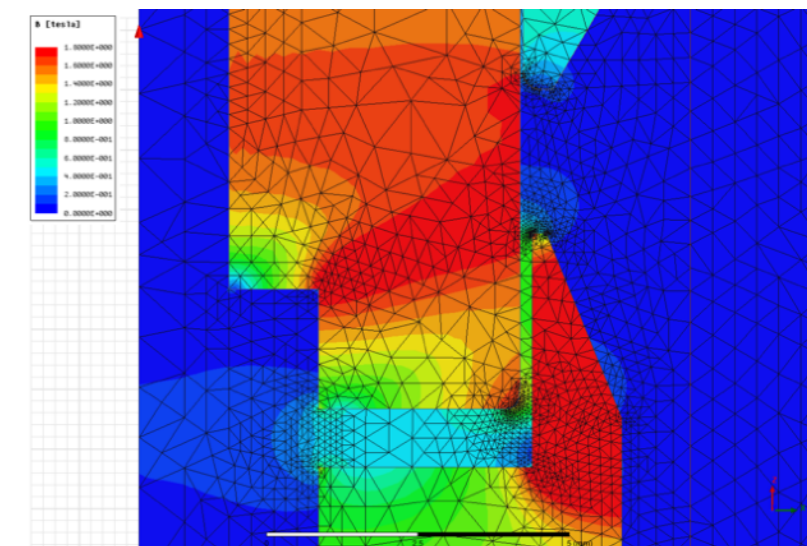
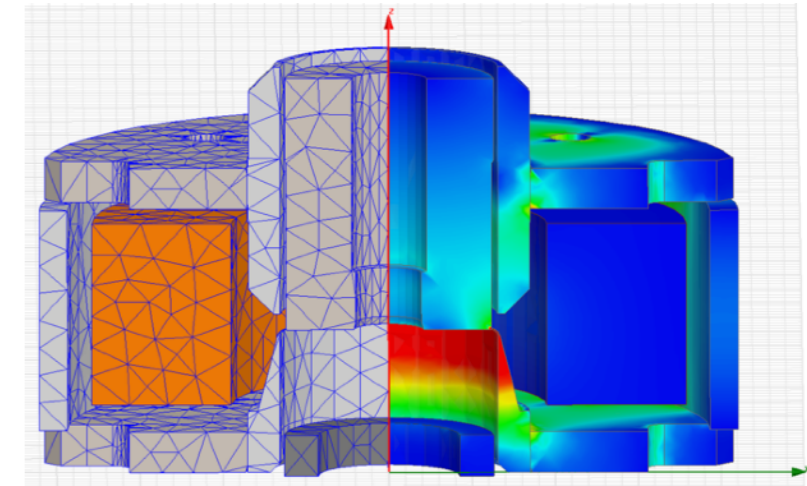
$$\hat{\mathbf{K}}_{\xi\eta} = \left[ \int_{\hat{K}} \partial_{\xi} \hat{N}_{\beta} \partial_{\eta} \hat{N}_{\alpha} \right]_{\alpha,\beta} = \frac{1}{2} \begin{bmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

we finally arrive at

$$\left[ \int_K \nabla N_{\beta}^K \cdot \nabla N_{\alpha}^K \right]_{\alpha,\beta} = |\det B_K| (c_{11}^K \hat{\mathbf{K}}_{\xi\xi} + c_{22}^K \hat{\mathbf{K}}_{\eta\eta} + c_{22}^K (\hat{\mathbf{K}}_{\xi\eta} + \hat{\mathbf{K}}_{\xi\eta}^T))$$

- Doing this calculations for all elements of the triangulation completes the assembly of the system matrix; the same has to be done for the righthand-sides (and possible boundary integrals due to Neumann - conditions...)

- Most commercially available codes contain sophisticated routines for pre-processing...
  - automated meshing with tets, quads or mixed elements
  - simple ways of local mesh refinement
  - defeaturing
- running the simulation...
  - apply boundary conditions, forces, stresses etc.
  - implicit and explicit solvers for time-evolution, stationary solver
- and post-processing the results
  - visualization of the solution(s)
  - animations and arbitrary XY-plots
- We will try to solve a simple Poisson equation with similar OpenSource tools
  - Gmsh for meshing, deal.II as FE library, VisIt for visualization



- There are a couple of sophisticated OpenSource FE libraries available:

- MOOSE framework (Multiphysics Object-Oriented Software Environment)



- OpenFOAM (OpenSource Field Operation and Manipulation), mainly used for CFD



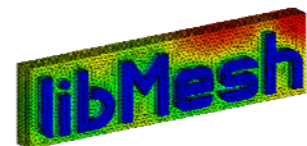
- The Fenics Project (uses Python scripting for solving PDEs)



- Agros2D (application package with GUI support for meshing and post-processing)



- libmesh (used by MOOSE under the hood)



- deal.ii (Differential Equations Analysis Library, used by Agros2D under the hood)

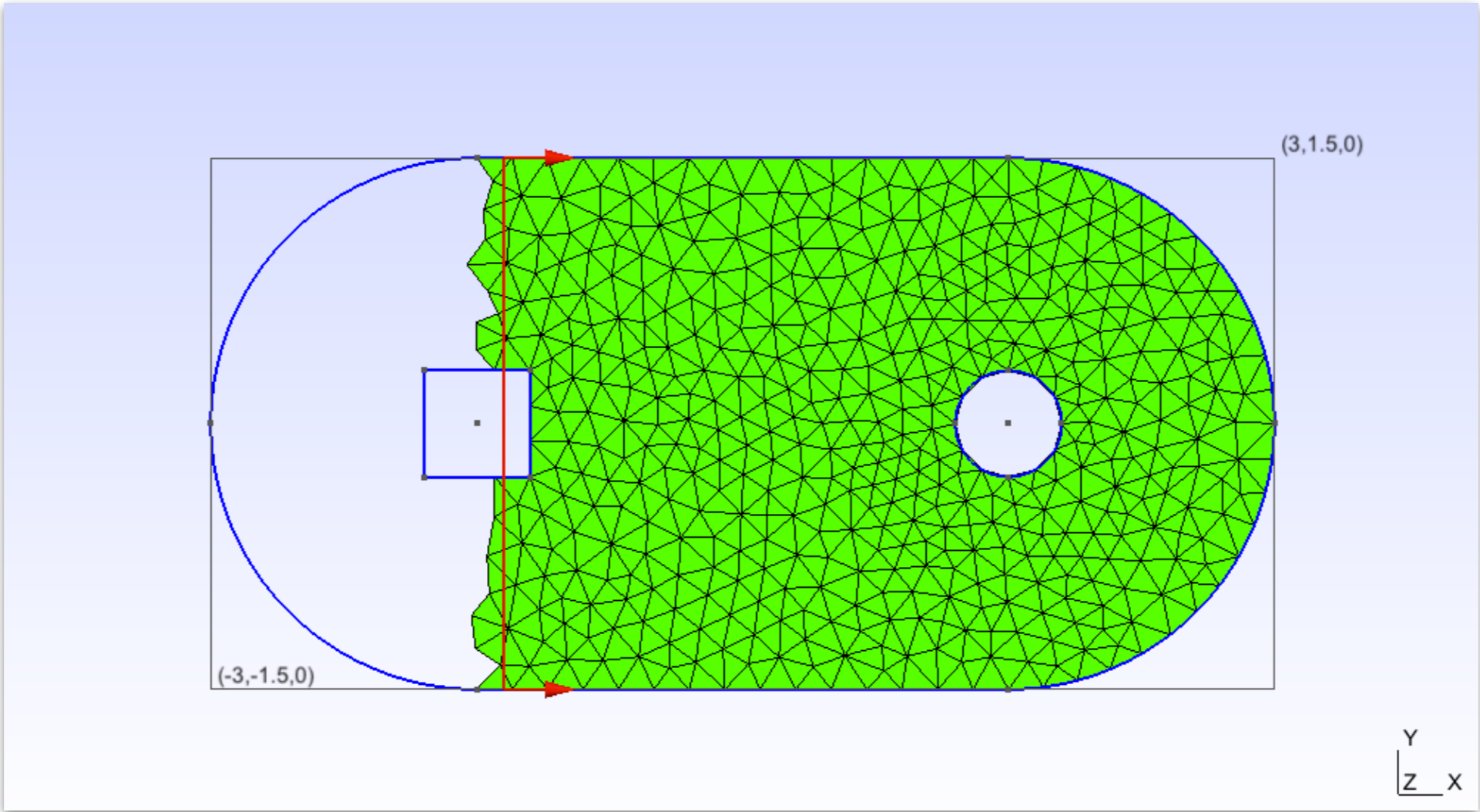


- The former four provide an additional abstraction layer to the more low-level APIs found in libmesh and deal.ii



# Demo

# Demo



# Demo

```

/* implementation of the assemble_system member function */
template <int dim>
void FEM_Example<dim>::assemble_system()
{
    // use Gauss quadrature with two quadrature points for every dimension
    QGauss<dim> quadrature_formula(2);

    // get values for shape functions, their gradients, the quadrature points and transformation coefficients
    FEValues<dim> fe_values(fe, quadrature_formula, update_values | update_gradients | update_quadrature_points | update_JxW_values);

    // get number of DoFs per cell, number of quadrature points, instantiate objects for rhs and coefficient arrays etc.
    const unsigned int dofs_per_cell = fe.dofs_per_cell;
    const unsigned int n_q_points = quadrature_formula.size();

    FullMatrix<double> cell_matrix(dofs_per_cell, dofs_per_cell);
    Vector<double> cell_rhs(dofs_per_cell);

    std::vector<types::global_dof_index> local_dof_indices(dofs_per_cell);

    const RightHandSide<dim> righthandside;
    std::vector<double> righthandside_values(n_q_points);

    // loop over all cells, quadrature points and DoFs to assemble the local matrices
    typename DoFHandler<dim>::active_cell_iterator cell = dof_handler.begin_active(), endc = dof_handler.end();
    for(; cell != endc; ++cell) {
        fe_values.reinit(cell);
        cell_matrix = 0;
        cell_rhs = 0;

        righthandside.value_list(fe_values.get_quadrature_points(), righthandside_values);

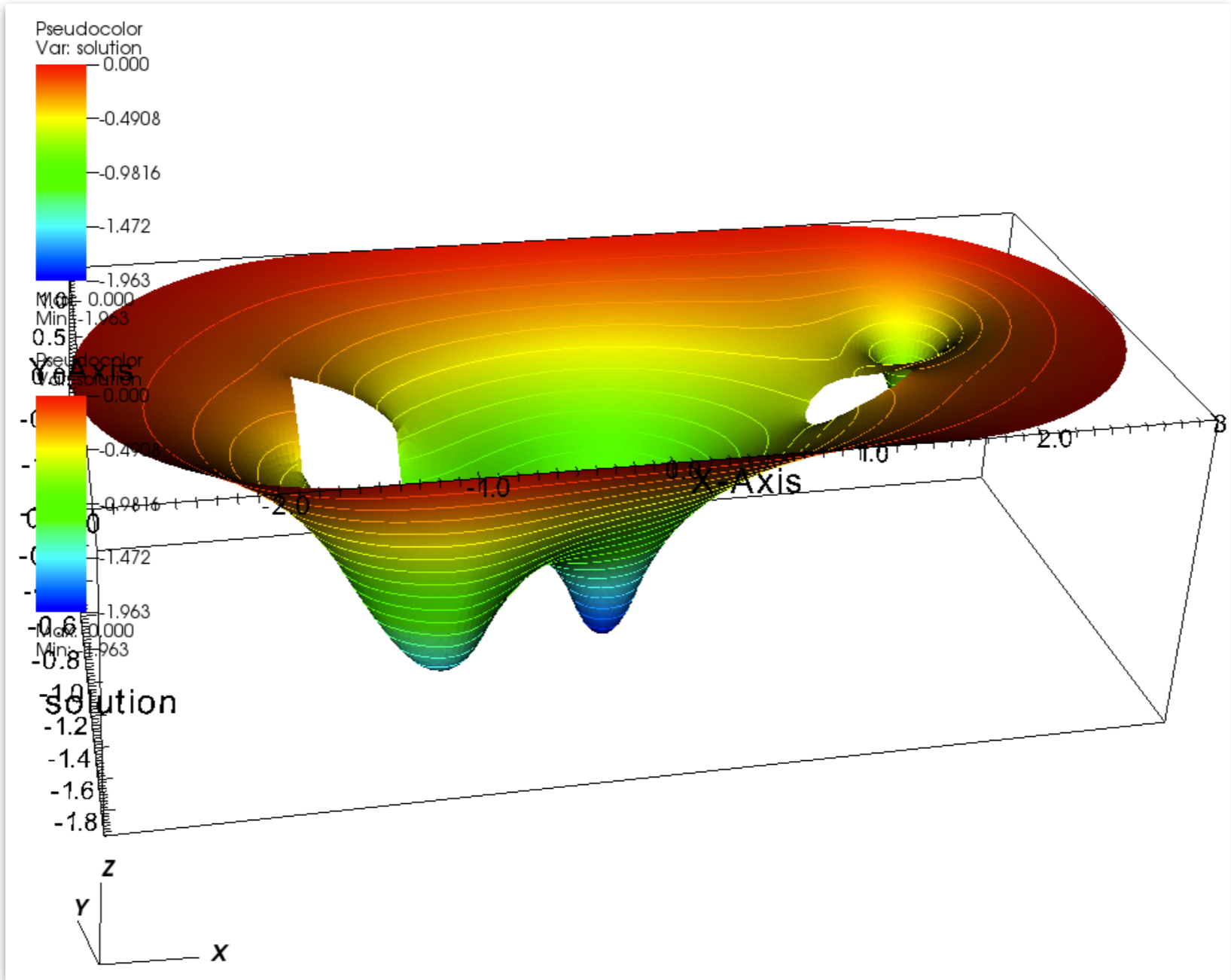
        for(unsigned int q_point = 0; q_point < n_q_points; ++q_point) {
            for(unsigned int i = 0; i < dofs_per_cell; ++i) {
                for(unsigned int j = 0; j < dofs_per_cell; ++j)
                    cell_matrix(i,j) += -1.0*fe_values.shape_grad(i,q_point) * fe_values.shape_grad(j,q_point) * fe_values.JxW(q_point);

                cell_rhs(i) += fe_values.shape_value(i, q_point) * righthandside_values[q_point] * fe_values.JxW(q_point);
            }
        }

        // copy the local contributions to the global matrix and rhs
        cell->get_dof_indices(local_dof_indices);
        constraints.distribute_local_to_global(cell_matrix, cell_rhs, local_dof_indices, system_matrix, system_rhs);
    }
}

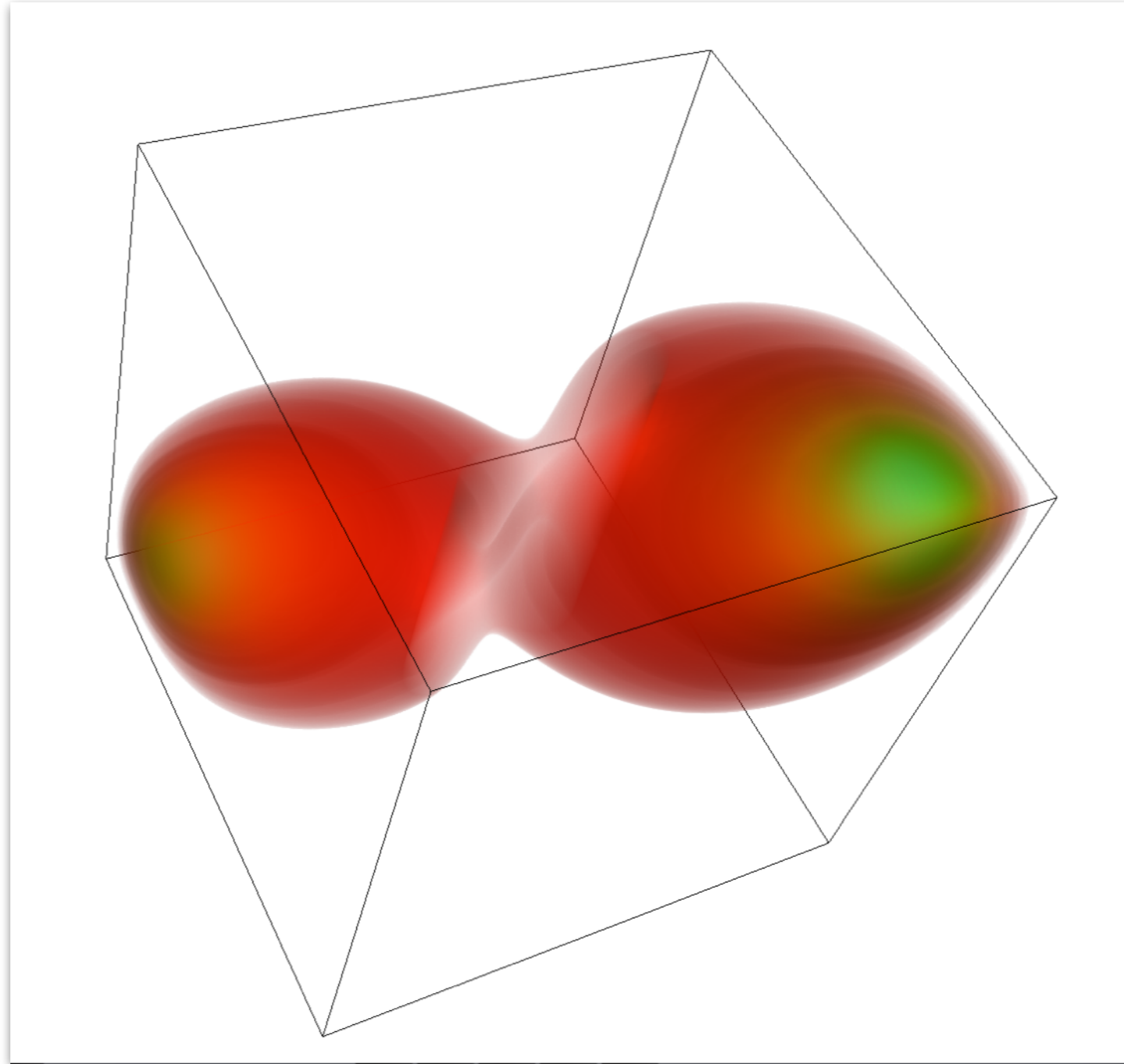
```

# Demo





# Demo



- Zhangxin Chen  
**Finite Element Methods and Their Applications**  
*Springer, 2005*
- Susanne C. Brenner, L. Ridgway Scott  
**The Mathematical Theory of Finite Element Methods**  
*Springer, 2007*
- J. N. Reddy  
**An Introduction to the Finite Element Method**  
*McGraw Hill, 2006*
- Dietrich Braess  
**Finite Elements: Theory, Fast Solvers and Applications in Solid Mechanics**  
*Cambridge University Press, 2007*
- Young W. Kwong, Hyochoong Bang  
**The Finite Element Method using MATLAB**  
*CRC Press, 1996*