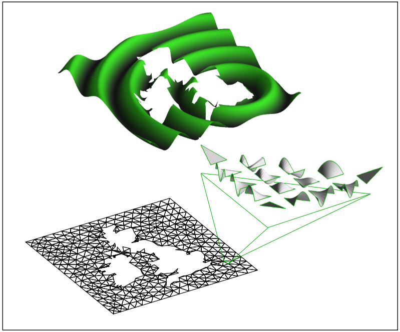
Nektar++: Technical Introduction



Chris Cantwell

7th February 2011

Imperial College London

Nektar++: A Technical Introduction

- Aim to introduce the *Nektar++* library and demonstrate its use in practice.
- Outline
  - Define the problem
    - Mesh Definition
    - Nektar++ XML file format
  - Solving the problem
    - Basic ingredients
    - Helmholtz equation in 2D
    - Unsteady Advection-Diffusion
- Hands-on tutorial.

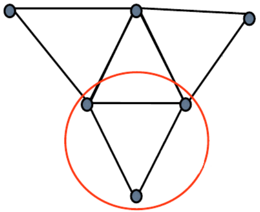
Useful prerequisites: C/C++, basic use of terminal

Imperial College London

Mesh Definition

Mesh defined in terms of:

- Vertices
- Edges
- Elements



Support for a range of elements:

- 1D: Segment
- 2D: Triangle, Quadrilateral
- 3D: Tetrahedron, Hexahedron

Mesh definition specified using XML.

XML file

- e**X**tensible **M**arkup **L**anguage - similar concept to HTML.
- A hierarchial document
  - tags (XML 'elements'), attributes, content.

```
<VERTEX>
  <V ID="0"> 0.1 1.0 0.0 </V>
</VERTEX>
```

- Open tag with <NAME>
- Close tag with matching </NAME>
- Each tag has a name (no uniqueness requirement)
- Each tag may contain one of more attributes (ID="0")
- Each tag may contain element text <V...> 0.1 1.0 0.0 </V>
- ...or other elements (<VERTEX><V>...</V></VERTEX>)

XML file

```xml
<NEKTAR>
    <GEOMETRY DIM="1" SPACE="2">
        <VERTEX>
            ...
        </VERTEX>
        <ELEMENT>
            ...
        </ELEMENT>
        <COMPOSITE>
            ...
        </COMPOSITE>
        <DOMAIN> C[0] </DOMAIN>
    </GEOMETRY>
    <EXPANSIONS>
      <E COMPOSITE="C[0]" NUMMODES="8" TYPE="MODIFIED"/>
    </EXPANSIONS>
    <CONDITIONS>
      ...
    </CONDITONS>
</NEKTAR>
```

Imperial College London

Mesh Definition

- Each component has unique index
- Tag name specifies component type
- Each vertex has 3 coordinates

  <V ID="0"> 0.5 1.0 0.0 </V>

- Edges defined by two vertices

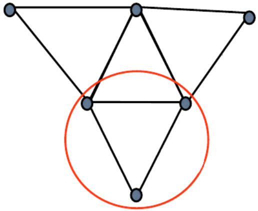
  <E ID="0"> 3 6 </E>

  (We also define faces in 3D)
- Elements defined by set of edges (2D) or faces (3D)

  <T ID="0"> 4 5 6 </T>
  <Q ID="20"> 17 18 21 20 </Q>

- Composites define groups of entities

  <C ID="0"> Q[0 − 4] </C>
  <C ID="1"> E[0 − 8] </C>



Element types (tag names):

- Segment(<S>)
- Triangle (<T>)
- Quadrilateral(<Q>)
- Tetrahedron(<A>)
- Hexahedron(<H>)

Imperial College London

Mesh Generation

- Define geometry in GMSH

```
...
Point(1) = {0,0,0};
Line(1) = {16,29};
Line Loop(54) =
    {1,-18,-17,16};
Plane Surface(54) = {54};
...
```

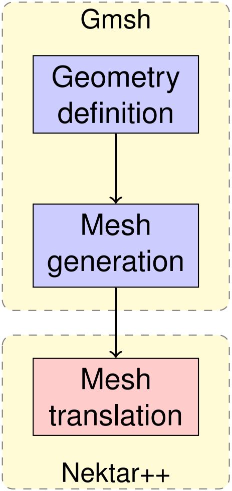
- Generate mesh

```
gmsh -2 example.geo
```

- Translate mesh

```
MeshConvert example.msh \
    example.xml
```

- Fill in boundary conditions, etc



Imperial College London

Using Nektar++: Basic Ingredients in 2D

Some principle components for using the Nektar++ framework:

- Geometry and expansions definition (SpatialDomains)

```
SpatialDomains::MeshGraph2D graph2D;
graph2D.ReadGeometry("example.xml");
graph2D.ReadExpansions("example.xml");
```

- Boundary conditions (SpatialDomains)

```
SpatialDomains::BoundaryConditions bcs(&graph2D);
bcs.Read("example.xml");
```

- Continuous Galerkin solution field (MultiRegions)

```
MultiRegions::ContField2DSharedPtr Exp;
Exp = MemoryManager<MultiRegions::ContField2D>::
              AllocateSharedPtr(graph2D,bcs);
```

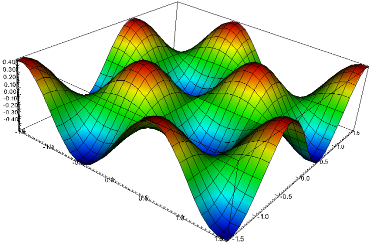
OR Discontinuous Galerkin solution field (MultiRegions)

```
MultiRegions::DisContField2DSharedPtr Exp;
Exp = MemoryManager<MultiRegions::DisContField2D>::
              AllocateSharedPtr(graph2D,bcs);
```

Using Nektar++: Helmholtz2D Demo

Solve Helmholtz problem

$$\nabla^2 u - \lambda u = f.$$



- Forcing function:

$$f = -(\lambda + 2\pi^2)\sin(\pi x)\sin(\pi y)$$

- Analytic solution:

$$u = \sin(\pi x)\sin(\pi y)$$

Specifying forcing and analytic solution:

```
<FORCING>
  <F VAR="u" VALUE="-(Lambda + 2*PI*PI)*sin(PI*x)*sin(PI*y)" />
</FORCING>

<EXACTSOLUTION>
  <F VAR="u" VALUE="sin(PI*x)*sin(PI*y)" />
</EXACTSOLUTION>
```

Imperial College London

Using Nektar++: Boundary Conditions

- Use composite to select boundary mesh elements

```
<C ID="1"> E[0,31,3,32,38,39,11,12,22]</C>
```

- Define boundary regions using these composites

```
<BOUNDARYREGIONS>
  <B ID="0"> C[1] </B>
</BOUNDARYREGIONS>
```

- Define type and value for each boundary region

```
<BOUNDARYCONDITIONS>
  <REGION REF="0">
    <D VAR="u" VALUE="sin(PI*x)*sin(PI*y)" />
  </REGION>
</BOUNDARYCONDITIONS>
```

- Types: Dirichlet(`D`), Neumann(`N`)

Using Nektar++: Helmholtz2D Demo

- Read in forcing function

```
Array<OneD,NekDouble> fce = Array<OneD,NekDouble>(nq);
SpatialDomains::ConstForcingFunctionShPtr ffunc
          = bcs.GetForcingFunction(bcs.GetVariable(0));
```

- ...evaluate it at each quadrature point...

```
for(i = 0; i < nq; ++i)
{
    fce[i] = ffunc->Evaluate(x[i],y[i],z[i]);
}
```

- ...and create a corresponding solution field on the same mesh

```
MultiRegions::ContField2DSharedPtr Fce;
Fce = MemoryManager<MultiRegions::ContField2D>::
              AllocateSharedPtr(*Exp);
Fce->SetPhys(fce);
```

Using Nektar++: Helmholtz2D Demo

- Finally, we solve for $\hat{u}$

```
Exp->HelmSolve( Fce->GetPhys(),
                Exp->UpdateContCoeffs(),
                lambda,
                true);
```

- ... and transform back into physical space u

```
Exp->BwdTrans(  Exp->GetContCoeffs(),
                Exp->UpdatePhys(),
                true);
```

Imperial College London

Using Nektar++: Helmholtz2D Demo

Run `Helmholtz2D`:

```
./Helmholtz2D Test_HEL.xml
```

…and visualise output.

Using Nektar++: Advection-Diffusion-Reaction Solver

`ADRSolver` provides a suite of solvers for a range of steady and unsteady problems.

- Location: `Nektar++/solvers/ADRSolver`
- Some solvers require additional session information,

```
<SOLVERINFO>
 <I PROPERTY="EQTYPE" VALUE="UnsteadyAdvectionDiffusion" />
 <I PROPERTY="Projection" VALUE="Continuous"/>

 <I PROPERTY="DiffusionAdvancement" VALUE="Implicit"/>

 <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder2"/>
</SOLVERINFO>
```

- For unsteady problems, specify time integration parameters,

```
<P> TimeStep      = 0.00001     </P>
<P> NumSteps      = 2000        </P>

<P> IO_CheckSteps = 2000        </P>
<P> IO_InfoSteps  = 2000        </P>
```

Imperial College London

Using Nektar++: Advection-Diffusion-Reaction Solver

Example: Diffusion equation

Run `ADRSolver`:

```
./ADRSolver ImplicitDiffusion.xml
```

Using Nektar++: Incompressible Navier-Stokes Solver

- Location: `Nektar++/solvers/IncNavierStokesSolver`
- Support for high-order pressure and time-dependent boundary conditions

```
<BOUNDARYCONDITIONS>
  <REGION REF="0">
    <D VAR="u" USERDEFINEDTYPE="TimeDependent"
           VALUE="-cos(x)*sin(y)*exp(-2*t*Kinvis)" />
    <D VAR="v" USERDEFINEDTYPE="TimeDependent"
           VALUE="sin(x)*cos(y)*exp(-2*t*Kinvis)" />
    <N VAR="p" USERDEFINEDTYPE="H" VALUE="0"/>
  </REGION>
  <REGION REF="1">
    <D VAR="u" USERDEFINEDTYPE="TimeDependent"
           VALUE="-cos(x)*sin(y)*exp(-2*t*Kinvis)" />
    <D VAR="v" USERDEFINEDTYPE="TimeDependent"
           VALUE="sin(x)*cos(y)*exp(-2*t*Kinvis)" />
    <D VAR="p" USERDEFINEDTYPE="TimeDependent"
           VALUE=" -0.25*(cos(2*x)+cos(2*y))*exp(-4*t*Kinvis)"/>
  </REGION>
</BOUNDARYCONDITIONS>
```

Further Information

- **Nektar++ website**: `www.nektar.info`
  - Example usage
  - Educational Material
  - Code documentation (Doxygen)
- **Code Examples**: `Nektar++/library/Demos/`
- **Book**:

  G.E. Karniadakis and S.J. Sherwin, *Spectral/hp element methods for computational fluid dynamics (2nd ed.)*, Oxford University Press (2005).
- **Papers**:
  - *A Generic Framework for Time-Stepping PDEs: general linear methods, object-orientated implementation and application to fluid problems*, Peter E.J. Vos, Sehun Chun, Alessandro Bolis, Claes Eskilsson, Robert M. Kirby and Spencer J. Sherwin , Int J. CFD, Submitted , 2011
  - . . . more on the website. . .

    `http://www2.imperial.ac.uk/ssherw/spectralhp/pubs/`

Imperial College London