https://gitlab.com/Project-Rat

Jeroen van Nugteren
A tri and quad-mesh generator based on distance functions.

Not focus of today.
Heavily parallelized and vectorised open source Multi-Level Fast Multipole Method (MLFMM) capable of calculating the magnetic field and vector potential from line currents.
Library for modelling 3D coil geometries using a set of simple classes. From the models the magnetic field can be calculated at any point.
MLFMM Recap: Multipoles and Localpoles

• The multipole method uses **multipoles** and **localpoles** to represent the field of the sources.

• Multipoles represent the field of **sources inside** a sphere at any **target point outside** the sphere.

• Localpoles represent the field of **sources outside** a sphere at any **target point inside** the sphere.

• In this sense they are essentially opposites.
MLFMM Recap: Reducing Complexity

- Consider a system with $N$ sources and $M$ targets.
- Each line represents a field evaluation.
- Straightforward Biot-Savart integration leads to complexity $O(NXM)$.
- By using the Multipoles and Localpoles as a middle-man, the complexity is reduced to $O(N+M)$. 

**Direct Biot Savart Method**

Sources $\rightarrow$ Targets

$O(NXM)$

**Multi-Level Fast Multipole Method (MLFMM)**

Sources $\rightarrow$ Multipole $\rightarrow$ LocalPole $\rightarrow$ Targets

$O(N+M)$
MLFMM RECAP

Information moves from red to blue

M2L
Multipole to Local

M2M
Multipole to Multipole

S2M
Source to Multipole

Sources / Targets

Sources

S2T
Source to Target

L2L
Local to Local

L2T
Local to Target

Figure Inspired by
What is New/Different in RAT-MLFMM

- It is written in C++ to avoid dependency on proprietary software (more work)
- This means that post processing is done in VTK and Paraview (OpenSource)
- In RAT all the steps of the MLFMM are vectorised using (mostly) dense matrix-matrix products
- As these are part of BLAS (CPU) and CUBLAS (GPU) they can be executed in a heavily optimized way
- The Armadillo library is used as a wrapper around BLAS as it results in code similar to MATLAB
- The sources and targets are de-coupled from the MLFMM. This allows for running almost any 1/R problem. Like stellar dynamics.
- The GPU S2T step is written using tiles making better use of shared memory.
- Rat does not have a GUI yet
Numerical Scaling of RAT-MLFMM

- There is both CPU and GPU versions available for either Direct and MLFMM method.
- We compare calculation for a random distribution of sources and targets in a unit cube.
- Direct method scales with $O(N^2)$ and MLFMM scales with $O(N)$.
- Direct algorithm is actually very efficient on GPU.
Setting up a Model

- The model including geometry and calculations is actually build as a nodeograph consisting of C++ objects
- This can later be used to build a simple interface
- For example a simple nodeograph could look like this:

  ![Nodeograph Diagram]

- We’re heavily relying on inheritance and polymorphism, the names shown here are the base class names
- Each object has its own set of properties
Modelling a Solenoid

• The most minimalistic example is a solenoid

• The code looks like this

```c
1 // create a circular path object
2 rat::mdl::ShPathCirclePr circle = rat::mdl::PathCircle::create(radius, num_sections, delem);
3
4 // create a rectangular cross section object
5 rat::mdl::ShCrossRectanglePr rectangle = rat::mdl::CrossRectangle::create(0, dcoil, 0, wcable, delem);
6
7 // create a coil object
8 rat::mdl::ShModelCoilPr coil = rat::mdl::ModelCoil::create(circle, rectangle);
9 coil->set_number_turns(num_turns);
10 coil->set_operating_current(operating_current);
```
Calculating Field

• Now lets calculate the field on the coil itself by attaching the model to a calculation object

• The code looks like this

```cpp
13 // create a source representation for the coil and set them up
14 rat::mdl::ShCalcMeshPr mesh = rat::mdl::CalcMesh::create(coil, coil);
15
16 // perform calculation and write output file
17 mesh->calculate(); mesh->write();
```

• When executed the output VTK files are written to specified location (in this case .vtu for unstructured mesh)
Post Processing - Solenoid
Modelling a “Custom” Racetrack Coil

• Lets model a custom coil, for example sake a racetrack this is done through a PathGroup and is very similar to laying lego rails

• In code it looks like this

```cpp
  // create an arc
  rat::mdl::ShPathArcPr path_arc = rat::mdl::PathArc::create(radius, arma::datum::pi/2, delem, dcoil);

  // create straight section
  rat::mdl::ShPathStraightPr path_straight = rat::mdl::PathStraight::create(ell/2, delem);

  // add sections to a pathgroup
  rat::mdl::ShPathGroupPr racetrack = rat::mdl::PathGroup::create();
  racetrack->add_path(path_straight); racetrack->add_path(path_arc);
  racetrack->add_path(path_straight); racetrack->add_path(path_straight);
  racetrack->add_path(path_straight); racetrack->add_path(path_straight);
  racetrack->add_path(path_straight); racetrack->add_path(path_straight);

  The order in which the sections are added matters
```

• Note there is actually a wrapper: PathRacetrack
Post Processing - Rectangle
Cable Level Detail

- A path can be used as a base for a CablePath as shown in this nodegraph

- In code it looks like this

```c
// create cable
rat::mdl::ShPathCablePr path_cable = rat::mdl::PathCable::create(racetrack);
path_cable->set_turn_step(dcable + 2*dinsu);
path_cable->set_num_turns(num_turns);
path_cable->set_idx_incr(idx_incr);
path_cable->set_idx_start(idx_start);
path_cable->set_num_add(num_add);
path_cable->set_offset(dinsu);
```

- Cable path settings include the start position, number of turns, the section in which the turn is incremented and more
Transformations and Fresca2 Model

- Transformations can be added to model and path objects to translate, rotate, mirror, bend, flip, reverse them.
- As a more elaborate example let me show Fresca2 nodegraph:

  ```
  PathFlared → ModelCoil → ModelGroup
  CrossRect → ModelCoil → ModelGroup
  PathFlared → ModelCoil → ModelGroup
  CrossRect → ModelCoil → ModelGroup
  PathFlared → ModelCoil → ModelGroup
  Translate

  first pole
  ModelGroup
  full magnet
  ModelGroup
  second pole
  ModelCoil
  Mirror

  CalcGrid
  CalcMesh
  CalcInductance
  ```

- For code see: examples/fresca2.cpp
Post Processing – Fresca2
Available Path Types

• Sections:
  • Arc
  • Straight
  • Bezier *(Frenet-Serret)*
  • Bspline

• Extensions:
  • Cable
  • Connect
  • Group
  • Offset
  • Profile

• Premade Shapes:
  • Circle
  • Rectangle
  • Flared
  • D-shape
  • CCT
  • CCT-custom
  • Clover
  • Equation
  • Axis

Cable drawn using septic Bezier spline and Frenet-Serret equations
Types of Calculation

• Current calculation types are:
  • **Line** – The field A/B is calculated on a provided path
  • **Grid** – The field A/B is calculated in a volume grid of points
  • **Mesh** – The field A/B is calculated on the mesh of the coil
  • **Surface** – The field is calculated on the surface of the coil
  • **Harmonics** – The coil harmonics are calculated along a provided path
  • **Inductance** – The inductance matrix and stored energy is calculated
  • **Tracks** – The field B is calculated on a mesh after which it is used for particle tracing