A new mesh library for DOLFIN

Anders Logg
logg@simula.no

Simula Research Laboratory

FEniCS'06 in Delft, November 8-9 2006

Outline

Introduction

Tutorial

Algorithms

Benchmarks

DOLFIN needs a new mesh library

- Old DOLFIN mesh implemented in 2002–2003
- Local data stored in classes `Vertex`, `Cell`, etc.
- Dimension dependent interface:

```
for (EdgeIterator e(mesh); !e.end(); ++e)
    ...

for (FaceIterator f(mesh); !f.end(); ++f)
    ...
```

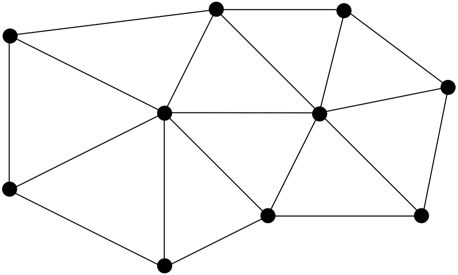
- Specialized to simplicial meshes: triangles or tetrahedra

Design goals for the new mesh library

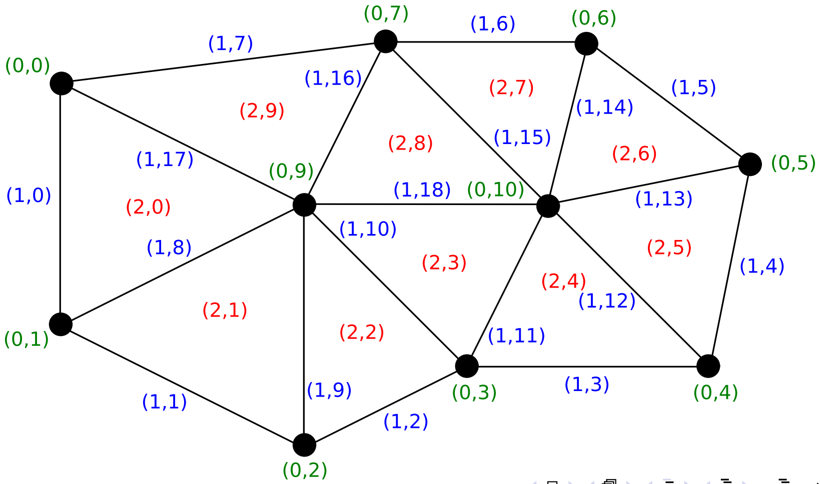
- Simple
  - No fancy data structures
  - Only `unsigned int*` and `double*`
- Intuitive
  - Choose suitable abstractions
  - Simple transition from old DOLFIN mesh
- Generic
  - Not specialized to simplicial meshes
  - Dimension-independent interface
- Fast
  - Minimize object-oriented overhead

Mesh abstractions

- Mesh = (Topology, Geometry)
- Topology = ({Mesh entities}, Connectivity)
- Mesh entity = (d, i)
- Connectivity = {Incidence relations $d - d'$}



Mesh entities



Named mesh entities

Entity	Dimension	Codimension
Vertex	0	D
Edge	1	$D-1$
Face	2	$D-2$
Facet	$D-1$	1
Cell	D	0

- Mesh entity defined by (d, i)
- Named mesh entities: Vertex, Edge, Face, Facet, Cell

Implementation

- Implemented in C++, about 4000 lines (including comments)
- Ships with DOLFIN 0.6.3

- `Mesh`
- `MeshTopology`, `MeshGeometry`
- `MeshEntity`, `MeshEntityIterator`
- `Vertex`, `Edge`, `Face`, `Facet`, `Cell`
- `MeshFunction`
- `MeshEditor`
- `UnitSquare`, `UnitCube`

Input/output

```
Mesh mesh(''mesh.xml'');
```

```
File file(''mesh.xml'');
Mesh mesh;
file >> mesh;
```

```
File file(''mesh.xml'');
Mesh mesh;
file << mesh;
```

Conversion to the new DOLFIN XML format

- Use `dolfin-convert` to convert to the DOLFIN XML format
- Conversion from Medit (tetgen) and Gmsh
- Conversion from old DOLFIN XML:

```
dolfin-convert --input old-xml old.xml new.xml
```

- Convert all meshes in current directory:

```
convertall
```

Built-in meshes

- Simple built-in meshes: `UnitSquare`, `UnitCube`
- Contributions are welcome: `UnitDisc`?, `UnitSphere`?

```
UnitSquare mesh(16, 16);
```

```
UnitCube mesh(256, 256, 256);
```

Building meshes

- Use class MeshEditor
- Specialized to simplicial meshes in 1D, 2D, 3D

```
Mesh mesh;
MeshEditor editor(mesh, CellType::triangle, 2, 2);
editor.initVertices(4);
editor.initCells(2);
editor.addVertex(0, 0.0, 0.0);
editor.addVertex(1, 1.0, 0.0);
editor.addVertex(2, 1.0, 1.0);
editor.addVertex(3, 0.0, 1.0);
editor.addCell(0, 0, 1, 2);
editor.addCell(1, 0, 2, 3);
editor.close();
```

Mesh iterators

Basic iteration:

```
unsigned int D = mesh.topology().dim();
for (MeshEntityIterator c(mesh, D); !c.end(); ++c)
  for (MeshEntityIterator v(c, 0); !v.end(); ++v)
    v->foo();
```

Iteration with named iterators:

```
for (CellIterator c(mesh); !c.end(); ++c)
  for (VertexIterator v(c); !v.end(); ++v)
    v->foo();
```

Mesh functions

- A discrete function on a mesh
- Implemented by the class MeshFunction
- Different from the class Function
- Takes a value on each mesh entity of given fixed dimension
- Templated over value type:
    - Material parameters (double)
    - Markers for mesh refinement (bool)
    - Inter-mesh connectivity (unsigned int)

```
MeshFunction<bool> marked_for_refinement;
for (CellIterator c(mesh); !c.end(); ++c)
{
  if ( marked_for_refinement(*c) )
    ...
}
```

Extracting boundaries

- A BoundaryMesh is a Mesh
- Simple boundary extraction:

```
Mesh mesh;
BoundaryMesh boundary(mesh);
```

- Mappings from the boundary to the mesh:

```
MeshFunction<unsigned int> vertices,
MeshFunction<unsigned int> cells;
BoundaryMesh boundary(mesh, vertices, cells);
```

Mesh refinement

- Uniform mesh refinement implemented
- Adaptive mesh refinement / coarsening will be added again

```
Mesh mesh;
for (int i = 0; i < 3; ++i)
  mesh.refine();
```

Python interface

- Generated automatically by SWIG
- Python iterators implemented for mesh entities
- C++ arrays mapped to Numeric arrays (will be NumPy)

```
from dolfin import *

mesh = UnitSquare(16, 16)
mesh.refine()
for c in cells(mesh):
    for v in vertices(c):
        ...
```

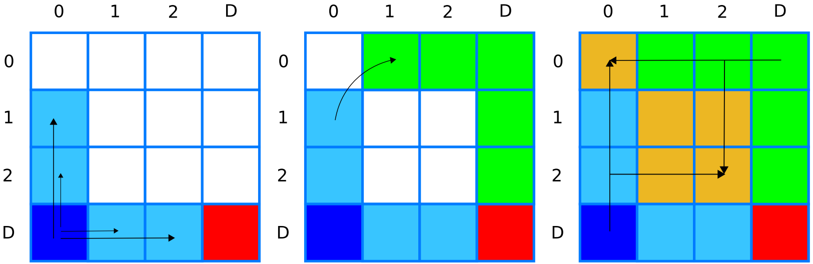
Mesh connectivity

- Mesh of topological dimension D
- Connectivity $D - 0$ given (cells – vertices)
- Need to compute connectivity $d - d'$ for $0 \leq d, d' \leq D$
- Compute only as needed

	0	1	2			D
0						
1						
2						
D	X					

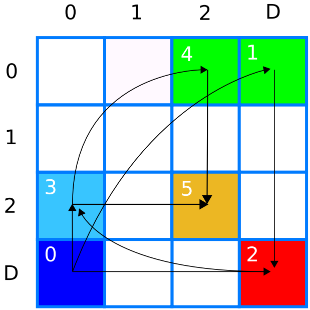
Computing mesh connectivity

- Build $D - d$ and $d - 0$ from $D - 0$ and $D - D$ for $0 < d < D$
- Compute $d - d'$ from $d' - d$ for $d < d'$ (transpose)
- Compute $d - d'$ from $d - d''$ and $d'' - d'$ (intersection)
- All algorithms are $\mathcal{O}(n^p)\mathcal{O}(N)$ for small n and p



Example: computing $2 - 2$ (faces $-$ faces)

```
for (FaceIterator f0(mesh); !f0.end(); ++f0)
  for (FaceIterator f1(f0); !f1.end(); ++f1)
    // Iterators automatically initialize 2 - 2
```

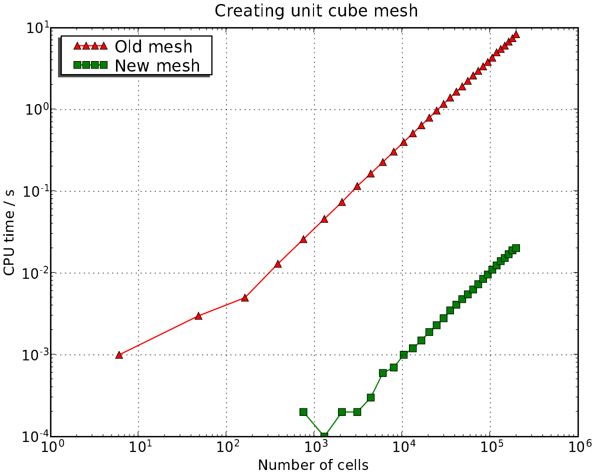


Simple benchmarks

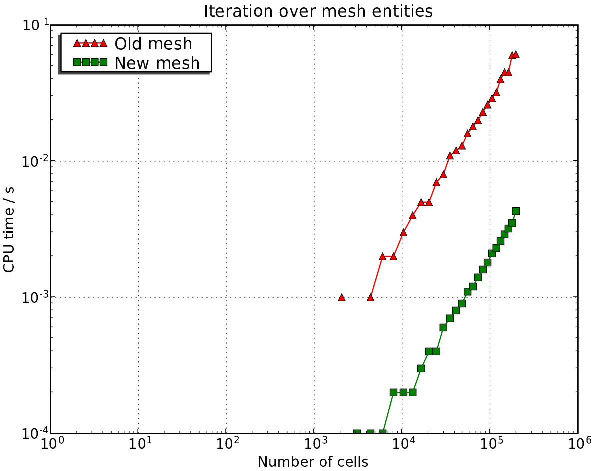
- Creating unit cube mesh
- Iteration over mesh entities
- Uniform mesh refinement
- Memory usage

- Speedup: a factor 10–100
- Reduced memory usage: a factor 10

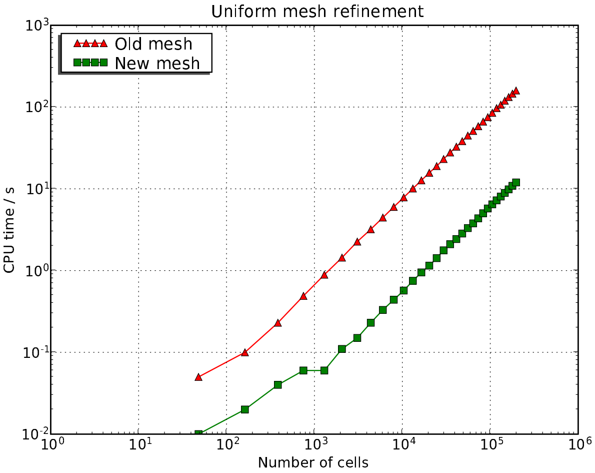
Creating unit cube mesh



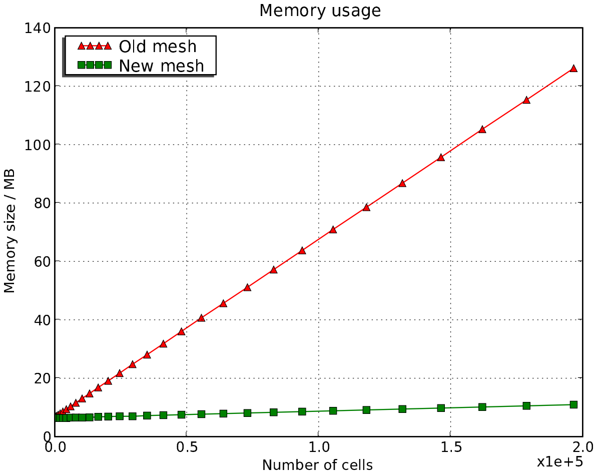
Iteration over mesh entities



Uniform mesh refinement



Memory usage



Future plans

- Adaptive mesh refinement / coarsening
- Extend functionality for ALE methods
- Extend functionality for parallel assembly
- Graphical mesh editor (Simula/Kalkulo)