ManiFEM

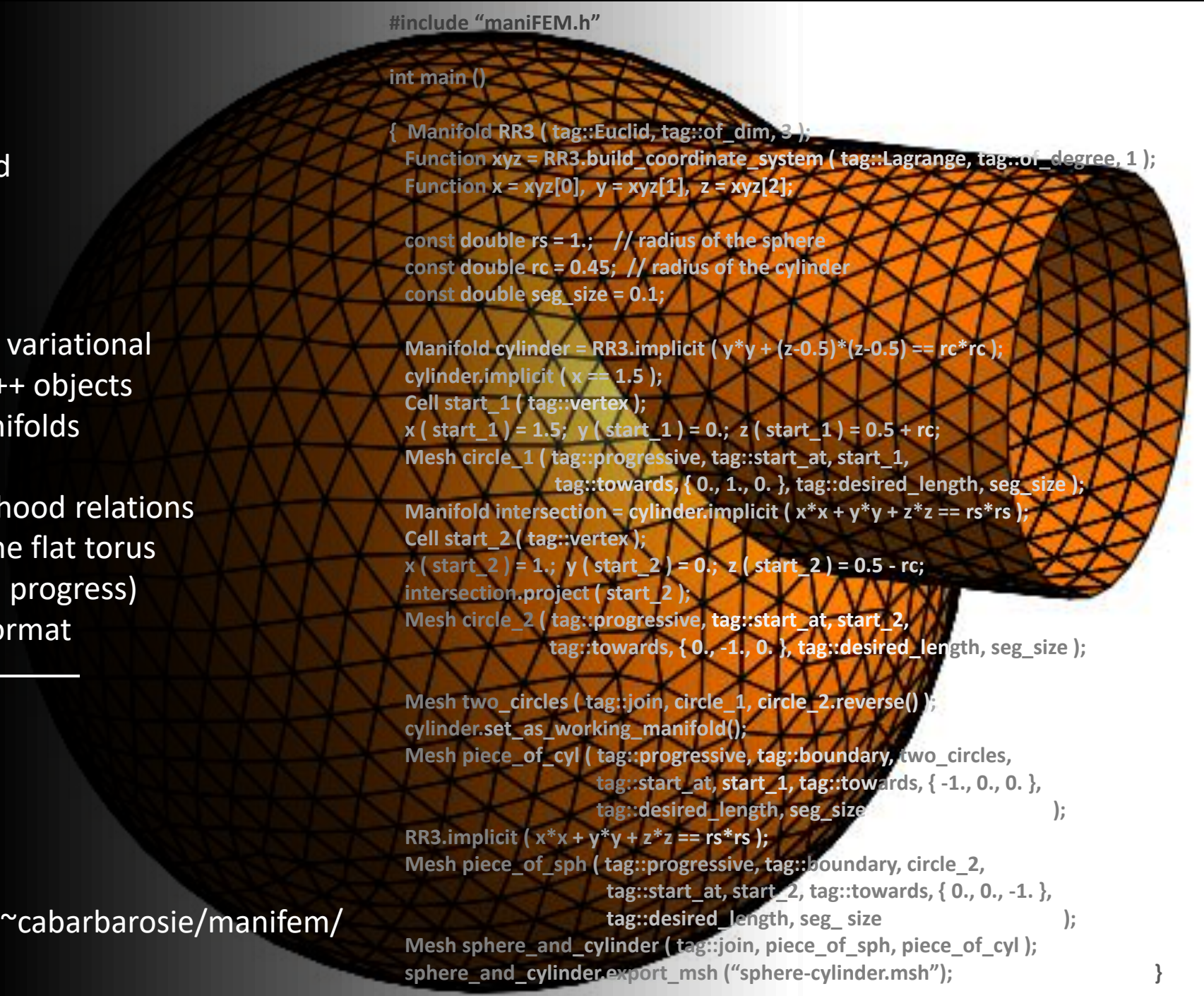
a free C++ library for meshes and finite elements on manifolds

- STL-based object-oriented library
- cells, meshes, finite elements and variational formulations implemented as C++ objects
- define curves and surfaces as manifolds
- easy navigation within the mesh, based on topological neighbourhood relations
- non-embeddable manifolds like the flat torus are being implemented (work in progress)
- meshes can be exported in msh format

---

authors : Cristian Barbarosie
Anca-Maria Toader
Sérgio Lopes

https://webpages.ciencias.ulisboa.pt/~cabarbarosie/manifem/



```
#include "maniFEM.h"

int main ()

{  Manifold RR3 ( tag::Euclid, tag::of_dim, 3 );
   Function xyz = RR3.build_coordinate_system ( tag::Lagrange, tag::of_degree, 1 );
   Function x = xyz[0],  y = xyz[1],  z = xyz[2];

   const double rs = 1.;    // radius of the sphere
   const double rc = 0.45;  // radius of the cylinder
   const double seg_size = 0.1;

   Manifold cylinder = RR3.implicit ( y*y + (z-0.5)*(z-0.5) == rc*rc );
   cylinder.implicit ( x == 1.5 );
   Cell start_1 ( tag::vertex );
   x ( start_1 ) = 1.5;  y ( start_1 ) = 0.;  z ( start_1 ) = 0.5 + rc;
   Mesh circle_1 ( tag::progressive, tag::start_at, start_1,
                   tag::towards, { 0., 1., 0. }, tag::desired_length, seg_size );
   Manifold intersection = cylinder.implicit ( x*x + y*y + z*z == rs*rs );
   Cell start_2 ( tag::vertex );
   x ( start_2 ) = 1.;  y ( start_2 ) = 0.;  z ( start_2 ) = 0.5 - rc;
   intersection.project ( start_2 );
   Mesh circle_2 ( tag::progressive, tag::start_at, start_2,
                   tag::towards, { 0., -1., 0. }, tag::desired_length, seg_size );

   Mesh two_circles ( tag::join, circle_1, circle_2.reverse() );
   cylinder.set_as_working_manifold();
   Mesh piece_of_cyl ( tag::progressive, tag::boundary, two_circles,
                       tag::start_at, start_1, tag::towards, { -1., 0., 0. },
                       tag::desired_length, seg_size                          );
   RR3.implicit ( x*x + y*y + z*z == rs*rs );
   Mesh piece_of_sph ( tag::progressive, tag::boundary, circle_2,
                       tag::start_at, start_2, tag::towards, { 0., 0., -1. },
                       tag::desired_length, seg_ size                         );
   Mesh sphere_and_cylinder ( tag::join, piece_of_sph, piece_of_cyl );
   sphere_and_cylinder.export_msh ("sphere-cylinder.msh");                    }
```