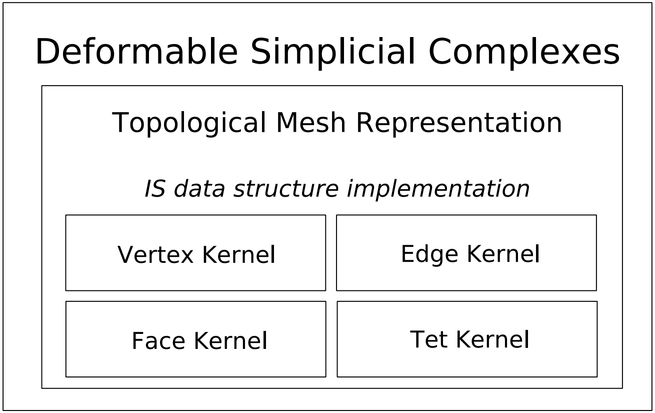
The 3D DSC Implementation: User Interface

Marek K. Misztal

Informatics and Mathematical Modelling, Technical University of Denmark
mkm@imm.dtu.dk

DSC 2011 Workshop
Kgs. Lyngby, 25th August 2011

Introduction


Deformable Simplicial Complexes
Topological Mesh Representation
IS data structure implementation
Vertex Kernel
Edge Kernel
Face Kernel
Tet Kernel

Simplices

All types of simplices: *tetrahedra*, *faces*, *edges* and *nodes* are represented in the data structure used by the 3D DSC. Simplices of the same dimension (3, 2, 1 or 0) are stored together in array-based *kernels*, supporting delayed removal and roll-back functionality.

There are two ways of addressing individual simplices:

Simplices

All types of simplices: *tetrahedra*, *faces*, *edges* and *nodes* are represented in the data structure used by the 3D DSC. Simplices of the same dimension (3, 2, 1 or 0) are stored together in array-based *kernels*, supporting delayed removal and roll-back functionality.

There are two ways of addressing individual simplices:

- *keys* – a key is simply an integer index of a simplex. Having a key, the user may obtain simplex data by calling one of the functions: `find_node`, `find_edge`, `find_face`, `find_tetrahedron` from the DSC class.

Simplices

All types of simplices: *tetrahedra*, *faces*, *edges* and *nodes* are represented in the data structure used by the 3D DSC. Simplices of the same dimension (3, 2, 1 or 0) are stored together in array-based *kernels*, supporting delayed removal and roll-back functionality.

There are two ways of addressing individual simplices:

- *keys* – a key is simply an integer index of a simplex. Having a key, the user may obtain simplex data by calling one of the functions: `find_node`, `find_edge`, `find_face`, `find_tetrahedron` from the DSC class.
- *kernel iterators* – DSC class provides the following functions: `nodes_begin`, `nodes_end`, `edges_begin`, etc. Simplex data can be obtained by simply dereferencing the iterator.

Traits

The users can decide what data they want to associate with the simplex through type traits. However, the minimum set of traits is required for the DSC algorithm to function (positions of the vertices, labels of the tetrahedra). Those default traits (from which user defined traits MUST inherit) are defined in file `traits.h`.

Our implementation of DSC also uses traits to cache such values associated with simplices as:

Traits

The users can decide what data they want to associate with the simplex through type traits. However, the minimum set of traits is required for the DSC algorithm to function (positions of the vertices, labels of the tetrahedra). Those default traits (from which user defined traits MUST inherit) are defined in file `traits.h`.

Our implementation of DSC also uses traits to cache such values associated with simplices as:

- lengths of the edges,

Traits

The users can decide what data they want to associate with the simplex through type traits. However, the minimum set of traits is required for the DSC algorithm to function (positions of the vertices, labels of the tetrahedra). Those default traits (from which user defined traits MUST inherit) are defined in file `traits.h`.

Our implementation of DSC also uses traits to cache such values associated with simplices as:

- lengths of the edges,
- areas of the faces,

Traits

The users can decide what data they want to associate with the simplex through type traits. However, the minimum set of traits is required for the DSC algorithm to function (positions of the vertices, labels of the tetrahedra). Those default traits (from which user defined traits MUST inherit) are defined in file `traits.h`.

Our implementation of DSC also uses traits to cache such values associated with simplices as:

- lengths of the edges,
- areas of the faces,
- volumes and quality values of the tetrahedra.

Vertex traits

```
namespace dsc
{
class default_node_traits{
public:
  CGLA::Vec3d v;
  default_node_traits();
  default_node_traits(double x, double y, double z);
  default_node_traits(const default_node_traits& t);

  bool is_boundary(); // true if on domain boundary
  bool is_marked(); // true if on interface
};
}
```

Edge traits

```
namespace dsc
{
class default_edge_traits{
public:
  double length;
  default_edge_traits();
  default_edge_traits(const default_edge_traits& t);

  bool is_boundary();
  bool is_marked();
};
}
```

Edge traits

```
namespace dsc{
class default_face_traits
{
  double area;
  default_face_traits();
  default_face_traits(const default_face_traits& t);

  bool is_boundary();
  bool is_marked();
};
}
```

Tetrahedron traits

```
namespace dsc{
class default_tetrahedron_traits
{
public:
  double volume;
  double quality;
  unsigned int label;

  default_tetrahedron_traits();
  default_tetrahedron_traits(const
                           default_tetrahedron_traits& t);
};
}
```

IS mesh

The IS data structure is transparent to the simplex traits. Our default instantiations takes the aforementioned simplex traits as template parameters.

```
typedef OpenTissue::is_mesh::t4mesh<default_node_traits,
                                    default_tetrahedron_traits,
                                    default_edge_traits,
                                    default_face_traits>

                                    default_mesh_type;
```

Simplex sets

The basic container for simplices in our framework is *simplex set*: a collection of four containers of `std::set` type, each of them storing, respectively, node, edge, face and tetrahedron keys.

Simplex set provides four iterator types:

```
simplex_set_type::node_set_iterator
simplex_set_type::edge_set_iterator
simplex_set_type::face_set_iterator
simplex_set_type::tetrahedron_set_iterator
```

Simplex set iterators

Simplex set provides the following iterators:

```
node_set_iterator nodes_begin();
node_set_iterator nodes_end();
edge_set_iterator edges_begin();
edge_set_iterator edges_end();
face_set_iterator faces_begin();
face_set_iterator faces_end();
tetrahedron_set_iterator tetrahedra_begin();
tetrahedron_set_iterator tetrahedra_end();
```

Simplex set operations

. . . and the following operations:

```
node_set_iterator insert(node_key_type const &);
edge_set_iterator insert(edge_key_type const &);
face_set_iterator insert(face_key_type const &);
tetrahedron_set_iterator insert(tetrahedron_key_type const &);

void add(simplex_set_type &); // union
void difference(simplex_set_type &); // difference
void intersection(simplex_set_type &); // intersection

bool contains(node_key_type const &);
bool contains(edge_key_type const &); //etc...

void erase(node_key_type const &);
void erase(edge_key_type const &); //etc...

void clear();
```

Vertex attributes

In order to facilitate vertex displacement, we need a map, which associates node keys to new vertex positions:

```
/// deformable_simplicial_complex.h
namespace dsc{
template<class T,
         class mesh_type>
class vertex_attributes
{
public:
  vertex_attributes();
  void clear();
  void associate(const typename mesh_type::node_key_type &,
                 const T &)
}; // Associative map
}
```

DSC iterators

The DSC mesh provides iterators over all four kernels:

```
/// deformable_simplicial_complex.h
namespace dsc{
template< ... > // traits, mesh type etc...
class deformable_simplicial_complex
{
  // kernel iterators
  node_iterator nodes_begin();
  node_iterator nodes_end();
  edge_iterator edges_begin();
  edge_iterator edges_end();
  face_iterator faces_begin();
  face_iterator faces_end();
  tetrahedron_iterator tetrahedra_begin();
  tetrahedron_iterator tetrahedra_end();
};
}
```

Simplex data retrieval

Simplex trait data can be also retrieved once we have a simplex key, by using `find` functions:

```
template< ... > // traits, mesh type, etc...
class deformable_simplicial_complex
{
  // getting simplex data
  node_type & find_node(const node_key_type &);
  edge_type & find_edge(const edge_key_type &);
  face_type & find_face(const face_key_type &);
  tetrahedron_type &
     find_tetrahedron(const tetrahedron_key_type &);
};
```

DSC mesh traversal

The DSC mesh traversal is facilitated by simplex sets and **star**, **closure**, **boundary** and **link** operations:

```
template< ... > // traits, mesh type, etc.
class deformable_simplicial_complex
{
  // mesh traversal
  void boundary(const simplex_key &, simplex_set &);
  void link(const simplex_key &, simplex_set &);
  void star(const simplex_key &, simplex_set &);
  void closure(const simplex_key &, simplex_set &);

  void star(const simplex_set &, simplex_set &);
  void closure(const simplex_set &, simplex_set &);
};
```

DSC instantiation

```
/// dsc_default.h
namespace dsc{
  typedef vertex_attributes<CGLA::Vec3d,
                            default_mesh_type>
                            default_vertex_attributes;

  typedef deformable_simplicial_complex<
                             default_node_traits,
                             default_edge_traits,
                             default_face_traits,
                             default_tetrahedron_traits,
                             default_mesh_type,
                             default_vertex_attributes,
                             // quality measures...>
                             dsc_default;
} // namespace dsc
```

Usage example

```
namespace dsc
{
void deformation(dsc_default * dsc,
                 double time_step)
{
  default_vertex_attributes att;

  deformation_helper(dsc, att, time_step);
  dsc->move_vertices(att);

  dsc->garbage_collect();
}
}
```

Usage example: continuation

```
void deformation_helper(dsc_default * dsc,
                        default_vertex_attributes & att,
                        double time_step)
{
  dsc_default::node_iterator nit = dsc->nodes_begin();
  dsc_default::node_iterator nn_it = dsc->nodes_end();
  while (nit != nn_it)
  {
    if (dsc->find_node(nit.key()).is_marked())
    {
      CGLA::Vec3d p = nit->v;
      CGLA::Vec3d u = ...; // Compute new vertex positions
      CGLA::Vec3d n_pos = nit->v + u*step;
      att.associate(nit.key(), n_pos);
    }
    ++nit;
  }
}
```

Exercises

- Exercises will be held in VR lab, room 047, building 305 (go downstair to the basement and cross the library).

Exercises

- Exercises will be held in VR lab, room 047, building 305 (go downstair to the basement and cross the library).
- In order to obtain ECTS credits for the PhD workshop, we expect the students to deliver a short report/log no later than the **9th September 2011**.

Exercises

- Exercises will be held in VR lab, room 047, building 305 (go downstair to the basement and cross the library).
- In order to obtain ECTS credits for the PhD workshop, we expect the students to deliver a short report/log no later than the **9th September 2011**.
- The report should contain a brief introduction (including description of the methods you have applied), code snippets (only including the code that you wrote) and screenshots.