HyENA library modifications (w.r.t. hyena-core version 2013-11-04)

referenceelements.hpp

```
//#define or ||
#include <iso646.h>
```

"or" is not a standard C++ keyword for "||", iso646.h defines such operators

```
static const unsigned int edge_node_ids[num_edges][num_edge_nodes+1];
```

Arrays of length 0 are not C++ standard, while gcc allows this, MSVC does not, other modifications due to this issue are tagged "ZLA".

referenceelements.tpl

All modifications due to 0-length arrays.

```
template<> const unsigned int ReferenceElement<LINE, CONSTANT>::
corner_node_ids[]={-1};
template<> const unsigned int ReferenceElement<LINE, LINEAR>::
inner_node_ids[] = {-1};
template<> const unsigned int ReferenceElement<TRIANGLE, CONSTANT>::
corner_node_ids[]={-1};
template<> const unsigned int ReferenceElement<TRIANGLE, CONSTANT>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{-1},{-1},{-1}};
template<> const unsigned int ReferenceElement<TRIANGLE, LINEAR>::
inner_node_ids[] = {-1};
template<> const unsigned int ReferenceElement<TRIANGLE, LINEAR>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{-1},{-1},{-1}};
template<> const unsigned int ReferenceElement<TRIANGLE, QUADRATIC>::
inner_node_ids[] = {-1};
template<> const unsigned int ReferenceElement<TRIANGLE, QUADRATIC>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{3},{4},{5}};
template<> const unsigned int ReferenceElement<QUADRANGLE, CONSTANT>::
corner_node_ids[]={-1};
template<> const unsigned int ReferenceElement<QUADRANGLE, CONSTANT>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{-1},{-1},{-1},{-1}};
template<> const unsigned int ReferenceElement<QUADRANGLE, LINEAR>::
inner_node_ids[] = {-1};
template<> const unsigned int ReferenceElement<QUADRANGLE, LINEAR>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{-1},{-1},{-1},{-1}};
template<> const unsigned int ReferenceElement<QUADRANGLE, QUADRATIC>::
edge_node_ids[num_edges][num_edge_nodes+1] = {{4},{5},{6},{7}};
```

collointegrator3d.tpl

Removed a restriction on the vertex-degrees (mostly for testing):

```
// In theory there could be any number of supporting elements, but
// everything more than 20 probably does not make sense. In that
// case an error message is thrown.
//if( num_supp < 20 ){
  if( true ){ // for suspension based meshes, vertex-degree can be high
```

colloduffy.hpp

```
const unsigned int ApproxTraits<TRIANGLE,CONSTANT>::edge_collo_pos[1]={-1};
const unsigned int ApproxTraits<TRIANGLE,LINEAR>::inner_collo_pos[1]={-1};
const unsigned int ApproxTraits<TRIANGLE,LINEAR>::edge_collo_pos[1]={-1};
const unsigned int ApproxTraits<TRIANGLE,QUADRATIC>::inner_collo_pos[1]={-1};
const unsigned int ApproxTraits<QUADRANGLE,CONSTANT>::edge_collo_pos[1]={-1};
const unsigned int ApproxTraits<QUADRANGLE,LINEAR>::inner_collo_pos[1]={-1};
const unsigned int ApproxTraits<QUADRANGLE,LINEAR>::edge_collo_pos[1]={-1};
const unsigned int ApproxTraits<QUADRANGLE,QUADRATIC>::inner_collo_pos[1]={-1};
```

(ZLA)

storageadapter.hpp

To allow incremental matrix assembly as elements are added to the mesh, we introduce offset, crop and clip values to the storage adapter, to make sure only the desired entries of the matrix are written to the correct positions. Line numbers of each modification are given in brackets.

```
int x_store_offset_, y_store_offset_, x_crop_, y_crop_, x_clip_, y_clip_;
//offset position in storage matrix (54)

StorageAdapterLHS(MATRIX& mat, int x_store_offset=0, int y_store_offset=0,
int x_crop=0, int y_crop=0,int x_clip=-1, int y_clip=-1) //(58)
    {
        x_store_offset_=x_store_offset; y_store_offset_=y_store_offset;
        x_crop_=x_crop; y_crop_=y_crop;
        x_clip_=(x_clip<0)?mat_.rows():x_clip;
        y_clip_=(y_clip<0)?mat_.cols():y_clip;
    } //(61-66)
int i_off=i+x_store_offset_, j_off=j+y_store_offset_; //(77)
if(i_off<x_clip_&& _off<y_clip_&&i_off>=x_crop_&&j_off>=y_crop_) //(78)
mat_(i_off,j_off) += entry; //(83)
```

enumerators.H

For handling fractures, we introduce additional constants `CRACK` and `CUNKNOWN`, for `BC_TYPE` and `LDOF_TYPE` respectively (lines 67 and 79), and according ostream conversions:

```
case CRACK:         out << "CRACK";    break; //(325)
case CUNKNOWN:       out << "CUNKNOWN";  break; //(340)
```

numbertraits.H

For MSVC compatibility changed `std::fabs` to `std::abs` on lines 85 and 128.

mesh.tpl

Added element IDs to error message on line 361:

```
std::string id1 = boost::lexical_cast<std::string>(dummy[m-2]->getId());
std::string id2 = boost::lexical_cast<std::string>(dummy[m-1]->getId());
HYENA_ERROR_MSG("Wrong orientation of elements "+id1+" "+id2);
```

mergedata.hpp

In our applications, mesh elements and nodes are added as fractures propagate, this causes the node-numbering to be in no particular order. To make sure all results get stored in the correct position, we use a modified version of `operator()` – note that this will probably only work for linear triangle elements.

```
template<typename LDOF_ARRAY,
         typename IN_DATA,
         typename OUT_DATA>
void operator()(const LDOF_ARRAY& ldofs,
                const IN_DATA& in_data,
                     OUT_DATA& out_data,
                int idx_offset=0, int node_base_idx=0) const
{
  for(unsigned int cnt=0;cnt<ldofs.size();++cnt) {
    if(idx_offset==0){ // default case, use original version
      out_data[ldofs[cnt]->getID()] = in_data[cnt];
    }else{
// using idx_offset for cu_ldofs
//(assume these correspond to nodes in the mesh, but are unordered)
      unsigned int node = ldofs[cnt]->getGDof()->getSuperElement(0)
        ->getElement()->getNode( ldofs[cnt]->getGDof()
        ->getReferenceElementIdx(0) )->getInputId();
      unsigned int target = (node-node_base_idx)*3+ldofs[cnt]->getLIDX();
      out_data[target] = in_data[ldofs[cnt]->getIDX()+idx_offset];
    }
  }
}
```

Usage is as follows:

```
//merging du_ldofs using default version
merge_data(du_ldofs, u, displacements);
//merging cu_ldofs using offset and node index
merge_data(cu_ldofs, u, displacements, du_ldofs.size(), NODE_BASE_INDEX);
```

galerkinassembler.hpp

As of version 2010, MSVC only supports integral types in OpenMP par-for loops. This workaround is probably not very efficient. Mods around line 280:

```
#ifdef _MSC_VER && ASSEMBLE_GALERKIN_OMP
#pragma omp parallel for schedule(guided)
   for (long it_p=0; it_p<x_se.size(); ++it_p) {
      typename x_se_array_type::const_iterator iter_x=x_se.begin()+it_p;
#else
   // start loop over row support
   typename x_se_array_type::const_iterator iter_x;
   // possibly parallel x-loop
#ifdef ASSEMBLE_GALERKIN_OMP
#pragma omp parallel for schedule(guided)
#endif
   for (iter_x=x_se.begin();iter_x<x_se.end(); ++iter_x) {
#endif
```

And around line 330:

```
#ifdef _MSC_VER && ASSEMBLE_GALERKIN_OMP
#pragma omp parallel for schedule(runtime)
   for (long it_p=0; it_p<se.size(); ++it_p) {
      typename x_se_array_type::const_iterator iter_x=se.begin()+it_p;
#else
   typename x_se_array_type::const_iterator iter_x;
#ifdef ASSEMBLE_GALERKIN_OMP
#pragma omp parallel for schedule(runtime)
#endif
   for (iter_x=se.begin(); iter_x<se.end(); ++iter_x) {
#endif
```

We also pass offset, crop and clip parameters from the caller to the storage adapter (around line 150).

```
template<typename MATRIX>
void operator()(const x_ldof_array_type& x_ldofs,
                const y_ldof_array_type& y_ldofs,
                MATRIX& matrix, int x_store_offset=0, int y_store_offset=0,
                int x_crop=0, int y_crop=0,int x_clip=-1, int y_clip=-1)
const{
  StorageAdapterLHS<MATRIX> store(
    matrix,x_store_offset,y_store_offset,x_crop,y_crop,x_clip,y_clip);
  assemble(x_ldofs, y_ldofs, store);
}
```

Finally, we added error checks at lines 754 and 760 respectively:

```
if( x_gdof==NULL ) HYENA_ERROR(); if (…
```

```
if( y_gdof==NULL ) HYENA_ERROR(); if (…
```

massassembler.hpp

As earlier, avoid OpenMP loops on iterators for MSVC compatibility.

```
#ifdef  MSC VER && ASSEMBLE GALERKIN OMP
#pragma omp parallel for schedule(guided)
      for (long it_p=0; it_p<x_se.size(); ++it_p) {
         typename x_se_array_type::const_iterator iter_x=x_se.begin()+it_p;
#else
      // start loop over row support
      typename x_se_array_type::const_iterator iter_x;
#ifdef ASSEMBLE_GALERKIN_OMP
#pragma omp parallel for schedule(guided)
#endif
      for (iter_x=x_se.begin(); iter_x<x_se.end(); ++iter_x) {
#endif
```