CustomShape class

Yaolu Liu

January 11, 2019

This short document explains how the CustomShape class is structured.

1 CustomShape

Inside “utilities.h”, the function pointers are declared.

Listing 1: Function pointer declaration

```
@SolidModel.h
  ShapeGradsFunc          getShapeGrads_;
  ShapeFuncsFunc          getShapeFuncs_;

@Utilities.h
// A pointer to a function that computes the spatial derivatives of
// the interpolation matrix. This is the so-called B-matrix.

typedef void        (*ShapeGradsFunc)

  ( const Matrix&      b,
    const Matrix&      g );

// Similar stuff, from the shape functions N_i,
// compute the shape matrix N

typedef void        (*ShapeFuncsFunc)

  ( const Matrix&      sfuncs,
    const Vector&      n );
```

```
@SolidModel.cpp
  // The role of this line is to return a function pointer
  // of type ShapeFuncsFunc
  getShapeFuncs_ = getShapeFuncsFunc( rank_ );


// A function that returns a pointer to a function that computes the
// B-matrix given the number of spatial dimensions.

ShapeFuncsFunc   getShapeFuncsFunc

  ( idx_t              rank )

```

```
{
  JEM_ASSERT ( rank >= 1 && rank <= 3 );


  if        ( rank == 1 )
  {
    return & get1DShapeFuncs;
  }
  else if ( rank == 2 )
  {
    return & get2DShapeFuncs;
  }
  else
  {
    return & get3DShapeFuncs;
  }
}

//-----------------------------------------------------------------------
//   get2DShapeFuncs
//-----------------------------------------------------------------------

void                      get2DShapeFuncs

  ( const Matrix&        s,
    const Vector&        n )
{
  JEM_ASSERT ( s.size(0) == 2 &&
               s.size(1) == 2 * n.size() );

  const idx_t  nodeCount = n.size ();

  s = 0.0;

  for ( idx_t inode = 0; inode < nodeCount; inode++ )
  {
    idx_t   i = 2 * inode;

    s(0,i + 0) = n[inode];
    s(1,i + 1) = n[inode];
  }
}
```

2 Shape function

```
  // The shape object is created here
  String shapeProp = joinNames ( myName_, SHAPE_PROP );
  shape_  = newInstance<CustomShape> ( shapeProp, conf, props );

  // Calculate the shape functions
  Matrix      sfuncs     = shape_->getShapeFunctions ();

  // The custom shape its child object to execute (@CustomShape.h)
  Matrix CustomShape::getShapeFunctions () const
  { return child_->getShapeFunctions (); }

  // declaration and instantiation
  Ref<IShape>          child_;                     //@CustomShape.h
  child_ = IShapeFactory::newInstance ( name, conf, props );  //@CustomShape.cpp

  //Returns a matrix containing the values of the shape functions in the
  //integration points of this shape.If the returned matrix is denoted by h, the
  //h(i,j) equals the value of the i-th shape function in the j-th integration p
  virtual Matrix jive::geom::ParametricShape::getShapeFunctions ( ) const
```

The shape function for two-dimensional element are:

$$N_i = \frac{1}{4}(1 + \xi_i\xi)(1 + \eta_i\eta) \tag{1}$$

Therefore, the shape function for each element is the same.

3 Strain calculation

```
   shape_->setGradsForIntegration ( ipWeights, coords, ie );
   shape_->getStrain ( strain, b, elemDisp, ip, ie );
```

This getStrain function belongs to the “CustomShape” class

```
//-----------------------------------------------------------------------
//  setGradsForIntegration
//-----------------------------------------------------------------------

void CustomShape::setGradsForIntegration

  ( const Vector&  ipWeights,
    const Matrix&  coords,
    const idx_t    ie )

{
// get gradients in integration points


//Fills the three-dimensional array g with the spatial derivatives of
//the shape functions in the integration points of this shape: g(i,j,k) is set
//to the derivative with respect to the i-th coordinate of the j-th shape
//function in the k-th integration point. As a bonus, the vector w is filled
//with the global integration weights: w[i] is set to the integration weight
//of the i-th integration point. The matrix c should contain the global node
//coordinates of this shape; c(i,j) should be equal to the i-th coordinate
//of the j-th node.

  // grads (i,j,k) = dN_j/dX_i(X(kth ip))
  child_->getShapeGradients ( grads_, ipWeights, coords );

  // store element number for check

  ieIntegr_ = ie;
}


//-----------------------------------------------------------------------
//  getStrain_
//-----------------------------------------------------------------------

void  CustomShape::getStrain

  ( const Vector&  strain,
    const Matrix&  b,
    const Vector&  disp,
    const idx_t    ip,
    const idx_t    ie ) const

{
    getBMatrix ( b, ip, ie );
    matmul     ( strain, b, disp );

}

//-----------------------------------------------------------------------
//  getBMatrix
//-----------------------------------------------------------------------

void CustomShape::getBMatrix

  ( const Matrix&  b,
    const idx_t    ip,
    const idx_t    ie ) const

{
  // first check whether the right setGrads has been called for this element

  JEM_ASSERT ( ie == ieIntegr_ );

  // fill the B matrix (gradsri_ argument is ignored if !sri_)

  fillBMatrix_ ( b, grads_ (ALL,ALL,ip), gradSri_ );
}
```

The fillBMatrix_ is a function pointer of type “bMatFunc_”.

```
  bMatFunc_              fillBMatrix_;

    typedef void           (*bMatFunc_)

    ( const Matrix&     b,
      const Matrix&     g,
      const Matrix&     gsri );

  fillBMatrix_ = getBMatFunc_ ( rank_, sri_ );

//-----------------------------------------------------------
// getBMatFunc_
//-----------------------------------------------------------

CustomShape::bMatFunc_ CustomShape::getBMatFunc_

 ( const idx_t rank,
   const bool  sri )

{
    if ( rank == 1 ) return & get1DBMat_;
    if ( rank == 2 ) return & get2DBMat_;
    else             return & get3DBMat_;
}

//----------------------------------------------------------------------
// get2DBMat_
//----------------------------------------------------------------------

void CustomShape::get2DBMat_

  ( const Matrix&       b,
    const Matrix&       g,
    const Matrix&       gsri )

{
  get2DShapeGrads ( b, g );
}


//------------------------------------------------------------------------
//   get2DShapeGrads
//------------------------------------------------------------------------


void              get2DShapeGrads

  ( const Matrix&   b,
    const Matrix&   g )

{
  JEM_ASSERT ( b.size(0) == 3 &&
                 g.size(0) == 2 &&
                 b.size(1) == 2 * g.size(1) );

  const idx_t  nodeCount = g.size (1);

  b = 0.0;

  for ( idx_t inode = 0; inode < nodeCount; inode++ )
  {
    idx_t  i = 2 * inode;

    b(0,i + 0) = g(0,inode);
    b(1,i + 1) = g(1,inode);

    b(2,i + 0) = g(1,inode);
    b(2,i + 1) = g(0,inode);
  }
}
```

4 Comments on function pointer

Defining a function pointer type, T, only sort of gives a structure of how does the inputs and outputs look like. Any object with the defined function pointer type T has no meaning before it is passed to the pointer of a real function, but of course, this real function should have the same input and output format as the function pointer type T.