```cpp
//============================================================================
//  Copyright (c) Kitware, Inc.
//  All rights reserved.
//  See LICENSE.txt for details.
//  This software is distributed WITHOUT ANY WARRANTY; without even
//  the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
//  PURPOSE.  See the above copyright notice for more information.
//
//  Copyright 2014 Sandia Corporation.
//  Copyright 2014 UT-Battelle, LLC.
//  Copyright 2014 Los Alamos National Security.
//
//  Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation,
//  the U.S. Government retains certain rights in this software.
//
//  Under the terms of Contract DE-AC52-06NA25396 with Los Alamos National
//  Laboratory (LANL), the U.S. Government retains certain rights in
//  this software.
//============================================================================

#ifndef vtk_m_worklet_MarchingCubes_h
#define vtk_m_worklet_MarchingCubes_h

#include <vtkm/VectorAnalysis.h>

#include <vtkm/exec/CellDerivative.h>
#include <vtkm/exec/ParametricCoordinates.h>

#include <vtkm/cont/ArrayHandle.h>
#include <vtkm/cont/ArrayHandleCompositeVector.h>
#include <vtkm/cont/ArrayHandleGroupVec.h>
#include <vtkm/cont/ArrayHandleIndex.h>
#include <vtkm/cont/ArrayHandlePermutation.h>
#include <vtkm/cont/DataSet.h>
#include <vtkm/cont/DeviceAdapter.h>
#include <vtkm/cont/DynamicArrayHandle.h>
#include <vtkm/cont/Field.h>

#include <vtkm/worklet/DispatcherMapTopology.h>
#include <vtkm/worklet/ScatterCounting.h>
#include <vtkm/worklet/WorkletMapTopology.h>

#include <vtkm/worklet/MarchingCubesDataTables.h>

namespace vtkm {
namespace worklet {

/// \brief Compute the isosurface for a uniform grid data set
template <typename FieldType, typename DeviceAdapter>
class MarchingCubes
{
public:
  typedef vtkm::cont::ArrayHandle<FieldType> WeightHandle;
  typedef vtkm::cont::ArrayHandle<vtkm::Vec<vtkm::Id,2> > IdPairHandle;

  class ClassifyCell : public vtkm::worklet::WorkletMapPointToCell
  {
  public:
    typedef void ControlSignature(
        FieldInPoint<Scalar> inNodes,
        TopologyIn topology,
        FieldOutCell<> outNumTriangles,
        WholeArrayIn<IdComponentType> numTrianglesTable);
    typedef void ExecutionSignature(_1, _3, _4);
    typedef _2 InputDomain;

    FieldType Isovalue;

    VTKM_CONT_EXPORT
    ClassifyCell(FieldType isovalue) :
      Isovalue(isovalue)
    {
    }

    template<typename InPointVecType,
             typename NumTrianglesTablePortalType>
    VTKM_EXEC_EXPORT
    void operator()(const InPointVecType &fieldIn,
                    vtkm::IdComponent &numTriangles,
                    const NumTrianglesTablePortalType &numTrianglesTable) const
    {
      vtkm::IdComponent caseNumber =
          (  (fieldIn[0] > this->Isovalue)
           | (fieldIn[1] > this->Isovalue)<<1
           | (fieldIn[2] > this->Isovalue)<<2
           | (fieldIn[3] > this->Isovalue)<<3
           | (fieldIn[4] > this->Isovalue)<<4
           | (fieldIn[5] > this->Isovalue)<<5
           | (fieldIn[6] > this->Isovalue)<<6
           | (fieldIn[7] > this->Isovalue)<<7 );
      numTriangles = numTrianglesTable.Get(caseNumber);
    }
  };

  /// \brief Compute isosurface vertices and scalars
  class IsosurfaceGenerate : public vtkm::worklet::WorkletMapPointToCell
  {
    typedef vtkm::Vec< vtkm::Id2,3 > Vec3Id2;
    typedef vtkm::Vec< vtkm::Vec<vtkm::Float32,3>, 3 > Vec3FVec3;
    typedef vtkm::Vec< vtkm::Vec<vtkm::Float64,3>, 3 > Vec3DVec3;

  public:
    struct InterpolateIdTypes : vtkm::ListTagBase< Vec3Id2 > { };
    struct Vec3FloatTypes : vtkm::ListTagBase< Vec3FVec3, Vec3DVec3> { };

    typedef typename vtkm::cont::ArrayHandle<vtkm::IdComponent>::
    ExecutionTypes<DeviceAdapter>::PortalConst IdPortalConstType;
    IdPortalConstType EdgeTable;

    typedef void ControlSignature(
        TopologyIn topology, // Cell set
        FieldInPoint<Scalar> fieldIn, // Input point field defining the contour
        FieldInPoint<Vec3> pcoordIn, // Input point coordinates
        FieldOutCell<Vec3> interpolationWeights,
        FieldOutCell<InterpolateIdTypes> interpolationIds,
        FieldOutCell<Vec3FloatTypes> vertexOut, // Vertices for output triangles
        FieldOutCell<Vec3FloatTypes> normalsOut, // Estimated normals (one per tri vertex)
        WholeArrayIn<IdComponentType> TriTable // An array portal with the triangle table
        );
    typedef void ExecutionSignature(
        CellShape, _2, _3, _4, _5, _6, _7, _8, VisitIndex, FromIndices);

    typedef vtkm::worklet::ScatterCounting ScatterType;
    VTKM_CONT_EXPORT
    ScatterType GetScatter() const
    {
      return this->Scatter;
    }

    VTKM_CONT_EXPORT
    IsosurfaceGenerate(FieldType isovalue,
                       bool generateNormals,
                       const vtkm::worklet::ScatterCounting& scatter,
                       IdPortalConstType edgeTable) : EdgeTable(edgeTable),
                                                      Isovalue(isovalue),
                                                      GenerateNormals(generateNormals),
                                                      Scatter(scatter) {  }

    template<typename CellShapeTag,
             typename FieldInType, // Vec-like, one per input point
             typename CoordType, // Vec-like (one per input point) of Vec-3
             typename WeightType,
             typename IdType,
             typename VertexOutType, // Vec-3 of Vec-3 coords (for triangle)
             typename NormalOutType, // Vec-3 of Vec-3
             typename TriTablePortalType, // Array portal
             typename IndicesVecType>
    VTKM_EXEC_EXPORT
    void operator()(
        CellShapeTag shape,
        const FieldInType &fieldIn, // Input point field defining the contour
        const CoordType &coords, // Input point coordinates
        WeightType &interpolationWeights,
        IdType &interpolationIds,
        VertexOutType &vertexOut, // Vertices for output triangles
        NormalOutType &normalsOut, // Estimated normals (one per tri vertex)
        const TriTablePortalType &triTable, // An array portal with the triangle table
        vtkm::IdComponent visitIndex,
        const IndicesVecType &indices) const
    {
      // Compute the Marching Cubes case number for this cell
      vtkm::IdComponent caseNumber =
          (  (fieldIn[0] > this->Isovalue)
           | (fieldIn[1] > this->Isovalue)<<1
           | (fieldIn[2] > this->Isovalue)<<2
           | (fieldIn[3] > this->Isovalue)<<3
           | (fieldIn[4] > this->Isovalue)<<4
           | (fieldIn[5] > this->Isovalue)<<5
           | (fieldIn[6] > this->Isovalue)<<6
           | (fieldIn[7] > this->Isovalue)<<7 );

      // Interpolate for vertex positions and associated scalar values
      const vtkm::Id triTableOffset =
          static_cast<vtkm::Id>(caseNumber*16 + visitIndex*3);
      for (vtkm::IdComponent triVertex = 0; triVertex < 3; triVertex++)
      {
        const vtkm::IdComponent edgeIndex =
            triTable.Get(triTableOffset + triVertex);
        const vtkm::IdComponent edgeVertex0 =
          this->EdgeTable.Get(2*edgeIndex + 0);
        const vtkm::IdComponent edgeVertex1 =
          this->EdgeTable.Get(2*edgeIndex + 1);
        const FieldType fieldValue0 = fieldIn[edgeVertex0];
        const FieldType fieldValue1 = fieldIn[edgeVertex1];
        const FieldType interpolant =
            (this->Isovalue - fieldValue0) / (fieldValue1 - fieldValue0);
        vertexOut[triVertex] = vtkm::Lerp(coords[edgeVertex0],
                                          coords[edgeVertex1],
                                          interpolant);

        interpolationIds[triVertex][0] = indices[edgeVertex0];
        interpolationIds[triVertex][1] = indices[edgeVertex1];
        interpolationWeights[triVertex] = interpolant;

        //conditionally do these only if we want to generate normals
        if(this->GenerateNormals)
        {
          const vtkm::Vec<vtkm::FloatDefault,3> edgePCoord0 =
              vtkm::exec::ParametricCoordinatesPoint(
                fieldIn.GetNumberOfComponents(), edgeVertex0, shape, *this);
          const vtkm::Vec<vtkm::FloatDefault,3> edgePCoord1 =
              vtkm::exec::ParametricCoordinatesPoint(
                fieldIn.GetNumberOfComponents(), edgeVertex1, shape, *this);

          const vtkm::Vec<vtkm::FloatDefault,3> interpPCoord =
              vtkm::Lerp(edgePCoord0, edgePCoord1, interpolant);

          normalsOut[triVertex] =
            vtkm::Normal(vtkm::exec::CellDerivative(
                           fieldIn, coords, interpPCoord, shape, *this));
        }

      }
    }

  private:
    const FieldType Isovalue;
    bool GenerateNormals;
    ScatterType Scatter;
  };

  class ApplyToField : public vtkm::worklet::WorkletMapField
  {
  public:
    typedef void ControlSignature(FieldIn<Scalar> interpolationLow,
                                  FieldIn<Scalar> interpolationHigh,
                                  FieldIn<Scalar> interpolationWeight,
                                  FieldOut<Scalar> interpolatedOutput);
    typedef void ExecutionSignature(_1, _2, _3, _4);
    typedef _1 InputDomain;

    VTKM_CONT_EXPORT
    ApplyToField() {}

    template <typename Field>
    VTKM_EXEC_EXPORT
    void operator()(const Field& low,
                    const Field& high,
                    const FieldType &weight,
                    Field &result) const
    {
      result = vtkm::Lerp(low, high, weight);
    }
  };


  MarchingCubes() {}

  template<typename CellSetType,typename StorageTagField,typename StorageTagVertices,typename StorageTagNormals, typename CoordinateType>
  void Run(const float &isovalue,
           const CellSetType& cellSet,
           const vtkm::cont::CoordinateSystem& coordinateSystem,
           const vtkm::cont::ArrayHandle<FieldType, StorageTagField>& field,
           vtkm::cont::ArrayHandle< vtkm::Vec<CoordinateType,3>, StorageTagVertices > vertices,
           vtkm::cont::ArrayHandle< vtkm::Vec<CoordinateType,3>, StorageTagNormals > normals)
  {
    // Set up the Marching Cubes case tables
    vtkm::cont::ArrayHandle<vtkm::IdComponent> edgeTable =
        vtkm::cont::make_ArrayHandle(vtkm::worklet::internal::edgeTable,
                                     24);
    vtkm::cont::ArrayHandle<vtkm::IdComponent> numTrianglesTable =
        vtkm::cont::make_ArrayHandle(vtkm::worklet::internal::numTrianglesTable,
                                     256);
    vtkm::cont::ArrayHandle<vtkm::IdComponent> triangleTableArray =
        vtkm::cont::make_ArrayHandle(vtkm::worklet::internal::triTable,
                                     256*16);

    vtkm::cont::ArrayHandle<vtkm::IdComponent> numOutputTrisPerCell;

    // Call the ClassifyCell functor to compute the Marching Cubes case numbers
    // for each cell, and the number of vertices to be generated
    ClassifyCell classifyCell(isovalue);

    typedef typename vtkm::worklet::DispatcherMapTopology<
                                      ClassifyCell,
                                      DeviceAdapter> ClassifyCellDispatcher;
    ClassifyCellDispatcher classifyCellDispatcher(classifyCell);

    classifyCellDispatcher.Invoke(field,
                                  cellSet,
                                  numOutputTrisPerCell,
                                  numTrianglesTable);

    vtkm::worklet::ScatterCounting scatter(numOutputTrisPerCell, DeviceAdapter());
    IsosurfaceGenerate isosurface(isovalue,
                                  true, //always generate normals.
                                  scatter,
                                  edgeTable.PrepareForInput(DeviceAdapter()));

    vtkm::worklet::DispatcherMapTopology<IsosurfaceGenerate, DeviceAdapter>
        isosurfaceDispatcher(isosurface);
    isosurfaceDispatcher.Invoke(
          cellSet,
          field,
          coordinateSystem.GetData(),
          vtkm::cont::make_ArrayHandleGroupVec<3>(this->InterpolationWeights),
          vtkm::cont::make_ArrayHandleGroupVec<3>(this->InterpolationIds),
          vtkm::cont::make_ArrayHandleGroupVec<3>(vertices),
          vtkm::cont::make_ArrayHandleGroupVec<3>(normals),
          triangleTableArray);
  }

  template<typename ArrayHandleIn, typename ArrayHandleOut>
  void MapFieldOntoIsosurface(const ArrayHandleIn& fieldIn,
                              ArrayHandleOut& fieldOut)
  {
    typedef typename vtkm::cont::ArrayHandleCompositeVectorType<IdPairHandle>
      ::type IdType;
    typedef vtkm::cont::ArrayHandlePermutation<IdType,ArrayHandleIn>
      FieldPermutationHandleType;

    FieldPermutationHandleType
      low(vtkm::cont::make_ArrayHandleCompositeVector(this->InterpolationIds,
                                                      0),fieldIn);
    FieldPermutationHandleType
      high(vtkm::cont::make_ArrayHandleCompositeVector(this->InterpolationIds,
                                                       1),fieldIn);

    ApplyToField applyToField;
    vtkm::worklet::DispatcherMapField<ApplyToField,DeviceAdapter>
      applyToFieldDispatcher(applyToField);

    applyToFieldDispatcher.Invoke(low,
                                  high,
                                  this->InterpolationWeights,
                                  fieldOut);
  }

  private:
  WeightHandle InterpolationWeights;
  IdPairHandle InterpolationIds;
};

}
} // namespace vtkm::worklet

#endif // vtk_m_worklet_MarchingCubes_h
```