Multipoint/bulk calls to CSM functions

Eugene Rose

10 December 2022

Proud history, bright future.

Requirements

- GEOINT Standardization Request 10 Sep 2020 “Multi-Point/Bulk Point Calls to CSM Functions”
- “There is a need for the CSM API to support the ability to make batch calls and pass multiple points into the CSM methods at a given time and retrieve the results for multiple points. The CSM API currently does not support such functionality. The primary driver for such updates is to improve computational efficiency and reduce processing times.”

Background

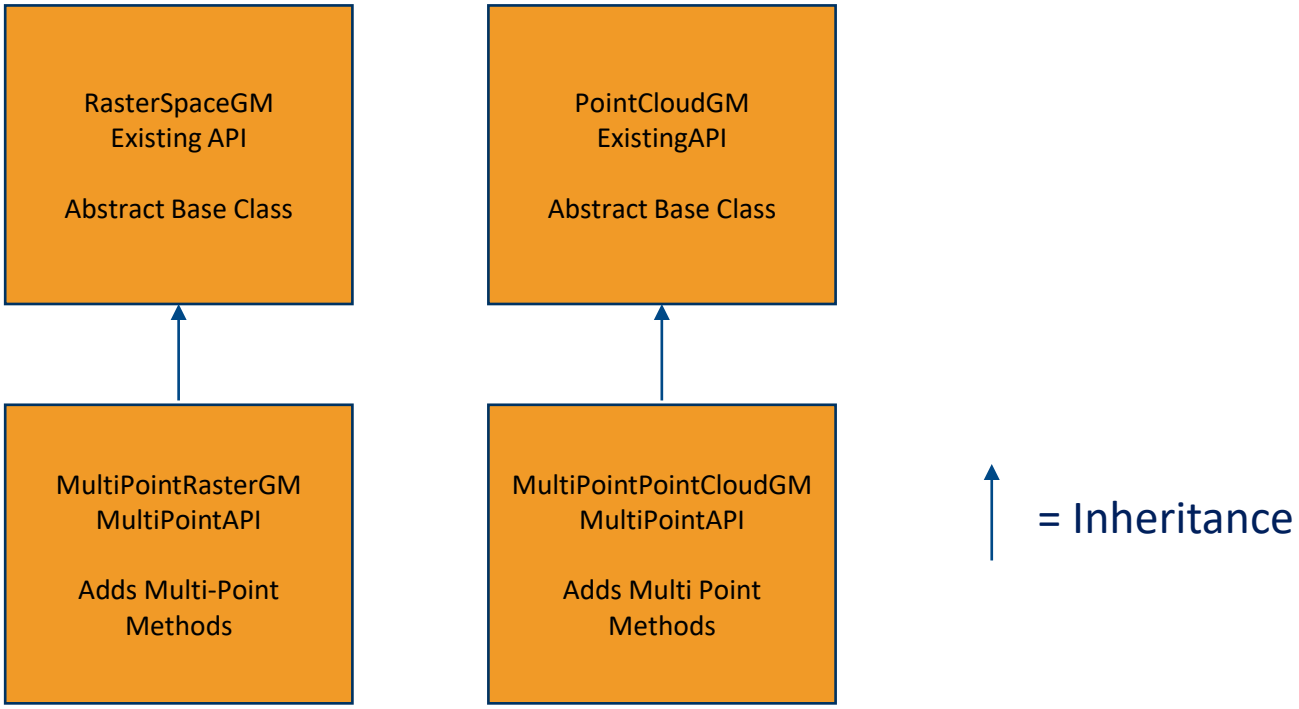
- CSM currently set up to deal with one point at a time
- Leads to performance issues
- Magnified by exploitation of new high point count data sets
- Intent is not to modify the rigor of the solution, only speed it up
- Also need to update error handling to handle multi-point/bulk.

Preliminary Design (PD) Overview

- Main Design Objectives
  - Do not impact existing models. No recompiling required.
  - Allow model developers to choose which methods to implement as Multi-Point. It should not be necessary to implement all.
  - Support both Raster and PointCloud multi-point models.
  - Encourage developers to make conscious decision on when to implement non-optimized methods in a “multi-point model”.
- Secondary Design Objectives
  - Support testing by allowing developers to add multi-point methods to any existing RasterGM model.

PD High Level Object Design

Existing GeometricGM Models



New and Modified File descriptions

New Files	Purpose
csmMultiPoint.h	New Multi-Point structs and aliases.
MultiPointRasterGM.h	New model class that derives from RasterGM and adds 19 Multi-point methods.
MultiPointRasterGM.cpp	Implementation file for MultiPointRasterGM.h
MultiPointUtils.h	Convenience class to provide standard implementations of non-optimized methods in a new “NonOptimized” namespace.
MultiPointUtils.cpp	Implementation of “NonOptimized” methods. All methods take a RasterGM reference as an argument and call that model’s single point methods in a loop.
MPRGWrapper.h	Defines a class that adds MultiPoint functionality to any RasterGM. The class holds a reference to an existing RasterGM. Useful for testing.
MPRGWrapper.cpp	Implementation of MPRGWrapper methods.

Modified Files	Reason
None	

MultiPointRasterGM Methods

multiGroundToImage – 2 versions
multiImageToGround – 2 versions
multiImageToProximateImagingLocus
multiImageToRemoteImagingLocus
multiGetIlluminationDirection
multiGetImageTime
multiGetSensorPosition – 2 versions
multiGetSensorVelocity – 2 versions
multiComputeSensorParitals – 2 versions
multiComputeAllSensorPartials – 2 versions
multiComputeGroundPartials
multiGetUnmodeledError
multiGetUnmodeledCrossCovariance

MultiPoint support data structures defined in csmMultiPoint.h

New Structs	Utility
ImageCoordWithHeight	Combines a csm::ImageCoord and a double height in a single struct
ImageCoordCovarianceWithHeight	Combines a csm::ImageCoordCovar, a double height and a double height and a double height variance

MultiPoint Aliases defined in csmMultiPoint.h

```
using ImageCoordPair                 = std::pair<ImageCoord,ImageCoord>;
using ImageEcefCoordPair             = std::pair<ImageCoord,EcefCoord>;
using MultiImageCoord                = std::vector<ImageCoord>;
using MultiImageCoordCovar           = std::vector<ImageCoordCovar>;
using MultiImageCoordWithHeight      = std::vector<ImageCoordWithHeight>;
using MultiImageCoordCovarWithHeight = std::vector<ImageCoordCovarWithHeight>;
using MultiEcefCoord                 = std::vector<EcefCoord>;
using MultiEcefCoordCovar            = std::vector<EcefCoordCovar>;
using MultiEcefLocus                 = std::vector<EcefLocus>;
using MultiEcefVector                = std::vector<EcefVector>;
using MultiImageCoordPair            = std::vector<ImageCoordPair>;
using MultiImageEcefCoordPair        = std::vector<ImageEcefCoordPair>;
using SensorPartialsVctr             = std::vector<RasterGM::SensorPartials>;
using MultiSensorPartialsVctr        = std::vector<SensorPartialsVctr>;
using ModelCoordPair                 = std::pair<ModelCoord,ModelCoord>;
using MultiModelCoordPair            = std::vector<ModelCoordPair>;
using ModelEcefCoordPair             = std::pair<ModelCoord,EcefCoord>;
using MultiModelCoord                = std::vector<ModelCoord>;
using MultiModelCoordCovar           = std::vector<ModelCoordCovar>;
using MultiModelEcefCoordPair        = std::vector<ModelEcefCoordPair>;
using MultiDbl                       = std::vector<double>;
using MultiDblVctr                   = std::vector<MultiDbl>;
```

MultiPoint Utilities (Object MultiPointUtil)

- Defines a new namespace “csm::Unoptimized”
- Exported methods take a RasterGM reference as the first argument
- MultiPointUtilities.cpp – implements the unoptimized methods by calling the passed RasterGM’s single-point methods in a loop.
- For example:

```
//*****************************************************************************
// MultiPointUnoptimized::groundToImage
//*****************************************************************************
MultiImageCoord
MultiPointUnoptimized::groundToImage(const RasterGM& model,
                                     const MultiEcefCoord& groundPts,
                                     double desiredPrecision,
                                     MultiDbl* achievedPrecisions,
                                     WarningList* warnings)
{
   MultiImageCoord coords;
   const size_t NUM_P = groundPts.size();

   if (NUM_P > 0)
   {
      const bool passPrec = achievedPrecisions &&
                            (achievedPrecisions->size() >= NUM_P);
      coords.resize(NUM_P);
      for (size_t i = 0; i < NUM_P; ++i)
      {
         coords[i] = model.groundToImage(groundPts[i],
                                         desiredPrecision,
                                         (passPrec                  ?
                                          &(*achievedPrecisions)[i] :
                                          nullptr),
                                         warnings);
      }
   }

   return coords;
}
```

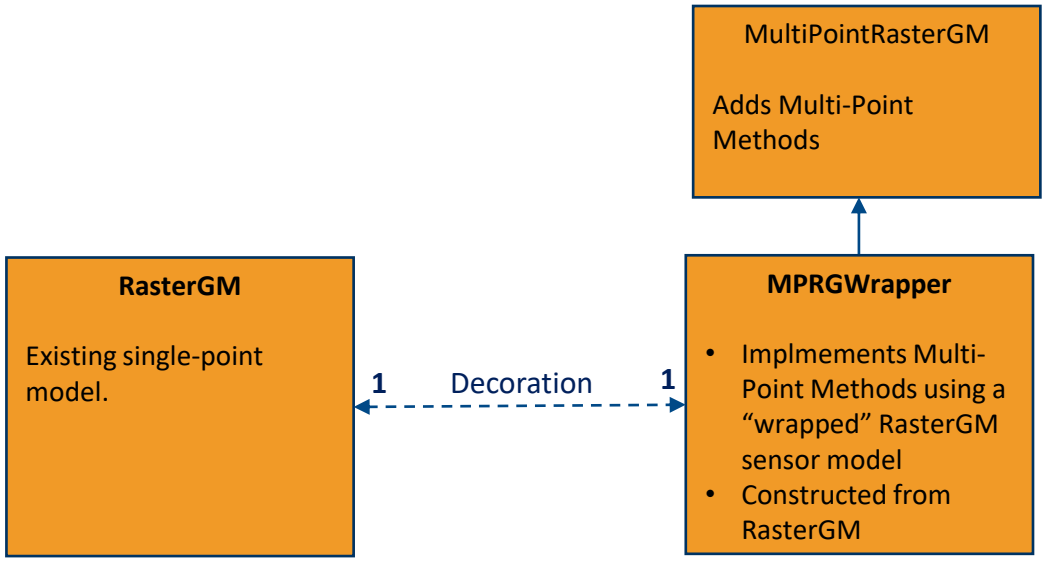
Using MultiPointUtils

```
MultiImageCoord TestMultiPointRasterGM::multiGroundToImage(
                const MultiEcefCoord& groundPts,
                double desiredPrecision ,
                MultiDbl* achievedPrecisions ,
                WarningList* warnings ) const
{
    return      MultiPointUnoptimized::groundToImage((RasterGM&)*this,
                groundPts, desiredPrecision, achievedPrecisions, warnings);
}
```

- In this example, the new sensor model “TestMultiPointRasterGM” is implementing the multiGroundToImage method using MultiPointUtils.
- Note that the developer should see this simply by looking at the way the function is called, with the “MultiPointUnoptimized” namespace explicitly called out. A conscious decision is made and it is reinformed by the calling sequence.
- In the same model, an optimized method can be implemented:

```
MultiEcefCoord TestMultiPointRasterGM::multiImageToGround(const
               MultiImageCoordWithHeight& pts,
               double desiredPrecision ,
               MultiDbl* achievedPrecisions ,
               WarningList* warnings ) const
{
// Do multithreaded implementation of imageToGround
// ….
}
```

MPRGWrapper



- Based on C++ Decorator Design Pattern
- The Wrapper class (MPRGWrapper) is used by clients (SETs) to add functionality dynamically (at runtime) to an existing RasterGM class model.
- The MPRGWrapper implementations are non-optimized, they use the methods in MultiPointUtils.
- This can be used for testing optimized versus unoptimized methods.

CSMWrapper Example

Code Snippet

```
// create an unoptimized model from an existing model
TestRasterGM testRGM;
MPRGWrapper testWrapper;
std::shared_ptr<RasterGM> sp = std::make_shared<TestRasterGM>(testRGM);
testWrapper.setRasterModel(sp);

// add timing code for unoptimized method
MultiImageCoord m2test = testWrapper.multiGroundToImage(me1);

// create the hopefully optimized version of the model
// (“TestMultiPointRasterGM is a brand new model which inherits from MultiPointRasterGM).
TestMultiPointRasterGM testMPR;

// add timing code for the optimized method
EcefCoord gp1(1000, 1000, 1000);
EcefCoord gp2(2000, 2000, 2000);
ImageCoord ic1 = testMPR.groundToImage(gp1);

// output a comparison of the performance
```