**SFM Pipeline**

**Data Types**

<http://104.236.14.11/doxygen/documentation/html/annotated.html>

*Unity<var>* - list of whatever data type var is

*Feature<var>* - info of a particular feature

Declared in ??

Usually var is SIFT\_Descriptor

*SIFT\_Descriptor* - data type to hold key points of a feature??

Declared in Feature.cu

Has attributes: floats *theta* & *sigma*, array of chars *values*

*DMatch* - Match data type but also holds euclidean distance of a Match

Derived from Match

Distance is a float and is squared

*Match* - pair of KeyPoints

Has boolean (invalid) & pair of KeyPoints

*MatchSet* - used to pass around sets of MultiMatches & KeyPoints

Has 2 pointers: Unity<MultiMatch>\* & Unity<KeyPoint>\*

Used SFM.cu line 118

*KeyPoint* -

Simple to fill out matches

Has 1 int attribute and 1 float attribute

*MultiMatch* - holds reference to KeyPoints that make up multiview match

*FeatureMatch* - DMatch but with descriptors

Derived from DMatch

Holds 2 descriptors

*Bundle* - structure in aid in indexing lines and vectors

Contains struct *Line* which holds floats line & vec

Holds num of lines, index of a single line and a bool invalid

Bool is used to remove bad indices in bundle adjustments

*BundleSet* - set of lines (point vector format) and indices (stored as bundles)

Represent bundled lines for reprojection

**Feature Extraction** (starts line 62)

SFM.cu main:

Calls a function from **SIFT\_FeatureFactory.cu** (line 62) :

ssrlcv::**SIFT\_FeatureFactory**(orientationContriWidth, descriptorContriWidth)

creates object featureFactory

just sets variables

Calls a function from **MatchFactory.cu** (line 63) :

ssrlcv::**MatchFactory**<ssrlcv::**SIFT\_Descriptor**>()

declared using template class (line 7)

creates object matchFactory

ssrlcv::Unity<ssrlcv::Feature<ssrlcv::SIFT\_Descriptor>>\* seedFeatures (line 65)

seedFeatures is set to a null pointer

seedFeatures = featureFactory.generateFeatures(seed,false,2,0.8) (line 70)

Calls a function from **SIFT\_FeatureFactory.cu** :

…**generateFeatures**(image, bool dense, maxOrientations, orientationThreshold)

Generates SIFT features for whatever image inputed

Image is converted to BW

If image is dense:

convertImageToFlt() a normalizeImage a generatePixelGradients()

then creates features: **createFeatures**() \*

else (if image is not dense):

.

.

.

Either way returns var features

Not sure what features really is in else case (created on line 106)

**\*createFeatures**(imgSize, orientThreshold, maxOrients, pixelWidth, gradients, keyPoints)

**Feature Matching** (starts line 97)

SFM.cu main:

Calls a function from **MatchFactory.cu** (line 101) :

Unity<float>\* seedDistances =

ssrlcv::MatchFactory<T>::**getSeedDistances**(Unity<Feature<T>>\* features) (line 318)

takes in allFeatures[0] from SFM.cu

Checks if features are stored in GPU using getMemoryState()

If not, moves them (?) to the GPU using setMemoryState()

Calls **getSeedMatchDistances** (line 334)

Uses a grid of blocks just instantiated on lines 328,329 (getGrid(matchDistances->size(), grid);

Returns matchDistances which is *Unity<float>\** type

Calls a function from **MatchFactory.cu** (line 102) :

DMatch\* distanceMatches =

ssrlcv::MatchFactory<T>::**generateDistanceMatches**(query, queryFeatures ,target targetFeatures, seedDistances) (line 453)

New grid created (same way as in getSeedMatchDistances)

If seedDistances is null then **matchFeaturesBruteForce** is used (line 470)

Checks to make sure size of seedDistances is same as size of queryFeatures

If seedDistance is not null, **matchFeaturesBruteForce** is also used but with

extra parameters: seedDistances->device and this->relativeThreshold

Validates the matches

This is the euclidean distance between features

Transfers distancesMatches over to cpu memory (line 106)

Find the max distance from distanceMatches, maxDist which is a float (line 109)

Calls a function from **MatchFactory.cu** (line 113) :

matches =

ssrlcv::MatchFactory<T>::**getRawMatches**(Unity<DMatch>\* matches) (line 269)

This calls **convertMatchToRaw** (line 2164 & 2171)

Checks if the globalID is less than the number of matches

if so it creates Match by indexing into matches using the globalID

A MatchSet object is instantiated (matchSet)

For 2-view case:

matchSet.keyPoints is set (KeyPoint object)

matchSet.matches is set (MultiMatch object)

Memory state of those ^ and matches is set to cpu

For n-view case:

matchSet = matchFactory.**generateMatchesExhaustive**(images,allFeatures); (line 142)

Rest of steps are same as 2-view case

Instantiates PointCloudFactory object (demPoints), BundleSet object (bundleSet), and 2 MeshFactory objects (meshBoi & finalMesh)

**Initial Triangulation** (starts line 167)

2-view case:

Calls a function from **PointCloudFactory.cu** (line 174) :

bundleSet = demPoints.

**generateBundles**(MatchSet\* matchSet, std::vector<ssrlcv::Image\*> images) (line 848)

CPU method that sets up the GPU enabled line generation

To store lines and sets of lines as bundles

The images are used only for their camera parameters

Instantiates new (Unity) list of Bundles & new(Unity) list of Lines

Cases of standard projection and use of pushbroom camera are separated

For standard projection: **generateBundle** is used

For pushbroom camera: **generatePushbroomBundle** is used

All matchSet/bundle/line info transferred to the cpu and then cleared from gpu

New BundleSet object containing bundles and lines is returned

Calls a function from **PointCloudFactory.cu** (line 176) :

points = demPoints.

**twoViewTriangulate**(BundleSet bundleSet, float\* linearError) (line 313)

Lines and bundles from bundleSet are stored in GPU

Sets up a point cloud (Unity list of float3s)

Uses **computeTwoViewTriangulate()**  (line 4492) to triangulate using skew lines

to find their closest interception

Transfers points from point cloud and linear error back to CPU

Returns a point cloud

**Filtering** (starts line 180)

**Bundle Adjustment** (starts line 211)