基于GPU的高分一号影像正射校正的设计与实现

**基于GPU的高分一号影像正射校正的设计与实现**

**一 RPC正射校正的原理**

影 像正射校正的方法有很多，主要包含两大类：一类是严格的几何纠正模型，另一类是近似几何纠正模型。当遥感影像的成像模型和有关参数已知时，可以根据严格的 成像模型来校正图像，这种方法属于严格几何纠正，最具代表的是共线方程法。当传感器成像模型未知或者无法获取相关的辅助参数时，可以用假定的数学模型模拟 成像模型，对影像实现校正，这种方法属于近似几何纠正，主要有：几何多项式纠正、有理函数法、局部区域校正等模型。本文将主要对RPC正射校正模型进行展 开讨论。

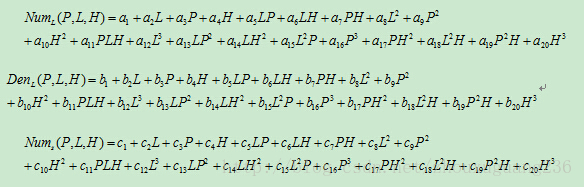
RPC模型将像点坐标d（line，sample）表示为以地面点大地坐标 D（Latitude，Longitude，Height）为自变量的比值。为了减小计算过程中舍入误差，增强参数求解的稳定性，需要把地面坐标和影像坐 标正则化到（-1，1）之间。对于一个影像，定义如下比值多项式：

  (1-1)

式中，

其中，b1和d1通常为1，（P,L，H）为正则化的地面坐标，（X，Y）为正则化的影像坐标，a1,a2,a3,...a20,b1,b2,b3,...b20,c1,c2,c3,...c20,d1,d2,d3,..d20为RPC参数。

正则化规则如下：



http://img.blog.csdn.net/20140913170643062?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvemhvdXh1Z3VhbmcyMzY=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

这 里，LAT\_OFF,LAT\_SCALE,LONG\_OFF,LONG\_SCALE,HEIGHT\_OFF,HEIGHT\_SCALE为物方坐标的正则化 参数。SAMP\_OFF,SAMP\_SCALE,LINE\_OFF,LINE\_SCALE为像方坐标的正则化参数。它们与RPC模型中4个多项式的80个 系数共同保存于卫星厂家提供给用户的RPC文件中。

在RPC模型中，可用一阶多项式来表示由光学投影引起的畸变误差模型，由二阶多项式趋近由地球曲率、投影折射、镜头倾斜等因素引起的畸变，可用三阶多项式来模拟高阶部分的其他未知畸变。

1.2 RPC模型的优化

   RPC模型的优化方法有两种：分为直接优化法和间接优化法。直接优化法是利用足够的地面控制点信息重新计算RPC系数。因此优化后的RPC系数可以直接代 入用于影像的定位和计算，而不需要对已经建立起来的模型进行改变。间接优化法是在像方或物方空间进行一些补偿换算，以减小一些系统误差。在间接优化法中， 仿射变换法是最常用的方法。

仿射变换公式如下：



式中，(x,y )是在影像上量测得到的GCP的影像坐标。

(sample,line)是利用RPC模型计算得到的GCP的影像坐标。

(e0,e1,e2,f0,f1,f2)是两组坐标之间的仿射变换参数。

利用少量的地面控制点即可解算出仿射变换的参数。这样就能利用仿射变换对由RPC模型计算得到的影像坐标解求出更为精确的行列坐标，从而达到优化RPC模型的目的。

本博客暂时先不考虑控制点和DEM，待以后完善。根据RPC系数，RPC正解变换就是从影像的行列号坐标到地面坐标点的变换，在RPC模型中，正解变换需要用到仿射变换六个参数以及RPC系数。RPC反变换是地面到影像行列号坐标的变换，只需要RPC系数就可以。

**二、RPC正射校正的CPU上实现**

RPC正射校正需要解决几个基本的问题，一个问题是第一部分提到的RPC正解和反解变换，另外还需要根据RPC系数填充多项式的系数。

1、计算各个多项式的项，即计算公式中(1-1)中的乘积项，也就是L,P,H相乘的那部分，其代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. void RPCComputeCoeffTerms( double dfLong, double dfLat, double dfHeight, double \*padfTerms )
2. {
3. padfTerms[0] = 1.0;
4. padfTerms[1] = dfLong;
5. padfTerms[2] = dfLat;
6. padfTerms[3] = dfHeight;
7. padfTerms[4] = dfLong \* dfLat;
8. padfTerms[5] = dfLong \* dfHeight;
9. padfTerms[6] = dfLat \* dfHeight;
10. padfTerms[7] = dfLong \* dfLong;
11. padfTerms[8] = dfLat \* dfLat;
12. padfTerms[9] = dfHeight \* dfHeight;
14. padfTerms[10] = dfLong \* dfLat \* dfHeight;
15. padfTerms[11] = dfLong \* dfLong \* dfLong;
16. padfTerms[12] = dfLong \* dfLat \* dfLat;
17. padfTerms[13] = dfLong \* dfHeight \* dfHeight;
18. padfTerms[14] = dfLong \* dfLong \* dfLat;
19. padfTerms[15] = dfLat \* dfLat \* dfLat;
20. padfTerms[16] = dfLat \* dfHeight \* dfHeight;
21. padfTerms[17] = dfLong \* dfLong \* dfHeight;
22. padfTerms[18] = dfLat \* dfLat \* dfHeight;
23. padfTerms[19] = dfHeight \* dfHeight \* dfHeight;
24. }

2、计算各个RPC项和L,P,H等的乘积和，代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. double RPCEvaluateSum( double \*padfTerms, double \*padfCoefs )
2. {
3. double dfSum = 0.0;
4. int i;
6. for( i = 0; i < 20; i++ )
7. dfSum += padfTerms[i] \* padfCoefs[i];
9. return dfSum;
10. }

3、正解变换，即影像到地面坐标的变换，这个函数需要用到迭代，具体代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. void RPCInverseTransform( stRPCInfo \*pRPC,double \*dbGeoTran,
2. double dfPixel, double dfLine, double dfHeight,
3. double \*pdfLong, double \*pdfLat )
4. {
5. double dfResultX, dfResultY;
7. //初始值使用放射变换系数求解
8. dfResultX = dbGeoTran[0] + dbGeoTran[1] \* dfPixel + dbGeoTran[2] \* dfLine;
10. dfResultY = dbGeoTran[3] + dbGeoTran[4] \* dfPixel + dbGeoTran[5] \* dfLine;
12. //开始用正变换的函数进行迭代
13. double dfPixelDeltaX=0.0, dfPixelDeltaY=0.0;
15. int    nIter;
16. for( nIter = 0; nIter < 20; nIter++ )
17. {
18. double dfBackPixel, dfBackLine;
20. //反算像点坐标，计算阈值
21. RPCTransform( pRPC, dfResultX, dfResultY, dfHeight,
22. &dfBackPixel, &dfBackLine );
24. dfPixelDeltaX = dfBackPixel - dfPixel;
25. dfPixelDeltaY = dfBackLine - dfLine;
27. dfResultX = dfResultX
28. - dfPixelDeltaX \* dbGeoTran[1] - dfPixelDeltaY \* dbGeoTran[2];
29. dfResultY = dfResultY
30. - dfPixelDeltaX \* dbGeoTran[4] - dfPixelDeltaY \* dbGeoTran[5];
32. if( fabs(dfPixelDeltaX) < 0.001
33. && fabs(dfPixelDeltaY) < 0.001 )
34. {
35. break;
36. }
38. }
40. //printf("迭代%d次\n",nIter);
42. \*pdfLong = dfResultX;
43. \*pdfLat = dfResultY;
44. }

4、反解变换，即地面坐标点到影像行列号坐标的变换，可由RPC系数直接求取，具体代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. void RPCTransform( stRPCInfo \*pRPC,
2. double dfLong, double dfLat, double dfHeight,
3. double \*pdfPixel, double \*pdfLine )
4. {
5. double dfResultX, dfResultY;
6. double adfTerms[20];
8. RPCComputeCoeffTerms(
9. (dfLong - pRPC->dfLONG\_OFF) / pRPC->dfLONG\_SCALE,
10. (dfLat - pRPC->dfLAT\_OFF) / pRPC->dfLAT\_SCALE,
11. (dfHeight - pRPC->dfHEIGHT\_OFF) / pRPC->dfHEIGHT\_SCALE,
12. adfTerms );
14. dfResultX = RPCEvaluateSum( adfTerms, pRPC->adfSAMP\_NUM\_COEFF )
15. / RPCEvaluateSum( adfTerms, pRPC->adfSAMP\_DEN\_COEFF );
17. dfResultY = RPCEvaluateSum( adfTerms, pRPC->adfLINE\_NUM\_COEFF )
18. / RPCEvaluateSum( adfTerms, pRPC->adfLINE\_DEN\_COEFF );
20. \*pdfPixel = dfResultX \* pRPC->dfSAMP\_SCALE + pRPC->dfSAMP\_OFF;
21. \*pdfLine = dfResultY \* pRPC->dfLINE\_SCALE + pRPC->dfLINE\_OFF;
22. }

至此，一些基本的准备做好了，那么下面就开始真正的校正了，RPC校正的大概的流程如下：

（1）根据输入未校正的影像得到输出影像的地理范围、地面分辨率大小以及仿射变换的留个参数。

（2）在输出的校正影像上，求得每一个像素对应的地理坐标，然后变换到原始影像上的行列号，根据一定的插值算法求得输出新影像上的对应的像素值。

（3）将对应的像素值写入到输出文件中。

基 本流程已经确定后，那么下面就开始讲如何对大数据进行分块，是按照输入影像分块还是输出影像分块，为了方便编程，一般采用输出影像进行分块，具体过程如 下：首先将输出影像按照行进行分块，然后计算每一个输出分块所对应的原始影像的行列号范围，再进行重采样操作，最后得到输出影像的像素值。在本博客中，每 多少行为一块分块只是本人自己的实验，实际产品和生产中需要动态检测内存等资源信息。

还有一个需要注意的是，假设影像是多波段的影像，数据怎么排列，为了效率，我创建的输出影像是按照BIP规格存储，就一个像素的所有波段的数据存储在一起，每一个分块的数据也按照BIP方式读取，写入也按照BIP格式写入。

综上所述，CPU上的正射校正可以比容易写出来，影像的IO还是一如既往的用GDAL，具体代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. bool ImageWarpRPC(const char \* pszSrcFile, const char \* pszDstFile, const char \* pszDstWKT,
2. double dResX, double dResY,  const char \* pszFormat)
3. {
4. GDALAllRegister();
5. CPLSetConfigOption("GDAL\_FILENAME\_IS\_UTF8", "NO");
7. // 打开原始图像
8. GDALDatasetH hSrcDS = GDALOpen(pszSrcFile, GA\_ReadOnly);
9. if (NULL == hSrcDS)
10. {
11. return 0;
12. }
14. GDALDataType eDataType = GDALGetRasterDataType(GDALGetRasterBand(hSrcDS, 1));
15. int nBandCount = GDALGetRasterCount(hSrcDS);
16. double dbGeonTran[6];
17. GDALGetGeoTransform(hSrcDS,dbGeonTran);
19. const char\* pszItem = GDALGetMetadataItem(hSrcDS,"INTERLEAVE","IMAGE\_STRUCTURE");
21. //获得原始图像的行数和列数
22. int nXsize = GDALGetRasterXSize(hSrcDS);
23. int nYsize = GDALGetRasterYSize(hSrcDS);
25. // 提取RPC系数
26. char\*\* pszMetadata = GDALGetMetadata(hSrcDS,"RPC");
27. GDALRPCInfo sRPCInfo;
28. if (pszMetadata != NULL)
29. {
30. GDALExtractRPCInfo( pszMetadata, &sRPCInfo );
31. }
33. stRPCInfo stInfo;
34. memcpy(&stInfo,&sRPCInfo,sizeof(stInfo));
36. // 计算输出图像四至范围、大小、仿射变换六参数等信息
37. double adfGeoTransform[6] = {0};
38. double adfExtent[4] = {0};      //minx,maxx,miny,maxy
39. int    nPixels = 0, nLines = 0;
40. double dbX[4];
41. double dbY[4];
43. //计算四个角点的地理坐标
44. RPCInverseTransform(&stInfo,dbGeonTran,0,0,100,dbX,dbY);
45. RPCInverseTransform(&stInfo,dbGeonTran,nXsize,0,100,dbX+1,dbY+1);
46. RPCInverseTransform(&stInfo,dbGeonTran,nXsize,nYsize,100,dbX+2,dbY+2);
47. RPCInverseTransform(&stInfo,dbGeonTran,0,nYsize,100,dbX+3,dbY+3);
48. adfExtent[0] = min(min(dbX[0],dbX[1]) , min(dbX[2],dbX[3]));
49. adfExtent[1] = max(max(dbX[0],dbX[1]) , max(dbX[2],dbX[3]));
50. adfExtent[2] = min(min(dbY[0],dbY[1]) , min(dbY[2],dbY[3]));
51. adfExtent[3] = max(max(dbY[0],dbY[1]) , max(dbY[2],dbY[3]));
53. int nMinCellSize = min(dbGeonTran[1],fabs(dbGeonTran[5]));
55. nPixels = ceil( fabs(adfExtent[1] - adfExtent[0]) / dbGeonTran[1] );
56. nLines = ceil ( fabs(adfExtent[3] - adfExtent[2]) / fabs(dbGeonTran[5]) );
57. adfGeoTransform[0] = adfExtent[0];
58. adfGeoTransform[3] = adfExtent[3];
59. adfGeoTransform[1] = dbGeonTran[1];
60. adfGeoTransform[5] = dbGeonTran[5];

63. // 创建输出图像
64. GDALDriverH hDriver = GDALGetDriverByName( pszFormat );
65. if (NULL == hDriver)
66. {
67. return 0;
68. }
70. char \*\*papszOptions = NULL;
71. papszOptions = CSLSetNameValue( papszOptions, "INTERLEAVE", "PIXEL" );
72. GDALDatasetH hDstDS = GDALCreate( hDriver, pszDstFile, nPixels, nLines, nBandCount, eDataType, papszOptions );
73. if (NULL == hDstDS)
74. {
75. return 0;
76. }
77. GDALSetProjection( hDstDS,  pszDstWKT);
78. GDALSetGeoTransform( hDstDS, adfGeoTransform );
80. //然后是图像重采样
81. double dfPixel = 0;
82. double dfLine = 0;
83. int nFlag = 0;
84. float dfValue = 0;
85. CPLErr err = CE\_Failure;
87. int nOffset = 0;                //内存偏移量
88. int nBandList[4] = {1,2,3,4};

91. //分块的做法
92. int nSubHeight = 2000;
93. int nBlockNum = (nLines+nSubHeight-1)/nSubHeight;       //块的数量
95. /\*double \*pLineXs = new double[nPixels\*nSubHeight];
96. double \*pLineYs = new double[nPixels\*nSubHeight];\*/
97. unsigned short \*pValues = new unsigned short[nPixels\*nBandCount\*nSubHeight];
98. unsigned short \*pSrcValues = NULL;
100. for (int nIndex = 0; nIndex < nBlockNum; nIndex ++)
101. {
102. //最后一块特殊处理
103. int nBottomIndex = (nIndex + 1) \*nSubHeight;
104. if (nBlockNum - 1 == nIndex)
105. {
106. nBottomIndex = nLines;
107. }
108. //分块的左上角地理坐标
109. double dbX1 = adfGeoTransform[0] + 0\* adfGeoTransform[1]
110. + nIndex\*nSubHeight  \* adfGeoTransform[2];
111. double dbY1 = adfGeoTransform[3] + 0 \* adfGeoTransform[4]
112. + nIndex\*nSubHeight  \* adfGeoTransform[5];
114. //分块的右下角地理坐标
115. double dbX2 = adfGeoTransform[0] + nPixels  \* adfGeoTransform[1]
116. + nBottomIndex  \* adfGeoTransform[2];
117. double dbY2 = adfGeoTransform[3] + nPixels  \* adfGeoTransform[4]
118. + nBottomIndex  \* adfGeoTransform[5];
120. //分块的右上角地理坐标
121. double dbX3 = adfGeoTransform[0] + nPixels \* adfGeoTransform[1]
122. + nIndex\*nSubHeight  \* adfGeoTransform[2];
123. double dbY3 = adfGeoTransform[3] + nPixels \* adfGeoTransform[4]
124. + nIndex\*nSubHeight  \* adfGeoTransform[5];
126. //分块的左下角地理坐标
127. double dbX4 = adfGeoTransform[0] + 0 \* adfGeoTransform[1]
128. + nBottomIndex  \* adfGeoTransform[2];
129. double dbY4 = adfGeoTransform[3] + 0 \* adfGeoTransform[4]
130. + nBottomIndex  \* adfGeoTransform[5];

133. //由输出的图像地理坐标系变换到原始的像素坐标系
135. RPCTransform(&stInfo,dbX1,dbY1,100,&dfPixel,&dfLine);
136. int nCol1 = (int)(dfPixel + 0.5);
137. int nRow1 = (int)(dfLine + 0.5);
139. RPCTransform(&stInfo,dbX2,dbY2,100,&dfPixel,&dfLine);
140. int nCol2 = (int)(dfPixel + 0.5);
141. int nRow2 = (int)(dfLine + 0.5);
143. RPCTransform(&stInfo,dbX3,dbY3,100,&dfPixel,&dfLine);
144. int nCol3 = (int)(dfPixel + 0.5);
145. int nRow3 = (int)(dfLine + 0.5);
147. RPCTransform(&stInfo,dbX4,dbY4,100,&dfPixel,&dfLine);
148. int nCol4 = (int)(dfPixel + 0.5);
149. int nRow4 = (int)(dfLine + 0.5);
151. int nMinRow = min( min( nRow1,nRow2 ), min( nRow3 , nRow4));
152. if (nMinRow < 0)
153. {
154. nMinRow = 0;
155. }
157. int nMaxRow = max( max( nRow1,nRow2 ), max( nRow3 , nRow4));
158. if (nMaxRow >= nYsize)
159. {
160. nMaxRow = nYsize-1;
161. }
163. int nHeight = nMaxRow-nMinRow+1;
165. //读取原始数据
166. pSrcValues = new unsigned short[nXsize\*nHeight\*nBandCount];
167. int nDataSize = GDALGetDataTypeSize(eDataType)/8;
168. ((GDALDataset\*)hSrcDS)->RasterIO(GF\_Read,0,nMinRow,nXsize,nHeight,
169. pSrcValues,nXsize,nHeight,eDataType,nBandCount,nBandList,nBandCount\*nDataSize,
170. nXsize\*nBandCount\*nDataSize,1\*nDataSize);
172. //这一块适合做并行
174. if (HasCUDA())
175. {
176. DWORD t1 = GetTickCount();
178. /\*RPCWarpCL(stInfo,adfGeoTransform,pSrcValues,nXsize,nHeight,nMinRow,
179. nYsize,pValues,nPixels,nSubHeight,nIndex\*nSubHeight,nBandCount,nIndex,nBlockNum);\*/
180. RPCWarpCuda(stInfo,adfGeoTransform,pSrcValues,nXsize,nHeight,nMinRow,
181. nYsize,pValues,nPixels,nSubHeight,nIndex\*nSubHeight,nBandCount,nIndex,nBlockNum);
183. DWORD t2 = GetTickCount();
185. double tt = (t2-t1);
186. printf("%f毫秒\n",tt);
187. }
189. else
190. {
191. for (int nRow = nIndex\*nSubHeight; nRow < nBottomIndex; nRow ++)
192. {
193. for (int nCol = 0; nCol < nPixels; nCol ++)
194. {
195. nOffset = ( (nRow - nIndex\*nSubHeight)\*nPixels + nCol ) \* nBandCount;
197. double dbX = adfGeoTransform[0] + nCol \* adfGeoTransform[1]
198. + nRow  \* adfGeoTransform[2];
199. double dbY = adfGeoTransform[3] + nCol \* adfGeoTransform[4]
200. + nRow  \* adfGeoTransform[5];
202. RPCTransform(&stInfo,dbX,dbY,100,&dfPixel,&dfLine);
203. int nColIndex = (int)(dfPixel + 0.5);
204. int nRowIndex = (int)(dfLine + 0.5);
206. int nOffsetSrc = ( (nRowIndex - nMinRow)\*nXsize + nColIndex ) \* nBandCount;
208. //超出范围的用0填充
209. if (nColIndex < 0 || nColIndex >= nXsize || nRowIndex < 0 || nRowIndex >= nYsize)
210. {
211. for (int nBandIndex = 0; nBandIndex < nBandCount; nBandIndex ++)
212. {
213. pValues[nOffset++] = 0;
214. }
215. }
217. else
218. {
219. for (int nBandIndex = 0; nBandIndex < nBandCount; nBandIndex ++)
220. {
221. pValues[nOffset++] = pSrcValues[nOffsetSrc++];
222. }
223. }

226. }//for
227. }//for
228. }//else
230. delete []pSrcValues;
232. //写入数据
233. if (nIndex < nBlockNum - 1)
234. {
235. ( (GDALDataset\*)hDstDS )->RasterIO(GF\_Write,0,nIndex\*nSubHeight,nPixels,nSubHeight,
236. pValues,nPixels,nSubHeight,(GDALDataType)eDataType,nBandCount,nBandList,nBandCount\*nDataSize,
237. nPixels\*nBandCount\*nDataSize,1\*nDataSize);
238. }
240. else if (nIndex == nBlockNum - 1)
241. {
242. int nRealIndex = nIndex\*nSubHeight;
243. nSubHeight = nLines - nIndex\*nSubHeight;
244. ( (GDALDataset\*)hDstDS )->RasterIO(GF\_Write,0,nRealIndex,nPixels,nSubHeight,
245. pValues,nPixels,nSubHeight,(GDALDataType)eDataType,nBandCount,nBandList,nBandCount\*nDataSize,
246. nPixels\*nBandCount\*nDataSize,1\*nDataSize);
247. }
248. }
250. /\*delete []pLineXs;
251. delete []pLineYs;\*/
252. delete []pValues;
254. GDALClose(hSrcDS );
255. GDALClose(hDstDS );
257. return 1;
258. }

上 面的代码还是有很多瑕疵的，比如计算输出影像的范围，比较保险的做法是根据原始影像的四条边来计算，还有影像的数据类型还不支持泛型，这些都将在以后不断 完善。细心的读者可能已经发现有RPCWarpCuda和RPCWrapCL的函数，不错，这正是CUDA实现和opencl上的实现，其实大家都知道影 像几何校正的时间大部分都花费在影像的重采样上，所以本博客对于正射校正的GPU实现也主要针对坐标的正反算和重采样上。下面开始GPU之旅吧！

**三、RPC正射校正的GPU实现**

自 从NVIDIA推出CUDA，GPGPU的发展速度非常迅速，以前要编写GPU上处理非图形的编程，需要将问题域转化为图形问题，而GPGPU的出现，大 大简化了GPU编程，随着技术的进一步发展，之后又出现opencl，这是跨平台的一个标准，最先由苹果公司联合几家公司推出的标准，opencl的具体 实现交给各个硬件厂商。

下面就着重介绍CUDA上的实现。CUDA上实现的函数声明如下：

extern "C" void RPCWarpCuda(stRPCInfo& stInfo,

                                                                 double\*pfGeoTransform,

                                                                 unsignedshort\* poDataIn,

                                                                 intnWidthIn,

                                                                 intnHeightIn,

                                                                 intnMinRowIn,

                                                                 intnSrcHeight,

                                                                 unsignedshort\* poDataOut,

                                                                 intnWidthOut,

                                                                 intnHeightOut,

                                                                 intnMinRowOut,

                                                                 intnBandCount,

                                                                 intnBlockIndex,

                                                                 intnBlockNum);

CUDA 的运行模型是由多个线程运行同一个计算过程，在一个计算设备上，划分为不同的线程块，每一个线程块由不同的线程组成，这样组成了一个两层的线程组织体系， 对于图像处理来说，最适合划分为二维网格，每一个线程计算一个输出像素，在RPC正射校正模型中，波段数、RPC的参数以及归一化的参数以及输出影像的仿 射变换系数在每一个线程都是一样的，所以只需拷贝这些参数到GPU上一次，这些参数在CUDA设备上需要声明为\_\_constant\_\_的，如下：

\_\_constant\_\_ double dbGeoTrans[6];

\_\_constant\_\_    double      dfLINE\_OFF;

\_\_constant\_\_    double      dfSAMP\_OFF;

\_\_constant\_\_    double      dfLAT\_OFF;

\_\_constant\_\_    double      dfLONG\_OFF;

\_\_constant\_\_    double      dfHEIGHT\_OFF;

    //缩放比例

\_\_constant\_\_    double      dfLINE\_SCALE;

\_\_constant\_\_    double      dfSAMP\_SCALE;

\_\_constant\_\_    double      dfLAT\_SCALE;

\_\_constant\_\_    double      dfLONG\_SCALE;

\_\_constant\_\_    double      dfHEIGHT\_SCALE;

    //系数

\_\_constant\_\_    double      adfLINE\_NUM\_COEFF[20];

\_\_constant\_\_    double      adfLINE\_DEN\_COEFF[20];

\_\_constant\_\_    double      adfSAMP\_NUM\_COEFF[20];

\_\_constant\_\_    double      adfSAMP\_DEN\_COEFF[20];

//波段数

\_\_constant\_\_ int nBandCount;

在RPCWarpCuda中，只在第一个块的数据传递过来的时候将这些数据拷贝的CUDA设备的常量内存上。代码如下，

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. if(0 == nBlockIndex)
2. {
3. cudaMalloc(&poDataOut\_d,nDataSizeOut);
5. //给参数常量分配空间
6. cudaMemcpyToSymbol(adfLINE\_DEN\_COEFF,stInfo.adfLINE\_DEN\_COEFF,sizeof(double)\*20);
7. cudaMemcpyToSymbol(adfLINE\_NUM\_COEFF,stInfo.adfLINE\_NUM\_COEFF,sizeof(double)\*20);
8. cudaMemcpyToSymbol(adfSAMP\_DEN\_COEFF,stInfo.adfSAMP\_DEN\_COEFF,sizeof(double)\*20);
9. cudaMemcpyToSymbol(adfSAMP\_NUM\_COEFF,stInfo.adfSAMP\_NUM\_COEFF,sizeof(double)\*20);
11. cudaMemcpyToSymbol(dfHEIGHT\_OFF,&stInfo.dfHEIGHT\_OFF,sizeof(double));
12. cudaMemcpyToSymbol(dfHEIGHT\_SCALE,&stInfo.dfHEIGHT\_SCALE,sizeof(double));
13. cudaMemcpyToSymbol(dfLAT\_OFF,&stInfo.dfLAT\_OFF,sizeof(double));
14. cudaMemcpyToSymbol(dfLAT\_SCALE,&stInfo.dfLAT\_SCALE,sizeof(double));
15. cudaMemcpyToSymbol(dfLINE\_OFF,&stInfo.dfLINE\_OFF,sizeof(double));
16. cudaMemcpyToSymbol(dfLINE\_SCALE,&stInfo.dfLINE\_SCALE,sizeof(double));
17. cudaMemcpyToSymbol(dfLONG\_OFF,&stInfo.dfLONG\_OFF,sizeof(double));
18. cudaMemcpyToSymbol(dfLONG\_SCALE,&stInfo.dfLONG\_SCALE,sizeof(double));
19. cudaMemcpyToSymbol(dfSAMP\_OFF,&stInfo.dfSAMP\_OFF,sizeof(double));
20. cudaMemcpyToSymbol(dfSAMP\_SCALE,&stInfo.dfSAMP\_SCALE,sizeof(double));
22. //传递放射变换系数
23. cudaMemcpyToSymbol(dbGeoTrans,pfGeoTransform,sizeof(double)\*6);
25. cudaMemcpyToSymbol(nBandCount,&nBands,sizeof(int));
26. }

由于输出影像是均匀分块，所以只需要第一次给输出的数据申请设备存在，这样可以减少设备内存分配的时间，在最后一次的时候将输出数据设备内存释放掉。

在 RPCWarpCuda中，只在第一个块的数据传递过来的时候将这些数据拷贝的CUDA设备的常量内存上。还有调用内很函数的时候将线程块设为32\*32 的，这刚好是1024个线程，现在很多设备都至少支持每个线程块都支持1024个线程，但是真正产品中需要检测设备的参数。一切准备工作就绪后，那么下面 就只需编写内核函数了，其实内核函数要做的工作主要是影像坐标和地面点的坐标之间的换算以及影像重采样，其核函数如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/zhouxuguang236/article/details/39253121)[在CODE上查看代码片](https://code.csdn.net/snippets/468348)

1. \_\_global\_\_ void RPCWarpKernel(
2. unsigned short\* poDataIn,
3. int nWidthIn,
4. int nHeightIn,
5. int nMinRowIn,
6. int nSrcHeight,
7. unsigned short\* poDataOut,
8. int nWidthOut,
9. int nHeightOut,
10. int nMinRowOut)
11. {
12. int idy = blockIdx.y\*blockDim.y + threadIdx.y;  //行
13. int idx = blockIdx.x\*blockDim.x + threadIdx.x;  //列
15. if (idx < nWidthOut && idy < nHeightOut)
16. {
17. //求出输出影像的实际行列号
18. int nRow = idy + nMinRowOut;
19. int nCol = idx;
21. double dbX = dbGeoTrans[0] + nCol \* dbGeoTrans[1]
22. + nRow  \* dbGeoTrans[2];
23. double dbY = dbGeoTrans[3] + nCol \* dbGeoTrans[4]
24. + nRow  \* dbGeoTrans[5];
26. double dfPixel = 0.0;
27. double dfLine = 0.0;
28. RPCTransformCUDA(dbX,dbY,100,&dfPixel,&dfLine);
30. //求出原始影像所在坐标，然后最邻近采样
31. int nColIndex = (int)(dfPixel + 0.5);
32. int nRowIndex = (int)(dfLine + 0.5);
34. int nOffsetSrc = ( (nRowIndex - nMinRowIn)\*nWidthIn + nColIndex ) \* nBandCount;
35. int nOffset = ( (nRow - nMinRowOut)\*nWidthOut + nCol ) \* nBandCount;

38. //超出范围的用0填充
39. if (nColIndex < 0 || nColIndex >= nWidthIn || nRowIndex < 0 || nRowIndex >= nSrcHeight)
40. {
41. for (int nBandIndex = 0; nBandIndex < nBandCount; nBandIndex ++)
42. {
43. poDataOut[nOffset++] = 0;
44. }
45. }
47. else
48. {
49. for (int nBandIndex = 0; nBandIndex < nBandCount; nBandIndex ++)
50. {
51. poDataOut[nOffset++] = poDataIn[nOffsetSrc++];
52. }
53. }
54. }
56. }

至此，CUDA上的实现就介绍到这里，关于OPENCL上的实现也和CUDA实现差不多，具体就不阐述了，我会上传代码，有兴趣的读者可以下载自己研究并改进。

**四、性能测试**

    我测试环境是硬件环境是GT750m，CUDA是5.5版本，opencl是CUDA上的实现。

测试数据时高分一号宽幅的数据，行数是13400，列数是12000,4个波段，数据类型是unsigned short，原始影像文件的大小是1.2GB。经过大量的测试，测试的结果如下：

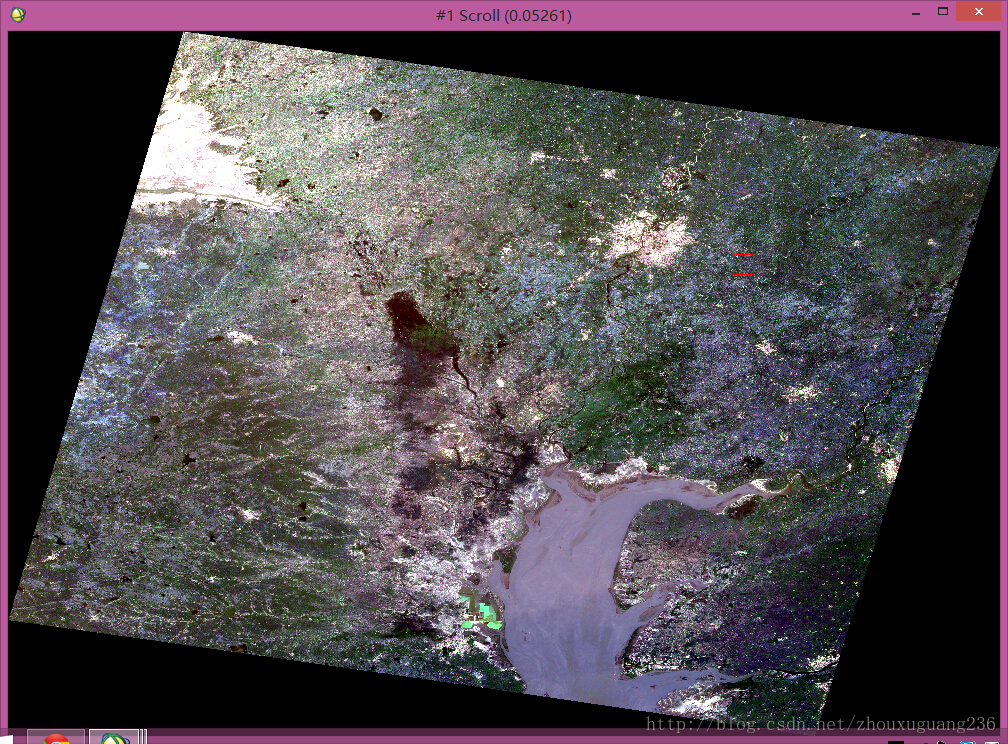
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CPU | CUDA | 加速比 | OPENCL | 加速比 |
| 包括IO时间 | 230.031s | 85.773s | 2.68 | 85.782s | 2.68 |
| 不包括IO时间 | 130.576s | 6.657s | 19.61 | 6.781s | 19.26 |

从 上面的结论可以看出，如果不统计影像IO的时间，CUDA和OPENCL都可以获得很大的加速比，加速比能够达到20左右，并且CUDA和OPENCL的 加速效果相当接近。影像IO的时间大约是80-100秒左右。如果计算影像IO的时间，加速效果大概是2.68倍左右，所以下一步该研究如何优化影像的 IO操作。

校正前后的图像比较如下：



校正前



校正后。

**五、结论与展望**

通过对RPC正射校正的的GPU实现，可以看出，GPU在遥感影像处理中可以发挥它巨大的优势，有GPU的用武之地。但是本文没对GPU优化进行进一步研究。下一步研究加入DEM以及加入控制点提高精度；对于影像IO部分，也需要深入的研究。代码下载地址是：<http://download.csdn.net/detail/zhouxuguang236/7910583>

对于相关的环境，可能需要读者自己配置了。也欢迎大家提出意见，共同进步。

来源： <<http://blog.csdn.net/zhouxuguang236/article/details/39253121#comments>>