A short description of the Spherical Harmonics Rotation Kernel

August 23, 2012

1 Spherical Harmonics

This document present very briefly how the rotation kernel is implemented in scafmm. Please refer to the papers: Parallelization of the Fast Multipole Method (Eric Lorenz), and Implementation of rotation-based operators for Fast Multipole Method in X10 (Andrew Haigh). Also the code $FRotationKernel$ and $FRotationOriginalKernel$ is well commented and refers direclty to the paper formulas.

Spherical Harmonics are centerer in 0 and have the form :

$$Y_l^m(\theta, \phi), 0 \leq l \leq \infty, -l \leq m \leq l \tag{1}$$

But for a computational reason, spherical harmonics are truncated to P term.

2 Initializing the multipoles

We create the multipole moments O_{lm} by:

$$O_{lm}(a) = \frac{a^l}{(l+|m|)!} P_{lm}(cos(\alpha)) e^{-im\beta} \tag{2}$$

Since we have a symetrie with :

$$O_{l,-m} = \bar{O_{l,m}}(-1)^m \tag{3}$$

We do not need to store all the multipole moments but only for $0 \leq m$. So in the memory our multipole vector will have a size $((P+2)*(P+1))/2$ and store the moments as:

$\omega_{0,0}$
$\omega_{1,0}\omega_{1,1}$
...
$\omega_{P,0}...\omega_{P,P}$

Multipole moments of the same order can be summed. In a SH point of view, spherical harmonics centered in the same position of a similar systems can be summed directly.

3 M2M

Translating multipole to another point is done by the formula:

$$O_{lm}(a+b) = \sum_{j=0}^{l} \sum_{k=-j}^{j} A_{jk}^{lm}(b) O_{jk}(a) \tag{4}$$

Translating multipole moments on the z axis is more easy :

$$O_{l,m}(a+b') = \sum_{j=|m|}^{l} \frac{b^{l-j}}{(l-j)!} O_{j,m}(a) \tag{5}$$

The idea is to rotate the spherical harmonics to make it pointed in direction of the parent cell. Then, we can translate it along the z axis and finally rotate it back. By doing so the target and the source will be expressed in the same system.

Such rotation has to be performed in two times : first a rotation around the z axis using the azimuth angle, then a rotation around y axis using the inclinaison angle. Azimuth and inclinaison are obtained by computing the relative difference between our current cell spherical harmonics position center and the target spherical harmonics position center. This difference is expressed in spherical coordinate such that $0 \leq Azimuth \leq 2\Pi$ and $0 \leq Inclinaison \leq \Pi$.

3.1 Rotating around z

This is a simple multiplication :

$$O_{l,m}(\alpha, \beta + \phi) = e^{-i\phi m} O_{l,m}(\alpha, \beta) \tag{6}$$

In the code, we precompute all possible rotations around z. There are 4 possibilities for the M2M and the L2L, and lets say $343 - 27$ for the M2L. Then, at the begining of M2M or M2L (not L2L since this is not multipole any more) and at the end of the M2M we multiply the multipole vector per another one term-to-term. We have $((P+2)*(P+1))/2$ complex multiplications.

```
static void RotationZVectorsMul(FComplexe* FRestrict dest, const
    FComplexe* FRestrict src, const int inSize = SizeArray){
      const FComplexe*const FRestrict lastElement = dest + inSize;
      const FComplexe*const FRestrict intermediateLastElement =
          dest + (inSize & ~0x3);
      // first the inSize - inSize%4 elements
      for(; dest != intermediateLastElement ;) {
          (*dest++) *= (*src++);
          (*dest++) *= (*src++);
          (*dest++) *= (*src++);
          (*dest++) *= (*src++);
      }
      // then the rest
      for(; dest != lastElement ;) {
          (*dest++) *= (*src++);
      }
  }
```

3.2 Rotating around y

Rotating around y is a little more tricky:

$$O_{l,m}(\alpha + \theta, \beta) = \sum_{k=-l}^{l} \sqrt{\frac{(l-k)!(l+k)!}{(l-|m|)!(l+|m|)!}} d^l_{k,m}(\theta) O_{l,k}(\alpha, \beta) \tag{7}$$

Finaly it is just sum of complex number after they have been multiply respectively per a real number. The sum goes from $-l$ to l, but multipole

for $m \leq 0$ are not stored. Let rewrite the formula with another notation to simplify :

$$D^l_{k,m} = \sqrt{\frac{(l-k)!(l+k)!}{(l-|m|)!(l+|m|)!}} d^l_{k,m}(\theta) \tag{8}$$

$$O_{l,m}(\alpha+\theta,\beta) = \sum_{k=-l}^{l} D^l_{k,m} O_{l,k}(\alpha,\beta) \tag{9}$$

But we have only $O_{l,m}$ for $0 \leq m$ so the formula becomes:

$$O_{l,m}(\alpha+\theta,\beta) = D^l_{0,m} O_{l,0}(\alpha,\beta) + \sum_{k=1}^{l} (-1)^k D^l_{-k,m} O_{l,k}(\bar{\alpha},\beta) + D^l_{k,m} O_{l,k}(\alpha,\beta) \tag{10}$$

Which can be factorized by working on the real and the imaginary part separatly :

$$Real(O_{l,m}(\alpha+\theta,\beta)) = D^l_{0,m} Real(O_{l,0}(\alpha,\beta)) + \sum_{k=1}^{l} ((-1)^k D^l_{-k,m} + D^l_{k,m}) Real(O_{l,k}(\alpha,\beta)) \tag{11}$$

$$Imag(O_{l,m}(\alpha+\theta,\beta)) = D^l_{0,m} Imag(O_{l,0}(\alpha,\beta)) + \sum_{k=1}^{l} (-(-1)^k D^l_{-k,m} + D^l_{k,m}) Imag(O_{l,k}(\alpha,\beta)) \tag{12}$$

The parts $D^l_{0,m}(\theta)$, $((-1)^k D^l_{-k,m}(\theta) + D^l_{k,m}(\theta))$ and $(-(-1)^k D^l_{-k,m}(\theta) + D^l_{k,m}(\theta))$ can be precomputed and store in a array. There are store in this order : first the coefficient needed for $m == 0$ which is used for real and imaginary. Then, the coefficients for $0 \leq m$, first for the real then for the imaginary. Finaly the code to preform this rotation becomes:

```
static void RotationYWithDlmk(FComplexe vec[], const FReal*
    dlmkCoef){
  FReal originalVec[2*SizeArray];
  FMemUtils::copyall((FComplexe*)originalVec,vec,SizeArray);
  // index_lm == atLm(l,m) but progress iteratively to write
     the result
```

```
        int index_lm = 0;
        for(int l = 0 ; l <= P ; ++l){
            const FReal*const FRestrict originalVecAtL0 = originalVec
                 + (index_lm * 2);
            for(int m = 0 ; m <= l ; ++m, ++index_lm ){
                FReal res_lkm_real = 0.0;
                FReal res_lkm_imag = 0.0;
                // To read all "m" value for current "l"
                const FReal* FRestrict iterOrignalVec =
                    originalVecAtL0;
                { // for k == 0
                    // same coef for real and imaginary
                    res_lkm_real += (*dlmkCoef) *
                        (*iterOrignalVec++);
                    res_lkm_imag += (*dlmkCoef++) *
                        (*iterOrignalVec++);
                }
                for(int k = 1 ; k <= l ; ++k){
                    // coef contains first real value
                    res_lkm_real += (*dlmkCoef++) *
                        (*iterOrignalVec++);
                    // then imaginary
                    res_lkm_imag += (*dlmkCoef++) *
                        (*iterOrignalVec++);
                }
                // save the result
                vec[index_lm].setRealImag(res_lkm_real,
                    res_lkm_imag);
            }
        }
    }
```

3.3 Rotating back

After the translation, we perform the inverse rotation. We just rotate first around y and the around z with inverse angle.

4 Other operators

The same principle apply to M2L and L2L.

4.1 M2L

At the entrance of M2L we use multipole so the rotation are the same. Then the translation is:

$$M_{l,m}(a-b') = \sum_{j=|m|}^{\infty} \frac{(j+l)!}{b^{j+l+1}} O_{j,-m}(a), \text{j bounded by P-l} \tag{13}$$

We then have local vector of the same size as multipole vector. We need to rotate back, but this time we use rotation for local/taylor expansion.

4.2 Rotating around y for taylor expansion

$$M_{l,m}(\alpha+\theta,\beta) = \sum_{k=-l}^{l} \sqrt{\frac{(l-|m|)!(l+|m|)!}{(l-k)!(l+k)!}} d^l_{km}(\theta) M_{l,k}(\alpha,\beta) \tag{14}$$

4.3 Rotating around z for taylor expansion

$$M_{l,m}(\alpha,\beta+\phi) = e^{i\phi m} M_{l,m}(\alpha,\beta) \tag{15}$$

4.4 L2L

We use the rotation for taylor expansion defined previously and the translation is:

$$M_{l,m}(a-b') = \sum_{j=l}^{\infty} \frac{b^{j-l}}{(j-l)!} M_{j,m}(a), \text{j bounded by P} \tag{16}$$

5 Optimization?

It is easy to see that there is a few possible rotations. For example, the rotation for the M2M (and L2L) are the same in any level and there are only 2 possibilities for the inclinaison ($0, 78$ and $2, 35$) and 4 possibilties for the azimuth ($\pi/4.3\pi/4$, $5\pi/4$ a $7\pi/4$).

For the M2L there are much more possiblities but it remains some relation and possible factorization.