Treecode, Fast Multipole, etc.

BY GUANGHUI HU

University of Macau

Email: garyhu@umac.mo
Web: http://ghhu.github.io

Abstract

In this note, the motivation, idea, algorithm, as well as the technical details will be introduced for the fast algorithm for calculating the gravitational interaction, Coulomb interaction, etc, among many particles.

1 Introduction

The integral we need to calculate is as follows

$$\phi(\vec{x}) = \int_{\Omega} \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y}$$

Of cousre a direct evaluation would result in $\mathcal{O}(N^2)$ complexity. To accelerate the calculation, following equavilent PDE can be considered

$$-\Lambda u(\vec{x}) = f(\vec{x}), \text{in} \Omega$$

$$u(\vec{x})|_b = \int_{\Omega} \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y}$$

With above PDE, using AMG to solve the linear equations would result in $\mathcal{O}$(N) complexity. However, the calculation of boundary value would result in $\mathcal{O}\left(N^{\frac{5}{3}}\right)$, so it is unacceptable.

The above problem appears in Hartree potential calculation in density functional theory, in which the electron cloud clustered around the nuclei position. In this case, we may

- Design a ball which can "completely" cover the electron cloud;
- Calculate the barycenter $p_{bc}$of that electron cloud;
- Calculate the Taylor expansion of the kernel $\frac{1}{|\vec{x}-\vec{y}|}$ at $\vec{p}_{bc}$.

In this case, we may obtain the following series

```
Maxima 5.43.2 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.12
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i2) f(x,y,z)
```

(%o2) $f(x,y,z)$

```
(%i3)
(%i3) kernel(x,y,z):=1/sqrt((x-u)^2 + (y-v)^2 + (z-w)^2)
```

(%o3) $\text{kernel}(x,y,z) := \frac{1}{\sqrt{(x-u)^2+(y-v)^2+(z-w)^2}}$

```
(%i4) diff(kernel(x,y,z),x,1)
```

(%o4) $-\frac{x-u}{((z-w)^2+(y-v)^2+(x-u)^2)^{\frac{3}{2}}}$

```
(%i5) diff(kernel(x,y,z),x,2)
```

(%o5) $\frac{3(x-u)^2}{((z-w)^2+(y-v)^2+(x-u)^2)^{\frac{5}{2}}} - \frac{1}{((z-w)^2+(y-v)^2+(x-u)^2)^{\frac{3}{2}}}$

```
(%i6) diff(diff(kernel(x,y,z),x,1),y,1)
```

(%o6) $\frac{3(x-u)(y-v)}{((z-w)^2+(y-v)^2+(x-u)^2)^{\frac{5}{2}}}$

```
(%i7)
(%i20)
```

```
(%i1) taylor(h(u,v,w),[u,v,w],[px,py,pz],2)
```

(%o1) $$h(\mathrm{px},\mathrm{py},\mathrm{pz}) + \left(\left(\left.\frac{d}{d\,u} h(u,\mathrm{py},\mathrm{pz})\right|_{u=\mathrm{px}}\right)(u-\mathrm{px}) + \left(\left.\frac{d}{d\,v} h(\mathrm{px},v,\mathrm{pz})\right|_{v=\mathrm{py}}\right)(v-\mathrm{py}) + \left(\left.\frac{d}{d\,w} h(\mathrm{px},\mathrm{py},w)\right|_{w=\mathrm{pz}}\right)(w-\mathrm{pz})\right) + \left(\left(\left.\frac{d^2}{d\,u^2} h(u,\mathrm{py},\mathrm{pz})\right|_{u=\mathrm{px}}\right)(u-\mathrm{px})^2 + \left(2\left(\left.\left.\frac{d^2}{d\,u\,d\,v} h(u,v,\mathrm{pz})\right|_{u=\mathrm{px}}\right|_{v=\mathrm{py}}\right)(v-\mathrm{py}) + 2\left(\left.\left.\frac{d^2}{d\,u\,d\,w} h(u,\mathrm{py},w)\right|_{u=\mathrm{px}}\right|_{w=\mathrm{pz}}\right)(w-\mathrm{pz})\right)(u-\mathrm{px}) + \left(\left.\frac{d^2}{d\,v^2} h(\mathrm{px},v,\mathrm{pz})\right|_{v=\mathrm{py}}\right)(v-\mathrm{py})^2 + 2\left(\left.\left.\frac{d^2}{d\,v\,d\,w} h(\mathrm{px},v,w)\right|_{v=\mathrm{py}}\right|_{w=\mathrm{pz}}\right)(w-\mathrm{pz})\,(v-\mathrm{py}) + \left(\left.\frac{d^2}{d\,w^2} h(\mathrm{px},\mathrm{py},w)\right|_{w=\mathrm{pz}}\right)(w-\mathrm{pz})^2\right)\Big/2 + \cdots$$

```
(%i1) diff(exp(-x^2),x,1)
```

(%o1) $-2\,x\,e^{-x^2}$

```
(%i2)
```

In summary, from above Taylor expansion, we obtain the following ($\vec{p}=\vec{p}_{bc}$ for simplicity below), and

$$\vec{p}=(p_1,p_2,p_3) \text{ and } \vec{x}=(x_1,x_2,x_3).$$

$$\phi(\vec{x})=\int_\Omega \frac{1}{|\vec{x}-\vec{y}|}\rho(\vec{y})d\vec{y}$$

$$\begin{aligned}
&=\int_\Omega \frac{1}{|\vec{x}-\vec{p}|}\rho(\vec{y})d\vec{y}\\
&\quad+\int_\Omega \frac{(x_1-p_1)}{|\vec{x}-\vec{p}|^3}(y_1-p_1)\rho(\vec{y})d\vec{y}+\int_\Omega \frac{(x_2-p_2)}{|\vec{x}-\vec{p}|^3}(y_2-p_2)\rho(\vec{y})d\vec{y}+\int_\Omega \frac{(x_3-p_3)}{|\vec{x}-\vec{p}|^3}(y_3-p_3)\rho(\vec{y})d\vec{y}\\
&\qquad+\frac{1}{2}\int_\Omega\left(\frac{3(x_1-p_1)^2}{|\vec{x}-\vec{p}|^5}-\frac{1}{|\vec{x}-\vec{p}|^3}\right)(y_1-p_1)^2\rho(\vec{y})d\vec{y}+\frac{1}{2}\int_\Omega\left(\frac{3(x_2-p_2)^2}{|\vec{x}-\vec{p}|^5}-\frac{1}{|\vec{x}-\vec{p}|^3}\right)(y_2-p_2)^2\rho(\vec{y})d\vec{y}\\
&\qquad+\frac{1}{2}\int_\Omega\left(\frac{3(x_3-p_3)^2}{|\vec{x}-\vec{p}|^5}-\frac{1}{|\vec{x}-\vec{p}|^3}\right)(y_3-p_3)^2\rho(\vec{y})d\vec{y}+\int_\Omega\frac{3(x_1-p_1)(x_2-p_2)}{|\vec{x}-\vec{p}|^5}(y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y}\\
&\qquad+\int_\Omega\frac{3(x_1-p_1)(x_3-p_3)}{|\vec{x}-\vec{p}|^5}(y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y}+\int_\Omega\frac{3(x_2-p_2)(x_3-p_3)}{|\vec{x}-\vec{p}|^5}(y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y}
\end{aligned}$$

$$\begin{aligned}
&=\frac{1}{|\vec{x}-\vec{p}|}\int_\Omega\rho(\vec{y})d\vec{y}+\frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})\cdot\int_\Omega(\vec{y}-\vec{p})\rho(\vec{y})d\vec{y}\\
&\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_1-p_1)^2-((x_1-p_1)^2+(x_2-p_2)^2+(x_3-p_3)^2))\int_\Omega(y_1-p_1)^2\rho(\vec{y})d\vec{y}\\
&\qquad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_1-p_1)(x_2-p_2))\int_\Omega(y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y}\\
&\qquad\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_1-p_1)(x_3-p_3))\int_\Omega(y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y}\\
&\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_1-p_1)(x_2-p_2))\int_\Omega(y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y}\\
&\qquad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_2-p_2)^2-((x_1-p_1)^2+(x_2-p_2)^2+(x_3-p_3)^2))\int_\Omega(y_2-p_2)^2\rho(\vec{y})d\vec{y}\\
&\qquad\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_2-p_2)(x_3-p_3))\int_\Omega(y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y}\\
&\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_1-p_1)(x_3-p_3))\int_\Omega(y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y}\\
&\qquad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_2-p_2)(x_3-p_3))\int_\Omega(y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y}\\
&\qquad\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}(3(x_3-p_3)^2-((x_1-p_1)^2+(x_2-p_2)^2+(x_3-p_3)^2))\int_\Omega(y_3-p_3)^2\rho(\vec{y})d\vec{y}
\end{aligned}$$

$$\begin{aligned}
&=\frac{1}{|\vec{x}-\vec{p}|}\sum_K\int_K\rho(\vec{y})d\vec{y}+\frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})\cdot\sum_K\int_K(\vec{y}-\vec{p})\rho(\vec{y})d\vec{y}\\
&\quad+\frac{1}{2|\vec{x}-\vec{p}|^5}\text{vectorize}\left(\begin{bmatrix}3(x_1-p_1)^2-r_{\vec{x}\vec{p}}^2 & 3(x_1-p_1)(x_2-p_2) & 3(x_1-p_1)(x_3-p_3)\\ 3(x_1-p_1)(x_2-p_2) & 3(x_2-p_2)^2-r_{\vec{x}\vec{p}}^2 & 3(x_2-p_2)(x_3-p_3)\\ 3(x_1-p_1)(x_3-p_3) & 3(x_2-p_2)(x_3-p_3) & 3(x_3-p_3)^2-r_{\vec{x}\vec{p}}^2\end{bmatrix}\right)
\end{aligned}$$

.

$$\text{vectorize}\left(\sum_K \begin{bmatrix} \int_K (y_1-p_1)^2\rho(\vec{y})d\vec{y} & \int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)^2\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K (y_3-p_3)^2\rho(\vec{y})d\vec{y} \end{bmatrix}\right),$$

where $r_{\vec{x}\vec{p}} = \sqrt{(x_1-p_1)^2+(x_2-p_2)^2+(x_3-p_3)^2}$ denotes the distance between two points $\vec{x}\langle\text{infix-and}\rangle\vec{p}$.

In above formula, vectorize(M) means vectorization of a matrix M. For example, if the matrix M is given by $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, then vectorize$(M)$ gives a column vector $[\,a\ c\ b\ d\,]^T$. Please check wikipedia page for details.

Remark 1. For two matrices M and N, the operation vectorize$(M)\cdot$vectorize(N) is equivalent to the operation $M \circ N$ then the summation of all elements in the result matrix. Here $\circ$ means the Hadamard product (elementwise product).

As a summary, we have the formula for the calculation of $\phi(x)$ below

$$\begin{aligned} \phi(\vec{x}) = & \frac{1}{|\vec{x}-\vec{p}|}\sum_K \int_K \rho(\vec{y})d\vec{y} + \frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})\cdot\sum_K \int_K (\vec{y}-\vec{p})\rho(\vec{y})d\vec{y} \\ & + \frac{1}{2|\vec{x}-\vec{p}|^5}\text{vectorize}\left(\begin{bmatrix} 3(x_1-p_1)^2-r^2_{\vec{x}\vec{p}} & 3(x_1-p_1)(x_3-p_3) \\ 3(x_1-p_1)(x_2-p_2) & 3(x_2-p_2)(x_3-p_3) \\ 3(x_1-p_1)(x_3-p_3) & 3(x_3-p_3)^2-r^2_{\vec{x}\vec{p}} \end{bmatrix}\right) \\ & \cdot\text{vectorize}\left(\sum_K \begin{bmatrix} \int_K (y_1-p_1)^2\rho(\vec{y})d\vec{y} & \int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)^2\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K (y_3-p_3)^2\rho(\vec{y})d\vec{y} \end{bmatrix}\right) \\ = & \frac{1}{|\vec{x}-\vec{p}|}\sum_K \int_K \rho(\vec{y})d\vec{y} + \frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})\cdot\sum_K \int_K (\vec{y}-\vec{p})\rho(\vec{y})d\vec{y} \\ & + \frac{1}{6|\vec{x}-\vec{p}|^5}\text{vectorize}\left(\begin{bmatrix} 3(x_1-p_1)^2-r^2_{\vec{x}\vec{p}} & 3(x_1-p_1)(x_3-p_3) \\ 3(x_1-p_1)(x_2-p_2) & 3(x_2-p_2)(x_3-p_3) \\ 3(x_1-p_1)(x_3-p_3) & 3(x_3-p_3)^2-r^2_{\vec{x}\vec{p}} \end{bmatrix}\right) \end{aligned}$$

.

$$\begin{aligned} & \text{vectorize}\left(\sum_K \begin{bmatrix} 3\int_K (y_1-p_1)^2\rho(\vec{y})d\vec{y} & 3\int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & 3\int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ 3\int_K (y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & 3\int_K (y_2-p_2)^2\rho(\vec{y})d\vec{y} & 3\int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ 3\int_K (y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} & 3\int_K (y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} & 3\int_K (y_3-p_3)^2\rho(\vec{y})d\vec{y} \end{bmatrix}\right) \\ = & \frac{1}{|\vec{x}-\vec{p}|}\sum_K \int_K \rho(\vec{y})d\vec{y} + \frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})\cdot\sum_K \int_K (\vec{y}-\vec{p})\rho(\vec{y})d\vec{y} \\ & + \frac{1}{6|\vec{x}-\vec{p}|^5}\text{vectorize}\left(\begin{bmatrix} 3(x_1-p_1)^2-r^2_{\vec{x}\vec{p}} & 3(x_1-p_1)(x_3-p_3) \\ 3(x_1-p_1)(x_2-p_2) & 3(x_2-p_2)(x_3-p_3) \\ 3(x_1-p_1)(x_3-p_3) & 3(x_3-p_3)^2-r^2_{\vec{x}\vec{p}} \end{bmatrix}\right) \end{aligned}$$

.

$$\text{vectorize}\left(\sum_K \begin{bmatrix} \int_K (3(y_1-p_1)^2-r^2_{\vec{y}\vec{p}})\rho(\vec{y})d\vec{y} & \int_K 3(y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K 3(y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K 3(y_1-p_1)(y_2-p_2)\rho(\vec{y})d\vec{y} & \int_K (3(y_2-p_2)^2-r^2_{\vec{y}\vec{p}})\rho(\vec{y})d\vec{y} & \int_K 3(y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} \\ \int_K 3(y_1-p_1)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K 3(y_2-p_2)(y_3-p_3)\rho(\vec{y})d\vec{y} & \int_K (3(y_3-p_3)^2-r^2_{\vec{y}\vec{p}})\rho(\vec{y})d\vec{y} \end{bmatrix}\right),$$

where $r_{\vec{y}\vec{p}} = \sqrt{(y_1-p_1)^2+(y_2-p_2)^2+(y_3-p_3)^2}$ denotes the distance between two points $\vec{y}$ and $\vec{p}$.

Remark 2. It is noted that a new term $\int_K ((y_1-p_1)^2+(y_2-p_2)^2+(y_3-p_3)^2)\rho(\vec{y})d\vec{y}$ is added on the diagonal line of the second matrix, this is equivalent to add a matrix $\int_K ((y_1-p_1)^2+(y_2-p_2)^2+(y_3-p_3)^2)\rho(\vec{y})d\vec{y}I$, where I is the identity matrix. After a simple calculation, it can be found that there is no affection on the final result, since the trace of the first matrix is zero.

Remark 3. The reason we introduce an extra term in the diagonal element of the second matrix is that, in this case, we will have a unified form for the expression in both matrice. Further, if we use Kronecker delta function δ_{ij}, then terms in each matrix can be given by $3r_ir_j - \delta_{ij}r_ir_j$.

In above formula, quantities such as monopole, dipole, quadrupole could be defined as

$$\begin{aligned} \text{Monopole} &:= \int_\Omega \rho(\vec{y})d\vec{y} \\ \text{Dipole} &:= \int_\Omega (\vec{y}-\vec{p})\rho(\vec{y})d\vec{y} \\ \text{Quadrupole } Q_{ij} &:= \int_\Omega (3(y_i-p_i)(y_j-p_j) - \delta_{ij}r^2_{\vec{y}\vec{p}})\rho(\vec{y})d\vec{y} \end{aligned}$$

In the scene that all particles are clustered in a small region, then multipole expansion give us a way that the potential on all points far from that small region can be calculated in an efficiency approach. Then the potential around that small region can be calculated directly by using the integral. Then the accuracy and efficiency can be well balanced.

However, if particles distribute all over the domain, directly using multipole expansion would no effectively save time. We need tree code algorithm here.

The idea is that, to calculate $\phi(\vec{x}) = \int_\Omega \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y}$, the following splitting would be used, i.e.,

$$\phi(\vec{x}) = \int_\Omega \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y} = \sum_{\text{near field}} \int_K \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y} + \sum_{\text{far field}} \int_K \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\vec{y}$$

In above two terms, we will keep the first one unchanged, due to the large error introduced when the approximation is used. For the second term, due to the large value of the quantity $|\vec{x}-\vec{y}|$, accurate approximation can be expected. The above multipole expansion is actually a charming choice.

More specifically, if K is a direct neighbor of the point $\vec{x}$, then $\phi(\vec{x})$ += $\int_K \frac{\rho(\vec{y})}{|\vec{x}-\vec{y}|} d\ \vec{y}$, and if K is far from $\vec{x}$, then $\phi(\vec{x})$ +=$\frac{1}{|\vec{x}-\vec{p}|}\int_K \rho(\vec{y})d\vec{y} + \frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p}) \cdot \int_K (\vec{y}-\vec{p})\rho(\vec{y})d\vec{y}$.

Remark 4. It is noted that the integral $\int_K \rho(\vec{y})d\,\vec{y}$, and $\int_K (\vec{y}-\vec{p})\rho(\vec{y})d\,\vec{y}$ has been calculated in advance, we only need to calculate the quantities $\frac{1}{|\vec{x}-\vec{p}|}$ and $\frac{1}{|\vec{x}-\vec{p}|^3}(\vec{x}-\vec{p})$.

Remark 5. If for all K elements in the current mesh, we use above approximation, it would also result in $\mathcal{O}(N^2)$ complexity. The essential idea on the acceleration is that, for those K elements which are quite far from $\vec{x}$, we merge them together, and produce an integrated approximation, instead of the sum of the approximation from each element.

2 Implementation of the fast algorithm

The idea on accelerating the calculation, is to generate clustered patch by merging those elements together in the far field. Basically, the clustered patch would be larger with farther distance, comparied with the point $\vec{x}$ for the calculation of $\phi(\vec{x})$.

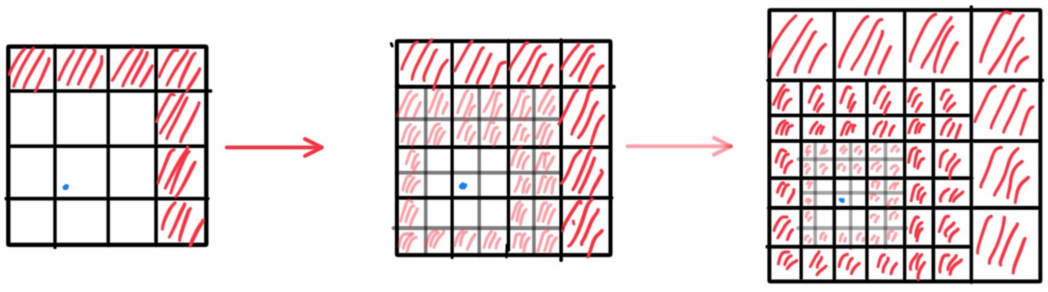


Figure 1. The merged patch for the calculation of $\phi(\vec{x})$ with that blue point $\vec{x}$.

It looks like we have following two questions, before the problem can be solved

- How can we generate those clustered patch efficiently, and,
- How to generate monopole, dipole, quadrupole in each element (including element and those clustered patch)

Answers are as follows

- Using tree structure, and,
- Choosing the mass center of each element (including clustered patch), and calculate the contribution to those -pole by the additivity.

2.1 Codes in AFEPack related to fast algorithm implementation.

Good news is that the mesh in AFEPack is managed by the tree structure (fork tree in 2D, and octree in 3D). Hence those merged patch just corresponds to the parent node in the tree.

We need to use the following class from AFEPack.

2.1.1 Class HGeometry<DIM>

The class HGeometry<DIM> in AFEPack is given below.

```
/**
 * Hierarchy geometry. This is the basis class to make the hierarchy geometry
 * to be able to refined. It store the information of the realation ship of the
 * hierarchy tree structure.
 */
template <int DIM, int DOW=DIM>
  class HGeometry : public HGeometryInfo<DIM>, public HGeometryBase
  {
  public:
  enum { dim = DIM, dow = DOW };
  typedef HGeometry<0,dow> vertex_t;
  typedef HGeometry<dim-1,dow> bound_t;
  typedef HGeometry<dim,dow> this_t;
  typedef this_t child_t;
  typedef this_t parent_t;

  int index;
  std::vector<vertex_t *> vertex;
  std::vector<bound_t *> boundary;
  parent_t * parent;
  std::vector<child_t *> child;
  bmark_t bmark;

  public:
  HGeometry();
  virtual ~HGeometry() {}

  public:
  bool isRefined() const;
  bool isIncludePoint(const Point<DOW>&) const;
  void refine();
  void checkIntegrity() const;

  friend std::ostream& operator<< <>(std::ostream&, const HGeometry<DIM,DOW>&);
  };
```

2.1.2 Class GeometryTree<DIM>

The calss GeometryTree<DIM> in AFEPack is given below.

```
/**
 * Hierarchy geometry tree. This is the class to manage all those macro elements, as
 * the roots of all those hierarchy geometries.
 */
template <int DIM, int DOW=DIM>
  class HGeometryTree
  {
  public:
  enum { dim = DIM, dow = DOW };

  private:
  typedef HGeometry<DIM,DOW> entry_t;
  public:
  typedef std::list<entry_t *> container_t;
  private:
  container_t root_element;

  bool _is_locked; /**
                    * 我们在进行半正则化以前对树进行加锁，在完成正则化
                    * 以后对树进行解锁，使得在此过程中数据具有一致性。
                    * 注意我们只是避免两个同时出现的不同的半正则化+正则
                    * 化操作发生冲突，并不能完全避免破坏性地使用。
                    */

  public:
  typedef _Deref_iterator<typename container_t::iterator, entry_t> RootIterator;
  typedef _Deref_iterator<typename container_t::const_iterator, const entry_t>
ConstRootIterator;

  typedef HTools Tools;
```

```
    public:
    HGeometryTree() : _is_locked(false) {};
    virtual ~HGeometryTree() {clear();};

    protected:
    bool lock() {
      if (_is_locked) return false;
      else {
        _is_locked = true;
        return true;
      }
    }
    void unlock() {
      _is_locked = false;
    }

    public:
    container_t& rootElement() { return root_element; }
    const container_t& rootElement() const { return root_element; }

    unsigned int n_rootElement() const {return root_element.size();}

    RootIterator beginRootElement() {return RootIterator(root_element.begin());}
    RootIterator endRootElement() {return RootIterator(root_element.end());}

    ConstRootIterator beginRootElement() const {return
  ConstRootIterator(root_element.begin());}
    ConstRootIterator endRootElement() const {return
  ConstRootIterator(root_element.end());}

    void clear(); // this method need very complex implementation, left for future
    void checkIntegrity();

    bool is_locked() const { return _is_locked; }
    bool& is_locked() { return _is_locked; }

    /**
     * This is the routine used to read in mesh data generated by the software "easymesh".
     * For 2 dimensional case only.
     */
    void readEasyMesh(const std::string&);

    /**
     * This is the routine used to read in the mesh data in the internal data format.
     */
    void readMesh(const std::string&);

    friend class IrregularMesh<DIM, DOW>;
    };
```

2.1.3 Class HElement<DIM>

The class HElement<DIM> in AFEPack is given below.

```
/**
 * Hierarcy element. Hierarchy element is the basic component of the irregular
 * mesh. The hierarchy tree to construct the irregular mesh is different from the
 * hierarchy geometry tree. It have only the tree of the elements, but no tree
 * of those lower dimensional geometries. In fact, the irregular mesh is a subtree
 * of the hierarchy element geometry tree.
 */
template <int DIM, int DOW=DIM>
  class HElement : public HGeometryInfo<DIM>, public HGeometryBase {
 public:
 enum { dim = DIM, dow = DOW };
 typedef HGeometry<dim,dow> h_element_t;
 typedef HElement<dim,dow> element_t;
 typedef element_t parent_t;
 typedef element_t child_t;

 public:
    typedef int                             ElementType;
    static const ElementType                NOT_ACTIVE = -1;

    // for 2 dimensional case
    static const ElementType                TRIANGLE = 0;
    static const ElementType                QUADRILATERAL = 1;

    // for 3 dimensional case
    static const ElementType                TETRAHEDRON = 0;
    static const ElementType                TWIN_TETRAHEDRON = 1;
    static const ElementType                FOUR_TETRAHEDRON = 2;
    public:
    int                                     index;
    double                                  indicator;
    int                                     value; // default: -1
    h_element_t *                           h_element; // default: NULL
    element_t *                             parent; // default: NULL
    std::vector<element_t *>               child; // default: NULL
    public:
    HElement();
    HElement(const element_t&);
    virtual ~HElement();
    public:
    element_t& operator=(const element_t&);
    /* bool is_dummy() const { return h_element->is_dummy(); } */
    bool isRefined() const;
    bool isIncludePoint(const Point<DOW>&) const;
    void refine();
    void checkIntegrity() const;

    friend std::ostream& operator<< <>(std::ostream&, const HElement<DIM, DOW>&);
  };
```

2.1.4 Class IrregularMesh<DIM>

The class IrregularMesh<DIM> in AFEPack is given below.

```
  /**
   * IrregularMesh is an image of a complete subtree of the hierarchy element
   * geometry tree. It's related with the hierarchy tree and a regular mesh
   * which is generated from itself to used by finite element space. It mainly
   * provides many iterator to used by other routines to access all those
   * elements, or certain selective elements in the mesh. It provides the most
   * important operation to make the irregular mesh a semi-regular mesh --
   * semiregularize, and the operation to construct the related regular
   * mesh -- regularize. The mesh adaptation are mainly depended on those
   * operations.
   */
  template <int DIM, int DOW=DIM>
    class IrregularMesh
    {
    public:
    enum { dim = DIM, dow = DOW };
    typedef RegularMesh<DIM,DOW> mesh_t;
    typedef HGeometryTree<DIM,DOW> tree_t;
    typedef IrregularMesh<DIM,DOW> ir_mesh_t;

    protected:
    typedef HGeometry<DIM,DOW> h_element_t;
    typedef HElement<DIM,DOW> element_t;
    typedef std::list<element_t *> container_t;
    typedef HTools Tools;

    private:
    tree_t * geometry_tree; // default: NULL
    container_t root_element; // default: NULL
    mesh_t * regular_mesh; // default: NULL

    public:
    typedef _Deref_iterator<typename container_t::iterator, element_t> RootIterator;
    typedef _Deref_iterator<typename container_t::const_iterator, const element_t>
  ConstRootIterator;
```

```
typedef RootFirstElementIterator<DIM, DOW> RootFirstIterator;
typedef ActiveElementIterator<DIM, DOW> ActiveIterator;

public:
IrregularMesh();
explicit IrregularMesh(tree_t&);
IrregularMesh(const ir_mesh_t&);
virtual ~IrregularMesh();

public:
void clear();
ir_mesh_t& operator=(const ir_mesh_t&);

public:
RootIterator beginRootElement() {return root_element.begin();}
RootIterator endRootElement() {return root_element.end();}

ConstRootIterator beginRootElement() const {return root_element.begin();};
ConstRootIterator endRootElement() const {return root_element.end();};

RootFirstIterator beginRootFirstElement();
RootFirstIterator endRootFirstElement();

ActiveIterator beginActiveElement();
ActiveIterator endActiveElement();

public:
/**
 * 使用几何遗传树对非正则网格进行重新初始化。其中第二个参数 is_bare
 * 是新加的，如果 is_bare 为 true，则仅仅设置几何遗传树的指针为
 * htree 的地址；否则就使用几何遗传树的信息彻底重构所有的信息。一般我
 * 们会使用后一个功能，前一个功能是为了从序列化的文档中读入几何遗传树
 * 和非正则网格的数据后，直接设定几何遗传树的指针而用
 *
 * @param htree 几何遗传树
 * @param is_bare 是否用几何遗传树的信息重构整个非正则网格的数据结构
 *
 */
void reinit(tree_t& htree, bool is_bare = false);
tree_t& geometryTree() const {return *geometry_tree;};
container_t& rootElement() {return root_element;};
mesh_t& regularMesh() {return *regular_mesh;};
const mesh_t& regularMesh() const {return *regular_mesh;};

virtual void semiregularize();
void regularize(bool renumerate = true);
void globalRefine(unsigned int i = 1);
void randomRefine(double percent = 50.0);
void writeFormatted(const std::string&);

void copyTree(const ir_mesh_t&);
void copyNonnegtiveSubtree(const ir_mesh_t&);

void copyTree(const element_t *, element_t *);
void copyNonnegtiveSubtree(const element_t *, element_t *);

void deleteTree(element_t *);

friend std::ostream& operator<< <>(std::ostream&, IrregularMesh<DIM, DOW>&);
friend class IrregularMeshPair<DIM, DOW>;

protected:
void checkIntegrity();
void setGeometryTree(tree_t *);

void semiregularizeHelper(bool&, int&);
void semiregularizeHelper(bool&, element_t&, int&);

void prepareSemiregularize();
void prepareSemiregularizeHelper(h_element_t *);

void renumerateElement();

void refineElement(element_t& h_ele);

public:
friend class RegularMesh<DIM, DOW>;
};
```

2.1.5 How to understand above classes

First of all, HGeometry<DIM> defines the geometry element, which would describe the geometry of each node in tree. Then HGeometryTree<DIM> defines a tree structure, from which we may find codes below

```
private:
  typedef HGeometry<DIM,DOW> entry_t;
public:
  typedef std::list<entry_t *> container_t;
private:
  container_t root_element;
```

As we can see from above, std::list<HGeometry<DIM, DOW> *> is defined, which is used to define a quantity container_t for root element.

```
int index;
std::vector<vertex_t *> vertex;
std::vector<bound_t *> boundary;
parent_t * parent;
std::vector<child_t *> child;
bmark_t bmark;
```

Above is codes from HGeometry<DIM>, from which we can see that vertex and boundary are used for describing the geometry of the element, while parent and child are used to describe the tree structure (the parent nodes and children nodes of current node).

By defining HElement<DIM>, the fundamental element for mesh is defined. From the code

```
typedef HGeometry<dim,dow> h_element_t;
typedef HElement<dim,dow> element_t;
typedef element_t parent_t;
typedef element_t child_t;

int                                     index;
double                                  indicator;
int                                     value; // default: -1
h_element_t *                           h_element; // default: NULL
element_t *                             parent; // default: NULL
std::vector<element_t *>               child; // default: NULL
```

we can see that there is a corresponding HGeometry<DIM> h_element_t in each HElement. Also, there are parent and child HElement<DIM>.

Finally, we have IrregularMesh<DIM>, which define an irregular mesh.

```
typedef RegularMesh<DIM,DOW> mesh_t;
typedef HGeometryTree<DIM,DOW> tree_t;
typedef IrregularMesh<DIM,DOW> ir_mesh_t;

protected:
typedef HGeometry<DIM,DOW> h_element_t;
typedef HElement<DIM,DOW> element_t;
typedef std::list<element_t *> container_t;
typedef HTools Tools;

private:
tree_t * geometry_tree; // default: NULL
container_t root_element; // default: NULL
mesh_t * regular_mesh; // default: NULL
```

From above it can be seen that a geometry_tree is inside, and a container_t (std::list<HElement<DIM> *>) is defined. It looks like that the quantity HGeometry<DIM, DOW> h_element_t; is useless (if we want to know the HGeometry of current HElement, we can just pick it up from HElement<DIM>).

Some points on coding the algorithm in AFEPack

- We need to generate monopole, dipole, quadrupole for each node in the tree. One way to save such an information, is to revise HElement, i.e., to cearte new vectors for those information. However, this is not appropriate to keep the class HElement general.
- Another way to do such a thing, is to create a vector. The length of the vector is same to the total number of nodes in the tree. Then how can we make the correspondence between HElement and the element from the vector? The answer could be std::map. By using std::map, we may bind two values together. It is noted that the first value should be "ordered", in the sense that the operator "less" would work based on the first key value.
- In our case, we have this IrregularMesh<DIM> and all HElement<DIM> inside. This is a tree structure, in which the loop should be done along some trajectory (not randomly). If we dont modify the class HElement<DIM>, then one possible approach for the purpose is to create an nodecache (for each node in h_tree), then make a vector or list like std::vector<nodeCache>, or std::list<nodeCache>. To build the connection between h_tree and nodecache, is to use std::map<HElement<DIM>*, nodeCache *> (this is doable since we have order property for pointers.)

2.2 How to proceed on coding

- What do we need to implement the tree code algorithm?

  For each node in the octree, which corresponds to an element in geometry, we need to generate two vectors (or set), one is called distant_element, and one is called neighbor_element. It is noted that whichever vector the element is from, the element must be in the same layer in the tree w.r.t. the current element(node).
- How to implement?

  For elements in distant_element vector, we use multipole expansion to collect the interaction. For elements in neighbor_element vector, we go deeper to figure out the new distant_element and neighbor_element vectors.

  To make the implementation efficient, it is better to save above two vectors before the calculation. Then How?

  - Loop all nodes (element) in the tree, to build a std::vector<std::set> for each vertex, to save element information which contain current vertex, level-by-level style (std::vector<⟨text-dots⟩>).
  -

2.3 Show me the code

```
#include <iostream>
#include <AFEPack/HGeometry.h>
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <assert.h>
#include <omp.h>
#include <fstream>
#include <iomanip>

const int DIM = 2;

struct hElementCache
{
  std::set<HElement<DIM>* > neighbor_elements;
  std::set<HElement<DIM>* > nonNeighbor_elements;


  AFEPack::Point<DIM> ref_point;
  double monopole;
  std::array<double, DIM> dipole;
  std::array<double, DIM * DIM> quadrupole;




};

void record_current_hElement_node_neighbors(HElement<DIM>& the_HElement,
                                            int level,

std::vector<std::vector<std::set<HElement<DIM>* > > >& node_neighbors,
                                            int& n_hElement)
{
  n_hElement ++;
  HGeometry<DIM>* the_geometry = the_HElement.h_element;
  std::vector<HGeometry<0, DIM>* > the_vertex = the_geometry->vertex;
  const int& n_vtx = the_vertex.size();
  for(int i = 0;i < n_vtx;++ i){
    node_neighbors[the_vertex[i]->index][level].insert(&the_HElement);
  }

  if(the_HElement.value == 0) return;
  else{
    const int& n_child = the_HElement.child.size();
    for(int i = 0;i < n_child;++ i){
      record_current_hElement_node_neighbors(*the_HElement.child[i], level + 1,
node_neighbors, n_hElement);
    }
  }

}

void fill_neighbor_elements_for_current_hElement(HElement<DIM>& the_hElement,
                                                 int& the_idx,
                                                 std::vector<hElementCache>&
hElement_cache,
                                                 std::map<HElement<DIM>*, int>&
map_HElement_to_int,
                                                 int& level,

std::vector<std::vector<std::set<HElement<DIM>* > > >& node_neighbors)
{
  //std::cout << "The current the_idx is " << the_idx << std::endl;
  map_HElement_to_int.insert({&the_hElement, the_idx});/// create a pair in map
  hElementCache& the_hElement_cache = hElement_cache[the_idx];
  HGeometry<DIM>* the_hGeometry = the_hElement.h_element;
  std::vector<HGeometry<0, DIM> *> the_vtx = the_hGeometry->vertex;
  const int& n_vtx = the_vtx.size();
  for(int i = 0;i < n_vtx;++ i){
    const int& idx_vtx = the_vtx[i]->index;
    /// if we implement the following only, we would introduce an
    /// extra layer outside the REAL boundary of current region. This
    /// is a bug.
    // TO FIX THIS BUG: we need revise
    /// neighbor_elements during the generation of
    /// nonNeighbor_elements, in which a COMPLETE_SET is defined by
    /// merging all child elements.
    /// SO: 1. we leave below unchanged.
    ///     2. when complete_set is obtained, we remove those elements
    ///        from neighbor_elements which are not in complete_set
    the_hElement_cache.neighbor_elements.insert(node_neighbors[idx_vtx][level].begin(),
                                                node_neighbors[idx_vtx][level].end());
  }

  if(the_hElement.value == 0) {/// if this is a leaf node
    //-- level;
    return;
  }
  else{
    const int& n_child = the_hElement.child.size();
    for(int i = 0;i < n_child;++ i){
      fill_neighbor_elements_for_current_hElement(*(the_hElement.child[i]),
                                                  ++ the_idx,
                                                  hElement_cache,
                                                  map_HElement_to_int,
                                                  ++ level,
                                                  node_neighbors);
      -- level;
    }
  }
}


/// Attention: Besides generating nonNeighbor_elements, we also revise neighbor_elements.
void fill_nonNeighbor_elements_for_current_hElement(HElement<DIM>& the_hElement,
                                                    std::vector<hElementCache>&
hElement_cache,
                                                    std::set<HElement<DIM>* >&
complete_set,
                                                    std::map<HElement<DIM>*, int>&
map_HElement_to_int)
{
  if(the_hElement.value == 0){///if this is a leaf node
    return;
  }

  /// do not use clear(), due to repeatedly memory allocation and deallocation
  //complete_set.clear();/// resize the complete_set as 0;

  complete_set.erase(complete_set.begin(), complete_set.end());

  std::map<HElement<DIM>*, int>::iterator the_map_it =
map_HElement_to_int.find(&(the_hElement));
  const int& idx_hElement = the_map_it->second;
  hElementCache& the_hElementCache = hElement_cache[idx_hElement];

  std::set<HElement<DIM>* >::iterator
    the_set_it = the_hElementCache.neighbor_elements.begin(),
    end_set_it = the_hElementCache.neighbor_elements.end();
  for(;the_set_it != end_set_it;++ the_set_it){
    //complete_set.insert(&(*the_set_it));
    if((*the_set_it)->value == 0) continue;///if this is a leaf node

    const int& n_child = (*the_set_it)->child.size();
    for(int i = 0;i < n_child;++ i){
      complete_set.insert((*the_set_it)->child[i]);
    }

  }
  /// now we have the complete set w.r.t. the_hElement. The next step,
  /// is to generate nonNeighbor_elements for each child in
  /// the_hElement, w.r.t. this complete set.
  const int& n_child = the_hElement.child.size();
  for(int i = 0;i < n_child;++ i){
    HElement<DIM>* the_child = the_hElement.child[i];
    the_map_it = map_HElement_to_int.find(the_child);
    const int& idx_hElement_child = the_map_it->second;
    hElementCache& the_hElementCache_child = hElement_cache[idx_hElement_child];

    /// to generate nonNeighbor_elements
    std::set_difference(complete_set.begin(),
                        complete_set.end(),
                        the_hElementCache_child.neighbor_elements.begin(),
                        the_hElementCache_child.neighbor_elements.end(),
                        std::inserter(the_hElementCache_child.nonNeighbor_elements,

the_hElementCache_child.nonNeighbor_elements.begin()));

    /// After we generate nonNeighbor_elements, we use it to generate neighbor_elements
    the_hElementCache_child.neighbor_elements.clear();/// to clear neighbor_elements.
    std::set_difference(complete_set.begin(),
                        complete_set.end(),
                        the_hElementCache_child.nonNeighbor_elements.begin(),
                        the_hElementCache_child.nonNeighbor_elements.end(),
                        std::inserter(the_hElementCache_child.neighbor_elements,

the_hElementCache_child.neighbor_elements.begin()));

  }

  /// now we are ready for a recursive call
  for(int i = 0;i < n_child;++ i){
    fill_nonNeighbor_elements_for_current_hElement(*(the_hElement.child[i]),
                                                   hElement_cache,
                                                   complete_set,
                                                   map_HElement_to_int);
  }

}

void write_current_hElement_neighbor(std::fstream& file,
                                     std::fstream& mpFile,
                                     HElement<DIM>* the_hElement,
                                     std::map<HElement<DIM>*, int>& map_HElement_to_int,
                                     std::vector<hElementCache>& hElement_cache,
                                     RegularMesh<DIM>& r_mesh
                                     )
{
  std::map<HElement<DIM>* , int>::iterator the_map_it =
map_HElement_to_int.find(the_hElement);
  const int& idx_hElementCache = the_map_it->second;
  hElementCache& the_hElementCache = hElement_cache[idx_hElementCache];

  //std::set<HElement<DIM>* >& the_neighbor_elements =
the_hElementCache.neighbor_elements;
  std::set<HElement<DIM>* >::iterator
    the_set_it = the_hElementCache.neighbor_elements.begin(),
    end_set_it = the_hElementCache.neighbor_elements.end();
  for(;the_set_it != end_set_it;++ the_set_it){
    HGeometry<DIM>* current_geometry = (*the_set_it)->h_element;
    const int& n_vtx = current_geometry->vertex.size();
    mpFile << "draw ";
    for(int j = 0;j < n_vtx;++ j){
      if(j != n_vtx - 1){
        mpFile << "(" << 10. *r_mesh.point(current_geometry->vertex[j]->index)[0] << "cm,
"
               << 10.* r_mesh.point(current_geometry->vertex[j]->index)[1] << "cm)--";
      }
      else{
        mpFile << "(" << 10.* r_mesh.point(current_geometry->vertex[j]->index)[0] << "cm,
"
               << 10.* r_mesh.point(current_geometry->vertex[j]->index)[1] <<
"cm)--cycle;" << std::endl;
      }

      for(int k = 0;k < DIM;++ k){
        file << r_mesh.point(current_geometry->vertex[j]->index)[k] << " ";
      }
      file << std::endl;
    }
  }

  if(the_hElement->parent == NULL) return;

  write_current_hElement_neighbor(file, mpFile, the_hElement->parent,
map_HElement_to_int, hElement_cache, r_mesh);
}

void write_current_layer(std::fstream & file,
                         int& level,
                         HElement<DIM>* the_hElement,
                         std::map<HElement<DIM>*, int>& map_HElement_to_int,
                         std::vector<hElementCache>& hElement_cache,
                         RegularMesh<DIM>& r_mesh
                         )
{
  if(the_hElement->parent != NULL){
    write_current_layer(file,
                        ++ level,
                        the_hElement->parent,
                        map_HElement_to_int,
                        hElement_cache,
                        r_mesh);
  }
  int current_level_for_plot = 1;

  if(level < current_level_for_plot) return;
  std::map<HElement<DIM>*, int>::iterator the_map_it =
map_HElement_to_int.find(the_hElement);
  const int& idx_hElementCache = the_map_it->second;
  hElementCache& the_hElementCache = hElement_cache[idx_hElementCache];
  std::set<HElement<DIM>* >& neighbor_elements = the_hElementCache.neighbor_elements;
  std::set<HElement<DIM>* >& nonNeighbor_elements =
the_hElementCache.nonNeighbor_elements;

  //file << "beginfig(" << level << ")" << std::endl;

  std::set<HElement<DIM>* >::iterator the_set_it, end_set_it;
  the_set_it = neighbor_elements.begin();
  end_set_it = neighbor_elements.end();
  for(;the_set_it != end_set_it;++ the_set_it){
    HElement<DIM>* current_hElement = *the_set_it;
    HGeometry<DIM>* current_hGeometry = current_hElement->h_element;
    std::vector<HGeometry<0, DIM>* > the_vtx = current_hGeometry->vertex;
    const int& n_vtx = the_vtx.size();

    file << "fill ";
    for(int j = 0;j < n_vtx;++ j){
      if(j != n_vtx - 1){
        file << std::fixed <<std::setprecision(16) << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--";
      }
      else{
        file << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle withcolor white;"
             << std::endl;
      }
    }
    file << "draw ";
    for(int j = 0;j < n_vtx;++ j){
      if(j != n_vtx - 1){
        file << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--";
      }
      else{
        file << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle;"
             << std::endl;
      }
    }

  }


  the_set_it = nonNeighbor_elements.begin();
  end_set_it = nonNeighbor_elements.end();
  for(;the_set_it != end_set_it;++ the_set_it){
    HElement<DIM>* current_hElement = *the_set_it;
    HGeometry<DIM>* current_hGeometry = current_hElement->h_element;
    std::vector<HGeometry<0, DIM>* > the_vtx = current_hGeometry->vertex;
    const int& n_vtx = the_vtx.size();

    file << "fill ";
    for(int j = 0;j < n_vtx;++ j){
      if(j != n_vtx - 1){
        file << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--";
      }
      else{
        if(level == current_level_for_plot){
          file << "("
               << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
               << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle withcolor 0." << 7 <<
"white;"
               << std::endl;
        }
        else{
          file << "("
               << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
               << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle withcolor white;"
               << std::endl;
        }
      }
    }
    file << "draw ";
    for(int j = 0;j < n_vtx;++ j){
      if(j != n_vtx - 1){
        file << "("
             << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
             << r_mesh.point(the_vtx[j]->index)[1] << "cm)--";
      }
      else{
        if(level == current_level_for_plot){
          file << "("
               << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
               << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle;"
               << std::endl;
        }
        else{
          file << "("
               << r_mesh.point(the_vtx[j]->index)[0] << "cm, "
               << r_mesh.point(the_vtx[j]->index)[1] << "cm)--cycle dashed evenly scaled
8;"
               << std::endl;
        }
      }
    }

  }

  level --;

  //file << "endfig;" << std::endl;

  // if(the_hElement->parent == NULL) return;

  // write_current_layer(file,
  //                     ++ level,
  //                     the_hElement->parent,
  //                     map_HElement_to_int,
  //                     hElement_cache,
  //                     r_mesh);


}



int main(int argc, char* argv[])
{
  HGeometryTree<DIM> h_tree;
  h_tree.readMesh(argv[1]);

  IrregularMesh<DIM> ir_mesh;
  ir_mesh.reinit(h_tree);
  ir_mesh.globalRefine(atoi(argv[2]));
  ir_mesh.semiregularize();
  ir_mesh.regularize(false);
  RegularMesh<DIM>& r_mesh = ir_mesh.regularMesh();

  const int n_node = r_mesh.n_geometry(0);

  std::cout << "There are total " << n_node << " nodes" << std::endl;

  /// we will build a neighbor element list for every node in the
  /// mesh. In this list, we also split the information into different
  /// layers, i.e., root layer, root + 1 layer, root + 2 layer, etc,
  /// since we need to find the neighbor elements in each layers.
  /// 1. By default, we introduce 10 layers. More layers will be added when needed.
  /// 2. This list will be updated when the mesh is updated.
  std::vector<std::vector<std::set<HElement<DIM>* > > > node_neighbors(n_node,

std::vector<std::set<HElement<DIM>*> >(10));

  std::vector<HElement<DIM> *> root_element_list;
  IrregularMesh<DIM>::RootIterator
    the_root = ir_mesh.beginRootElement(),
    end_root = ir_mesh.endRootElement();
  for(;the_root != end_root;++ the_root){
    root_element_list.push_back(&(*the_root));
  }
  const int& n_root_element = root_element_list.size();


  int n_hElement = 0;
  the_root = ir_mesh.beginRootElement();
  for(;the_root != end_root;++ the_root){
    int level = 0;
    record_current_hElement_node_neighbors(*the_root, level, node_neighbors, n_hElement);
  }

  std::cout << "There are total " << n_hElement << " hElements" << std::endl;
  std::vector<hElementCache> hElement_cache(n_hElement);
  std::map<HElement<DIM>*, int> map_HElement_to_int;

  double b_time = omp_get_wtime();
  /// fill neighbor_elements in each HElement first.
  int the_idx = 0;
  int level = 0;
  //the_root = ir_mesh.beginRootElement();
  //for(;the_root != end_root;++ the_root){
  /// can not parallelize following code, due to the_idx, the index for increasing
hElement_cache
  for(int i = 0;i < n_root_element;++ i){
    HElement<DIM>* current_root = root_element_list[i];
    fill_neighbor_elements_for_current_hElement(*current_root, the_idx, hElement_cache,
map_HElement_to_int, level, node_neighbors);
    ++ the_idx;
  }

  std::cout << "Filling neighbor_elements uses: " << omp_get_wtime() - b_time << "
seconds" << std::endl;

  b_time = omp_get_wtime();
  /// fill nonneighrbor_elements in each HElement, then.
  std::set<HElement<DIM> *> complete_set;
  /// firstly, we generate nonneighrbor_elements for those root elements
  the_root = ir_mesh.beginRootElement();
  for(;the_root != end_root;++ the_root){
    complete_set.insert(&(*the_root));
  }

  the_root = ir_mesh.beginRootElement();
  for(;the_root != end_root;++ the_root){
    std::map<HElement<DIM>*, int>::iterator the_it =
map_HElement_to_int.find(&(*the_root));
    const int& idx_hElementCache = the_it->second;
    hElementCache& the_hElementCache = hElement_cache[idx_hElementCache];
    std::set_difference(complete_set.begin(), complete_set.end(),
                        the_hElementCache.neighbor_elements.begin(),
the_hElementCache.neighbor_elements.end(),
                        std::inserter(the_hElementCache.nonNeighbor_elements,
the_hElementCache.nonNeighbor_elements.begin()));

    /// for debug
    assert(complete_set.size() == (the_hElementCache.neighbor_elements.size() +
the_hElementCache.nonNeighbor_elements.size()));
  }

  std::cout << "Filling root nonNeighbor_elements uses: " << omp_get_wtime() - b_time <<
" seconds" << std::endl;



  b_time = omp_get_wtime();
#pragma omp parallel
  {
#pragma omp for private(complete_set)
    for(int i = 0;i < n_root_element;++ i){
      HElement<DIM>* current_root = root_element_list[i];
      fill_nonNeighbor_elements_for_current_hElement(*current_root,
                                                     hElement_cache,
                                                     complete_set,
                                                     map_HElement_to_int);
    }
  }


  std::cout << "fill all nonRoot nonNeighbor_elements uses: " << omp_get_wtime() - b_time
<< " seconds " << std::endl;

  ////////////////////////////////////////////////// output hierarchical structure of a
given leaf hElement//////
  // int tmp = 0;
  // std::fstream file, mpFile;
  // file.open("hierarchy_point_tree.dat", std::ios::in|std::ios::out|std::ios::trunc);
  // mpFile.open("hierarchicalStructure.mp",
std::ios::in|std::ios::out|std::ios::trunc);
  // mpFile << "beginfig(1)" << std::endl;
  IrregularMesh<DIM>::ActiveIterator
    the_activeElement = ir_mesh.beginActiveElement(),
    end_activeElement = ir_mesh.endActiveElement();
  // for(;the_activeElement != end_activeElement;++ the_activeElement){
  //   tmp ++;
  //   if(tmp < 20000) continue;
  //   write_current_hElement_neighbor(file,
  //                                   mpFile,
  //                                   &(*the_activeElement),
  //                                   map_HElement_to_int,
  //                                   hElement_cache,
  //                                   r_mesh
  //                                   );


  //   break;
  // }

  // the_root = ir_mesh.beginRootElement();
  // for(;the_root != end_root;++ the_root){
  //   HGeometry<DIM>* the_hGeometry = the_root->h_element;
  //   const int& n_vtx = the_hGeometry->vertex.size();
  //   mpFile << "draw ";
  //   for(int j = 0;j < n_vtx;++ j){
  //     if(j != n_vtx - 1){
  //    mpFile << "(" << 10. *r_mesh.point(the_hGeometry->vertex[j]->index)[0] << "cm, "
  //            << 10.* r_mesh.point(the_hGeometry->vertex[j]->index)[1] << "cm)--";
  //     }
  //     else{
```

```
  //    mpFile << "(" << 10.* r_mesh.point(the_hGeometry->vertex[j]->index)[0] << "cm, "
  //           << 10.* r_mesh.point(the_hGeometry->vertex[j]->index)[1] << "cm)--cycle;"
<< std::endl;
  //     }

  //     for(int k = 0;k < DIM;++ k){
  //    file << r_mesh.point(the_hGeometry->vertex[j]->index)[k] << " ";
  //     }
  //     file << std::endl;
  //   }
  // }


  // file.close();

  // mpFile << "endfig;" << std::endl << "bye;";
  // mpFile.close();
  /////////////////////////////////////////////////////////////////////////

  int tmp = 0;
  std::fstream mFile;
  level = 1;
  mFile.open("treeCode.mp", std::ios::in|std::ios::out|std::ios::trunc);
  mFile << "beginfig(1)" << std::endl;
  the_activeElement = ir_mesh.beginActiveElement();
  for(;the_activeElement != end_activeElement;++ the_activeElement){
    tmp ++;
    if(tmp < 23000) continue;
    write_current_layer(mFile,
                        level,
                        &(*the_activeElement),
                        map_HElement_to_int,
                        hElement_cache,
                        r_mesh
                        );

    HGeometry<DIM>* the_hGeometry = (*the_activeElement).h_element;
    std::vector<HGeometry<0, DIM>* >& the_vtx = the_hGeometry->vertex;
    const int& n_vtx = the_vtx.size();
    mFile << "fill ";
    for(int i = 0;i < n_vtx;++ i){
      if(i != n_vtx - 1){
        mFile << "(" << r_mesh.point(the_vtx[i]->index)[0]
              << "cm, " << r_mesh.point(the_vtx[i]->index)[1] << "cm)--";
      }
      else{
        mFile << "(" << r_mesh.point(the_vtx[i]->index)[0]
              << "cm, " << r_mesh.point(the_vtx[i]->index)[1] << "cm)--cycle withcolor
0.9red;"
              <<std::endl;
      }
    }
    break;
  }

  mFile << "endfig;" << std::endl;
  mFile << "bye;" << std::endl;
  mFile.close();

  r_mesh.writeOpenDXData("D.dx");

  return 0;


}
```

- First of all, above code generate a neighbor_elements and a nonNeighbor_elements lists for each HElement node in IrregularMesh;
- For 2D case, we generate the following figures to show the results

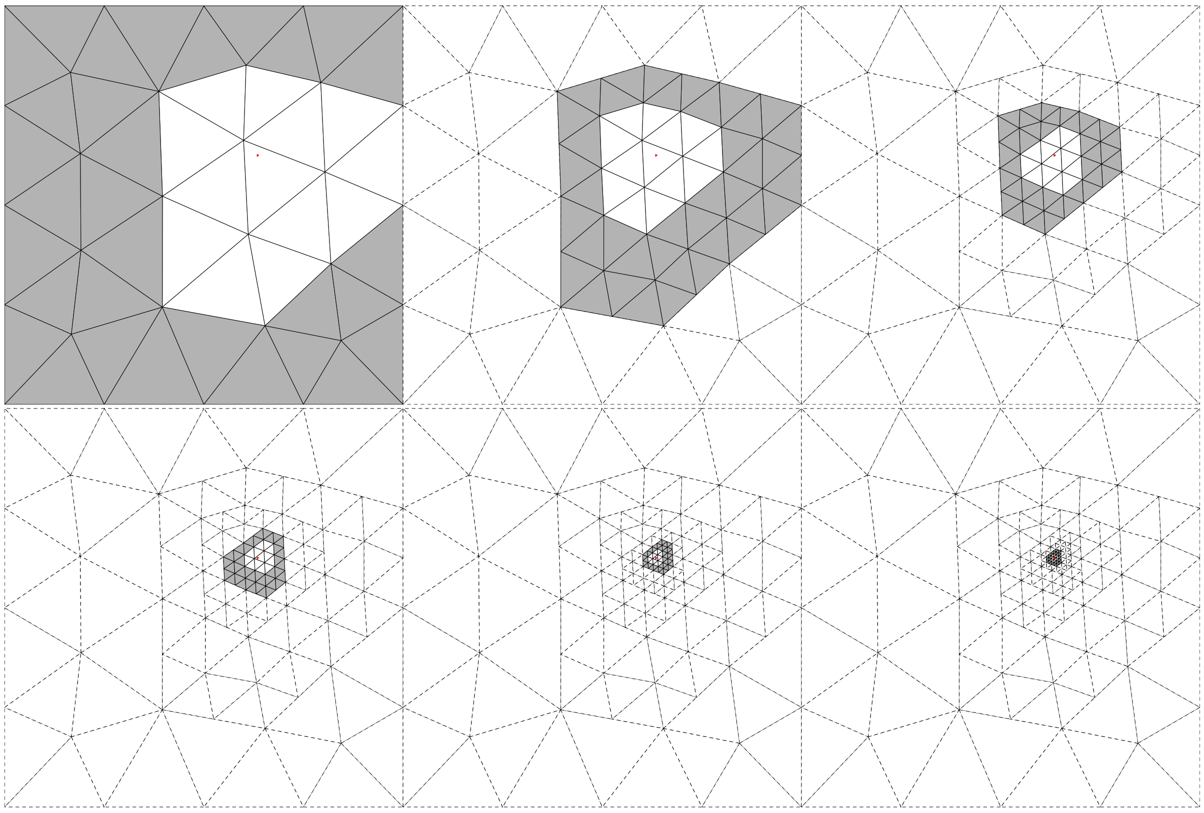


Figure 2. In above process, for the target element with red color, a series of non_neighbor elements are generated.

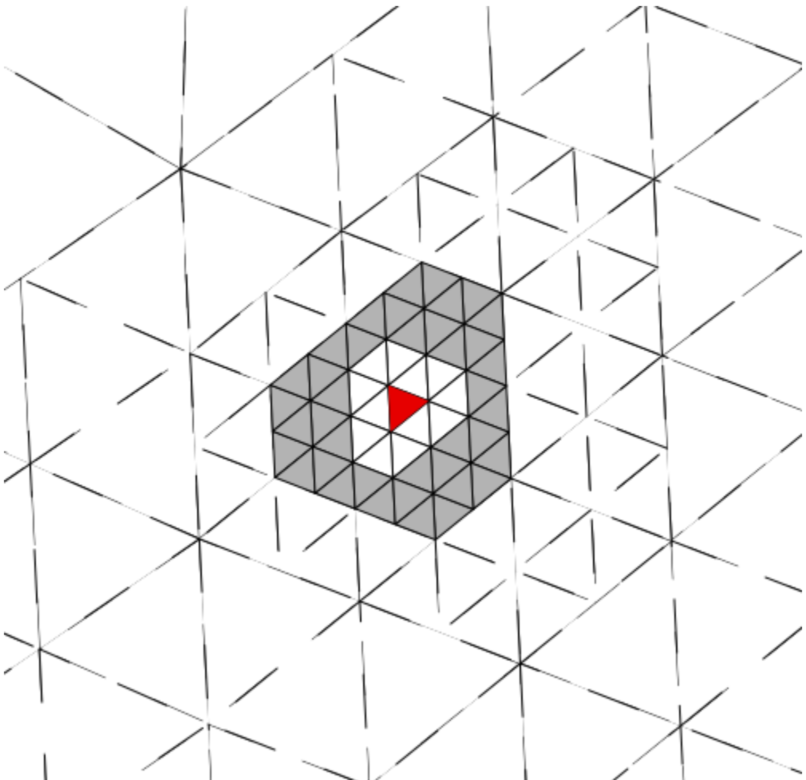


Figure 3.

2.4 Example

```
Maxima 5.43.2 http://maxima.sourceforge.net
using Lisp GNU Common Lisp (GCL) GCL 2.6.12
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
(%i10) diff(diff(exp(-pi*abs(x))*exp(-2*pi*abs(y))*exp(-3*pi*abs(z)),x),x)
```

(%o10) $\mathrm{pi}^2 e^{-3\mathrm{pi}|z|-2\mathrm{pi}|y|-\mathrm{pi}|x|}$

```
(%i11) diff(diff(exp(-pi*abs(x))*exp(-2*pi*abs(y))*exp(-3*pi*abs(z)),y),y)
```

(%o11) $4\,\mathrm{pi}^2 e^{-3\mathrm{pi}|z|-2\mathrm{pi}|y|-\mathrm{pi}|x|}$

```
(%i12) diff(diff(exp(-pi*abs(x))*exp(-2*pi*abs(y))*exp(-3*pi*abs(z)),z),z)
```

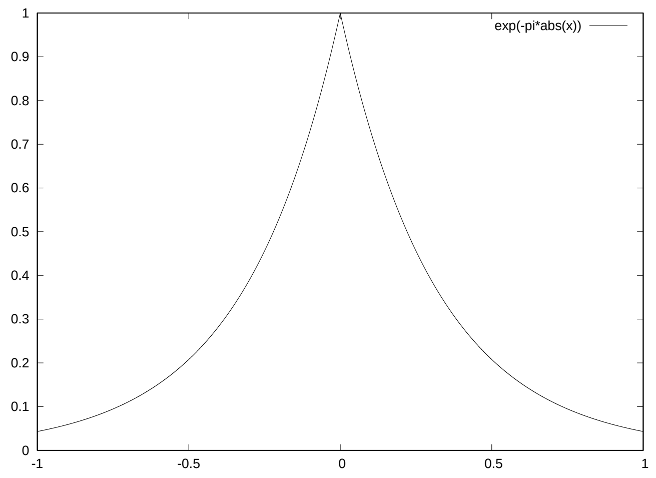
(%o12) $9\,\mathrm{pi}^2 e^{-3\mathrm{pi}|z|-2\mathrm{pi}|y|-\mathrm{pi}|x|}$

```
(%i13) exp(-pi*abs(x))*exp(-2*pi*abs(y))*exp(-3*pi*abs(z))
```

(%o13) $e^{-3\mathrm{pi}|z|-2\mathrm{pi}|y|-\mathrm{pi}|x|}$

```
(%i14)
```

```
   gnuplot 5.2 patchlevel 8
gnuplot] set sample 10000;set xrange [-1:1];plot exp(-pi*abs(x))
```



```
gnuplot]
```