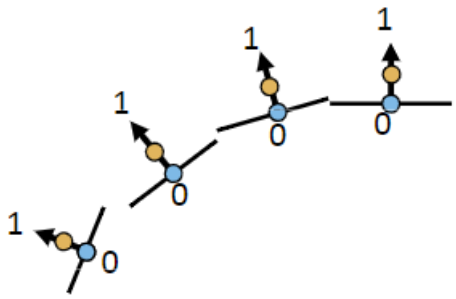
homework 10

本次作业实现了基本的RBF算法与泊松重建算法。

RBF Reconstruction

RBF Reconstruction是通过带法向的采样点来拟合一个隐式曲面出来，接着根据这个隐式方程通过Marching Cube算法来提取出表面的三角网格信息。RBF重建的思想非常简单，首先通过沿着法向构造额外点：



接着只要进行利用RBF函数插值求解就可以了。

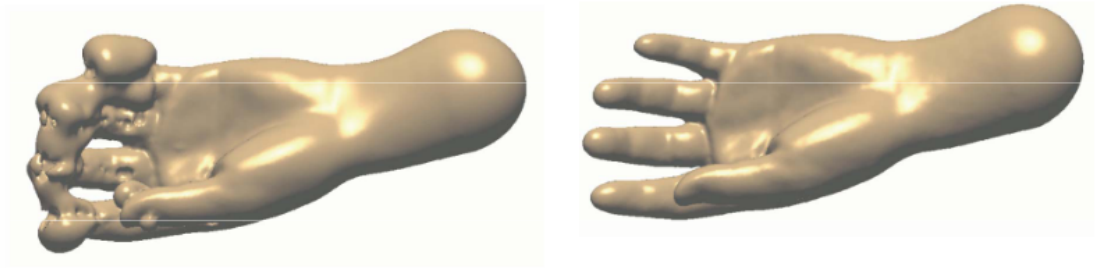
$$dist(\mathbf{p}_j) = \sum_i w_i \varphi_i(\mathbf{p}_j) = \sum_i w_i \varphi(\|\mathbf{p}_j - \mathbf{c}_i\|) = 0 \qquad 2n \text{ equations}$$

$$dist(\mathbf{p}_j + \varepsilon \mathbf{n}_j) = \sum_i w_i \varphi(\|(\mathbf{p}_j + \varepsilon \mathbf{n}_j) - \mathbf{c}_i\|) = \varepsilon \qquad 2n \text{ variables}$$

The on- and off-surface points are the centers $\mathbf{c}_i$

$$\begin{bmatrix} \phi(\|\mathbf{x}_1 - \mathbf{x}_1\|) & \cdots & \phi(\|\mathbf{x}_1 - (\mathbf{x}_n + \varepsilon \mathbf{n}_n)\|) \\ \vdots & \mathbf{K} & \vdots \\ \phi(\|(\mathbf{x}_n + \varepsilon \mathbf{n}_n) - \mathbf{x}_1\|) & \cdots & \phi(\|(\mathbf{x}_n + \varepsilon \mathbf{n}_n) - (\mathbf{x}_n + \varepsilon \mathbf{n}_n)\|) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ \mathbf{w} \\ \vdots \\ w_{2n} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \mathbf{d} \\ \varepsilon \\ \vdots \\ \varepsilon \end{bmatrix}$$

在实现过程中，发现对于ε的取值对重建结果也很重要，文章中也提到了这一点：



Without normal length validation

With normal length validation

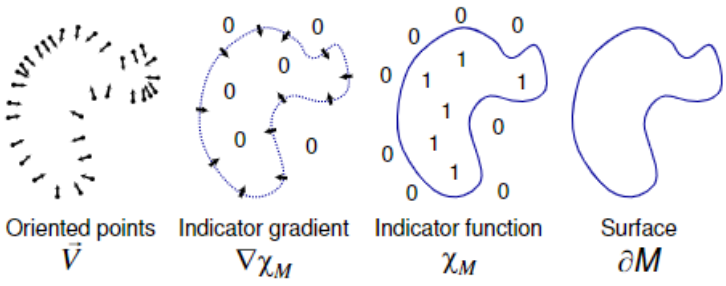
RBF的缺点很明显，它其实就是函数插值的缺点：

1. 点的个数多的时候，非常耗时且占用内存
2. 如果数据包含噪声，函数插值会震荡得非常厉害，也就是过拟合。

当然，针对这些也有一些改进的地方，例如Fast RBF使用了Greedy算法，来在错误比较大的地方逐步添加插值点的个数。但是它依然是比较简单的算法，无论是效果还是效率上都不如一些其他的算法。

Poisson Reconstruction

泊松重建与RBF类似的地方在于，他们都是拟合一个全局的隐式函数出来。RBF使用的函数是带符号的距离，函数值为点到曲面的距离，在曲面内部为负值，在曲面外部为正值；Poisson使用的是指示函数，在曲面内部的函数值为0，在曲面外部的函数值为1。仔细考虑一下这个指示函数，如果直接通过内外点线性插值，得到的数值是没有意义的。因为你在外部点不管离表面多远，函数值都是1，因此它们的关系不是线性可以得到的。直观上来讲，指示函数χ_M的梯度的方向应该与这些法向量的方向是一致的。因此，我们可以通过重建指示函数的梯度，进而得到指示函数，最后根据MarchingCube来提取出表面。整个算法的过程简单来说如下图：



但是如果仅仅通过上面的图是无法实现poisson重建的。首先，根据指示函数的定义，指示函数的在表面的梯度实际上是趋于无穷的，因为它不是一个光滑的函数，是从零直接变为一的。因此，我们需要引入一个光滑的操作，而通过散度定理可以证明，经过光滑后的指示函数的梯度，与光滑后的表面的法向量是一样的：

$$\nabla\left(\chi_M * \tilde{F}\right)(q_0) = \int_{\partial M} \tilde{F}_p(q_0)\,\vec{N}_{\partial M}(p)dp$$

上面的式子中，$\tilde{F}$是光滑函数，而$*$代表的是卷积操作，实际上也就是光滑操作。文章中定义了：

$$\tilde{F}_p(q) = \tilde{F}(q-p)$$

也就是经过一个平移操作候的光滑操作。由于样本点的离散性，并不是每一个点q的法向量都是已知的，因此我们需要对他进行分段的近似：

$$\begin{aligned}\nabla\left(\chi_M * \tilde{F}\right)(q) &= \sum_{s\in S}\int_{\mathscr{P}_s}\tilde{F}_p(q)\vec{N}_{\partial M}(p)dp \\ &\approx \sum_{s\in S}|\mathscr{P}_s|\,\tilde{F}_{s.p}(q)s.\vec{N} \equiv \vec{V}(q)\end{aligned}$$

上式中：$\mathscr{P}_s$为按照空间划分的样本点s的附近的表面，$s.p$为样本s的位置信息，而$s.\vec{N}$为样本位置的法向量。从这里也就可以看出来，一个点q的法向量是通过多个样本点的法向量加权相加得到的。这里的$\tilde{F}$是前面说到的光滑函数，也就是盒式滤波器。如果该点与s.p离得很近，那么也就是$\tilde{F}(0)$，权重很大，如果离得远权重就越小。实际上，这个就是以q为中心的一个卷积操作。如果样本是均匀的，实际上常数$|\mathscr{P}_s|$是可以忽略的。

在实际的实现中，作者将利用octree来进行空间的分割。Octree会对比较密的地方进行更高精度的划分，从而提高效率。对Octree $\mathscr{O}$的每个结点$o\in\mathscr{O}$，文章中会定义一个node function：

$$F_o(q) \equiv F\left(\frac{q-o.c}{o.w}\right)\frac{1}{o.w^3}$$

这里$o.w$是这个结点的宽度，而$o.c$是这个结点的中心，而

$$F(q) = \tilde{F}\left(\frac{q}{2^D}\right)$$

因此这个定义实际上是在不同的结点中，平滑操作会有所不同，从定义可以看出来，离结点中心越近权重是越高的。而由于上述函数中除了一个2^D，因此整个定义域的范围就被局限在半径为0.5的一个球内部了，这也就是三维情况下的高斯分布，最后将所有结点的函数张为函数空间：

$$\mathscr{F}_{\theta,F} := \mathrm{Span}\left\{F_o\right\}$$

这样也就可以得到一个$|\mathscr{O}|$维的基函数向量$\vec{F}$。文章中使用的光滑函数是高斯函数，在离散情况下，为了提高效率，比如图片中，高斯光滑一般用filter来近似，也就是高斯滤波，而这里用的是类似的方法，在文章中被称为高斯盒式滤波（box filter）：

$$\tilde{F}(x,y,z) \equiv (B(x)B(y)B(z))^{*n} \quad \text{with} \quad B(t) = \begin{cases} 1 & |t| < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

每次滤波可以看到是在0.5范围内取1，超出部分取0，三次连续这样的滤波器组合成的滤波器最终的范围会扩大到 1.5，因此滤波范围为：$[-1.5, 1.5]$。这时候盒式滤波其扩大到了$3\times3\times3$个voxel，这样也就可以理解文章中很令人费解的一句话：

> for the basis function of any octree node,there are at most 53-1=124 other nodes at the same depthwhose functions overlap with it.

关于高斯滤波可以查看：http://nghiaho.com/?p=1159。 所以虽然上面讨论了很长时间，光滑操作实际上就是一个高斯盒子滤波。

到这里平滑操作就定义好了，每个非样本点的法向量也就可以通过结点上定义的基函数来得到。为了得到更准确的结点法向量，文章使用了线性插值的方法来得到法向量：

$$\vec{V}(q) \equiv \sum_{s \in S} \sum_{o \in \mathrm{Ngbr}_D(s)} \alpha_{o,s} F_o(q) s \cdot \vec{N}$$

这也就是离散情况下求得法向量场的操作了。

其实上述式子可以通过先对每个样本点进行splat操作，也就是计算每个样本点在最近8个邻域结点中的权重向量，每个结点累计计算得到的对应的权重向量，最后得到每个结点的权重向量$\vec{N}_{weighted}$，最后计算每个结点的向量，只需要对基函数加权相加即可：

$$\vec{V}(o') = \sum_{o \in O} F_o(o'.c) \vec{N}_{weight}$$

也就是通过基函数将向量进行了表达。

求解Poisson方程

本来求得了向量场后，最简单的想法是直接通过积分就可以得到指示函数，但是现实却不是那么容易。因为向量场不一定是可积的，因此无法得到精确解，进而转化为求最小二乘误差的近似解：

$$\Delta \tilde{\chi} = \nabla \cdot \vec{V}$$

这里$\Delta \tilde{\chi}$指的是指示函数的拉普拉斯操作，而$\nabla \cdot \vec{V}$是对法向量求divergence。现在最后一个问题是，即使指示函数的梯度以及法向量是在函数空间$\mathscr{F}_{\theta,F}$内部的，但是$\Delta \tilde{\chi}$与$\nabla \cdot \vec{V}$却不一定在。为了解决这个问题，简单的寻找这两个变量在函数空间的投影即可，最后求解泊松方程就被近似为求解：

$$\sum_{o \in \mathscr{O}} \left\| \left\langle \Delta \tilde{\chi} - \nabla \cdot \vec{V}, F_o \right\rangle \right\|^2 = \sum_{o \in \mathscr{O}} \left\| \langle \Delta \tilde{\chi}, F_o \rangle - \left\langle \nabla \cdot \vec{V}, F_o \right\rangle \right\|^2$$

为了在矩阵形式上进行表达，方便求解，首先我们使用基函数来表示指示函数$\tilde{\chi} = \Sigma_o x_o F_o$，因此我们求解的实际上是指示函数在基函数上的权重$\vec{x}$。接着定义矩阵一个$|\mathscr{O}| \times |\mathscr{O}|$维度的矩阵$L$，而矩阵的值为：

$$L_{o,o'} \equiv \left\langle \frac{\partial^2 F_o}{\partial x^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial y^2}, F_{o'} \right\rangle + \left\langle \frac{\partial^2 F_o}{\partial z^2}, F_{o'} \right\rangle .$$

求解泊松方程也就变成了：

$$\min_{x \in \mathbb{R}^{|\mathscr{O}|}} \|L\vec{x} - v\|^2$$

具体实现

法向量估计

法向量的估计主要是通过kdtree找到最近点，接着平面拟合得到法向量，再通过最小生成树来得到法向量的朝向。

Marching Cube

对于MarchingCube算法，我使用了之前实现过的版本，只做了些许改动，用来适应当前的算法。代码来自于：https://github.com/MyEvolution/OnePiece/tree/master/src/Integration

RBF

对于RBF我只实现了最基本的方法，而没有实现Fast RBF，因此大大限制了我可以处理的点的个数。因此在实现中，对于输入点云，我会进行一个scale，将它放大或者缩小到一定范围，接着对点云进行法向量估计，最后进行下采样，得到一个相对稀疏的点云（一般点的个数在500到1500左右）。接着对这样的稀疏点云进行RBF插值，最后通过marching cube算法提取表面。

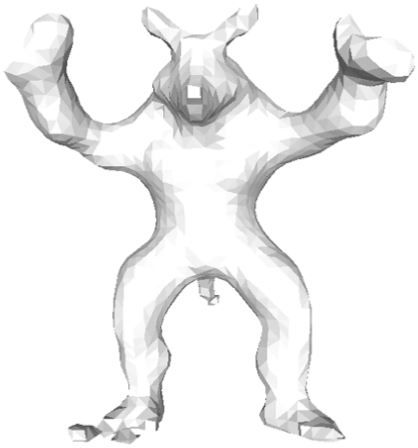
Poisson

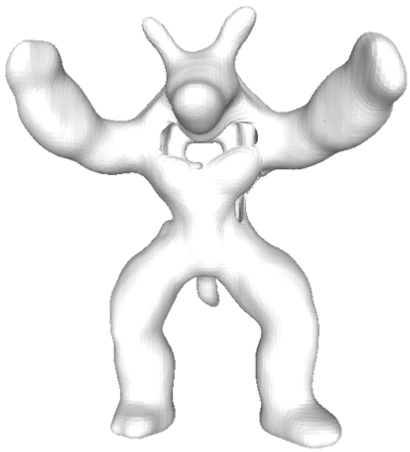
Poisson的实现基本是照着我对文章的理解实现做下来的，实际上这个算法还是相当复杂的。最后网格提取依然使用了体素的marching cube，但是这时候的体素需要小于八叉树的最小结点，而没有针对八叉树进行优化，因此在这里对体素进行indicator value的计算以及marching cube是非常慢的。目前的实现效果不如官方版本，效率也不如，我没有来得及做任何性能上的优化，虽然它包含了大量可以并行的操作。对于原文的一些细节我还有一些疑惑，我本打算给原作者发一封邮件咨询，因此用英语描述了自己算法的具体实现过程，现在没有足够的时间翻译成中文，所以我就把这部分放到作业中了，在结果展示的之后附录中。

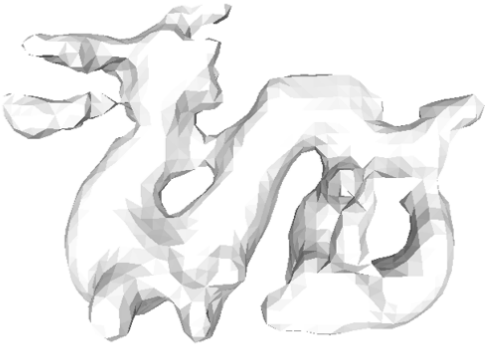
所有代码上传在https://github.com/MyEvolution/Dragon.

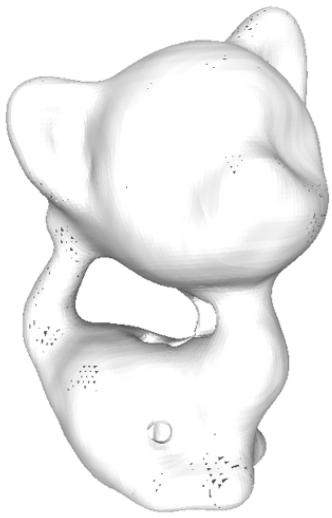
结果

RBF与Poisson对比（Poisson的八叉树深度为6）

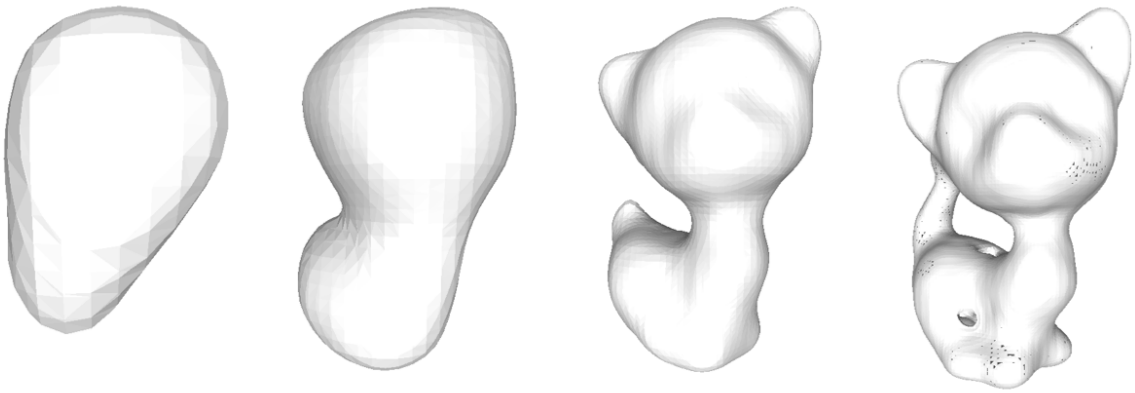






可以看到的是Poisson是逼近，而RBF实际上是经过给定点的（目前实现的poisson还有点问题）。随着poisson树的深度加大时候，表现力会更强，如下图（树的深度为3，4，5，6）：



更多内容可以查看附录的视频。

附录：Implementation of Poisson Surface Reconstruction

Build Octree

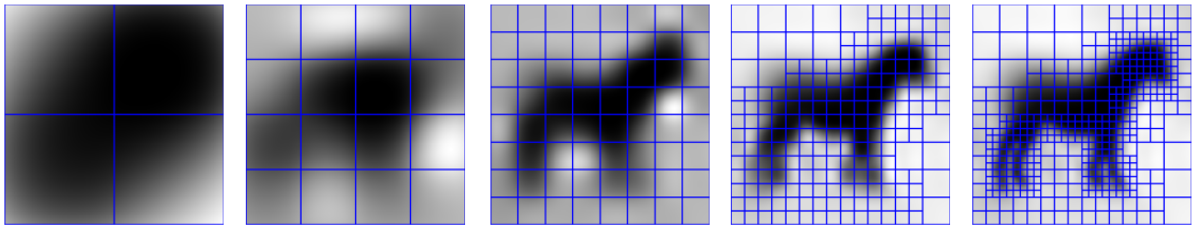
In my implementation, firstly i will compute the boundingbox of input point cloud. Then the root node o_{root} of octree $\mathcal{O}$ will be built:

$$o_{root}.w = 1.5 * \max(dist(x), \max(dist(y), dist(z)))$$

$dist(x), dist(y), dist(z)$ are the length of x-range, y-range and z-range of the bounding box. The center of root node o_{root} will be the center of bounding box. The max depth is denoted as D.

Node function

We solve the problem iterately. And in each iteration d, we only process the node with depth equals to d. I would use $\mathcal{O}$ to represent all the nodes whose depth is d.



I define a node function for each node. Each node function F_o is defined based on the following fomulation:

$$F_o(q) \equiv F\left(\frac{q - o.c}{o.w}\right)\frac{1}{o.w^3}$$

while

$$F(x,y,z) \equiv (B(x)B(y)B(z))^{*n} \quad \text{with} \quad B(t) = \begin{cases} 1 & |t| < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

This is the same as that depicted in paper. If I am not misunderstanding , each leaf node has a different node function. The discrete form of node function F_{o_i} can be written as:

$$F_{o_i} = \begin{bmatrix} F_{o_i}(o_1.c) & F_{o_i}(o_2.c) & \cdots & F_{o_i}(o_n.c) \end{bmatrix}^\top$$

So, the spanned function space is:

$$\mathbf{F} = \begin{bmatrix} F_{o_1}(o_1.c) & F_{o_2}(o_1.c) & \cdots & F_{o_n}(o_1.c) \\ F_{o_1}(o_2.c) & F_{o_2}(o_2.c) & \cdots & F_{o_2}(o_2.c) \\ \vdots & \vdots & \ddots & \vdots \\ F_{o_1}(o_n.c) & F_{o_2}(o_n.c) & \cdots & F_{o_n}(o_n.c) \end{bmatrix} = \begin{bmatrix} F_{o_1} & F_{o_2} & \cdots & F_{o_n} \end{bmatrix}$$

And because we only proccess the node with same width as this way, so this function space is **symmetric.**

Each column of $\mathbf{F}$ is the dicrete form of a node function, and each row is the function values of current node center in different node functions.

Further, the vector field (I only compute the vector fields of leaf node centers) can be computed through

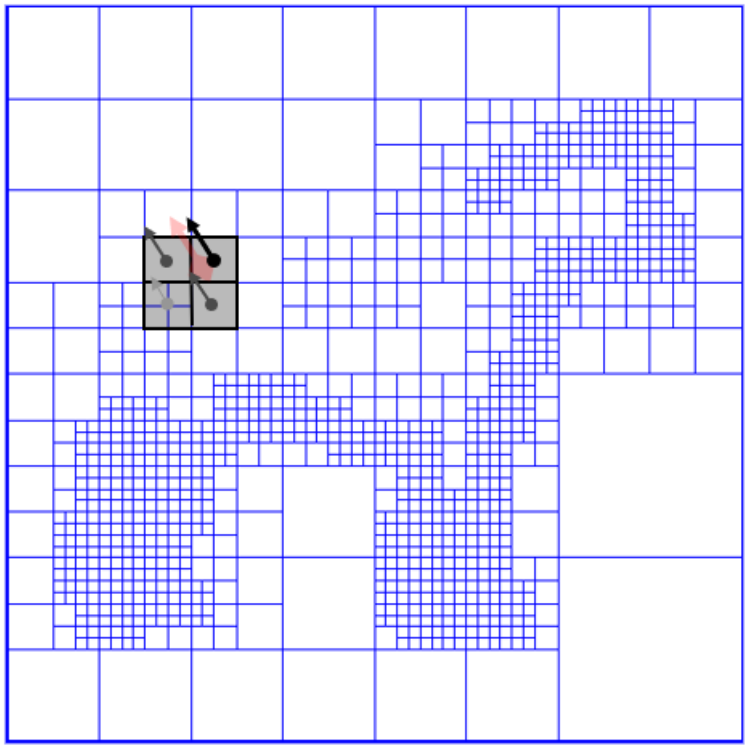
$$\vec{V}(o_i) \equiv \sum_{s \in S} \sum_{o \in \text{Ngbr}_D(s)} \alpha_{o,s} F_o(o_i.c) s \cdot \vec{N}$$

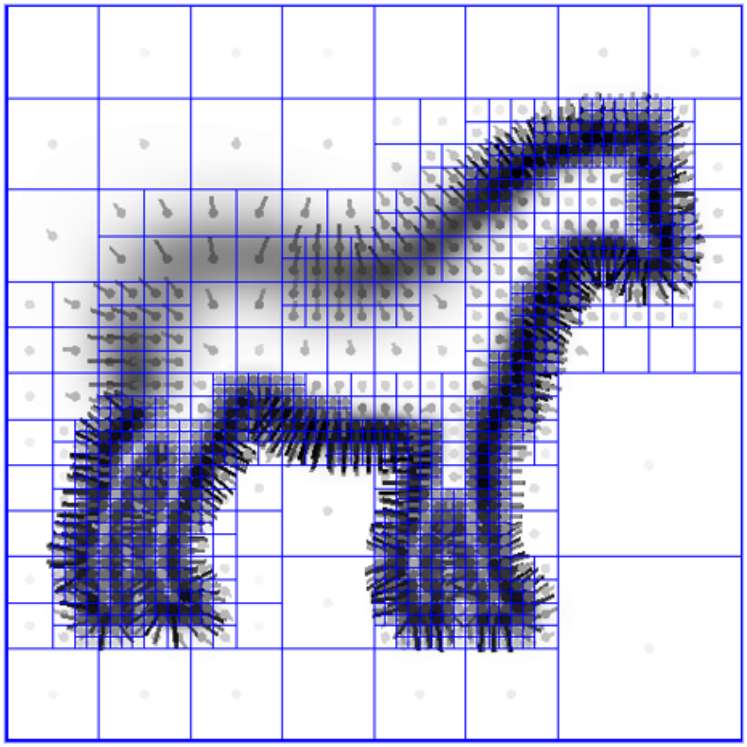
Choose 8 closest neighbor is really important, or the normal interpolation is not accurate, and this could lead to bad result. Acturally, these can be down by firstly splating the sample point, than each leaf node will have a accumulated weighted normal denoted as $N_{w,o}$, then the vector field of each leaf node can be computed as:

$$\vec{V}(o_i) = \sum_{o \in \mathcal{O}_d} F_o(o_i.c) N_{w,o_i} = \mathbf{F} \cdot \mathbf{N}_w$$

while is a $\mathbf{N}_w = \begin{bmatrix} N_{w,o_1} & N_{w,o_2} & \cdots N_{w,o_n} \end{bmatrix}^T$

Following figure shows the process (this figure shows a non-uniform splating, however, in my implementation, i only use uniform splating):





And the whole vector vield can be denoted as a matrix:

$$\mathbf{V} = \begin{bmatrix} \vec{V}(o_1) & \vec{V}(o_2) & \cdots \vec{V}(o_n) \end{bmatrix}$$

Poisson Equation

Further, I compute the partial derivatives as:

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial F_{o_1}(o_1.c)}{\partial x} & \frac{\partial F_{o_2}(o_1.c)}{\partial x} & \cdots & \frac{\partial F_{o_n}(o_1.c)}{\partial x} \\ \frac{\partial F_{o_1}(o_2.c)}{\partial x} & \frac{\partial F_{o_2}(o_2.c)}{\partial x} & \cdots & \frac{\partial F_{o_n}(o_2.c)}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_{o_1}(o_n.c)}{\partial x} & \frac{\partial F_{o_2}(o_n.c)}{\partial x} & \cdots & \frac{\partial F_{o_n}(o_n.c)}{\partial x} \end{bmatrix}$$

Also, I can get $\frac{\partial \mathbf{F}}{\partial \mathbf{y}}, \frac{\partial \mathbf{F}}{\partial \mathbf{z}}$.

I could also compute the laplacian operation as:

$$\Delta \mathbf{F}_x = \begin{bmatrix} \frac{\partial^2 F_{o_1}(o_1.c)}{\partial^2 x} & \frac{\partial^2 F_{o_2}(o_1.c)}{\partial^2 x} & \cdots & \frac{\partial^2 F_{o_n}(o_1.c)}{\partial^2 x} \\ \frac{\partial^2 F_{o_1}(o_2.c)}{\partial^2 x} & \frac{\partial^2 F_{o_2}(o_2.c)}{\partial^2 x} & \cdots & \frac{\partial^2 F_{o_n}(o_2.c)}{\partial^2 x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F_{o_1}(o_n.c)}{\partial^2 x} & \frac{\partial^2 F_{o_2}(o_n.c)}{\partial^2 x} & \cdots & \frac{\partial^2 F_{o_n}(o_n.c)}{\partial^2 x} \end{bmatrix}$$

Also I can get the $\Delta \mathbf{F}_y, \Delta \mathbf{F}_z$.

The divergence of vector field, for leaf node o_i, can be computed as:

$$\begin{aligned}\mathrm{div}\vec{V}(o_i) &= \left[\frac{\partial \vec{V}(o_i)(0)}{\partial x} \quad \frac{\partial \vec{V}(o_i)(1)}{\partial y} \quad \frac{\partial \vec{V}(o_i)(1)}{\partial z}\right] \\ &= \left[\sum_{o\in\mathcal{O}_d}\frac{\partial F_o(o_i.c)}{\partial x}\cdot N_{w,o_i}(0) \quad \sum_{o\in\mathcal{O}_d}\frac{\partial F_o(o_i.c)}{\partial y}\cdot N_{w,o_i}(1) \quad \sum_{o\in\mathcal{O}_d}\frac{\partial F_o(o_i.c)}{\partial z}\cdot N_{w,o_i}(2)\right]\end{aligned}$$

And it would be easy to get the divergence of $\mathbf{V}$:

$$\begin{aligned}\mathrm{div}\mathbf{V}_x &= \frac{\boldsymbol{\partial}\mathbf{F}}{\boldsymbol{\partial}\mathbf{x}}\cdot\mathbf{N}_w(0) \\ \mathrm{div}\mathbf{V}_y &= \frac{\boldsymbol{\partial}\mathbf{F}}{\boldsymbol{\partial}\mathbf{x}}\cdot\mathbf{N}_w(1) \\ \mathrm{div}\mathbf{V}_z &= \frac{\boldsymbol{\partial}\mathbf{F}}{\boldsymbol{\partial}\mathbf{x}}\cdot\mathbf{N}_w(2)\end{aligned}$$

Here, $\mathbf{N}_w(i)$ means the i-th column of matrix $\mathbf{N}_w$.

If we project the divergence onto function space $\mathbf{F}$, we can get:

$$\mathrm{div}\mathbf{V}_{\mathbf{F}} = \left[\mathbf{F}^\top\cdot\mathrm{div}\mathbf{V}_x \quad \mathbf{F}^\top\cdot\mathrm{div}\mathbf{V}_y \quad \mathbf{F}^\top\cdot\mathrm{div}\mathbf{V}_z\right]$$

And $\mathrm{div}\mathbf{V}_{\mathbf{F}}$ is a $|\mathcal{O}_d|\times 3$ matrix.

We use the linear combination of node functions to estimate the indicator function $\mathcal{X}$. Here, each indicator function can be written as:

$$\mathcal{X}(q) = \sum_{o\in\mathcal{O}_d} F_o(q)x_o$$

$\mathbf{X} = \left[x_{o_1} \quad x_{o_2} \quad \cdots \quad x_{o_n}\right]$ is a basis vector of indicator function.

The discrete form of indicator function is (we only compute the indicator value of the center of each leaf node):

$$\vec{\mathcal{X}} = \left[\mathcal{X}(o_1.c) \quad \mathcal{X}(o_2.c) \quad \cdots \quad \mathcal{X}(o_n.c)\right]^\top$$

After the laplacian operation, we can get:

$$\begin{aligned}\Delta\vec{\mathcal{X}}_x &= \Delta\mathbf{F}_x\cdot\mathbf{X} \\ \Delta\vec{\mathcal{X}}_y &= \Delta\mathbf{F}_y\cdot\mathbf{X} \\ \Delta\vec{\mathcal{X}}_z &= \Delta\mathbf{F}_z\cdot\mathbf{X}\end{aligned}$$

Project these value to function space $\mathbf{F}$:

$$\begin{aligned}\Delta\vec{\mathcal{X}}_{x,\mathbf{F}} &= \mathbf{F}^\top\cdot\Delta\mathbf{F}_x\cdot\mathbf{X} \\ \Delta\vec{\mathcal{X}}_{y,\mathbf{F}} &= \mathbf{F}^\top\cdot\Delta\mathbf{F}_y\cdot\mathbf{X} \\ \Delta\vec{\mathcal{X}}_{z,\mathbf{F}} &= \mathbf{F}^\top\cdot\Delta\mathbf{F}_z\cdot\mathbf{X}\end{aligned}$$

And the L matrix in paper is computed by adding these three matrix:

$$L = \Delta\vec{\mathcal{X}}_{x,\mathbf{F}} + \Delta\vec{\mathcal{X}}_{y,\mathbf{F}} + \Delta\vec{\mathcal{X}}_{z,\mathbf{F}}$$

Also,

$$v = F^\top \cdot \mathrm{div}\mathbf{V}_x + F^\top \cdot \mathrm{div}\mathbf{V}_y + F^\top \cdot \mathrm{div}\mathbf{V}_z$$

Finally, i solve the equation

$$\min_{x \in \mathbb{R}^{|\mathcal{O}|}} \|L\mathbf{X} - v\|^2$$

By using SparseLu decomposition of L.

ISO extraction

The indicate value of sample q is computed by $\mathcal{X}(q) = \sum_{o \in \mathcal{O}} F_o(q) x_o$, and the iso value is chose as:

$$\gamma = \frac{1}{|S|} \sum_{s \in S} \tilde{\chi}(s \cdot p)$$

Should mention that in this step, the we use all the nodes, not nodes who have the same depth. And I got some results like this:

The result is quiet strange. It's reasonable, however it's not completely correct. There are several holes (the computed surface turned inward the model in some areas, seems like the vector filed problem) on the model, and if the resolution is higher, the holes are more frequent to occur. I noticed that I am just compute the vector field using the uniform sampling strategy, so could this be the problem? Or there are some misunderstanding on some conception. Also i have not implemented octree based marching cube, instead i am using voxel based marching cube. So the process is quiet time consuming.

Also it's also a little confusing that how to solving the problem iterately, i think my implementation is not reasonable, because the $d+1$-layer did not actually using the d-layer's information.