SURFACE RECONSTRUCTION USING BOUNDARY INTEGRAL EQUATIONS

VICTOR RONG

Abstract. Reconstructing an implicit 3D shape representation from a set of point samples and their estimated oriented normals is a key subproblem of the surface reconstruction problem. The indicator function of the shape is a common choice for the implicit representation as the oriented normals are exactly its inward gradient at the boundary and thus the problem of solving for the indicator function can be interpreted as a Poisson equation. We solve this using an explicit integral equation and we explore using the Barnes-Hut algorithm to efficiently perform the evaluation. In order to apply the Barnes-Hut algorithm, we derive the multipole expansion for a novel kernel. Our implementation achieves better runtimes than the naive approach while incurring negligible error. However, we find that this numerical solution does not properly generate the 3D shape.

Key words. surface reconstruction, boundary integral equations, fast multipole method

AMS subject classifications. 65D18, 68U05

1. Introduction.

When raw 3D data is taken, it is in the form of point samples such as depth maps or LiDAR scans. Surface reconstruction is the problem of taking such data and converting it into more convenient representations such as triangle meshes. As it is important in a number of applications, many works have set out to find an accurate and efficient method. One of the most successful approaches in terms of both accuracy and runtime is the Poisson Surface Reconstruction algorithm [7, 8] which solves for the indicator function χ given a point cloud with oriented normals. In particular,

$$\nabla\chi = -\vec{n}\mid_{\partial\Omega}$$

where $\vec{n}$ is the vector field induced by the outward normals and $\partial\Omega$ is the surface of the solid which the points are sampled from. It can be shown using variational methods that the minimization problem $\min_\chi ||\nabla\chi + \vec{n}||$ is equivalent to the Poisson problem

$$\nabla^2\chi = -\nabla\cdot\vec{n}$$

and so we have transformed this graphics problem into a PDE problem. There are many techniques for dealing with such an equation. However, most are reliant on a regular grid domain while the point samples can be located arbitrarily. Instead, following [11], we use an explicit solution to this Poisson problem which can be derived from the Gauss lemma [13]. This solution is in the form of a boundary integral equation

$$\chi(x) = \int_{\partial\Omega} K(x,y)dy$$

for kernel function to be introduced. As we only have discrete point samples, this integral is discretized into a weighted sum.

Fast evaluation of such convolutions has been developed for physics applications, particularly that of the n-body gravitational potential problem. Given n point samples (acting as potential sources) and n evaluation points (acting as targets), the Fast Multipole Method provides evaluations in $O(n\log n)$ which can be further improved to $O(n)$ [5]. This algorithm operates by using the multipole expansion of the kernel function being evaluated. Early works only derived said formulas for a small, but common, set of kernel functions but it was later generalized to black-box kernels [14]. Our work uses a kernel which was not part of the small set. As such, we derive an expansion for this novel kernel. Using this expansion, we implement the $O(n \log n)$ Barnes-Hut algorithm, a simpler version of the FMM.

In summary, our contributions are as follows:

- We reduce the original kernel to a different, radially symmetric kernel and derive its multipole expansion with guarantees on the error.
- We extend this derivation to find the multipole expansion of a class of kernels.
- We provide a Python implementation for surface reconstruction in $O(n \log n)$ time using the Barnes-Hut algorithm.

2. Related Work.

Surface reconstruction. Despite the challenging nature of the problem, there have been a wide array of approaches to solving the surface reconstruction problem. The goal of these methods is to improve the accuracy of the reconstruction, the runtime which the it requires, and its robustness to noisy or partial data. Many early works approached the problem from a combinatorial perspective, focusing on drawing edge connections directly from the point cloud vertices. For example, Amental etal [1] use the Voronoi diagram of the point cloud to establish connectivity, and Bernardini etal [3] find triangles of the mesh by pivoting balls of various sizes about the points. Due to the local nature of these methods, they have trouble against noisy data. Implicit methods, on the other hand, are often more robust to noise while also demonstrating good speed and accuracy. These methods construct an implicit representation of the data and then apply the Marching Cubes algorithm to obtain a triangle mesh from occupancy samples [10]. Generally, either the signed distance function (SDF) or indicator function is used as they are simple and provide occupancy information. One of the earliest such methods [6], estimates the SDF at a set of target points. They find that the orientations of the normals are particularly important and develop a technique for reorienting them. Another work [4] represents the signed distance function with radial basis functions (RBFs). Like our method, this work uses FMM-based techniques to efficiently evaluate the RBFs. As mentioned earlier, Kazhdan etal. [7, 8] instead considers the indicator function χ and constructs the PDE $\nabla^2\chi = -\nabla \cdot \vec{n}$. They represent the indicator function in a basis of multiresolution functions with finite support and solve the problem as a linear system, using the positive definiteness of the Laplacian to efficiently invert the operator. Lu etal [11] also uses the Poisson formulation and instead considers the explicit solution based on integral equations. They propose a subquadratic algorithm based on the FMM for evaluating the integral, but their method gives no reason to expect low error relative to the naive evaluation method. Our work extends that of [11] and is able to achieve the same time complexity while having guaranteed bounds on the error.

Fast Multipole Methods. It is famously known that the n-body problem has no closed form solution in general, but the model can instead be simulated. As there are $O(n^2)$ interactions between them, we can directly compute the potentials on each body in $O(n^2)$. This can become impractical for large n and so algorithms have been developed to speed up the evaluation with some controllable error that can be made to hit machine precision. In an abuse of terminology, we will refer to any such method as a fast multipole method, though we note that many use this term to refer to a specific algorithm [5]. Rokhlin [12] was the first to break through the $O(n^2)$ algorithm, creating an $O(n)$ method for evaluating the potential in the 2D domain. The crucial observation was the interactions between two far clusters can be approximated well. This method derived a separable multipole expansion to the kernel as well as formulas for translating the expansions. Around the same time, an $O(n \log n)$ algorithm for evaluating the potential in 3D was published by Barnes and Hut, now called the Barnes-Hut algorithm [2]. This method also leveraged the clustering idea and used an octree structure to specify $O(n \log n)$ pairs of clusters. Due to the different natures of the 2D and 3D potentials, it was challenging to reach $O(n)$ in 3D as well. A decade after the Barnes-Hut algorithm was published, Greengard and Rokhlin [5] were finally able to do so. Their algorithm requires five operators on the multipole expansion of the kernel: source-to-multipole, multipole-to-multipole, multipole-to-local, local-to-local, and local-to-target [9]. In comparison, the Barnes-Hut algorithm only needs the source-to-multipole operator to achieve numerical accuracy at the cost of an extra $\log n$ factor in the time complexity. As it is cumbersome to derive each of these operators for any such kernel, Ying etal [14] proposed a black box method to approximate these operators.

3. Multipole Expansion. In this section, we specify the kernel we are working with and derive its multipole expansion for later use in the algorithm.

LEMMA 3.1 (Gauss Lemma [13, 11]). *Let Ω be an open region in $\mathbb{R}^3$ and let $\overline{\Omega}$ and $\partial\Omega$ be its closure and boundary, respectively. Let $\chi : \mathbb{R}^3 \to \mathbb{R}$ be such that*

$$\chi(x) = \int_{\partial\Omega} \frac{\partial G}{\partial \vec{n}(y)}(x, y)dy$$

for any $x \in \mathbb{R}^3$ and G the Green's function for the Laplace equation. Then χ is exactly the indicator function of Ω.

So the kernel $K(x, y) = \frac{\partial G}{\partial \vec{n}(y)}(x, y)$. In particular,

$$G(x, y) = -\frac{1}{4\pi}\frac{1}{||x - y||} \implies \frac{\partial G}{\partial \vec{n}(y)}(x, y) = -\frac{1}{4\pi}\frac{(x - y) \cdot \vec{n}(y)}{||x - y||^3}.$$

Note that Lu etal [11] set the kernel to zero when $||x-y||$ is sufficiently small in order to remove the singularities, but we allow for these to persist.

Instead of directly finding the multipole expansion of $K(x, y)$, we will work with a new radially symmetric kernel defined as $K_0(x, y) := -\frac{1}{4\pi}\frac{1}{||x-y||^3}$. Then

$$\begin{aligned} K(x, y) &= -\frac{1}{4\pi}\frac{(x - y) \cdot \vec{n(y)}}{||x - y||^3} \\ &= (x - y) \cdot \vec{n(y)} K_0(x, y) \\ &= (x \cdot n(y))\, K_0(x, y) - (y \cdot n(y))\, K_0(x, y). \end{aligned}$$

Before we can give our results on the multipole expansion of $K_0(x, y)$, we must establish a few lemmas on spherical harmonics from [5]. We use the convention

$$Y_n^m(\theta, \phi) := \sqrt{\frac{2n + 1}{4\pi}\frac{(n - m)!}{(n + m)!}} P_n^m(\cos(\theta))e^{im\phi},$$

where P_n^m are the associated Legendre polynomials. Also, Y_n^{m*} denotes complex conjugation and is equal to $(-1)^m Y_n^{-m}$.

Lemma 3.2 (Generating Function of Legendre Polynomials). *For $\mu < 1$,*

$$\frac{1}{\sqrt{1-2u\mu+\mu^2}} = \sum_{n=0}^{\infty} P_n(u)\mu^n$$

where $P_n(u)$ are the Legendre polynomials.

Lemma 3.3 (Derivatives of Legendre Polynomials). *For any $n \geq 1$,*

$$(2n+1)P_n(u) = P'_{n+1}(u) - P'_{n-1}(u).$$

Lemma 3.4 (Addition Theorem). *Let $x, y \in \mathbb{R}^3$ have spherical coordinates (r, θ, ϕ) and (ρ, α, β). Let γ be the angle between x and y at the origin. Then*

$$P_n(\cos(\gamma)) = \frac{4\pi}{2n+1} \sum_{m=-n}^{n} Y_n^{m*}(\alpha, \beta) Y_n^m(\theta, \phi).$$

With these in hand, we can now derive the multipole expansion of $K_0(x, y)$.

Theorem 3.5 (Multipole Expansion). *Let $x, y \in \mathbb{R}^3$ have spherical coordinates (r, θ, π) and (ρ, α, β) with $\rho < r$. Then*

$$-\frac{1}{4\pi}\frac{1}{||x-y||^3} = -\sum_{t=1}^{\infty} \frac{\rho^{t-1}}{r^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} Y_n^{m*}(\alpha, \beta) Y_n^m(\theta, \phi).$$

Furthermore, for any $p \in \mathbb{N}$,

$$\left| -\frac{1}{4\pi}\frac{1}{||x-y||^3} + \sum_{t=1}^{p} \frac{\rho^{t-1}}{r^{t+2}} (\cdots) \right| \leq O\left(\frac{1}{(r-\rho)^3} \left(\frac{\rho}{r}\right)^p \right).$$

Proof. Let γ be the angle between x and y at the origin. By the Law of Cosines,

$$\begin{aligned} ||x-y|| &= \sqrt{r^2 - 2r\rho\cos\gamma + \rho^2} \\ &= r\sqrt{1 - 2\left(\frac{\rho}{r}\right)\cos\gamma + \left(\frac{\rho}{r}\right)^2} \end{aligned}$$

Let $\mu = \frac{\rho}{r} < 1$ and $u = \cos\gamma$. Then we have

$$\begin{aligned} K_0(x, y) &= -\frac{1}{4\pi}\frac{1}{||x-y||^3} \\ &= -\frac{1}{4\pi}\frac{1}{r^3\sqrt{1-2u\mu+\mu^2}^3} \end{aligned}$$

Note from Lemma 3.2 that

$$\begin{aligned} \frac{1}{\sqrt{1-2u\mu+\mu^2}^3} &= \frac{1}{\mu}\frac{d\frac{1}{\sqrt{1-2u\mu+\mu^2}}}{du} \\ &= \sum_{t=1}^{\infty} P'_t(u)\mu^{t-1}. \end{aligned}$$

From Lemma 3.3, we have that $P_t'(u) = (2t-1)P_{t-1}(u) + (2t-3)P_{t-3}(u) + \dots$. Using Lemma 3.4, this can be written in a separate manner with spherical harmonics.

$$\begin{aligned}
P_t'(u) &= (2t-1)P_{t-1}(u) + (2t-3)P_{t-3}(u) + \dots \\
&= \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} (2n+1)P_n(u) \\
&= 4\pi \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} Y_n^{m*}(\alpha,\beta) Y_n^m(\theta,\phi).
\end{aligned}$$

Plugging this in, we get the desired expansion:

$$\begin{aligned}
K_0(x,y) &= -\frac{1}{4\pi}\frac{1}{||x-y||^3} \\
&= -\frac{1}{4\pi}\frac{1}{r^3\sqrt{1-2u\mu+\mu^2}^3} \\
&= -\frac{1}{4\pi}\frac{1}{r^3}\sum_{t=1}^{\infty} P_t'(u)\mu^{t-1} \\
&= -\frac{1}{4\pi}\sum_{t=1}^{\infty}\frac{\rho^{t-1}}{r^{t+2}}P_t'(u) \\
&= -\sum_{t=1}^{\infty}\frac{\rho^{t-1}}{r^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} Y_n^{m*}(\alpha,\beta) Y_n^m(\theta,\phi).
\end{aligned}$$

To show the convergence results, note that for $u \in [-1,1]$, we have $|P_n'(u)| \leq t^2$. Then the tail sum is

$$\begin{aligned}
\left|\sum_{t=p+1}^{\infty} P_t'(u)\frac{\rho^{t-1}}{r^{t+2}}\right| &\leq \sum_{t=p+1}^{\infty}\left|P_t'(u)\frac{\rho^{t-1}}{r^{t+2}}\right| \\
&\leq \sum_{t=p+1}^{\infty} t^2\frac{\rho^{t-1}}{r^{t+2}} \\
&= O\left(\frac{1}{r^3}\left(\frac{1}{1-\frac{\rho}{r}}\right)^3\left(\frac{\rho}{r}\right)^p\right) \\
&= O\left(\frac{1}{(r-\rho)^3}\left(\frac{\rho}{r}\right)^p\right)
\end{aligned}$$

□

THEOREM 3.6 (Multipole Expansion). *Let $x, y \in \mathbb{R}^3$ have spherical coordinates (r,θ,π) and (ρ,α,β) with $\rho < r$. Then for any $j \in \mathbb{N}$, there exist coefficients $C_{tnm} \in \mathbb{R}$ such that*

$$\frac{1}{||x-y||^{2j+1}} = \sum_{t=j}^{\infty}\frac{\rho^{t-j}}{r^{t+j+1}}\sum_{n=0}^{t-j}\sum_{m=-n}^{n} C_{tnm} Y_n^{m*}(\alpha,\beta) Y_n^m(\theta,\phi).$$

Note that the truncation of the first p terms can be evaluated in $O(p^3)$ operations assuming that the spherical harmonics can be evaluated in $O(1)$. We omit the proof of this generalization as it is similar to that of Theorem 3.5 and not relevant to our problem.

4. Algorithm. We now have all the tools necessary to perform the Barnes-Hut algorithm in this setting. A high level idea of the algorithm is shown in Algorithm 4.1 with further details below.

Algorithm 4.1 Barnes-Hut algorithm

```
Targets {x_i}_{1≤i≤n}, sources {y_j}_{1≤j≤n} with normals {n_j}_{1≤j≤n}
Build octree O from {x_i} ∪ {y_j}
for node o ∈ O do
   Build near neighbour list J_o
   Build interaction list I_o
   Compute mean ȳ_o, the average of sources y_j ∈ o
   Compute moments M_o^{tkm}, N_o^{tkm} from sources y_j ∈ o according to (4.1) and (4.2)
end for
Set χ_i = 0 as the indicator function at x_i for 1 ≤ i ≤ n
for node o ∈ O do
   for node o' ∈ I_o do
      for target x_i ∈ o do
         Update χ_i according to (4.3)
      end for
   end for
   for node o' ∈ J_o do
      if o or o' is a leaf node then
         for target x_i ∈ o do
            Update χ_i naively
         end for
      end if
   end for
end for
return χ
```

The initial input is a list of n points $y_i \in \mathbb{R}^3$ and the oriented normals n_i at these points, as well as a list of n targets $x_i \in \mathbb{R}^3$. For simplicity, the number of sources and targets is set to be the same, but this is not necessary. Then the desired output is the indicator function evaluated at each target. In other words, for all $1 \leq i \leq n$, we want

$$\begin{aligned}\chi(x_i) &= \int_{\partial\Omega} K(x_i, y_j)dy \\ &= \sum_{j=1}^{n} m_j K(x_i, y_j)\end{aligned}$$

where m_j are weights associated with each source in order to discretize the integral. We choose m_j to be proportional to the square of the average of the distance from y_j to the 10 sources nearest to it.

We construct an octree on the union of the sources and targets so that each leaf node only has one point. For each node o_i in the octree, the *near neighbours* of o_i are the nodes o_j at the same level whose bounding box touches the bounding box of o_i. Thus a node can have up to 27 near neighbours. Nodes at the same level as o_i and which are not near neighbours are said to be *well separated* from o_i. The *interaction list* of o_i are the well separated nodes of o_i which are also children of o_i's parent's near neighbours (see Figure 1). There can be up to 189 nodes in the interaction list for a single node.

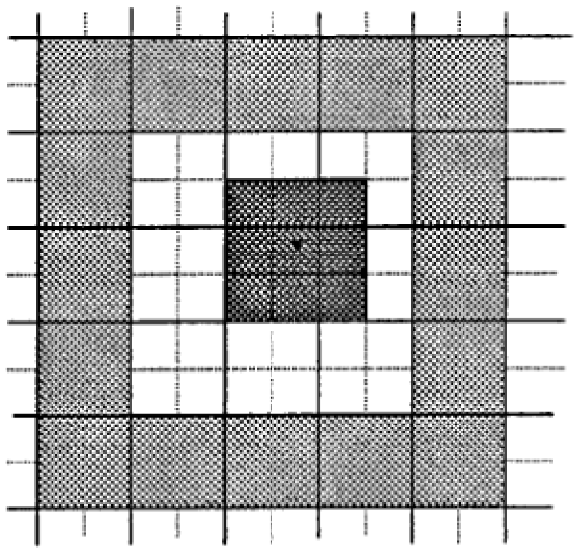

FIG. 1. *The interaction list displayed for a quadtree from [5]. The near neighbours of the marked square are the black squares and the interaction list is the ring of white squares about the near neighbours.*

For a node o_i and a node o_j in its interaction list, we can compute the forces from the s sources in O_j to the t targets in O_i in $O(s+t)$. For convenience, say the s sources are $y_1, y_2, \ldots, y_s$ and the t targets are $x_1, x_2, \ldots, x_t$. Let $\overline{y}$ be the mean of the sources. For each i, let us write $x_i - \overline{y}$ in spherical coordinates as (r_i, θ_i, ϕ_i). Similarly, we write $y_j - \overline{y}$ as $(\rho_j, \alpha_j, \beta_j)$. Note that the well-separated property guarantees that $\max_j \rho_j < \frac{1}{2} \min_i r_i$ which is important for the convergence of our method. The actual contribution of $y_1, y_2, \ldots, y_s$ to the indicator function at x_i is

$$
\begin{aligned}
\chi(x_i) \mathrel{+}= &\sum_{j=1}^{s} m_j K(x_i, y_j) \\
= &\sum_{j=1}^{s} m_j K(x_i, y_j) \\
= &\sum_{j=1}^{s} m_j \left((x_i - y_j) \cdot n_j\right) K_0(x_i, y_j)
\end{aligned}
$$

Using Theorem 3.5, we can expand $K_0(x_i, y_j)$ to the first p terms. In practice, we choose $p = 4$.

$$
\begin{aligned}
&= \sum_{j=1}^{s} m_j \left((x_i - y_j)\cdot n_j\right) K_0(x_i - \overline{y}, y_j - \overline{y}) \\
&\approx -\sum_{j=1}^{s} m_j \left((x_i - y_j)\cdot n_j\right) \sum_{t=1}^{p} \frac{\rho_j^{t-1}}{r_i^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} Y_n^{m*}(\alpha_j, \beta_j) Y_n^m(\theta_i, \phi_i) \\
&= -\sum_{t=1}^{p} \frac{Y_n^m(\theta_i, \phi_i)}{r_i^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} \sum_{j=1}^{s} m_j \left((x_i - y_j)\cdot n_j\right) \rho_j^{t-1} Y_n^{m*}(\alpha_j, \beta_j) \\
&= -\sum_{t=1}^{p} \frac{Y_n^m(\theta_i, \phi_i)}{r_i^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} \left[x_i \cdot \left(\sum_{j=1}^{s} m_j n_j \rho_j^{t-1} Y_n^{m*}(\alpha_j, \beta_j) \right) \right. \\
&\qquad\qquad\qquad\qquad\qquad \left. - \left(\sum_{j=1}^{s} m_j (y_j \cdot n_j) \rho_j^{t-1} Y_n^{m*}(\alpha_j, \beta_j) \right) \right].
\end{aligned}
$$

On the face of it, this seems to be $O(st)$ operations to compute this for all targets. But the inner terms are the only piece dependent on j, so this sum is separable. Define moments

$$M^{tkm} := \sum_{j=1}^{s} m_j (y_j \cdot n_j) \rho_j^{t-1} Y_n^{m*}(\alpha_j, \beta_j), \tag{4.1}$$

$$N^{tkm} := \sum_{j=1}^{s} m_j n_j \rho_j^{t-1} Y_n^{m*}(\alpha_j, \beta_j). \tag{4.2}$$

(Note that M is a scalar while N is a vector.) It takes $O(s)$ time to compute this for each node over its s contained sources. Then we can use these precomputed values to write our scary expression as

$$\chi(x_i) \mathrel{+}= -\sum_{t=1}^{p} \frac{Y_n^m(\theta_i, \phi_i)}{r_i^{t+2}} \sum_{\substack{k=1 \\ n=t+1-2k}}^{\lfloor \frac{t+1}{2} \rfloor} \sum_{m=-n}^{n} \left[x_i \cdot N^{tkm} - M^{tkm} \right] \tag{4.3}$$

which is $O(t)$ to do over all t targets. At the finest level, the naive algorithm is applied between near neighbours which is fine as there are $O(1)$ points. The number of times a point is contained in an octree node is bounded by its height. Assuming a relatively uniform distribution of points, the height of the octree is $O(\log n)$. Modifications are possible to account for all distributions, but we do not consider this. So given this assumption, the total time complexity for our algorithm is indeed $O(n \log n)$.

The indicator function can be used as input to an isosurface extraction algorithm such as Marching Cubes. We choose our target points close to the sources as we want to sample the surface closely. Then, we use an octree to fill in a regular grid so that Marching Cubes can be applied directly. We choose $\chi(x) = 0.5$ as our isosurface.

5. Experiments.

Software & Hardware. The implementation was done in Python3 using the NumPy library for vectorized mathematical operations. A number of other libraries were used for specialized purposes, particularly SciPy's spherical harmonics function, scikit-learn's nearest neighbours function for efficiently estimating the source weights, and scikit-image's Marching Cubes implementation. We use Open3D to load and visualize 3D shapes and Matplotlib for plotting. All experiments were run on a personal laptop with an Intel(R) Core(TM) i5-10300H CPU.

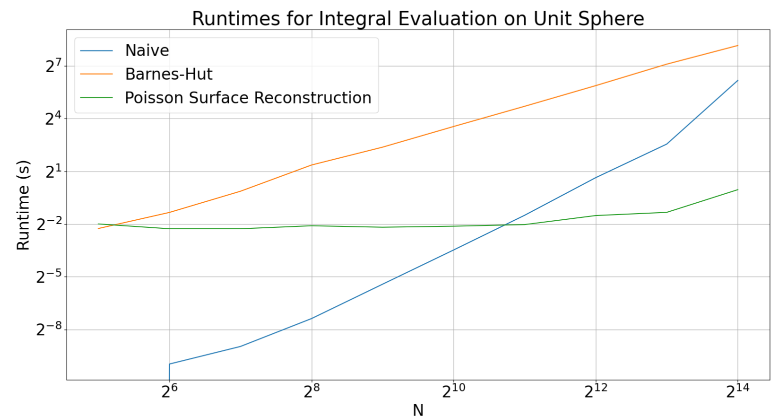

Fig. 2. *The runtime of our methods. Naive refers to the brute-force $O(N^2)$ method while Barnes-Hut is our $O(N \log N)$ method with $p = 4$. We also tested with Kazhdan etal's implementation of Screened Poisson Surface Reconstruction (SPSR) [8].*

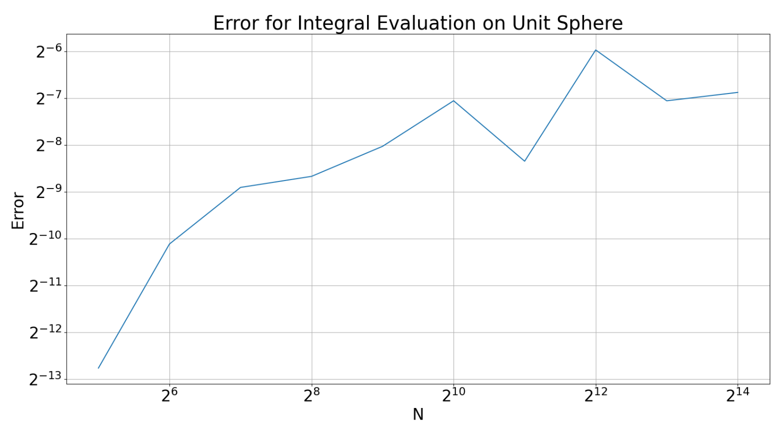

Fig. 3. *The absolute error between the naive method and the Barnes-Hut method for $p = 4$.*

Runtime and Error. Figure 2 shows the runtimes of our methods as N, the number of sources and targets, grows. In particular, on the log-log plot, Barnes-Hut has a slope slightly greater than 1 and Naive has a slope around 2, which aligns with what we would expect. The constant on Barnes-Hut is heavy and so it only outperforms the naive algorithm after $N = 2^{15}$. At this point, the naive algorithm hits a memory cap and so we cannot test further on our machine. We also tested against a library implementation of Screened Poisson Surface Reconstruction for a benchmark. This implementation has Marching Cubes wrapped into it which is likely dominating the runtime without depending on N, hence the plateau.

We also plot the error for fixed p and increasing N in Figure 3. The indicator obtained by the naive method is assumed to be the ground truth as this plot is concerned with the error from our numerical approximations of the kernel. As can be seen, $p = 4$ has reasonable error considering the values are generally in $[0, 1]$. As expected, the error increases as N does too from aggregation.

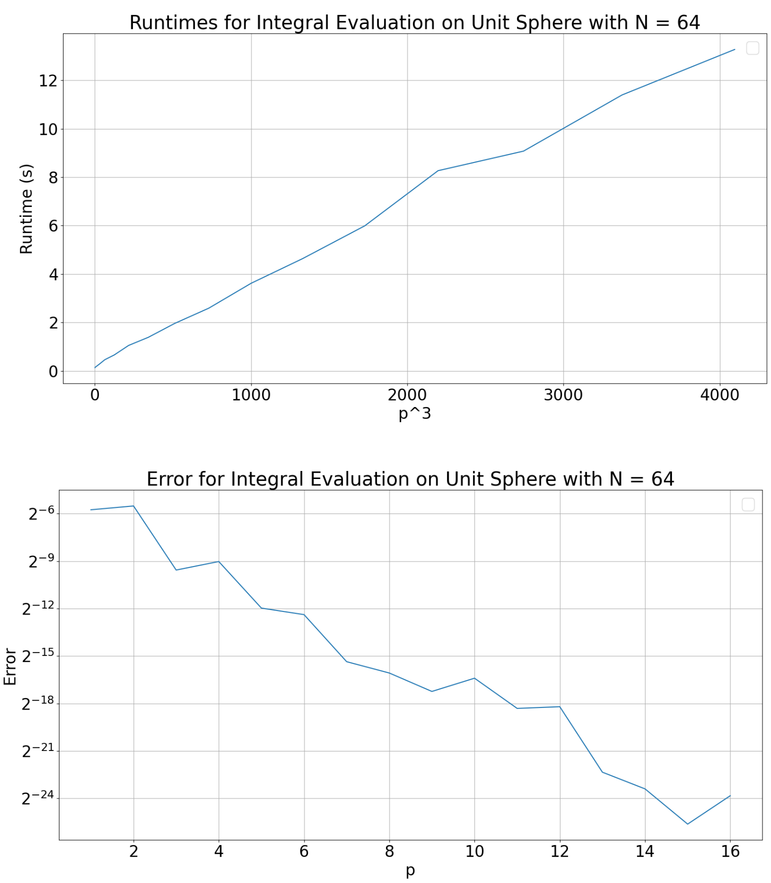

FIG. 4. *The runtime and error for the Barnes-Hut approach with varying p.*

We repeat these experiments with fixed N and changing p in Figure 4. The formulas for the multipole expansion suggest that the runtime should be proportional to $O(p^3)$ and we confirm this by plotting p^3 versus the runtime for fixed N. On the other hand, the lower plot suggests that the error has a geometric rate of convergence $O(e^{-\mu p})$ for some index μ as we expect from Theorem 3.5. Thus, there is a simple tradeoff between runtime and error for the choice of p.

Reconstruction Quality. The truncation error in the multipole expansion is not the only relevant error. The error arising from the use of the boundary integral equation is arguably more important. Figure 5 plots the indicator function generated by samples on a sphere against the distance of the points to the origin. The set of points for which $\chi > 0.5$ is almost precisely the set of points within a distance of 1 as desired.

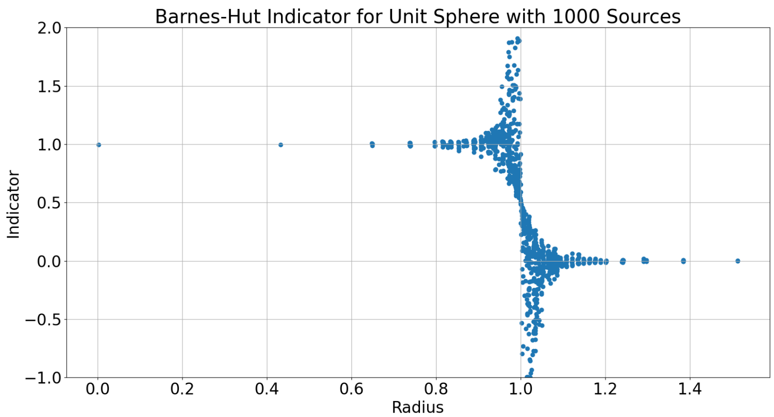

Fig. 5. *The indicator function for the unit sphere.*

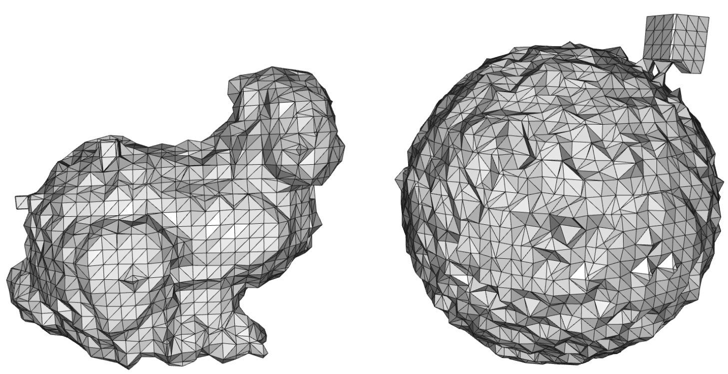

Fig. 6. *The Stanford bunny (left) and unit sphere (right) reconstructed using our algorithm. Each had* 2048 *sources and* 2048 *targets sampled close to the sources as their input. The indicator function was then filled into a* $32 \times 32 \times 32$ *grid and Marching Cubes was applied.*

However, for more complex meshes, we do not obtain good results. We see qualitatively in Figure 6 that the even though our method is accurate compared to the naive implementation of the boundary integral equation, it seems that the reconstruction is still lacking. As can be seen, our method was only loosely able to reconstruct the Stanford bunny. In fact, the bunny is missing its entire left half and the sphere has an unnecessary surface. The jaggedness is likely an artifact of the Marching Cubes algorithm, but this is in some ways inevitable as we only take 2048 evaluations.

This poor reconstruction quality can in part be explained by the singular nature of the boundary integral equation. The estimated indicator changes quickly at the surface and even slight noise could cause it to be thrown off as a result. In particular, the bunny's ears in Figure 6 are completely missing as they likely posed a challenge to the integration equation due to their proximity but opposing normal directions. The sampling of the point cloud also becomes crucial as a result of the singularities. Artificially producing more samples based on a point's normal could help alleviate this issue by implicitly smoothing out the sample set. Furthermore, the boundary integral equation is a global method. While the kernel does decay quickly, it is difficult to express finer details on the surfaces.

6. Conclusion.

In this paper, we proposed approaching the surface reconstruction problem from a boundary integral equation viewpoint. In order to apply fast methods, we derived a novel multipole expansion for the kernel in question. Our results show that our fast method achieves good error and runtime relative to the naive approach, but is unable to obtain even decent reconstruction quality. Future work should focus on mitigating the singularities and improving the runtime's constant factor. Finally, an interesting extension would be to leverage the structure of the boundary integral equation to perform streaming surface reconstruction of dynamic point clouds.

Acknowledgments. Thanks to Professor Burns for the great course.

REFERENCES

[1] N. Amenta, M. Bern, and M. Kamvysselis, *A new voronoi-based surface reconstruction algorithm*, in Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998, pp. 415–421.

[2] J. Barnes and P. Hut, *A hierarchical o (n log n) force-calculation algorithm*, nature, 324 (1986), pp. 446–449.

[3] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, *The ball-pivoting algorithm for surface reconstruction*, IEEE transactions on visualization and computer graphics, 5 (1999), pp. 349–359.

[4] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, *Reconstruction and representation of 3d objects with radial basis functions*, in Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001, pp. 67–76.

[5] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the laplace equation in three dimensions*, Acta numerica, 6 (1997), pp. 229–269.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, in Proceedings of the 19th annual conference on computer graphics and interactive techniques, 1992, pp. 71–78.

[7] M. Kazhdan, M. Bolitho, and H. Hoppe, *Poisson surface reconstruction*, in Proceedings of the fourth Eurographics symposium on Geometry processing, vol. 7, 2006.

[8] M. Kazhdan and H. Hoppe, *Screened poisson surface reconstruction*, ACM Transactions on Graphics (ToG), 32 (2013), pp. 1–13.

[9] H. Langston, L. Greengard, and D. Zorin, *A free-space adaptive fmm-based pde solver in three dimensions*, Communications in Applied Mathematics and Computational Science, 6 (2011), pp. 79–122.

[10] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, ACM siggraph computer graphics, 21 (1987), pp. 163–169.

[11] W. Lu, Z. Shi, J. Sun, and B. Wang, *Surface reconstruction based on the modified gauss formula*, ACM Transactions on Graphics (TOG), 38 (2018), pp. 1–18.

[12] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, Journal of computational physics, 60 (1985), pp. 187–207.

[13] W. Wendland, *On the double layer potential*, in Analysis, partial differential equations and applications, Springer, 2009, pp. 319–334.

[14] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, Journal of Computational Physics, 196 (2004), pp. 591–626.