FMM 算法复杂度争议整理：严格 O(N)是否成立

核心结论

- 工程 / 算法实现语境下：FMM 完全可以称为 “近似线性、实际 O (N)” 的算法。
- 严格的渐近复杂度、并要求误差随 N 增长仍保持某种固定标准时：结论并不绝对，复杂度甚至可能变成 $O(N\log^2 N)$。
- 争论的核心：FMM 的 O (N) 结论成立，依赖哪些前提假设？

讨论发展脉络

1. 初始共识：力计算本身为 O (N)

发起讨论的 DeadlockCode 提出，FMM 的 “力计算” 环节确实可以做到 O (N)。经过树结构划分后：

- 每个粒子只和有限数量的近邻 /interaction list 交互
- 每个 box 只和有限数量的远处 box 做 M2L 操作
- M2M、L2L 等树操作也是每个节点做有限量的工作

因此在树结构已存在的前提下：
$T_{\mathrm{FMM}} \sim O(N)$
这一点基本没有争议。

2. 第一个疑问：建树的复杂度如何计算

FMM 的总复杂度 = 建树开销 + FMM 计算开销。如果建树不是 O (N)，那么整个算法严格来说就不能直接宣称 O (N)。

对于均匀分布的粒子，建树复杂度没有问题：

```
Plain Text
整个空间
├── box 1
```

```
│   ├── particle
│   ├── particle
│   └── ...
├── box 2
│   └── ...
└── ...
```

若每个叶子节点最多放置固定数量的粒子，则：

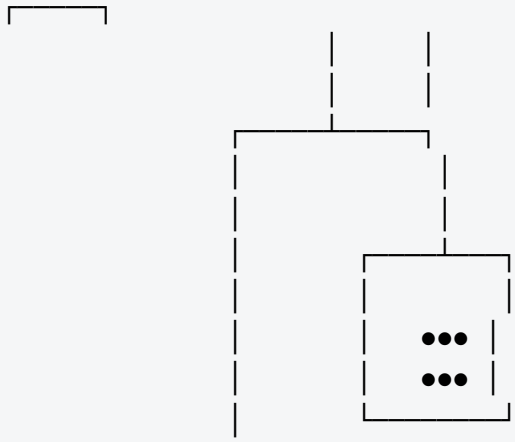
- 叶子节点数量与 N 成正比
- 树的深度相对稳定
- 每个粒子仅需被处理有限次

因此均匀分布下：
$T_{\text{tree}} = O(N)$

3. 非均匀分布带来的问题

如果粒子分布高度不均匀，局部密集区域需要不断细分空间：

Plain Text



此时树的深度会随 N 增长，记深度为 $p = p(N)$。若每个粒子都要经过约 p 层处理，则：
$T_{\text{tree}} \sim O(pN)$
若 $p = O(\log N)$，则建树复杂度变为 $O(N \log N)$，这是最初的核心疑问。

4. Adaptive FMM 的解释：机器精度限制

Adaptive FMM 相关论文提出了工程视角的解决方案：树深度会被机器精度限制，不可能无限细分。

假设浮点数可区分的最小空间尺度为 $\Delta x_{\min}$，当 box 尺寸小于该尺度后，继续细分没有实际意义。因此：

$p \le p_{\max}$
对于固定的计算机硬件，$p_{\max}$ 是常数，因此：
$O(pN) = O(N)$
这是工程领域认为 FMM 为 O (N) 的核心理由。

5. 对机器精度论证的反驳

DeadlockCode 不认可上述论证，他认为：不能因为机器存在有限精度，就把一个本随 N 增长的量当作常数。

在理论分析的 $N \to \infty$ 场景下，在达到机器精度极限之前，树深度依然会随 N 增长，即 $p(N)$ 单调递增。严格来说：
$O(pN) \neq O(N)$
只有当 N 大到树深度被机器精度完全限制后，p 才能视为常数。这体现了 “工程上的 O (N)” 与 “数学上的 O (N)” 的区别。

6. 更极端的视角：所有 O (N) 算法都可被质疑

评论者 G.Aaron.Fisher 提出了更宽泛的观点：几乎所有 O (n) 算法的复杂度都隐含 $n(1 + \epsilon \log n)$ 的项。

常规算法分析默认字长 / 精度为常数，但如果严格考量计算机的表示能力，这个假设本身就不成立。他以链表为例：

- N 个节点的链表，每个节点需要指针指向下一节点
- 要表示 N 个不同的内存地址，指针至少需要 $\log_2 N$ 比特
- 严格来说指针大小为 $O(\log N)$，单个节点存储量不是常数
- 整个链表的存储复杂度应为 $O(N \log N)$

按此逻辑，所有经典 O (N) 算法都可以被质疑。

7. 工程语境的边界

现实场景中，无论 N 是数百、数百万还是数十亿，指针都可以用固定大小（如 64 位）表示。要让 64 位指针不够用，需要 $N > 2^{64}$，这远超出实际应用的规模。

因此该观点的共识是：理论上可以严格挑刺，但工程实践中不采用这种极端标准。

8. 学术界的正式质疑：Aluru 的论文

视频作者 keyframe41 提到，Srinivas Aluru 发表过论文《Greengard’s N-body Algorithm is not order N》，对 FMM 的 O (N) 性质提出了比 “树深度” 更根本的质疑。

9. 深层问题：展开阶数 p 也可能随 N 增长

此处的 p 指 multipole/local 展开的阶数，例如球谐展开形式：

$$\Phi(x) \approx \sum_{n=0}^{p} \sum_{m=-n}^{n} a_{nm} Y_n^m(\theta, \phi)$$

展开阶数 p 决定近似精度，通常 p 越高，误差越小。

10. Aluru 的核心论点

若要求 $N \to \infty$ 时误差始终满足 $\epsilon < \epsilon_0$（固定误差标准），则随着粒子数量增长，累计误差会要求展开阶数同步提升，满足：

$p \propto \log N$

即 $p = O(\log N)$，而非常数。

11. 对复杂度的最终影响

FMM 的计算量与展开阶数相关，大致满足 $O(p^2 N)$ 的关系。代入 $p = O(\log N)$ 可得：

$O(p^2 N) = O(N \log^2 N)$

因此该论文得出结论：

$T_{\text{FMM}} = O(N \log^2 N)$

即在严格误差保证 + N 无限增长的理论模型下，FMM 的复杂度不是 O (N)，而是 $O(N \log^2 N)$。

12. 对比的公平性问题

需要注意的是，Barnes-Hut 算法没有严格的误差保证。

Barnes-Hut 通过 opening angle 准则 $\frac{d}{s} < \theta$ 判断是否将簇视为整体，只能说明 “在该参数下效果较好”，不具备 FMM 那样的严格高阶展开误差控制能力。

二者的对比如下：

- Barnes-Hut：$O(N \log N)$，无严格误差保证
- FMM：严格理论分析下可能为 $O(N \log^2 N)$，但可严格控制误差

不能简单认为 Barnes-Hut 复杂度更优，因为二者的误差约束前提完全不同。

13. Morton code 的建树方案

Morton 编码可以实现 O (N) 复杂度的树构建，基本路径为：

粒子坐标 → Morton 编码 → 排序/分组 → 生成树结构

该方法可以避免传统递归建树的部分开销，在合适的计算模型下可实现 $T_{\text{tree}} \approx O(N)$。

14. Morton code 未解决根本争议

Morton 编码方案依然依赖以下前提：

- 固定数值精度
- 有限的坐标表示范围
- 叶子节点粒子数可控
- 合理的空间分布与深度限制

因此它只是换了一种形式的假设，并没有从理论上彻底解决复杂度争议。

15. 实际工程中的性能表现

实际测试显示，FMM 树构建耗时约 4~5 ms / 帧，占比高于预期。原本认为树构建很快、数学计算是主要耗时，但实际树构建开销不可忽略。

对应的工程优化思路：不必每一帧都重建树，可以每 m 帧重建一次，在精度和构建开销之间做权衡。

16. 实际场景的树深度表现

在 12 万粒子、每个 box 最多 100 个粒子的测试中：

- 树深度约为 10
- 偶尔超过 10
- 极少达到 12

实际场景中 p 近似为很小的常数，而非随 logN 增长。即使理论上存在非线性增长趋势，现实中该效应也非常微弱。

讨论共识汇总

问题	讨论结论
FMM 的力计算环节	O (N)，基本无争议
均匀分布下的建树	可以认为是 O (N)
非均匀分布下的建树	严格理论分析存在疑问

树深度是否为常数	工程上通常可以，理论上未必
机器精度能否将深度视为常数	工程上可以，严格渐近分析有争议
Morton code 能否 O (N) 建树	可以，但仍依赖模型假设
展开阶数 p 是否为常数	固定精度 / 固定误差要求下通常如此处理
要求误差随 N 增长仍严格有界	可能需要 $p \sim \log N$
严格误差约束下的复杂度	论文指出可能为 $O(N\log^2 N)$
实际 FMM 性能是否接近 O (N)	是，且通常非常接近线性
FMM 是否因此不如 Barnes-Hut	不能这么判定，二者误差保证不同

FMM 无法达到纯 O (N) 的三个层次

1. **教科书 / 工程分析层次**

假设展开阶数 p 为常数、树深度 L 为常数、每个 box 的交互数量为常数，则 $T = O(N)$，这是常规语境下 FMM 是 O (N) 算法的来源。

2. **考虑空间非均匀性层次**

若树深度 $L = L(N)$，则建树复杂度为 $O(NL)$；若 $L \sim \log N$，整体复杂度为 $O(N\log N)$。

3. **严格误差约束层次**

若要求误差不随 N 增长而恶化，则需要 $p \sim \log N$，而展开操作本身为 $O(p^2)$，最终整体复杂度为 $O(N\log^2 N)$。

核心启示

这场讨论最核心的价值不是得出 “Nlog²N” 的结论，而是引出一个本质问题：

> O (N) 的结论，是在什么计算模型、什么误差要求、什么数据分布、什么精度假设下成立的？

复杂度分析本身建立在特定的计算模型之上，我们通常默认：

- 机器字长为常数
- 浮点精度为常数
- 算术运算为 O (1)
- 指针操作为 O (1)
- 最大树深度有界
- 展开阶数有界

在现实计算机的模型下，认为 FMM $\approx O(N)$ 是完全合理的。但如果切换到 N 无穷大、误差严格有界的理论模型，“FMM 是 O (N)” 就只是有条件的结论，而非无条件的数学真理。

最终总结

- 质疑方：FMM 计算看似 O (N)，但建树和展开阶数都可能随 N 增长，严格意义上真的是 O (N) 吗？
- 学术佐证：确实有论文证明，严格误差约束下 FMM 复杂度可能为 $O(N\log^2 N)$，因为保持误差需要展开阶数随 logN 增长。
- 工程视角：如果连 “机器字长为常数” 这个前提都不承认，几乎所有 O (N) 算法都能被质疑；工程实践中仍应将其视为常数。

最终共识：理论上可以对 FMM 的 O (N) 性质提出严肃质疑；工程实践中，在固定精度、有限规模、合理粒子分布和固定展开阶数的前提下，将 FMM 称为 O (N) 算法完全合理，其实际性能也确实接近线性。

实际测试数据显示，12 万粒子、每 box 100 个粒子的场景下树深度仅约 10，极少超过 12，且 FMM 实际可处理的粒子规模比 Barnes-Hut 更大。

附录：评论原文

@DeadlockCode

Hey, Deadlock here. I'm the creator of one of the Barnes-Hut videos you referenced near the beginning.

First off, I just wanted to say I really enjoyed this video. It was fantastic to see all of it put together with beautiful animations to boot. I really related when you started talking about how there's seemingly barely any material explaining how the FMM algorithm actually works. It's great to finally see such a clear and well put-together explanation. You did a way better job explaining it than I think I ever would so I'm almost glad I didn't make a video on it because then the world might've never been blessed with this gem.

After making my Barnes-Hut video, I originally hoped to make one on the FMM as well but I never got to it because other projects interested me more. But while researching for the FMM there was always one thing that I kept getting stuck on: is it really O(N)?

The force calculation itself is clearly O(N), but the total complexity seems to depend on whether the acceleration structure itself can be built in O(N) as well.

For a uniform particle distribution, that part seems straight forward. You can bucket particles into regions, the number of particles per region stays bounded, and building the tree is O(N) from there.

Where I got stuck was the non-uniform case. The Adaptive FMM paper argues that the octree construction is O(pN), where p is used to bound the tree depth, and then treats p as bounded by a constant because it is limited by machine precision. At face value, that makes perfect sense; once you reach machine precision, further subdivision doesn't provide additional accuracy so you can stop.

But that argument always felt unsatisfying to me. Before you hit that machine precision limit, the depth will still grow with N, meaning the tree construction could behave more like O(N log N) unless I'm missing something. I feel like it only truly becomes O(N) once the depth stops growing because you've reached the limits of the machine precision. It just feels very different from saying the algorithm is O(N) in a more general sense.

I completely understand why the paper makes that assumption, treating machine

limits as constants is very common and often completely justified. I was just curious whether there is another way of looking at it which I'm missing.

I also came across other approaches like radix-sorted Morton codes, which can also build the tree in O(N), but they seem to rely on similar assumptions about fixed precision or particle distribution to make sure the number of particles per leaf is O(1).

I'd love to hear your thoughts on this. When you think about FMM, do you feel that the machine precision assumption is enough to justify calling the full algorithm O(N), or is there some deeper argument that avoids this concern altogether?

I also just want to acknowledge that this is mostly just a philosophical question for the fun of it. In reality the time complexity doesn't really matter; if the algorithm is better, it's better, and that's all that matters. The O(N) argument just never clicked with me so I'd love to hear how you see it, and maybe I can finally love this algorithm unconditionally without the sour taste of tree construction.

@G.Aaron.Fisher

Almost all O(n) algorithms are secretly O(n*(1+ε*log(n)). That epsilon n log n part can either relate to the size of object pointers, or the level of precision required to accurately aggregate values.

In practice (1+ε*log(n)) is O(1) for any n that anyone is ever going to care about.

As a straightforward example, a linked list where each node contains a pointer to the next will require that pointer to be at least log_2(n) bits in size. So for a large enough n, you need more than a long to point to the next node in the sequence. But by the point that the linked list requires 3 longs to represent a single pointer, the linked list has more elements than the universe has atoms. And if you stop there, 3 is O(1).

It's my professional opinion that you should pretend you have no knowledge of this forbidden lore in any job interviews or work conversations. The knowledge, while completely accurate, is nothing but a footgun if you've been cursed enough to learn of its existence.

@keyframe41

Really cool to see you here and I'm glad you liked the video this much! When I was researching the algorithm, I actually read a paper with basically the same argument about machine precision not being constant ('Greengard's N-body Algorithm is not order N' by Srinivas Aluru) and felt very mixed about it as well. He shows that the number of expansion terms p must be proportional to logN to maintain an error bound as N grows large. So he states that FMM is not O(N) but in fact O(Nlog^2N), even worse than Barnes-Hut.

However Barnes-Hut doesn't have any guarantee of error, so we're comparing them on two different standards here, and FMM still ends up on top.

I then stumbled onto the Morton codes approach which seemed better but didn't think too much of it as it felt like a lot of extra work to implement on top of the core algorithm. Just before I posted this video I asked gemini to vibecode the Morton codes tree creation and it turned out slightly worse than my pretty optimized approach, so that was probably a good decision. Still, for the implementation video I also plan to do some more tests with different numbers of particles just in case the time taken grows differently with the approach, and also check out if the NlogN cuve really appears.

If I recall, the tree creation takes up about 4-5ms per frame which was more than I anticipated, since I originally believed that most of the time would be on the FMM math, so the O(N) would overpower the O(NlogN). So, another thing I could test is building the tree every m-th frame to see how that would tradeoff accuracy.

One important thing about the tree I noticed while debugging is that the depth seems quite bounded. At 120k particles and 100 particles per box, the depth only occasionally goes past 10, and very rarely hits 12. So we aren't going over particles that many extra times, and before I do any of those tests my first instinct is expecting the 'nonlinear-ness' to be only very slight.

Personally, I'm entirely fine with accepting FMM as practically O(N) and forgetting about the tree creation, which I admittedly skimmed over in the video. I was worried that Barnes-Hut would perform better for me since you somehow got 1M particles to work but thankfully in practice my laptop can run twice as many particles (only 120k though) using FMM than it can with Barnes-Hut.

@DeadlockCode

@keyframe41 I'm glad to see my gut reaction against the O(N) claim mirrored in the literature, it makes me feel a little less crazy. I didn't go quite as deep, so I must have missed that paper. I'll definitely have to read it sometime.

Either way, I completely agree that FMM's value goes far beyond just the time complexity argument. It's still an amazing algorithm with a lot of really beautiful math behind it, and the practical performance is what matters most in the end.

Experimental measurements of the scaling would be really interesting to see. Will your final implementation be open source? I'd love to try it on my machine, or maybe even experiment with GPU acceleration if I find the time.

Either way, I'm really looking forward to the implementation video!