网格简化

计算机科学与技术 75 班 赵成钢 2017011362

2019 年 6 月

1 摘要

本次作业实现了 Michael Garland 在 Surface Simplification Using Quadric Error Metrics 中提出的基于边塌缩的增量网格简化算法，并在边选择引入了近似的搜索算法、在代价函数上引入了翻转代价和锐利特征代价，基于自己的想法实现了较好的效果。

Github 地址：https://github.com/LyricZhao/Surface-Simplification

2 代码实现与说明

2.1 框架

代码采用 C++ 实现，使用 Makefile 方便了管理和编译。其中 main.cpp 为程序的主入口，负责参数的处理。mesh.hpp 定义了网格类、三角面片类、顶点类、支持删除的堆和并查集，并实现了网格的简化、文件的读写等功能。

2.2 使用方式

在 Makefile 中修改编译器命令并键入 make 即可编译。具体使用方式为：simplify <input> <output> <ratio> (<t>)。其中 input 为输入 OBJ 文件的路径，output 为输出的 OBJ 文件的路径，ratio 为简化比（小数，即简化后面片数量对原数量的比），最后一个参数 t 为可选参数（默认为 0.01），表示在选择边过程中两点距离的上限。

2.3 本地环境

1. macOS 10.14.5
2. Homebrew GCC 9.1.0
3. 2.8 GHz Intel Core i7, 16GB RAM

2.4 可变参数

在 mesh.hpp 中有 DFS_T_SEARCH、FLIP_COST 和 SHARP_COST 三个宏变量，对应注释掉可以关闭其功能。其中若 DFS_T_SEARCH 开启，则在选择边的过程中会执行 DFS，会根据当前距离进行近似剪枝，如果注释会严格计算（也做了一些优化，但还是很慢），DFS 的速度非常快，而且二者效果相当；如果 FLIP_COST 开启，则在选择的边造成其他面片翻转时为代价函数乘一个惩罚系数；如果 SHARP_COST 开启，则在尖锐边缘的边会乘一个惩罚系数，这样可以尽量保持细节。具体的算法在后面会给出。

3 算法及优化

3.1 算法流程

1. 从存储中读取网格
2. 计算每个点对应的 Q 矩阵
3. 把已连接的边和距离小于参数 t 的两个点作为可移除对加入堆中
4. 根据定义的代价依次把最小的可移除对从网格中删除，并在堆中更新代价
5. 重复上一做法直到达到简化要求
6. 写入简化后的网格到存储

3.2 基础原理

在论文中，作者把点到所有所连接的平面的距离和平方作为 Error，并把平面写成方程系数的向量，这样有 $\Delta(v) = \Sigma_p(p^T v)^2 = \Sigma_p v^T(pp^T)v = v^T(\Sigma_p pp^T)v$，再把 $\Sigma_p pp^T$ 看成每个点的 Error 矩阵，这样在合并两个点时，Error 矩阵直接相加即可得到新的误差矩阵（平方和可加，也就是这个点到原来所有平面的距离平方和），而对于合并两个点之后的新点位置也就是 Error 最小的地方，根据一些微积分的知识即可算出，具体算式在论文中有体现，这里不再赘述。

3.3 图和堆的维护

对于图，我的实现方式是不存边，只存三角形面片，即在每个点的数据结构上存储每个和该点相邻的面片的信息，每个面片采用最小表示法的方式，用循环位移的方式把最小编号的点放在第一个（循环位移是为了法向量方向不变），并用 STL 中的 set 数据结构对表示后的结构存储，这样可以方便添加、删除面并由于采用了 Hash 所以天然可以判断面重叠问题。

对于堆的维护，我采用了 STL 中的优先队列，并用 STL 中的 map 来维护堆中元素的时间戳，每次操作时间戳加一，在插入时用元素对的编号组成 unsigned long long（小编号在前）作为 map 的键值更新最新的时间戳，同时也在插入的元素中打上当前时间的标记，而要删除元素的时候只需要把 map 中的时间戳改成-1，这样就支持了删除的操作。而对于取堆顶和删除堆顶的操作，则需要先把无用的堆顶删除，所谓无用也就是时间戳已经过期，这样基于时间的好处还有可以方便的判断一个对是不是在堆中避免重复插入。

而且注意到，在删除一对点时也产生了新的点，这里我没有真的在代码上实现一个新的点的插入，而是把信息都改到了标号较小的那个点上，而较大的那个点也没有真的删除，而是采用了并查集的方式，把两个点看成集合，把第二个点所在的集合合并到第一个点上，这样在所有的情况下所有是集合树树根的点就是没有被删除的点。

3.4 边的选择

如果完全参照论文中来，个人认为这是一个 $O(n^2)$ 的算法，而瓶颈就在于边的选择上，论文中除了真的存在的边，还划定了距离小于一个阈值的

边来作为可以选择的对，这样如果阈值充分大，这部分的开销已经到 $O(n^2)$ 了，也没有特别大的优化意义。这里我最开始实现的是，把所有点按照第一维排序，在两层循环时根据第一维的位置 break，可以精确地剪枝，另一个可以想到的是 KD 树的优化，不过考虑到这部分的上限就是 $O(n^2)$，我认为如果不考虑 t 的合理性，那 KD 树反而成了负优化，也就没有进一步实现。在实验中我发现，第一维剪枝的方式已经在保证了完全等价的情况下的近 50 倍速度的提高（枚举实在过慢），不过还是很慢，而且其实很大程度上也是因为点对太多，在龙的模型中 t 取 0.01 已经有 7000 万个点对了。

进一步的，我想了一个近似的改进的方法，也就是采用深度优先搜索的方式：枚举每个点，从枚举点出发，当距离（不是路径距离，是欧几里得距离）一旦超过 t 时就剪枝，虽然这样可能会漏掉很多点，而且在路径上不连接的点不可能没枚举到。但是在实验上还是取得了近似的可观的效果，而且非常快，相比于 t 为 0.01 时的 7000 万个对，通过这种做法直接降低到了 90 万个点，而且可以从对比图中看出效果非常好。这里我的想法是：如果一个点通过一个超过路径范围的点放置在合理的范围内，那它有非常大的概率和枚举点通过更短的路径直接连接进而也会被枚举到，针对第二个情况，我认为作者设置它的意图并非让路径不连接的点简化后连接起来（毕竟还是两个连通域），而是要让在路径上间接连接的点连起来，所以我觉得是合理的，或者更准确的说，这里的 t 参数更像是一种搜索阈值。

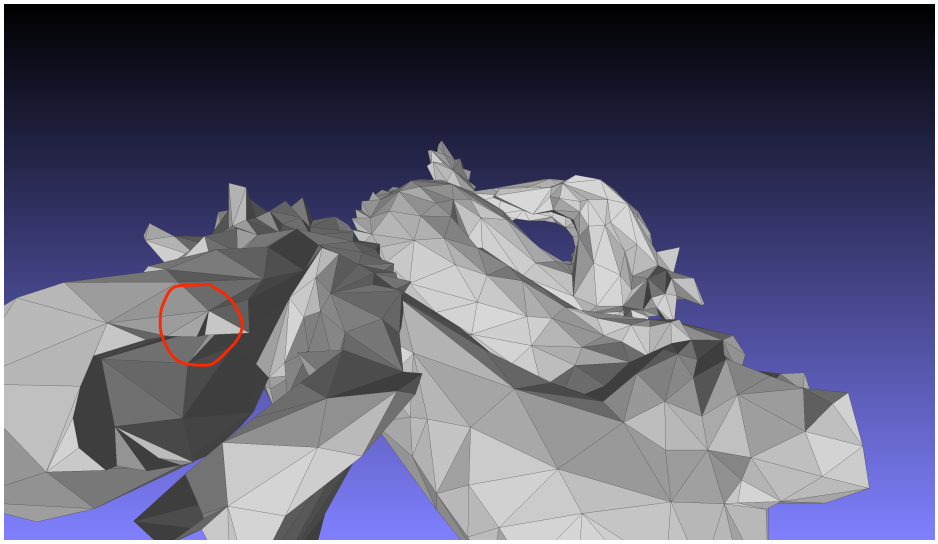

图 1: 没有添加 DFS，t=0，可以看到局部的不自然，比如红圈处

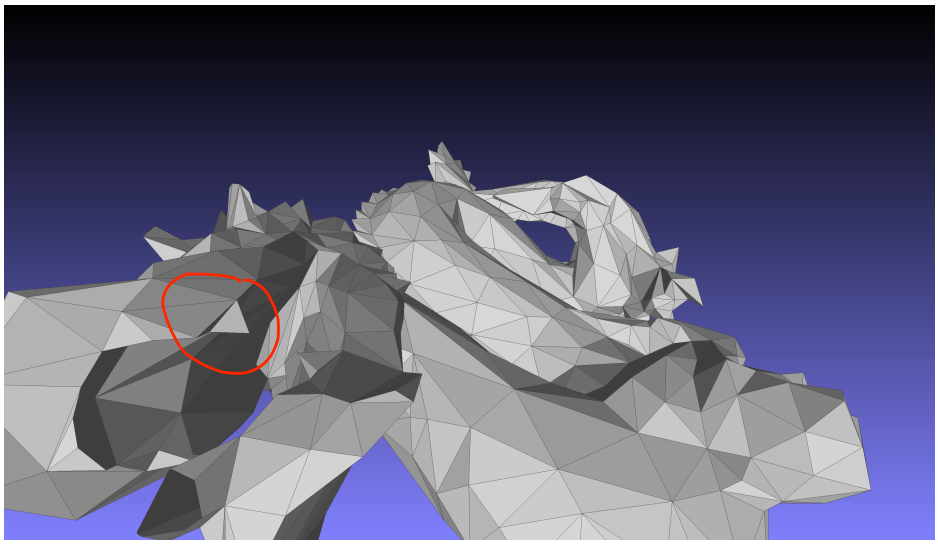

图 2: 添加了 DFS，t=0.01，可以看到局部更为平滑，毛刺较少

3.5 面翻转惩罚

对于在习题课上 PPT 的翻转的情况，我认为这种算法本身是无法避免的，而我们需要做的是减少这种情况的发生。首先，我对面翻转情况的估计是，在删除一对点后，原点相连的面片构成的新位置的面片法向量的变化角度的余弦的平均（因为面片有很多），这样的话就是平均越小越差，如果是负的是真的彻底翻转了。

而为了体现真正减少，我设定了几个阈值，当平均值再加 1.1（加 0.1 是为了防止除 0）小于 1.3 时候，对这个情况进行惩罚，惩罚的措施是对原代价（点到所有平面的距离平方和）除以 $(avg\_cos+1.1)/1.3$，这个数小于 1，所以代价会变大。综合不惩罚的情况就是，原代价除以 $min(1,(avg\_cos+1.1)/1.3)$。结果显示，这样的做法会一定程度上改变一些奇怪的空面片和翻转不可见的面片，其实效果不是特别明显，在细节处可见。

3.6 锐利惩罚

在上述算法中，我发现在低简化比时，面片的分布非常均匀，这就导致了很多原来是大块形状和原来是大量细节的地方非常平均，也就浪费了大量的面片在一些本来可以用简单图形来表示的地方。所以，我引入了一个锐

利的惩罚，我直觉上认为，大量的细节体现在边两侧的面片的夹角上，夹角越小就证明了这个地方越锐利，越是有可能成为细节的地方，而对于大面的形状而言，面片之间往往平滑过渡，夹角很小。

类似，上面的面翻转的惩罚，我先计算两侧面片的夹角余弦（越小越锐利，尤其是负数的时候），再设定阈值 1，在余弦值加 1.1 还小于 1 的情况下进行惩罚，类似的也就是原代价除以 $min(1, cos + 1.1)$。同样的，如图显示，简化后的模型有了进一步的改进，原来的细节之处被强调了一些，效果相对明显，也在一定程度上是一种边界保持。

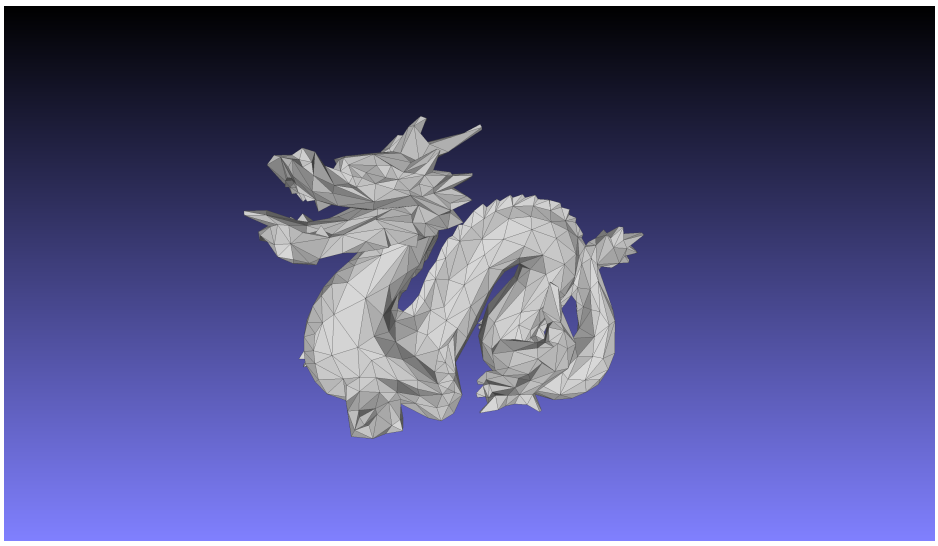

图 3: 没有强调锐利边缘，可以看到网格分布比较均匀

3.7 速度

在我个人的电脑上，使用 g++-9 编译，打开 O3 优化开关，对 21 万个面的龙模型进行 0.001 简化比的简化，得到了 208 个面的龙模型，只用了 15 秒左右，结果较为可观，下表是我对 21 万个面的龙模型做的一些速度的实验。

可以看到简化比对时间影响并不大，即维护的堆的规模差别不大，所差时间只体现在塌陷的次数上。而对 t 进行调整则意味着维护数据结构的规模的改变，不仅在选择边的时间会变少，后面塌陷也会因为规模减小而变快。

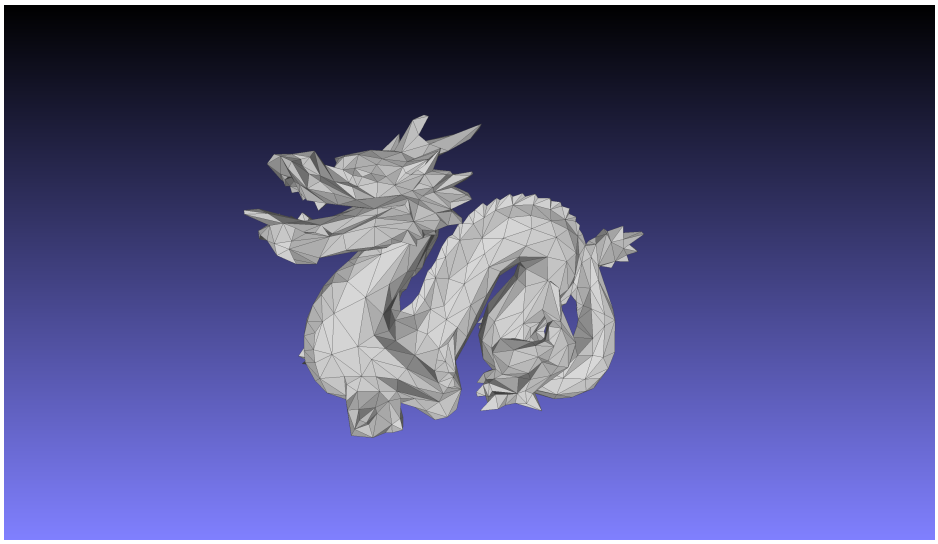

图 4: 强调锐利边缘，可以看到网格分布不均匀，细节处分布更多

Model	Ratio	t	Time(s)
Dragon	0.1	0.01	14.722
Dragon	0.001	0.01	15.679
Dragon	0.001	0	7.262

3.8 误差分析

为了进一步分析结果，我对点平均误差进行了计算，计算方法即为论文中 cost 的计算，也就是点到平面的距离平方和。我对龙模型进行了实验和分析，得到了下面的结果（两个 Error 分别指简化比为 0.01 和 0.001 时，DFS 的 t 参数取 0.01）。

可以看到，在高简化比时，DFS 操作可以一定程度上减小误差，而极低简化比时随着点之间的距离的增大，这种方式的效果就不好了。对于 FLIP 优化，同样的，在面片非常多的情况下可以在保证误差几乎不变的同时防止大量的面翻转，而面片很少时算法本身就已经很好了，对结果没有任何影响。而 SHARP 优化体现在 Error 结果上成了负优化，我的想法是算法本身就是为了 Error 最小，虽然引入了这个代价使得误差变大，但视觉细节效果的确有一定改观，也不失为一个尝试。

Method	Error(0.01)	Error(0.001)
t = 0	0.0119017	5.71766
DFS	0.0118876	5.85715
DFS + FLIP	0.0118882	5.85715
DFS + FLIP + SHARP	0.0121665	6.614
DFS + SHARP	0.0121549	6.614

3.9 其他实验

后面，我把所有优化打开，也对其他模型进行了实验，t 均取 0.01，简化比为 0.1。

Model	Faces(After)	Time(s)	Error
Arma	4638	1.58	0.000406692
Block	426	15.679	0.0558648
Buddha	12304	4.69	5.99644e-05
Bunny	7058	55.23	7.25597e-07
Dinosaur	400	0.12	44.0852

可以看到综合来讲算法的速度的效果均可观，我也检查了视觉输出，均有不错的效果。对于上面 bunny 时间过长的问题，我的发现是 t 过大导致的边选取规模过大，可以看到每个模型的坐标 Scale 均不相同导致误差的量级也不同，如果要调整到最佳的状态，还需要针对模型调整参数。进一步的，我认为这样的误差也并不直观，会导致不同坐标 Scale 产生不同的量级，没有可比性，进一步的改进也有待思考。本部分实验中的一些效果图见图 5 到图 8。

4 小结

本次我独立依据论文实现了经典的网格简化算法，受益匪浅，其中用了大量的程序语言技巧、数据结构和算法知识，是一个比较综合的应用，也提起了我对计算机图形学的进一步的兴趣。感谢老师和助教的耐心讲解。

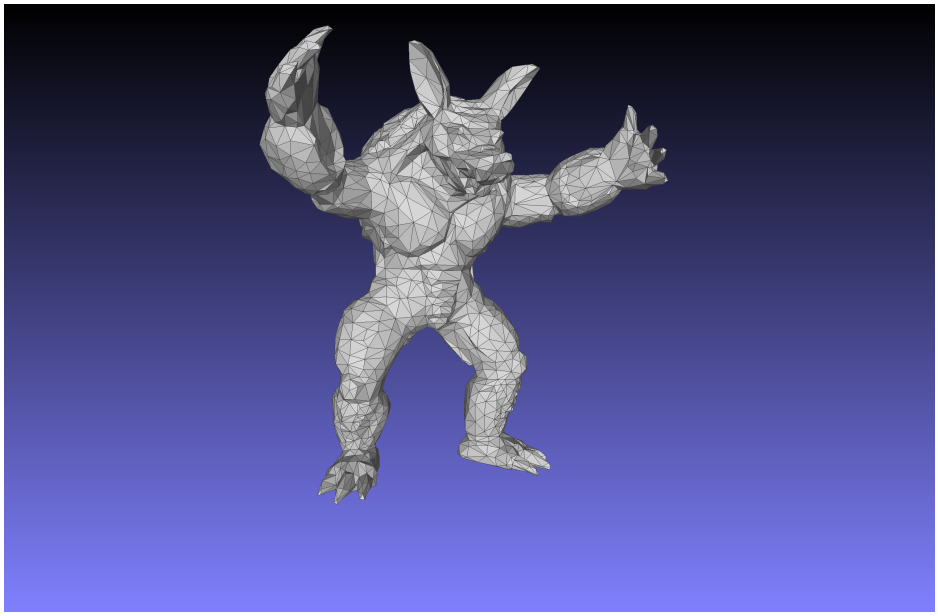

图 5: Arma

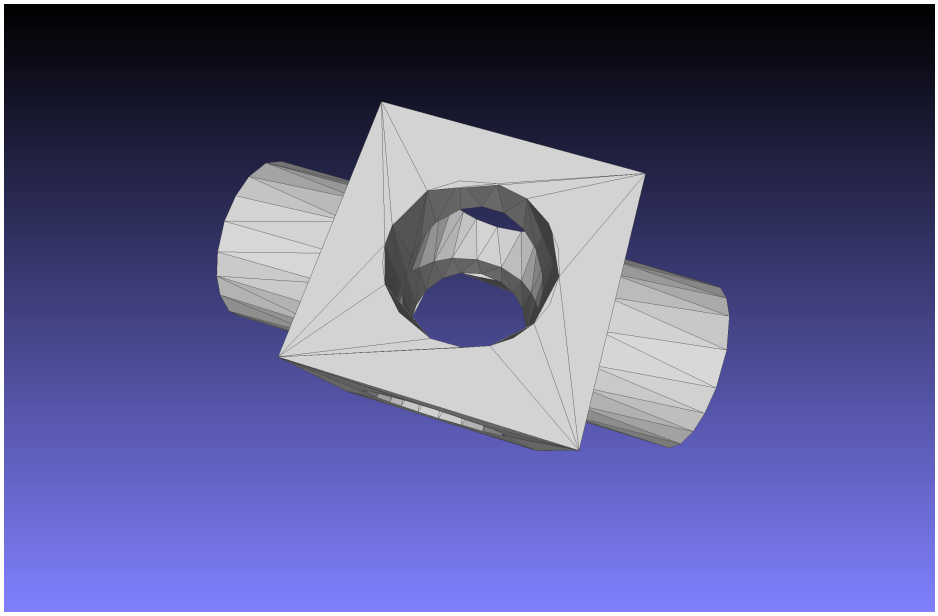

图 6: Block

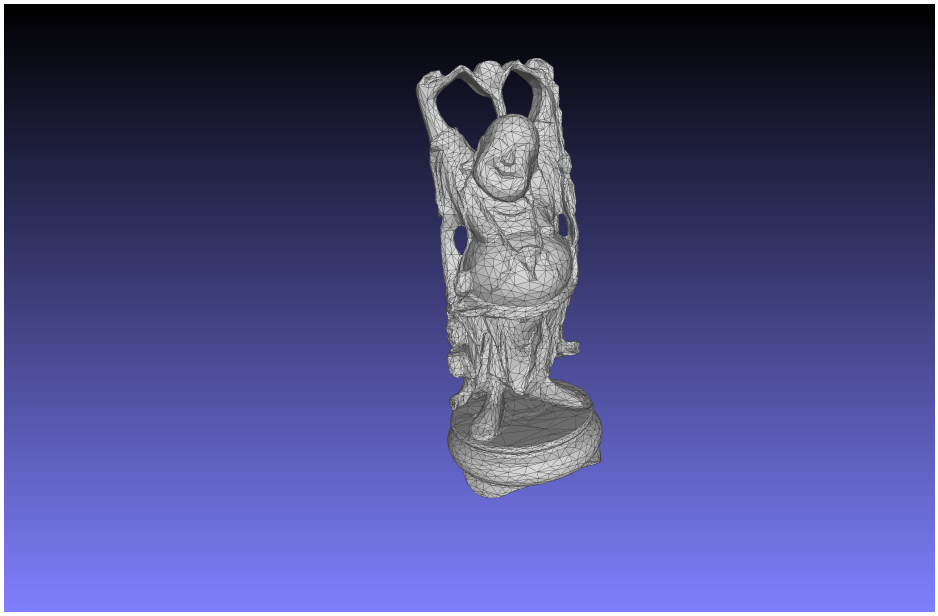

图 7: Buddha

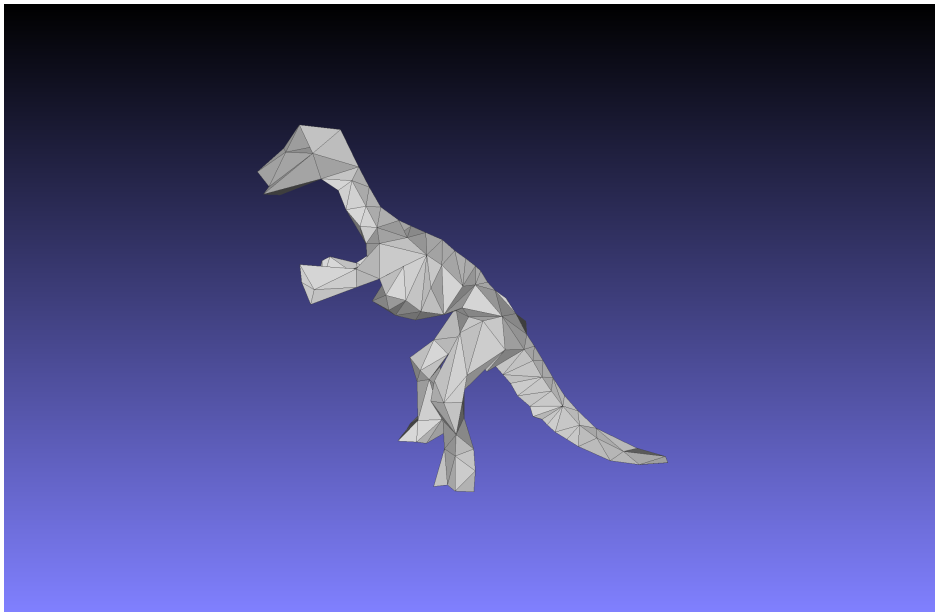

图 8: Dinosaur