三角网格上的离散测地线算法的研究与实现

卜鹏洋 江乐彬 伍宇昭

摘要

在微分几何中，曲面上两点间的测地线可理解为所有连接两点的曲线中拥有最小长度的一条。本实验就三角网格上某一顶点到其他所有顶点的测地线问题，分析了一系列前人设计的算法的效率及问题，同时实现了[1]中提出的改进版 CH 算法并制作了演示程序。

关键字：测地线，三角网格，CH 算法，窗口传播，剪枝

1. 引言

在微分几何中，曲面S上p,q两点间的测地线可以理解为在所有连接p,q两点的曲线Γ，Γ⊆S中，拥有最小长度的一条。在这种意义下，测地线可以被认为是直线这个概念在三维曲面上的推广。而微分几何[5]也告诉我们，曲面S上p,q两点间的测地线可以通过变分法计算得到。

不过，需要注意的是，微分几何中处理的都是理想的参数化曲面。而在实际的计算机应用中，特别是在计算机图形学中，很多情况下，曲面是通过一组离散的三角面片近似得出的。在这种情况下，测地线也要进行相应的离散化，即离散测地线。参考原始的测地线定义我们不难得出，所谓三角网格上S上p,q两点间的离散测地线，就是三角网格上连接p,q两点的最短路径。自然地，这种离散测地线可以认为是图论中最短路径问题在三角网格上的推广。

与图论中的最短路径问题相似，三角网格上的离散测地线算法也大致分为三个版本，即：

- 计算三角网格上某一点到其他所有点的测地线
- 计算三角网格上任意两点间的测地线
- 计算三角网格上所有可能点对(顶点对)间的测地线

在我们的实验中，我们关注第一个版本，即给定三角网格S上一点s，如何计算s到S上任意一点t的测地线。还有一点需要说明的是，本文中所说的三角网格上的点特指三角面片的顶点。这也符合大部分测地线相关工作的习惯。

2. 相关工作

对于计算三角网格上一点到其他所有点间的测地线这一问题，Sharir 和 Schorr 在 1986 年第一次给出了一个多项式时间复杂度的算法。这个算法的复杂度为 $O(n^3\log(n))$，其中 n 代表表面的复杂程度，但这一算法只能对凸多面体进行计算。1987 年，Mitchell 等人提出了 MMP 算法([2])。这种算法使用了类似于 Dijkstra 算法的流程，将复杂度降到了 $O(n^2\log(n))$，同时这种算法可以处理任意的多面体表面。1990 年，Chen 和 Han 提出了 CH 算法([3])，进一步将复杂度降到了 $O(n^2)$。

不过值得注意的是，很多实验结果(Kaneva and O'Rourke[2000, 6]，Surazhsky[2005, 4])表明，MMP 算法在绝大多数实际应用中的性能都要远远优于它在理论上的最坏情况，而CH算法尽管有着很好的理论性质，但其实际性能却要远逊于MMP算法。

[1]分析了CH算法在实际应用中效率低下的原因，即CH算法产生了过多的无用窗口，这些窗口对测地线没有任何贡献。“在许多样例中，超过*99%*的由CH算法产生的窗口都被抛弃掉了”。在这个基础上，[1]中的算法对CH算法进行了修正，采用了更加高效的窗口剪枝算法，减少了产生的无用窗口的数目，同时使用了优先级队列这一数据结构，进而提高了算法

效率。[1]中的实验说明，[1]中提出的CH修正算法在实际运行中要远远快于CH算法，同时这种算法也要比MMP算法拥有更快的运行速度。不过，这种CH修正算法的渐进复杂度却退回到了$O(n^2\log(n))$。

3. CH 算法概述及相关改进

3.1 旋转展开和窗口传播

旋转展开三角面片序列是一种常用的研究测地线的方法。设$f_1, f_2, \cdots, f_i, \cdots, f_m$是一组三角面片序列，其中$f_i$与$f_{i+1}$相邻，且$e_i = f_i \cap f_{i+1}$。$f_1$表示包含源点 S 的三角面片。关于这一组三角面片的旋转展开可用图 1 中的伪代码表示：

```
F := {f_1};
for i := 1 to m - 1 do
    Rotate F around e_i until
        F, f_{i+1} are co-plane and lie on
        different sides of e_i;
    F := F ∪ {f_{i+1}}.
```

图 1：三角面片的旋转展开

当上述过程完成后，所有的三角面片都被展开到同一平面上，而经过这些三角面片的最短路径也会被展开为直线段。

需要注意的是，对于一些三角面片序列而言，根本没有测地线通过它们，如图 2 所示。

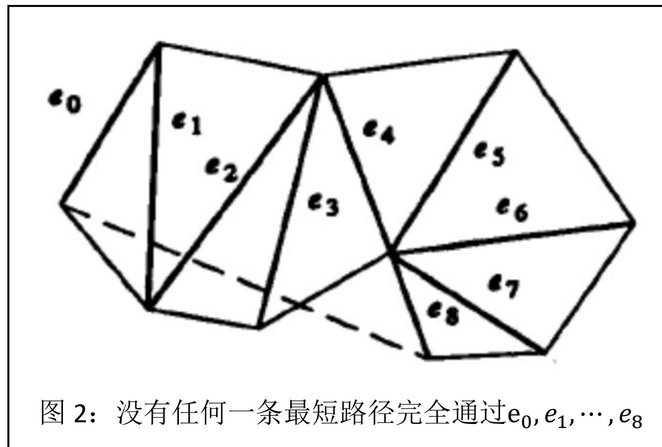


图 2：没有任何一条最短路径完全通过$e_0, e_1, \cdots, e_8$

为了处理这种情况，我们可以在每条边上维护如下一个三元组$(e_i, I_{e_i}, Proj_{e_i}^{I_{e_i}})$。其中$I_{e_i}$表示完成旋转展开后，在$f_i$所在平面的坐标系中，源点 S 的像的坐标。$Proj_{e_i}^{I_{e_i}}$表示$I_{e_i}$在$e_i$上的投影。所谓投影是指$e_i$上的一段闭区间，这一区间中的任何一点到源点 S 的测地线都可以被展成直线段。我们可以很容易地从$(e_{i-1}, I_{e_{i-1}}, Proj_{e_{i-1}}^{I_{e_{i-1}}})$计算出$(e_i, I_{e_i}, Proj_{e_i}^{I_{e_i}})$，具体说来，$Proj_{e_i}^{I_{e_i}}$可以看成是在点光源$I_{e_i}$的作用下，$Proj_{e_{i-1}}^{I_{e_{i-1}}}$形成的阴影与$e_i$的交，如图 3 所示。

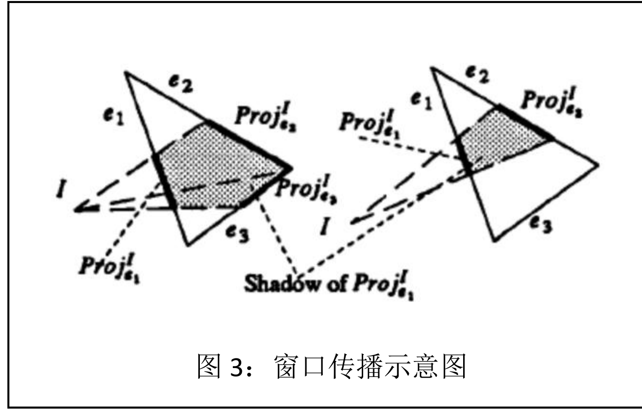


图 3：窗口传播示意图

上述从从$(e_{i-1}, I_{e_{i-1}}, Proj_{e_{i-1}}^{I_{e_{i-1}}})$计算$(e_i, I_{e_i}, Proj_{e_i}^{I_{e_i}})$的过程即为窗口传播。容易看出，窗口传播是以旋转展开为基础的。基于这两种技术，我们可以得到 CH 算法的基本框架，如图 4 所示。

```
I.  root := S;
    for all the edges e opposite to S do
        insert (e, S, e) as root's child;
    /* e is an edge of the face containing S,*/
    /* S is a vertex after triangulation.    */

II. for i := 1 to N do
    /* N is the number of the faces */
        for all the leaves (e, I, Proj_e^I) at the ith level do
            unfold I to Ī;
            /* Ī is co-planar with ΔABC,*/
            /* the shadowed face of e   */
        for e' := AB, CA do
            calculate Proj_e'^Ī;
            if Proj_e'^Ī nonempty then
                insert (e', Ī, Proj_e'^Ī) as
                (e, I, Proj_e^I)'s child;
```

图 4：CH 算法基本框架

3.2 “one angle, one split”原则

图 4 中给出的算法有可能达到指数量级的运算复杂度，因为所有阴影区域覆盖对角的窗口都会有两个子窗口。而使 CH 算法复杂度控制在$O(n^2)$的关键在于利用“one angle, one split”原则对两个子窗口进行剪枝，使得至多只有一个子窗口留下来。“one angle, one split”原则的原始表述如图 5 所示：

Lemma 1: *Given two nodes n_1 and n_2 on the same edge CB of ΔABC, whose shadows cover A, the vertex of $\angle CAB$, at most one of them can have two children which could be used to define a shortest sequence.*

图 5：“one angle, one split”原则

但是 CH 算法并没有明确地说明那个子窗口应该留下来，[3]中对这个原则的证明也是含糊不清的。我们在实验中提出了如下的基于“one angle, one split”原则的子窗口剪枝策略。

（one angle, one split 剪枝）

设$(BC, I_1, Proj_{BC}^{I_1})$, $(BC, I_2, Proj_{BC}^{I_2})$是 BC 边上阴影区域可以覆盖顶点 A 的两个窗口。AB_1是$(BC, I_1, Proj_{BC}^{I_1})$在 AB 边上的子窗口，$AB_2$是$(BC, I_2, Proj_{BC}^{I_2})$在 AB 边上的子窗口，若满足如下条件，则$AB_2$可以被剪枝：

1. $AB_2 \subseteq AB_1$
2. $\|I_1A\| \leq \|I_2A\|$
3. $\|I_1B_2\| \leq \|I_2B_2\|$

3.3 进一步的优化策略

虽然 CH 算法具有$O(n^2)$的算法复杂度，但在实际应用中，CH 算法的执行效率远低于 MMP 算法，尽管后者有着$O(n^2 \log n)$的算法复杂度。[1]中对这一现象进行了分析，认为 CH 算法执行效率低下的主要原因在于 CH 算法产生了过多的无用窗口。于是，[1]提出了如下的剪枝策略，减少了生成的无用窗口数量，提高了算法的运行效率。

THEOREM 3.2. *Let w be a window that enters $\Delta v_1v_2v_3$ through edge $\overline{v_1v_3}$. Assume that d_1, d_2, d_3 are respectively the minimum-so-far distance at the three vertices v_1, v_2, v_3. Then w can't define a shortest sequence if*

$$d + \|\overline{IB}\| > d_1 + \|\overline{v_1B}\|,$$

or

$$d + \|\overline{IA}\| > d_2 + \|\overline{v_2A}\|,$$

or

$$d + \|\overline{IA}\| > d_3 + \|\overline{v_3A}\|.$$

于此同时，[1]中还建议将所有窗口按照它们到源点的测地线距离进行排序，使用类似于 Dijkstra 算法的流程利用优先级队列由近及远地进行窗口传播。虽然这使得算法的复杂度从$O(n^2)$提升到$O(n^2 \log n)$，但[1]中的实验表明，采用这种策略可以大幅提高算法的运行效率。

另外，为了处理图 2 中测地线有可能变成折线的情况，以及所谓的 saddle point，[1]中建议对这两种点进行扩展。即将这两种点作为伪源点，从伪源点出发进行旋转展开和窗口传播。与[3]中关于“one angle, one split”原则的阐述相似，[1]中并没有给出这两种点的判断依据。对 saddle point 而言，我们可以使用[2]中的判据，即该点的邻接角之和大于 2π。但对类似于图 2 中的边界点的点而言，我们还没有找到合适的判据。

我们当前的算法会将所有顶点作为伪源点进行窗口扩展。一旦算法找出了源点 S 到某一顶点 V 的测地线，我们就会将 V 作为伪顶点进行窗口传播。显然，这样做会产生很多无用窗口，从而使算法的效率下降。为了尽量弥补这一缺陷，我们设计了如下的算法终止条件：

(算法终止条件)
一旦优先级队列弹出的顶点伪窗口数等于三角网格的顶点数，算法终止

通过采用这一终止条件，我们的算法避免了对剩余的大量无用窗口进行窗口传播，从而在一定程度上弥补了对所有点进行扩展这一缺陷。保证了算法的实际运行效率。

4. 实现细节

算法永远是纯粹而美好的。但当我们在计算机上实现算法时，由于各种各样实际条件的制约，以及各种各样的边界情况的出现，由算法转化的程序就会变得相对复杂，相对丑陋。也正因如此，我们才有必要说明我们觉得在实现过程中需要注意的问题，为后来者提供一些参考。

4.1 关于机器误差

在计算几何中，我们可以将很多几何关系判断（如共线、共面、两点是否在某一直线的同一侧等）通过向量运算转化为与 0 的比较(大于 0，小于 0，等于 0)。并且一般来讲，这种向量运算均为浮点数运算。但是，计算机在存储和运算浮点数时会不可避免地引入机器误差，从而影响最终的判断结果。为了尽量克服机器误差带来的影响，我们采用了如下策略：

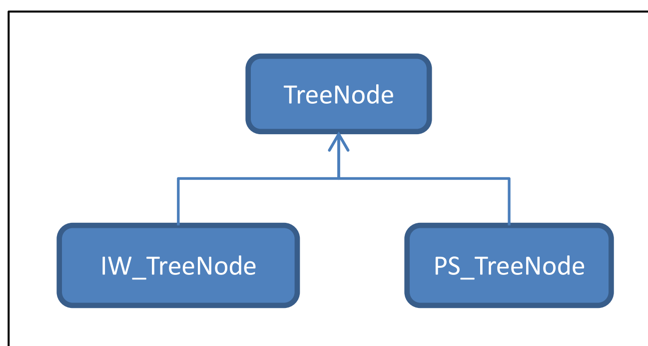
a. 使用双精度浮点数进行运算
b. 将原始模型文件放大到一定的倍数
c. 使用自定义的比较器进行比较，自定义的比较器如下图所示

```
#ifdef _WIN32
    const double PRECISION=1E-5;
#endif

template<typename T>
struct GeoComparator
{
#ifdef __linux__
   static const double PRECISION=1E-5;
#endif
   static bool Equal(const T &v1, const T &v2)
   {
       return abs((double)(v1-v2))<PRECISION;
   }

   static bool Less(const T &v1, const T &v2)
   {
       return v1<v2-PRECISION;
   }

   static bool Larger(const T &v1, const T &v2)
   {
       return Less(v2, v1);
   }

   static bool LessEqual(const T &v1, const T &v2)
   {
       return !Larger(v1, v2);
   }

   static bool LargerEqual(const T &v1, const T &v2)
   {
       return !Less(v1, v2);
   }
};
```

4.2 基本数据结构

4.2.1 窗口数据结构

本实验中的窗口有两种，即与各个顶点对应的伪源窗口(PS_TreeNode)和每条边上的间隔窗口(IW_TreeNode)。考虑到这两种窗口都会进入优先级队列，且又有一定的公共属性，因此，我们设计了如下的层次结构：



另外，考虑到在计算测地线时我们需要经常执行如下操作：

a. 遍历某一条边所对的顶点

b. 遍历某一顶点的邻居顶点

c. 遍历某一顶点的邻接边

我们在边对应的结构体中加入了 set<int> opposite_vertexs 这一属性，在点对应 PS_TreeNode 结构体中加入了 set<int> opposite_edges、set<int> adjacent_vertexs 这两个属性。这样，通过在读取模型文件时进行预处理，维护以上三个属性，我们就可以在计算测地线时很方便地执行上述操作。窗口定义的具体代码如下所示：

```
struct TreeNode
{
    NodeType type;

    double d;
    int depth;

    TreeNode *parent;
    set<TreeNode*> children;

    TreeNode(NodeType t) : type(t), d(DBL_MAX), depth(0), parent(NULL) {}
    virtual ~TreeNode() {}

    virtual double distance()const=0;
};

struct IW_TreeNode : TreeNode
{
    int idx;
    Edge *e;

    Vec3<double> s_img;
    Pair<double> proj;

    IW_TreeNode() : TreeNode(INTERVAL), idx(-1), e(NULL), s_img(-1., -1., -1.), proj(-1., -1.) {}
    ~IW_TreeNode() {}

    virtual double distance()const;
};

struct PS_TreeNode : TreeNode
{
    int idx;
    Vertex *v;

    set<int> opposite_edges;
    set<int> adjacent_vertexs;

    map<Edge*, IW_TreeNode*> occupied_windows;
    map<Edge*, Vec3<double> >occupied_imgs;

    PS_TreeNode() : TreeNode(PSEUDO_SOURCE), v(NULL) {}
    ~PS_TreeNode() {}

    virtual double distance()const { return d; }
};
```

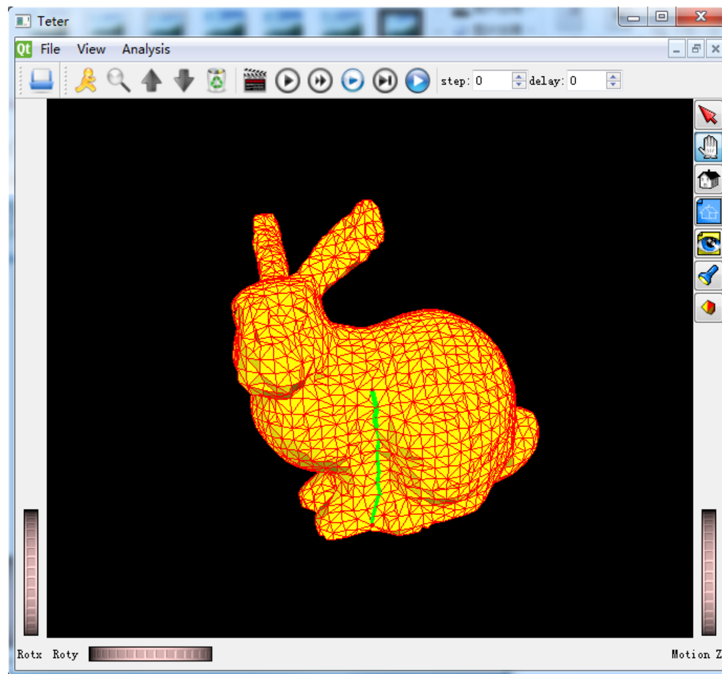
4.2.2 基于堆和 map 的优先级队列

在计算最短路径时，我们需要改变优先级队列中有些元素的优先级。为了使这一操作能够在$O(\log n)$的时间内完成，同时不影响其他操作的时间复杂度，我们使用了 map 这一数据结构维护优先级队列中每个元素在优先级队列中的位置信息。同时，为了能够最大程度上复用我们的代码，我们使用了模板技术实现了优先级队列，其对应的类定义如下所示：

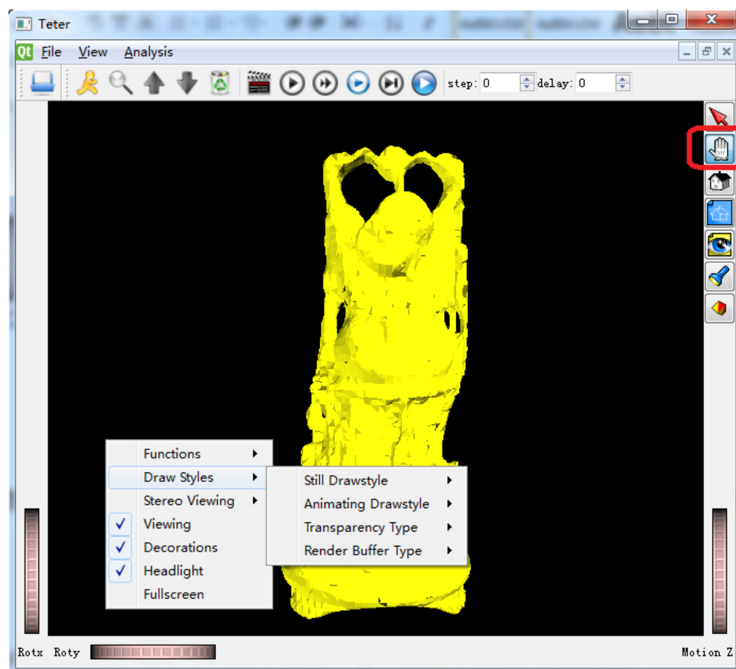
```
template<typename T>
struct Less
{
    bool operator()(const T &t1, const T &t2)const
    {
        return t1<t2;
    }
};

template<typename T, typename Compare=Less<T> >
class Priority_queue
{
   Compare compare;

   vector<T> _t_arr;
   map<T, int> _inv_idx;
   int _size;

   void minify_heap(int idx);
   void shift_up(int idx);

public:
   Priority_queue() : _size(0)
   {
       T pivot;
       _t_arr.push_back(pivot);
   }
   Priority_queue(const vector<T> &t_arr);

   void clear();

   T remove();
   void remove(const T &elem);
   void insert(const T &elem);

   bool is_empty()const { return _size==0; }
   void debug_output()const;
   //add by jlb
   int getSize(){return _size;}
   vector<T> getData()const{return _t_arr;}
};
```

5. 演示程序说明

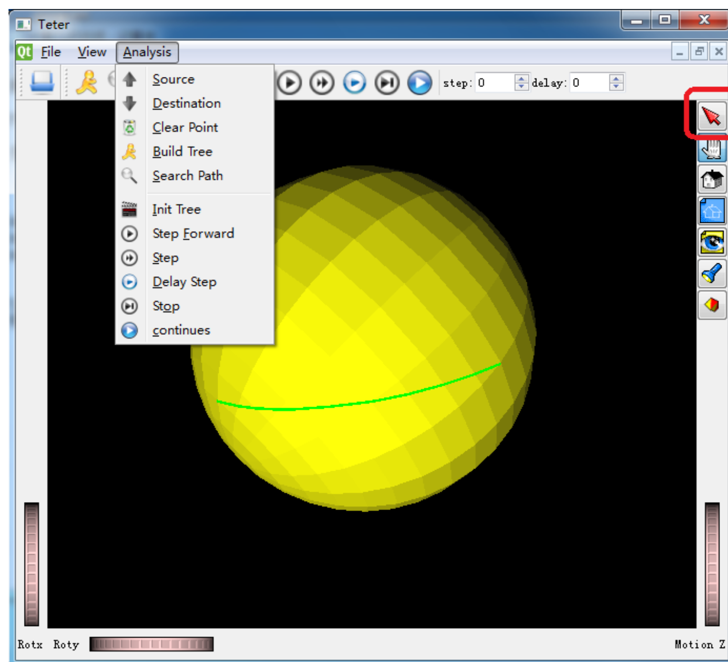
为了更好的演示测地线的构造过程，我们编写了一个演示程序。整个演示程序是基于 Qt 和 Coin3D 这两个库。其总体效果如下所示：



当我们将模型读入内存后，我们可以选择子窗口右侧的手型图标来旋转模型，滑动鼠标滚轮可以缩放模型，而Ctrl+左键则可以对模型进行拖动。而在非“指针”状态下，右键->Draw Styles->Still Drawstyle 则可以选择模型的显示风格。如下图所示：



当我们需要选择源点和目的点进行测地线的计算时，我们可以在选择 Analysis -> Source (Destination)之后，点击子窗口“指针”选择源点(目的点)。源点用小圆球标记，目的点用小正方体标记。通过选择 Analysis->Clear Point 则可以清除源和目的点的选择。接着选择 Analysis->Build Tree 则可以生成用于查询测地线的数据结构，最后单击 Analysis->Search Path，程序即可显示源到目的点的测地线。如下图所示：



如果我们需要观察用于查询测地线的数据结构的生成过程，在选择了源点后，我们可以单击 Analysis->Init Tree 来初始化查询测地线的数据结构，然后可以通过单击 Analysis->Step Forward/Step/Delay Step 观查构造数据结构过程中生成的窗口。这三个菜单项的意义如下：

a. Step Forward：

步进一个迭代步，对应于 build_geodesic_structure 中 while 的一次循环。

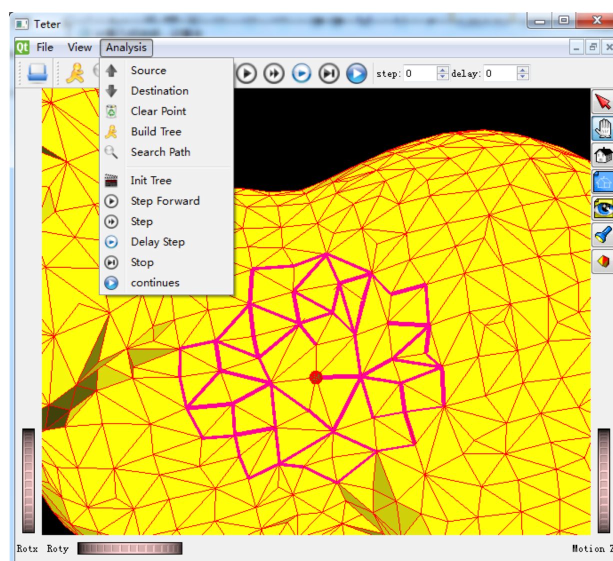
b. Step：

步进指定个迭代步，当主界面中的 step 为 0 时，会弹出窗口要求输入。

c. Delay Step：

指定每秒步进的次数（实际为步进指定步数后 sleep 1 秒）。当主界面中的 delay 为 0 时，会弹出窗口要求输入。

另外，若单击 Analysis->Stop，则程序会使 Delay Step 暂停。而单击 Analysis->continue 后，程序便构建完剩下的数据结构。如下图所示：



6. 实验总结及未来工作展望

在本次实验中，我们对测地线算法进行了调研，并实现了[1]中的改进 CH 算法，同时，对原始论文中一些没有阐述清楚的地方进行了研究。另外，我们编写了演示程序来直观地演示算法构建测地线距离的过程。

我们的实验也有不足的地方。我们的验证环节比较薄弱。我们针对球和立方体这两种情况进行了正确性验证，也使用了一些规模较大的数据进行压力测试(如 budda_simply.ply, $n > 7000$)，但总体上讲还是比较欠缺。最近我们针对测地线算法请教了刘永进老师，他给我们介绍了一些关于测地线算法的 benchmark。以后可以用这些 benchmark 对我们的实现进行进一步的验证。

对所有顶点进行扩展是当前实现的一个主要问题。以后可以进一步考虑确定扩展顶点的判别条件，从而进一步提高算法效率。

测地线算法是图形学中十分基础的一个算法，它可以衍生出很多应用。但若将测地线算法依托某一应用展现出来，却又在一定程度上湮没了测地线算法的基础性和重要性，因此我们没有展示测地线的应用。以后可以考虑实现一系列基于测地线的应用，从而更好地体现出测地线算法的应用价值。

7. 参考文献

1. Shiqing Xin and Guojin Wang, 2009, Improving Chen and Han’s Algorithm on the Discrete Geodesic Problem, ACM Transactions on Graphics
2. Mitchell, Mount and Papadimitriou, 1987, The discrete geodesic problem, SIAM
3. Chan and Han, 1990, Shortest paths on a polyhedron, SoCG
4. Surazhsky.V, Surazhsky.T, Kirsanov, Gortler, and Hoppe, 2005, Fast exact and approximategeodesics on meshes, ACM Transactions on Graphics
5. 微分几何，苏步青
6. KANEVA, B. AND O’ROURKE, J. 2000. An implementation of Chen & Han’s shortest paths algorithm. In Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG)

8. 写在最后

通过这次大实验，我们的工程能力得到了锻炼，在团队合作方面也有了很大进步。感谢我们 A 组的所有成员，如果没有所有成员的努力与奉献，没有彼此之间的合作与沟通，我们不可能完成这项实验。再次对各位表示衷心的感谢。