Consistent Mesh Parameterizations

——2003《计算几何》课程设计

蔡晓华，严寒冰，韩东，黄其兴

前言、报告概述

本次课程设计主要是想参考[Praun 01]，实现“一致的网格参数化(Consistent Mesh Parameterizations)”。其主要思想是：对一系列模型网格按照一定的基网格拓扑进行重新剖分和重新构建，从而使得这些网格有一致拓扑的良好性质，从而可以做很多有意义的应用。如何在给定基网格和一系列模型网格的情况下，进行拓扑一致的剖分是[Praun 01]中主要讨论的内容。

但为了达到这个目标，还必须完成另外 3 项工作：面片简化（[Garland 97]），参数化（[Floater 97]）和网格重构（[Guskov 00]）。

因此最终，我们完成了这 4 部分内容，并将其分别实现在 2 个程序框架内：cmp 完成了一致网格剖分，Huma 完成了面片简化，参数化，和重构。两部分的程序分别主要由蔡晓华和严寒冰完成，整个小组共同讨论算法原理和实现细节，并制作基网格，调测最终结果等。

上传文件中，cmp 目录下是程序 cmp，Huma 目录下是程序 Huma，data 目录下是模型数据（其中 basemesh 目录是基网格，chetah 和 hema 目录是 2 组模型，这两个目录之下的 15_base 和 24_base 子目录分别表示对应 15 点基网格和 24 点基网格的例子，test 目录是 1 个简单立方体测试模型），paper 目录下是参考的文章，others 目录下是一些结果和我们的一个简要的 ppt。

由于我们的实验有 2 个框架，而 2 个框架之间相对独立，以文件交换的形式传递数据，所以，以下的实验报告将分为：

一、问题背景

二、系统构成及简介

三、cmp 框架

四、Huma 框架

五、总结

六、参考文献

而，在三和四的每个框架中，分别讨论各自的：

1. 框架简介（背景 & 功能 & 输入输出 ）
2. 算法及原理
3. 系统设计 & 数据结构 & 输入输出
4. 测试、对比结果
5. 实现过程中遇到的问题及解决对策
6. 没有解决的问题

一、问题背景

数字几何处理（Digital Geometry Processing）是一个有关构建几何的信号处理算法的领域。由于曲面的非欧氏（non-Euclidean）特性，DGP算法的构建基本上要比构建经典信号处理算法难得多。声音（1维），图像（2维），视频（3维）现在都已经可以参数化到欧氏空间了，例如：一幅图像可以用覆盖平面一定区域的发光函数（irradiance function）给出。此外，图像总是用笛卡儿坐标系采样的。因此，例如计算两幅图像的平均值或者计算它们的归一化差等简单操作是非常容易的。

可几何问题并不是这样的。这有2个原因：（a）几何的非欧氏特性（b）通常表达几何的网格的采样模式和连接关系不尽相同。DGP算法涉及的多个模型需要一个共同的参数化表达（parameterization）和采样模式（sampling pattern）。作为许多算法的一个基本步骤，例如从纹理映射（texture mapping）和外形融合（shape blending），到物理模拟（physical simulation）、压缩（compression）和数据分析（data analysis）等，对一个单独的模型计算一个全局参数化并且重构（remeshing），本身是非常困难的问题。

“一致的网格参数化”（Consistent Mesh Parameterizations，简写为CMP）（[Praun 01]）讨论的是一组模型的参数化问题。当这一组模型考虑各自特征并且共有一个基模型时，就认为它们是一致参数化的。注意：这里所有的模型必须有同样的亏格。而实际在[Praun 01]中讨论的是零亏格的，可定向的流行曲面。

“一致的参数化表达”在给定所有模型之间的点的对应关系后，使得我们能够用同样的连接对每一个模型进行重构。因此，每一个模型网格上的点唯一地对应到其他网格上。这样，从一个模型到整个模型几何的某些应用，例如：从N模型的外形融合到纹理、细节或动画控制等的属性传递等等都有可能实现。如图1所示，许多模型，可能在几何上非常不同，通过参数化到同一个基模型上，并且用一致的连接重构，可以进一步的DGP处理所使用。此外，许多新的需要同时涉及若干模型的几何处理算法，例如：主分量计算（principal component

computations）等也第一次成为可能。如图2所示，通过一系列人脸模型，可以求得其平均人脸，这是一种主分量分析的一种应用。

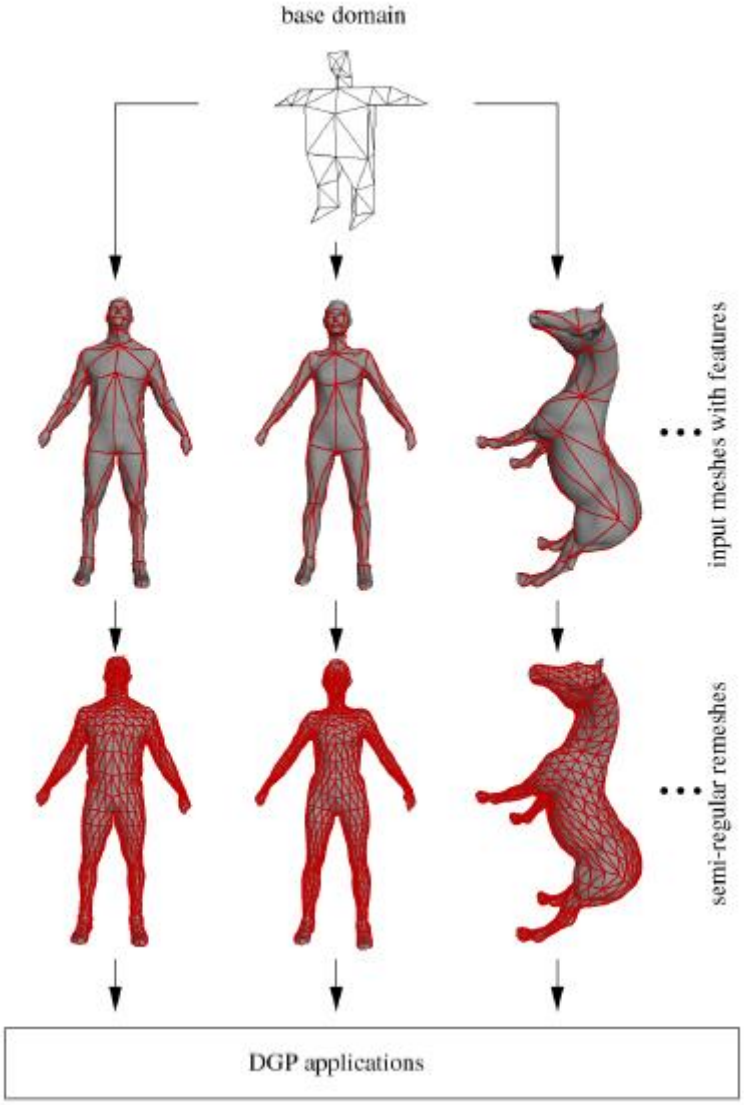


图1 多个模型通过与基网格的点（边）映射从而实现参数化。引用自[Praun 01]。

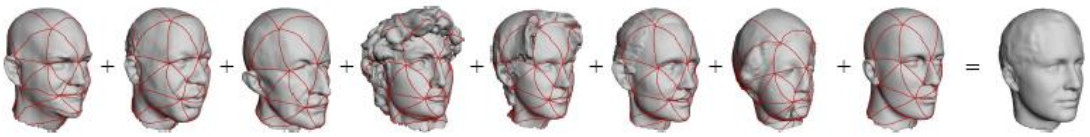


图2 通过一系列人脸模型求得其平均。引用自[Praun 01]。

已有的参数化方法存在一个问题：即使是非常相近的模型，可能最终对应到不同的基模型上，从而导致不一致的参数化。这并没有为什么可以解释，“一致的参数化方法”克服了这个问题。

二、系统构成及简介

“一致的网格参数化”中，首先给定基网格和模型网格的对应关系，例如，模型网格上选取特征点对应基网格顶点。然后计算了它们的一致的参数化剖分。得到参数剖分结果，就可以通过同一的连接关系进行重构。最后可以在重构的网格上做各种应用。

因此，这个课程设计可以首先大致划分为：数据准备（基网格，模型网格，对应点），模型网格剖分，参数化重构，应用，等4部分。

1．系统模块分析

对各部分的分析和考虑如下：

模型网格剖分（系统模块3）是[Praun 01]主要讨论的问题，也就是：如何在给定基网格和模型网格对应关系的情况下，对模型网格进行剖分。这部分内容是本课程设计的主要问题，较复杂和繁琐，我们在程序框架cmp中加以实现。此时得到的结果仅仅是模型网格上的剖分结构。

参数化（系统模块4）重构（系统模块5）由程序框架Hema实现。对于cmp输出的剖分结果，做局部参数化，并且按照剖分的连接关系重构模型。参数化和重构可以单独分开，但我们还是将其放在同一程序框架中，方便调测。此外，cmp框架中对模型网格加以剖分时，为了有更好的效果，参考[Praun 01]的内容，需要首先计算N维参数化（系统模块2）做为依据。考虑到N维参数化和一般（2维）参数化虽然概念上不同，但实际实现有很大的相似之处，所以，这部分内容也由Huma框架实现。

实际问题中，一般的模型都比较大，我们原有的chetah（猎豹）和hema（三角恐龙）模型都是5000个顶点，约10000面片，规模太大，在后续操作中，时间和空间复杂度都过大，无法计算。参考[Praun 01]的内容，采用简化后模型进行计算，因此需要模型简化功能（系统模块1）。这部分功能我们实现了2个独立的程序（最初考虑分2组，并行这个课程设计，然后各部分采用比较稳定的程序模块。模型简化是最开始的工作，由严寒冰和蔡晓华分别做了2个程序。之后由于

时间比较紧，网格剖分和参数化重构只能串行实现），提交的是集成在tracing中的版本。实际采用的chetah和hema都是1200顶点，约2400面片。

应用（系统模块6）也由tracing完成，在重构（remeshing）过程中，加以控制，得到的重构网格就可以是某个具体应用。例如：对于cheta和hema的剖分结果，参数化后，映射回网格时采用原网格的线性组合，当两者比例系数为1.0和0.0是，得到的就是cheta的重构网格，为0.0和1.0时，就是hema的重构网格，而0.5和0.5时，就是半cheta半hema的重构网格了。

此外，基网格由手工完成，模型网格和基网格的对应关系也手工完成。

通过上述分析，我们的整个系统程序模块6个，包含在2个框架内，按实验顺序，分别为：

（1） 系统模块1——模型简化，包含在Huma框架内。输入为复杂模型（obj后缀）；输出为简化模型（obj后缀）；

（2） 系统模块2——N维参数化，包含在Huma框架内。输入为模型网格（obj后缀），基网格和对应关系文件（bsm后缀）；输出为模型网格Floater参数（par后缀）。

（3） 系统模块3——模型网格一致剖分，cmp框架。输入为模型网格（ojb后缀），基网格（obj后缀），对应点关系（vmp后缀），模型网格参数（par后缀），后处理文件（pst后缀）；输出为剖分结果（bep文件），和一些log文件（后缀log）；

（4） 系统模块4——2维参数化，包含在Huma框架内。输入为剖分结果（bep后缀）；输出为内部数据，没有文件数据。

（5） 系统模块5——网格重构，包含在Huma框架内。输入为系统模块4的内部数据；输出为重构网格，可以直接查看。

（6） 系统模块6——应用，包含在Huma框架内。是系统模块5的扩展。输入为系统模块4的内部数据；输出为重构网格，可以直接查看。

其中，后缀obj表示object，bsm表示base mesh，par表示parameters，vmp表示vertex mapping，pst表示posthandle，bep表示base mesh edge and patch。

2．系统输入数据

在实验中，系统模块1——模型简化，用于生成模型网格，独立于其后的内容。我们于最初阶段，生成2个简化模型1200_chetah.obj（1200顶点，2396面片）和1200_hema.obj（1200顶点，2396面片）之后，不再被使用。而这两个简化后的模型作为整个系统输入——模型网格。这两个模型都采用普通的点面结构定义。

基网格是手工绘制的，我们制作了2个基网格，分别是：15v_base.obj（15顶点，26面片）和24v_base.obj（24顶点，44面片），采用点面结构定义。

此外还需要相应的点对应文件来表示模型网格如何对应到基网格上。

由于基网格只需要表示拓扑关系，实际形状是没有意义的，所以，框架Hema将基网格和点对应文件合成一个文件bsm。而cmp中，因为调测需要，要求显示基网格，判断其性质，所以，给以基网格顶点设置了伪坐标，需要输入obj格式的基网格模型，和另外的点对应文件vmp。

最后，对于实际问题，cmp还需要一些后处理手段，需要后处理pst文件。

所以，对于整个系统，输入文件包括：

1）模型网格（obj）：1200_chetah.obj和1200_hema.obj，输入到tracing和cmp；

2）基网格和点对应关系（bsm）：输入到tracing。

3）基网格（obj）：15v_base.ojb和24V_Basechetah.obj，输入到cmp；

4）点对应关系（vmp）：输入到cmp；

5）后处理文件（pst）：输入到cmp。

系统的输出为直接的显示结果。

3．实际系统构成

综合上述两个部分，系统构成可如下框图（图3）表示：

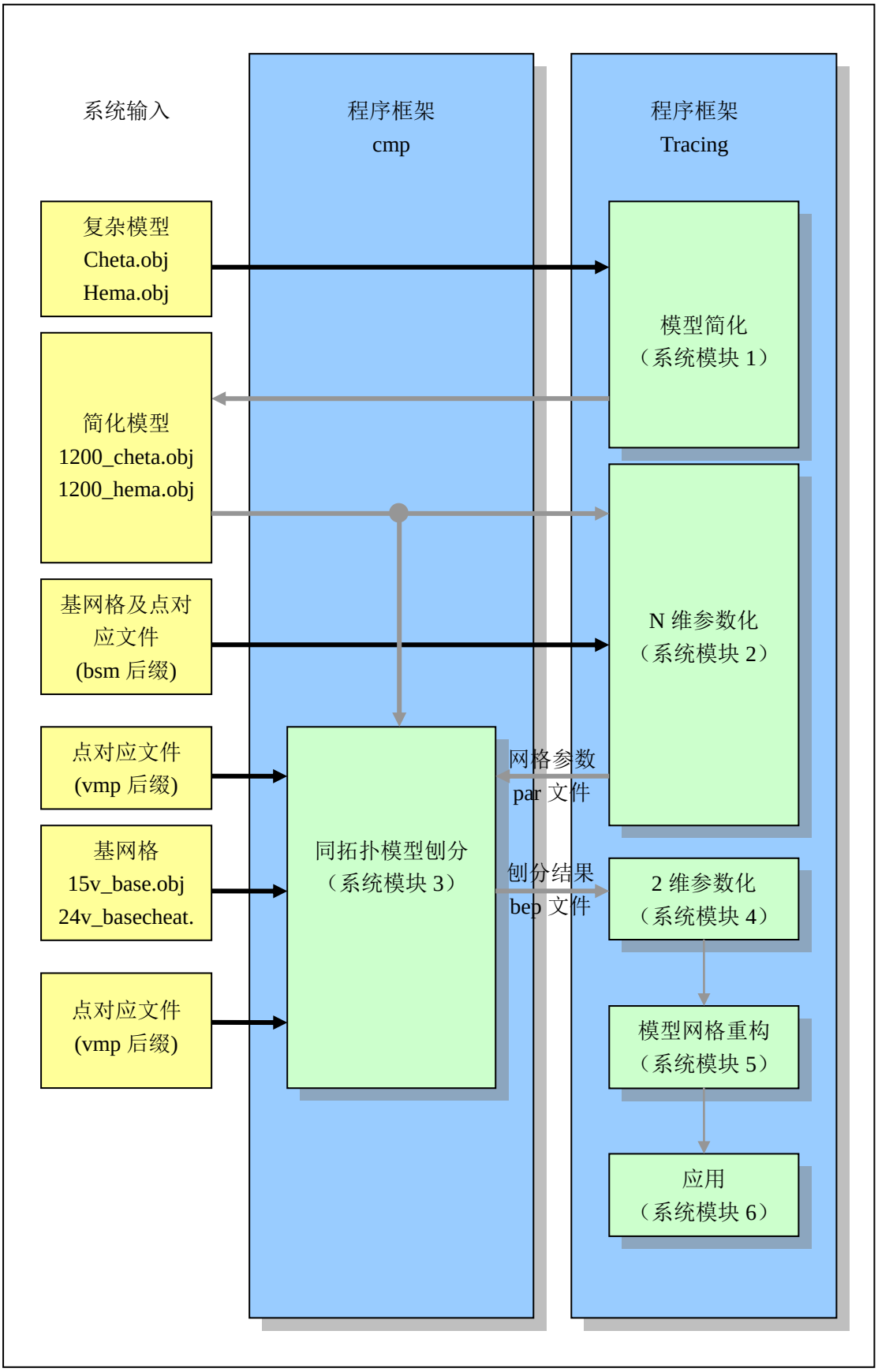

系统输入
程序框架
cmp
程序框架
Tracing
复杂模型
Cheta.obj
Hema.obj
模型简化
（系统模块 1）
简化模型
1200_cheta.obj
1200_hema.obj
基网格及点对
应文件
(bsm 后缀)
N 维参数化
（系统模块 2）
点对应文件
(vmp 后缀)
网格参数
par 文件
基网格
15v_base.obj
24v_basecheat.
同拓扑模型刨分
（系统模块 3）
刨分结果
bep 文件
2 维参数化
（系统模块 4）
点对应文件
(vmp 后缀)
模型网格重构
（系统模块 5）
应用
（系统模块 6）

图3 系统框图

三、cmp 框架

程序框架 cmp 是参考文献[Praun 01]完成，也是本次课程实验的主要内容之一，是一个较为复杂的启发式算法，目的是对模型网格进行同拓扑剖分。以下将按照报告概述中提到的顺序展开。

1. 框架简介（背景 & 功能 & 输入输出 ）

这个程序在上传的 cmp 目录下。程序的基本框架是以前同学留下的（稍作修改），包括 Common，Display 和 Model 等三个子目录，其他部分均为实验中编写的，主要的功能部分在 CmpExt 子目录中（ext 表示 extend），此外还有交互等写在 cmp 根目录文件中。用工具统计，总共 10110 行代码，此次实验部分为 6837 行代码。

这个程序功能是：对模型进行同拓扑剖分容，基本上实现了[Praun 01]中提到的每个问题。

程序的输入包括 4 个文件：模型网格（obj 文件，菜单 File/Open 选择），基网格（obj 文件，菜单 CMP/Load Base Mesh 选择），对应点关系（vmp 文件，菜单 CMP/Load Mapping Data 选择），网格点参数（par 文件，菜单 CMP/Load Vertex Para 选择）和后处理文件（pst 文件，菜单 CMP/Load Post Handle Data）。

程序的输出是模型剖分结构（bep 文件），供 Huma 框架做参数化重构，以及一些 log 文件，供调测用。

2. 算法及原理

Cmp 的原理中有 3 个问题：1）什么是一致的参数化，2）如何保证一致的参数化，3）如何更优，4）算法流程。

1）什么是一致的参数化

前面的概述等部分中已经提了很多“一致的网格参数化”概念，这里结合

实际操作，来进一步说明。

（1）什么是参数化

首先什么是参数化？简单的说，参数化就是将曲面映射到二维平面上（也有一些方法将曲面映射到球面上，因为球面也具有很好性质，和很多可用的方法），这样就可以利用欧氏空间的性质，进行很多处理，例如平均，变形等等。另外，一些经典的二维图像的处理方法，例如图像处理中的去噪声，卷积等等，也可以加以应用。参数化在映射时，需要面片的边界绑到的二维平面的边界上（例如正方形，圆，三角型等等），然后将面片中间部分按一定的方法映射到平面的边界内部，形象的说，也就是“拍平”。然而对于大且复杂的曲面，是不一定能够“拍平的”，因为拍平之后会有重叠的部分，例如将球拍平，所以，此时需要将其刨开。

（2）一致的网格参数化

“一致的网格参数化”是一种局部参数化。它将一个模型网格（复杂曲面）首先对应到一个简单的基网格上，使得基网格的每一个面片（patch）包含模型网格的许多小面片（facet）。由于，基网格的每个面片就是一个二维平面，所以，这样每个 patch 包含的 facet 必然可以进行参数化了——这种局部参数方法，就是对网格一部分一部分地分别求解和参数化。形象地可以认为，在模型网格上剪了若干刀（基网格地边数），使得网格成为若干块（基网格面片数），然后一一进行参数化。——与之相对应的还有一些全局参数化方法，例如，我们组朱旭平使用 Skeleton 方法进行的刨分，这种方法只剪一刀，然后将整个网格都映射到同一个平面地边界内。

我们将模型网格上对应到基网格顶点的点称为特征点，通过特征的选取，可以使得重构的参数化能尽量好地反映原模型地特点，譬如，对人脸而言，要将眼睛，鼻子，嘴等定义为特征点，这样可以体现不同人脸的特点。

利用“一致的网格参数化”，我们将不同地模型，例如豹子和三角龙，人和马等等，都对应到相同地基网格上，那么不同的模型之间就有了对应关系（特征点之间地对应，patch 之间地对应），那么对不同模型地各种 DGP 操作就有了依据和可能。对一组模型，还可以进行主分量分析。

（3）实际问题中的描述

在实际求解问题中，已知的是模型网格，基网格，特征点对应关系。这里考虑的都是三角形网格。需要得到是模型网格上和基网格有一致拓扑关系的剖分结构，也就是同拓扑剖分。同拓扑指：点，边，面分别一一对应，且点的邻接关系，和点相关的边及边序关系，面的边序关系也分别一一对应。

引入数学语言描述：

定义网格 $M=(P,K)$，其中 $P=\{p_i=(x_i,y_i,z_i)\in R^3 \mid 1\le i\le N\}$，而 K 是包含所有拓扑关系（例如邻接关系）的抽象单纯联合体（abstract simplicial complex）。联合体 K 是一系列子集 $\{1,\ldots,N\}$。每个子集包含 3 中类型：点 $\{i\}$，边 $\{i,j\}$ 和面 $\{i,j,k\}$。当 $\{i,j\}\in K$ 时，则两个点 $\{i\}$ 和 $\{j\}$ 是相邻，对于 $\{i\}$ 的 1-ring 相邻的点指来自集合 $V(i)=\{\{j\}\mid(i,j)\in K\}$ 的点。

复杂的模型网格定义为 $N_j=(Q_j,L_j)$，基网格定义为 $B=(P,L_o)$。问题就是已知模型网格 N_j 和基网格 B，并且给定 $N_0=(Q_0,L_0)$，其中 $Q_0\longleftrightarrow P$，$|Q_0|=|P|$，求得从 L_j 到 L_0 的对应关系 $L_0=f(Lj)$？

在实际问题中，就是给定了模型网格，基网格，和模型网格上对应到基网格的一系列特征点，要在模型网格上，求得基网格的 edge 所对应的模型网格的 edge 组成的 Curve，基网格的 facet 所对应的模型网格的 facet 组成的 patch。

2）如何保证一致的参数化

这里要说明如何求得参数化剖分，并保证其一致（同拓扑）。

（1）如何求得参数化

求参数化的方法，也就是求得网格的剖分结构，也就是求得 Curves 和 Patches。我们首先 tracing curve，就是在模型网格是进行遍历，求得其中满足同拓扑要求的 curves，进而，再以 curves 为边界，求得其包含的 patches。

Tracing curve 的方法，在[Praun 01]中提到的是一种叫“brush fire”的算法，可是我们没能找到相应的资料，根据其描述和用到的所写“BFS”，我们认为是一种类似图的广度优先遍历算法：从起点开始，用一定的测量准则（测地线距离），

按广度优先的原则遍历网格，一旦访问到终点就结束。

（2）保证同拓扑的条件

仅按照 brush fire 方法求得剖分结构往往不满足同拓扑结构的，为了保证同拓扑的性质，需要满足三个测试条件：a）相交测试，所有的 curves 只能在特征点相交；b）边序测试，以 curve 相连于某特征的点的其他特征点，应该同基网格的邻点有一致的顺序关系；c）生成树测试，避免将某点孤立起来。

分别具体说明如下：

条件（a）——相交测试。

所有的 curves 只能在特征点相交，这是显然的，否则，curves 之间包围的 patches 会有相交重叠，那么它的面对应关系和基网格就不一样了，因为基网格的面之间是不相交重叠的。

这是一种改进的 brush fire 方法。

条件（b）——边序测试。

图 4 是这个测试的实例。

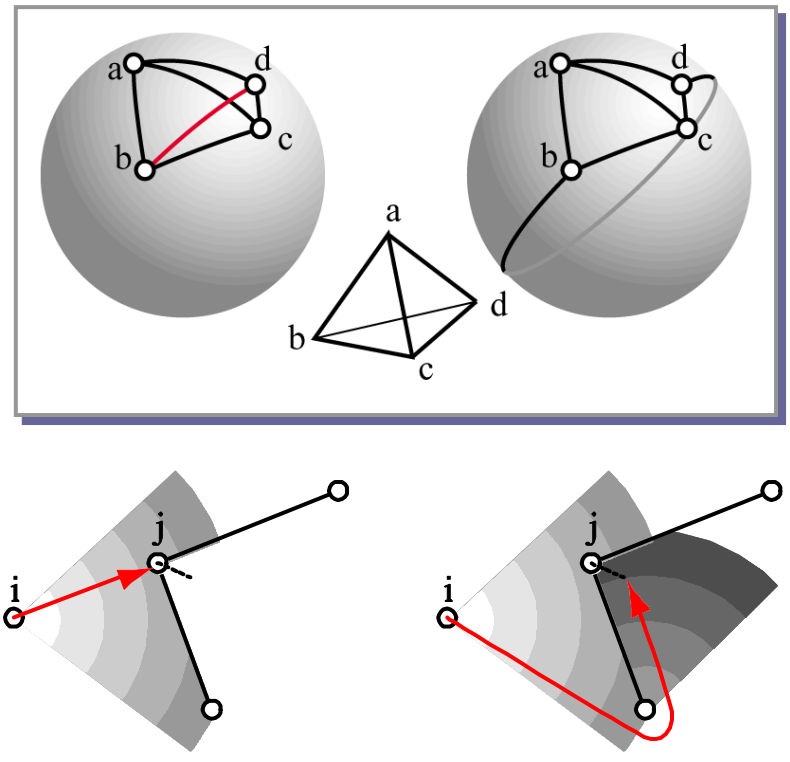


图 4 边序测试

在图 4 上图中，我们搜索一条从 b 到 d 的 curve（假设 bd 先于 ac 被搜索），那么出现了上图中左右 2 中情况，而实际的拓扑情况如上图中下所示。上左图是

错误的，因为，当上左图中，正确的 ac 在被搜索到后，bd 和 ac 就相交了。所以，正确的 curve 应该是上右图中的情况。

造成这种错误的原因是 brush fire 总是容易找到路进长度更短的路径。所以，在 tracing curve 的过程中，无论是出边，还是入边都要考虑其在基网格上对应的边序情况。如图 4 下图所示，左侧红线是 brush fire 第 1 次从 i 搜索到 j，可正确的边序应该是虚线所示的位置，所以，此时应该继续搜索，知道右侧图中，红线再次从 i 到 j。

条件（c）——生成树测试。

由于 curve 总是一条一条被确定，一种不恰当的 curve 顺序可能导致某个特征点被孤立，这样由于 curve 不能相交的限制，将不能找到一致拓扑的刨分界结构。如右图图 5 所示。

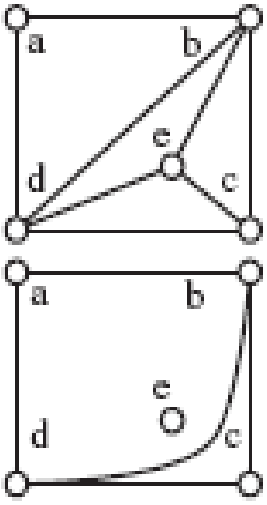


图 5 生成树测试

假设拓扑关系如图 5 上图所示。实际 tracing 的过程如图 5 下图所示。由于 ab，bd，da 先于 cc 被确定，c 被环绕包围，cc 在 curve 不能相交的条件下，将不能被正确找到。解决这个问题的方法是恰当的 curve 顺序，如果 cc 先于 bd 被确定，那么就能正确找到所有 curve 了。

通过观察可以发现，只要形成了环，就可能存在环绕包围的问题，所以，需要先搜索出生成树上，使得没有点被孤立，然后添加剩余的 curve，就不会有这个问题了。——这在[Praun 01]中有证明。

所以需要进行生成树测试，一旦够成环，就要沿后操作。

3）如何更优。

通过 2）的讨论，我们可以在模型网格上得到同基网格拓扑一致的刨分结构了，但这样的刨分可能很差，因为存在一个问题：扭曲（swirl）。

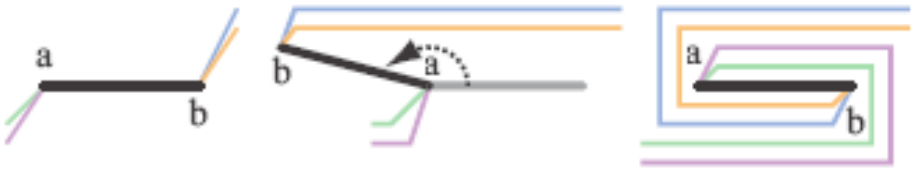


图 6 扭曲问题

考虑上图图 6，从拓扑结构上看，图中 3 个结构没有任何区别，而我们的 brush fire 方法，即使增加了条件（a），（b）和（c）也无法区别。因为网格上，两点间测地线距离相等的路径可能很多，我们无法只通过判断哪个扭曲更少。而，对于图 6 中右图的情况，不难想象其重构结果必然是很变形很大的。所以，我们需要进一步分析。

[Praun 01]提出了公式（1）

$$\sum_{C\in N}\int_{s\in C}\|g'\|+\|g''\|ds \qquad (1)$$

其中，C 表示 Curve，N 表示网格，g 是 C 的弧长函数。整个求和表示对网格上所有 Curve 函数的一阶倒数，二阶倒数求和。这是一个复杂的全局优化问题，因为曲线是非凸，并且混合了离散和连续函数。所以，难以实际求解。

于是采用启发方法。启发的线索包括：a）特征点约束，b）弧长距离最短，c）边的三角形翻转测试。分别说明如下：

启发线索（a）——特征点约束。

原始 brush fire 方法和改进 brush fire 方法都采用测地线距离作为测量。这种方法也就像没有风火势的一样，均匀的向四周扩散。在扩散过程中，并不考虑特征点的约束。所以，可能造成无法避免扭曲。

于是引入 n 维参数距离替代测地线距离。

将网格嵌入 n 维 $\Re^n$ 空间，n 是特征点个数。用 $I_i(k)$ 表示第 i 个特征点对空间第 k 个点的影响，而 $\sum_{i\in n} I_i(k)=1$。用 I_k 表示某个点在参数空间中的值，是一个 n 维向量。

初始时，对于特征点：使得 $I_i(i)=1$ 且 $I_i(\mathit{feature}\ j)=0$，既每个特征点不受其他特征点影响。对于非特征点：使得 $I_k=\sum_{\{l\}\in V(k)} w_{kl}I_l$，其中 $\sum_{\{l\}\in V(k)} w_{kl}=1$，既每个点由周围点的凸组合表示，$w_{kl}$ 是加权系数，不同的加权系数会表现出不同的性质。我们这里采用 Floater 加权系数，所以，这里简称为 n 维 Floater 参数化。

通过求解，可以得到每个点的 n 维解向量。

于是对于从 a 到 b 的搜索曲线 C(a,b)，不再匀速生长，而是以优先级 $P_k(a,b)=1-I_a(k)-I_b(k)$，向优先级小的方向生长。因为 $I_a(a)$ 和 $I_a(K)$ 分别表示

特征点 a 和 b 对 k 的影响，所以，$P_k(a,b)$ 表示其他特征点对 k 的影响，取值最小的路径，就是取其他特征点对这点影响最小的路径。

启发线索（b）——最短弧长约束。

采用的参数化距离，改善了扭曲，但不能本质上避免。所以需要进一步考虑。因为 curve 的顺序很大程度上影响最终效果，所以，我们希望最少扭曲的 curve 首先被确认。那么弧长最短的 curve 发生扭曲的可能性最小，弧长长的 curve 发生扭曲的可能性就大，所以需要对 curve 按弧长进行排序，从短到长的距离来依次确认。

启发线索（c）——边的三角形翻转测试

实际问题中还存在一种直接的可以检测到的扭曲，考虑下图图 7。

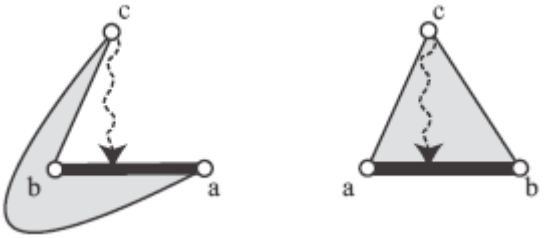


图 7 边的三角形翻转测试

考虑图 7 中的 2 个三角形（灰色区域）左图和右图，它们是拓扑相同的 2 种情况，不同的是 ab 发生了对换。右图的情况明显优于左图。所以，可以搜索从 c 到 ab 的路径，如果找到的路进在三角形 abc 内，如右图，那么认为没有发生翻转，否则这个三角形发生了翻转，如左图所示。

4）算法流程

通过上述分析，我们可以得到以下算法流程：

1. 生成弧长优先级队列（不考虑拓扑一致测试）
2. 当队列不空，取队列首元素（Curve）
3. 生成树测试，失败，则惩罚 后重新插入队列
4. 拓扑测试（相交，边序），失败，则重新生成弧长（需要考虑拓扑一致）
5. 边反转测试，失败，则惩罚后重新插入队列

6. 确认该 Curve，做为已有的限制条件

7. 当所有的 Curve 被确认，以 Curve 为边界搜索 Patch

程序核心就是此流程。

3. 系统设计 & 数据结构 & 输入输出

由于程序内容比较多（总共 1 万多行代买），主要介绍以下内容：1）模型表示，2）Tracing curve 功能的实现，3）界面和操作。

1）模型表示。

输入的模型文件为 obj，是通用的点面模型结构，既用 3 个 float 值表示点的空间坐标，然后用 3 个点索引表示面的构成，这 3 个点按是面的顶点按逆时针顺序排列的。

模型在程序中采用 DECL 结构，即半边结构。定义在类 CMesh 中。包含子类 CVertex（定义顶点），CEdge（定义半边），CFace（定义面）。其主要数据结构，简要说明如下（具体内容可参考\cmp\Model\mesh.h）：

```
class CVertex
{
    Vector3D    m_vPosition;    //点的坐标
    UINT*       m_piEdge;       //从该点发出的 halfedge,要根据点的度数动态创建，按顺时针序排列
    _UINTLIST m_lEdgeList;      //用来构造 m_piEdge 的临时链表
    short       m_nValence;     //点的度数
    bool        m_bIsBoundary; //是否为边界点
    Vector3D    m_vNormal;      //法向
}

class CEdge
```

```
{
    UINT    m_iVertex[2];    //该边的两端点，Vertex0－>Vertex1
    UINT    m_iWedge[2];     //该边指向的两个 Wedge，cmp 中未使用
    UINT    m_iTwinEdge;     //与该边方向相反的另一条边，如果为-1 则该边为边界
    UINT    m_iNextEdge;     //沿逆时针方向的下一条边
    UINT    m_iFace;         //该边所属的面，应该在它的左边
}

class CFace
{
    short     m_nType;       //几边形
    UINT*     m_piVertex;    //所有点
    UINT*     m_piEdge;      //所有边
    Vector3D  m_vNormal;     //法向
    Vector3D  m_vMassPoint;  //该面所有顶点的几何重心，在拾取时用该点代表面
    double    m_dArea;       //面积
}

class CMesh
{
    UINT        m_nVertex;          //点数
    CVertex*    m_pVertex;          //点表
    UINT        m_nEdge;            //边数
    CEdge*      m_pEdge;            //边表
    UINT        m_nFace;            //面数
    CFace*      m_pFace;            //面表
    UINT        m_nWedge;           //Wedge 数
```

```
    CWedge*         m_pWedge;                //Wedge 表
}
```

显示主要由类 CScene 和 CDisplay 实现。(具体内容可参考\cmp\Display 下文件，不再赘述。

2）Tracing curve 功能的实现

Tracing curve 的功能实现在类 CcmpExt 中。

（1）主要数据

此类继承了 CMesh，其它主要数据结构如下（除去了一些具体实现的控制变量）:

```
class CMeshExt : public CMesh
{
    const CMeshExt*     m_pBaseMesh;   //指向基网格
    CboolMatrix m_bmxBaseTopology;     //点之间连接关系，用于生成数判断
    CintMatrix   m_nmxTopologyPath;    //由点确定属于那条路径
    // 点对应关系
    UINT*  m_pnVtxMappingFromBaseMesh;//基网格到模型网格特征点
    UINT*  m_pnVtxMappingToBaseMesh;   //模型网格到基网格特征点
    // curve
    CTracingPath*   m_pTracingPath;          //curve 的存贮
    Min_Path_Queue      m_qMinWeightPath; //curve 的弧长优先级队列
    int*       m_pnVtxBelongTo;          //模型网格点属于哪条 curve
    int*       m_pnEdgeBelongTo;         //模型网格边属于哪条 curve
    double* m_pdVtxParaTable;            //Floater 参数化表
    // 一次 brush fire tracing 过程的数据
    UINT*  m_pnVtxInedge;                // tracing 到点的入边，用于反向遍历
    int*       m_peVtxTracingStatus;     //tracing 中点的状态
    Min_Vtx_Queue m_qMinWeightVertex;// tracing 中，参数 P_k(a,b) 队列
```

```
    // patch
    bool*    m_pbFaceVisited;              // face 是否访问标志
    UINT_List* m_plstSubface;              //patch 中包含的 face 列表
}
```

（2）主要接口函数

函数 TracingObjectMesh

除了读入数据的接口，显示用的接口，主要的功能接口函数为：

```
BOOL TracingObjectMesh(bool bLogFile = true)
{
    ReInitTracingData();                   //初始化
    if (!TracingDirectPath(bLogFile)) //不考虑拓扑，生成初始优先级队列
        return FALSE;
    if (m_bQualifiedTracing) {             //是否进行拓扑一致 tracing 标志
        if (!TracingQualifiedPath(bLogFile)) { //核心函数，搜索 curve
            if (bLogFile)
                SaveTracingPath(m_sTracingPath);
            return FALSE;
        }
        if (!TracingSubface(bLogFile))//搜索 patch
            return FALSE;
    }
    return TRUE;
}
```

返回值是成功与否标志。TRUE 表示成功，FALSE 表示失败。

输入参数 bLogFile 表示是否要记录过程中的辅助文件，true 表示是，false 表示否。

此函数主要调用 3 个函数 TracingDirectPath，TracingQualifiedPath 和 TracingSubface 完成了（2.4）算法流程中的 1—7 全部内容。其中

TracingDirectPath 完成 1，生成弧长优先级队列（不考虑拓扑一致测试）。

TracingQualifiedPath 完成 2－6，是核心的刨分过程。

TracingSubface 完成 7，当所有的 Curve 被确认，以 Curve 为边界搜索 Patch。

以下，分别简要说明。

函数 TracingDirectPath

实现不考虑拓扑结构的情况下的 curve 的 tracing。使用 m_pdVtxParaTable 中的 Floater 参数值，从每条 curve 的初始点 brush fire 收缩到终止点。搜索过程中用队列 m_qMinWeightVertex 记录每个点的 Floater 参数权中，并置访问标记 m_peVtxTracingStatus，和访问前进顺序 m_pnVtxInedge。当到达终止点，反向遍历 m_pnVtxInedge 就得到了 path。

将每条 path 的初始路径存储在 m_pTracingPath 中，并计算弧长，将其插入弧长优先级队列 m_qMinWeightPath。

函数 TracingQualifiedPath

此函数是核心的刨分函数，完成（2.4）中流程 2－6 的操作。首先取出弧长优先级队列 m_qMinWeightPath 中的最短 Curver。然后进行以下操作：

3．生成树测试，失败，则惩罚 后重新插入队列

生成树测试用矩阵 m_bmxBaseTopology 来判断。当每条 Curve 被确认时，相应的顶点 i 和 j 之间置 1，表示连通，此外和 i 相连的所有顶点与和 j 连的所有顶点置 1，表示其连通。测试时，只要某 curve 两顶点之间为 1，那么该曲线不能确认，要沿后操作。

当确认了基网格顶点数－1 条 curve 之后，改测试取消。

4．拓扑测试（相交，边序），失败，则重新生成弧长（需要考虑拓扑一致）

相交测试由 m_pnVtxBelongTo 来判断。确认 Curve 时，curver 经过的顶点置标志，当检测 curve 时，遍历其 curve 经过的顶点，如果有标记，那么就相交了。

边序测试比较繁琐，需要数出基网格的邻接顺序，然后数出 curve 的邻接顺序加以判断。这个测试对于出边和入边都需要。

如果测试失败，需要重新 tracing。此时，在 tracing 的过程中就需要进行上

述两项测试。

5．边反转测试，失败，则惩罚后重新插入队列

对某条 curve，取其基网格的面片的另外一个顶点，从该点出发，用 brush fire（不需要拓扑测试）tracing 一条到该 curve 的路径，终止条件改为到 curve 结束。然后和边序测试类似，判断入边的方向

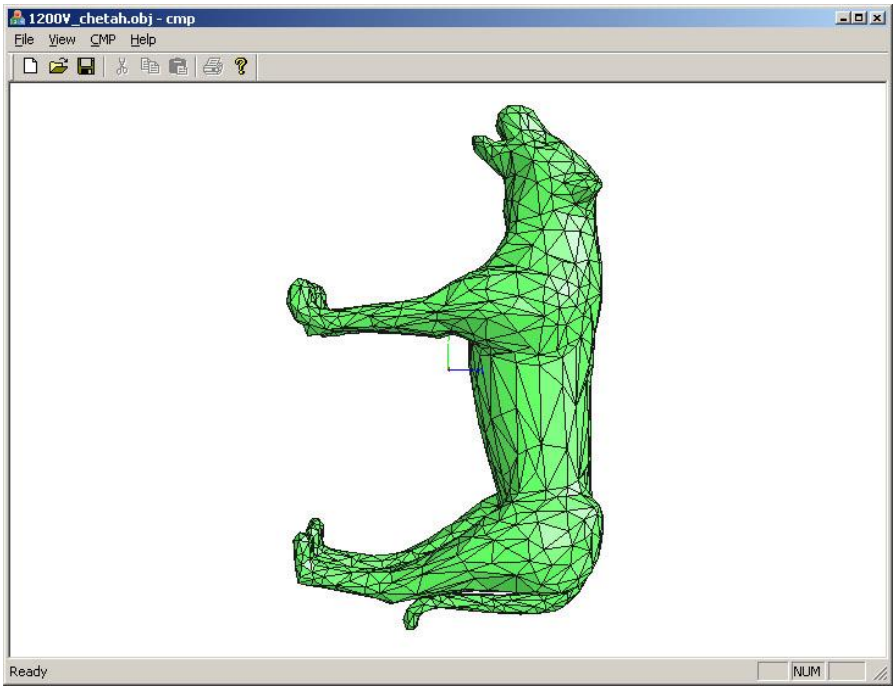
6．确认该 Curve，做为已有的限制条件

通过上述各项测试的 curve 既作为接受的 curve。修改 m_pTracingPath 中确认标记，m_pnVtxBelongTo 标记和 m_pnEdgeBelongTo 标记。这些标记，作为限制条件，限制以后的 tracing。

函数 TracingSubface

此函数完成以 Curve 为边界搜索 Patch。此函数首先根据基网格信息和 curve 信息，找到一个属于该 path 的 face，然后采用广度遍历的方法，遍历其相邻的 face。相邻的 face 由当前 face 的边的对偶边得到，所以，只要加入限制条件，不访问打上标记的边的对偶边的面，就能将 face 限制在 curve 边界内。标记由上一函数实现在 m_pnEdgeBelongTo 中，此处再添加 m_pbFaceVisited 作为访问标记。m_plstSubface 输出结果。

（3）界面和操作操作



Cmp 的主要操作界面如上图所示。并且程序执行过程中由后台 console 窗口，输出信息。

读入文件，及顺序：

1）模型网格，obj 文件，菜单 File/Open，如 1200V_chetah.obj。

2）基网格，obj 文件，菜单 CMP/Load Base Mesh，如 15v_base.obj。

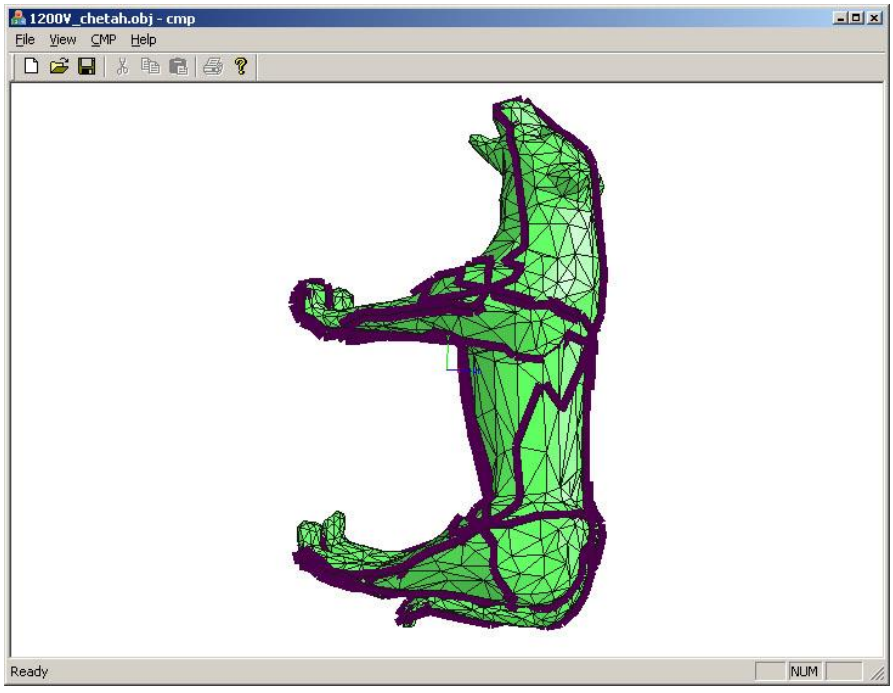
3）对应点关系，vmp 文件，菜单 CMP/Load Mapping Data，如 15_1200V_chetah.vmp。

4）网格点参数，par 文件，菜单 CMP/Load Vertex Para，如 15_1200V_chetah.par。

5）后处理文件，pst 文件，菜单 CMP/Load Post Handle Data，如 15_1200V_chetah.pst。这个文件的内容，将在后文讨论。

执行：

菜单 CMP/Tracing Path。其结果如下图所示：



图中的黑线表示得到的刨分路径。可反复执行

模型操作：

鼠标右键：有上下文菜单，可以选择显示模式和选取模式。

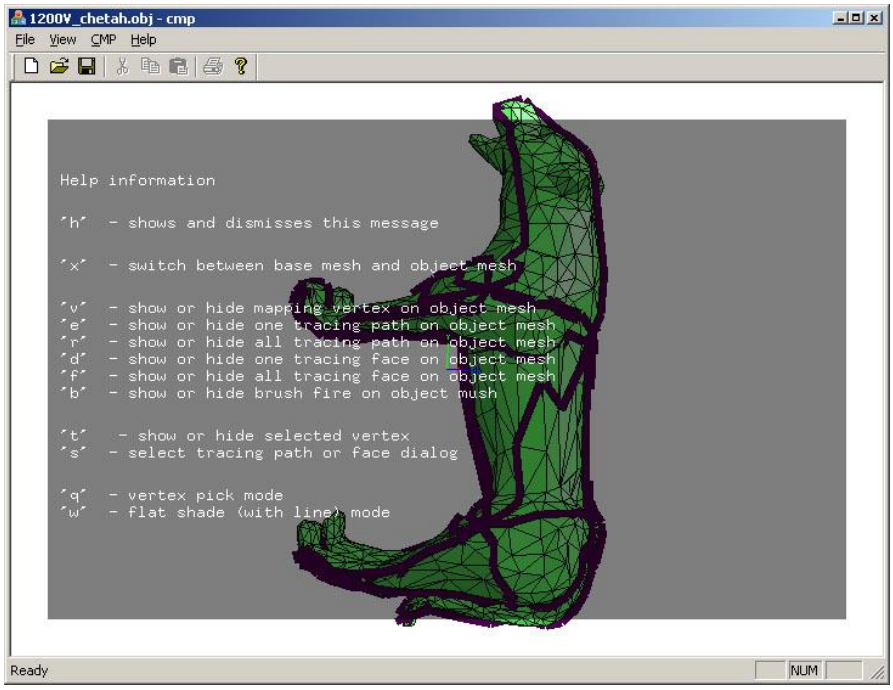
旋转：Alt+鼠标拖动

平移：Ctrl+鼠标拖动

缩放：Shift+鼠标拖动

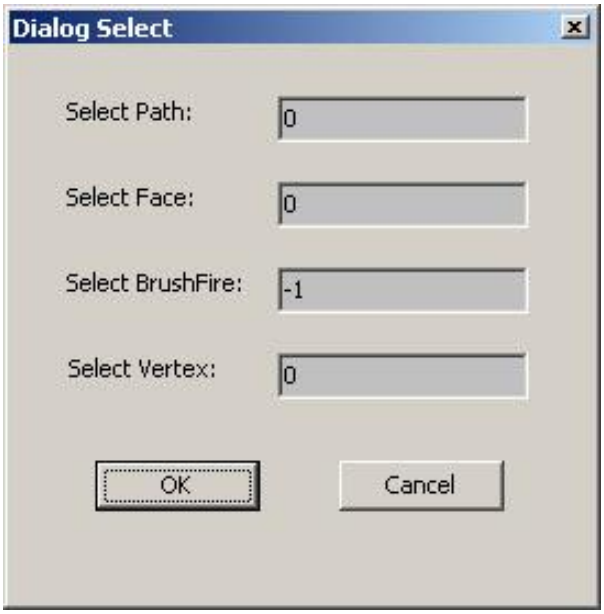
快捷键：

程序有很多快捷键。按‘h’可得到帮助信息界面。不再赘述。快捷键基本对应了菜单 View 下的操作。



显示选择对话框：

菜单 View/Select Path and Face 或者快捷键‘s’。



其中 Path 选择某条 curve，Face 选择某个 patch，BrushFire 选择某条 curve 过程中的 brush fire 访问过的所有边，vertex 选择某个点。选则后，需要打开相

应的显示标记。可查看 View 菜单或者 help 信息。

控制选择对话框：

菜单 CMP/Control Select。

这个对话框设定 cmp 的 tracing。

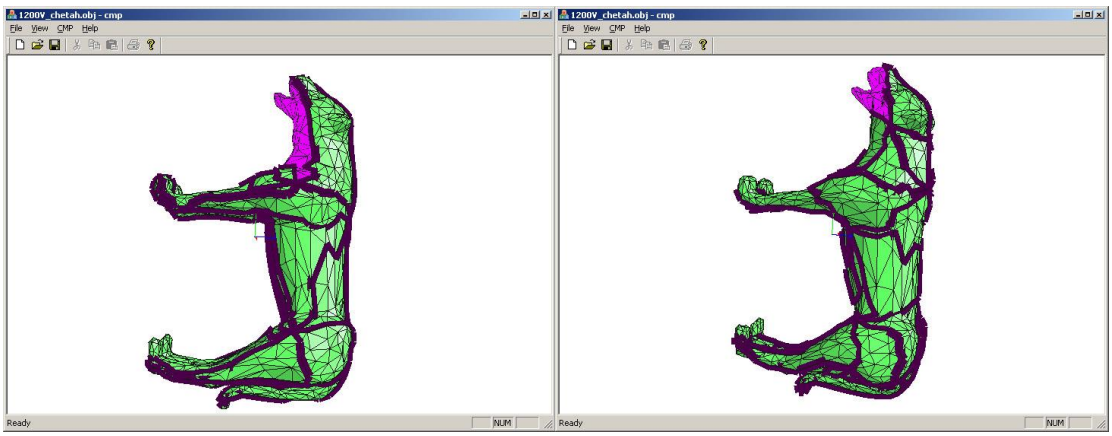
Direct Tracing（缺省不选），选择是否要做拓扑一致 tracing，选择的化，只是简单的 brush fire tracing，不做拓扑一致测试，可以差看原始结果。

Log Detail Path and Face fiel（缺省选择），是否要记录详细过程文件。

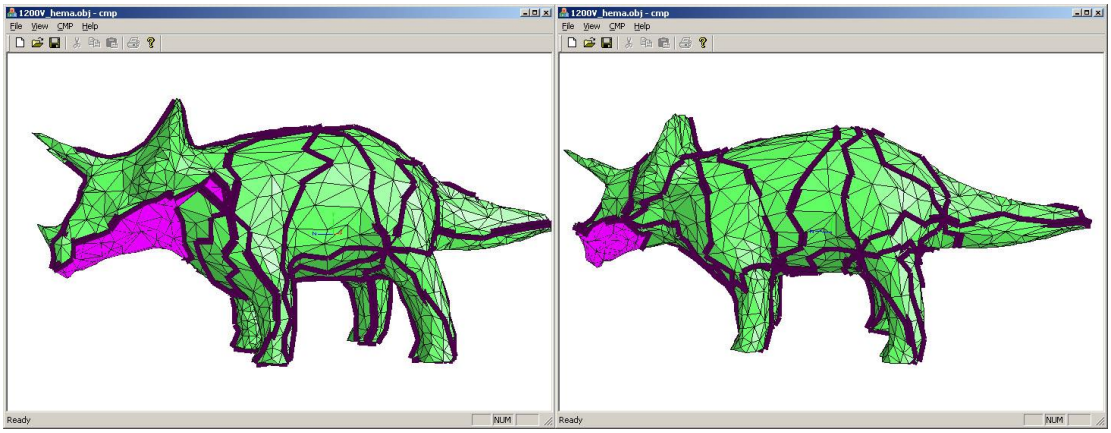
Post Handle Num，是可供选择的后处理项目，和允许的后处理项目。将在下文讨论。

4. 测试、对比结果

对不同的基网格，得到的分隔是不同的。



上两个图分别时模型网格采用 1200_chetah.obj 时，采用 15v_base.obj 和 24v_Basechetah.obj 做为基网格的结果。其中，红色的区域是两个两种情况下，嘴附近的 patch。



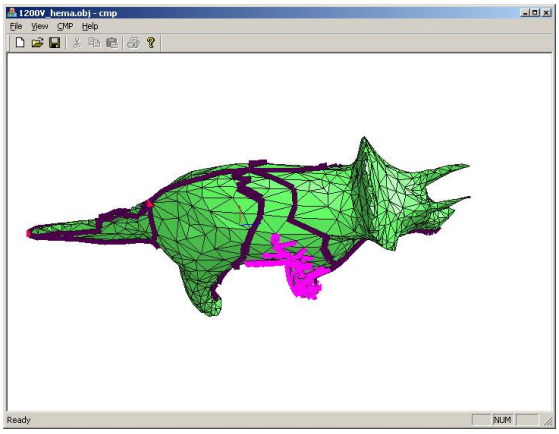
同样，上两个图分别时模型网格采用 1200_hema.obj 时，采用 15v_base.obj 和 24v_Basechetah.obj 做为基网格的结果。

从 path 可以看出基网格模型点多一些的情况下，path 比较合理，上左图中有狭长的效果。

5. 实现过程中遇到的问题及解决对策

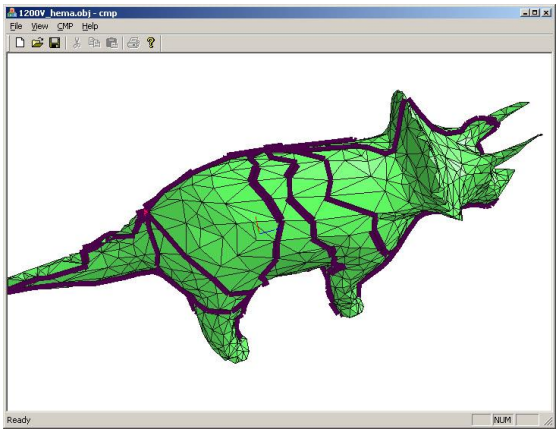
实现过程中，其他问题都已解决。只有模型的问题困扰我们。由于我们的模型是由普通的简化模型程序生成，没有对 cmp 问题做处理和优化，所以，按照一般的简化规则：平坦的地方面片少，陡峭的地方面片多。所以网格模型比较不均匀。这样带来一个无法避免的问题：模型上的 edge 不够多，因为 curve 之间不能相交，也不能共用边，那么无解。

我们最初使用 400 个顶点的模型网格，在 2 组基网格下都没有找到解，后来使用 800 个顶点的模型网格，也失败了。最终使用了 1200 个点的基网格能够找到解。所以，这个程序对模型网格是有一定要求的，这个要求难以用数学描述。

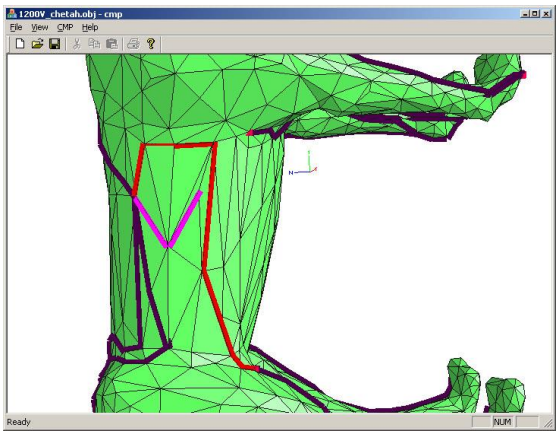


在 1200 点时，由于网格仍旧不均

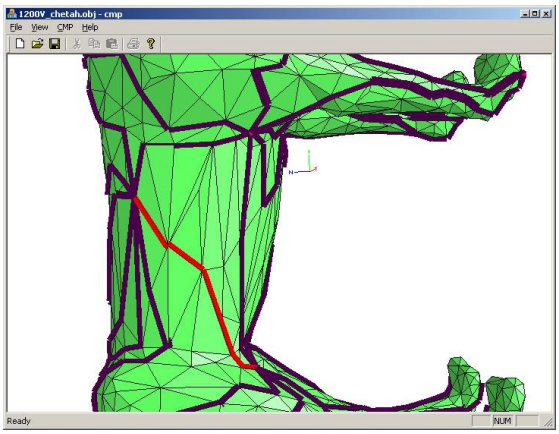
匀，参数化，可能造成 2 条路径实际上将某点孤立起来。如右上图所示。紫色表示 brush fire。由于 2 条黑色路进将网格前进的路径堵住,从右前腿特征点出发的路径无法连到背上的特征点。



稍稍将左边的路径向左侧移开一格，这个问题就解决了，如右图所示，中间的黑色路径就是找到的结果。



又如右图,红色的 curve 是一条明显弯曲非常明显的曲线。这是由于网格不均匀，偏差 1 个就可能差了非常多。这种情况下，也找不到某些 curve，如图中紫色的 curve，被挡住了。将红色路径向下调一格，这个问题就可以解决。如右图所示。红色的 curve 被纠正了很多。它的左侧有 3 条黑色的 curve。但它们仍旧有明显的扭曲。



所以，这个模型网格不均匀造成的。我们解决方案是人工干预。需要填写后处理里文件（pst）。对某些 curve 做一定的调整。

模型均匀，则需要的干预就少，否则就多。我们使用的后处理如下表所示：

	15V_base.obj	24V_Basechetah.obj
1200_ chetah	4	3
1200_hema	9	2

后处理调整可以通过交互方式了进行，选择点选取模式（上下文菜单中，或者快捷键‘q’）可以在 console 中看到点的编号，按顺序记录填写到 pst 文件中即可。

这个问题本质上必须通过对网格模型处理来解决。

四、Huma 框架

一．框架简介

功能：

Huma 程序主要完成了以下几项功能：

1．模型简化。Operation->Simplify

执行该命令后，会弹出一对话框，要求输入简化后要求的模型顶点数量。

2.简化模型输出。Operation->Output Model，输出模型为当前目录下 Model.obj

3．划分一致性网格。Operation->Consist Remesh。

执行这一功能前，还要首先输入基网格文件名，由 Operation->Open BSM File 输入文件路径。Operation->Mesh Dens 输入重采样的网格密度。缺省情况下为 15，表示一个 patch 的每一条边被分为 15 份。

执行 Operation->Consist Remesh 的过程中，程序会到打开 bsm 文件的相同路径下去读取同名的 bep 文件，因此在一致化参数前保证有同名的 bep 文件存在。

4．输出一致性网格。Operation->Output Remesh，输出模型为当前目录下 ReModel.obj

5．Floater N 维参数化。Operation->Param(ND)，输出为 N 维 floater 参数化结果，结果文件包存在当前目录下 Out.par。这个文件被用于模型分片的 Tracing 操作。

6．形状渐变。Operation->Shape Blending，这是一致化网格划分的一个典型应用。执行 Shape Blending 命令时，依次输入两个模型的文件名，这两个模型是上面一致参数化执行的结果文件。Shape Blending 命令的执行结果会产生五个名为 m*.obj 的结果文件，结果文件保存在当前路径下。

输入、输出：

Huma 程序的需要的输入文件有：

模型网格文件：*.obj

基网格文件：*.bsm

模型网格剖分后文件：*.bep

Huma 程序的需要的输出文件有：

简化后的模型网格文件：*.obj

一致参数化网格划分后的模型文件：*.obj

N 维 floater 参数化的结果文件：*.par

形状渐变的结果文件：*.obj

二．算法及原理

2．1 网格简化

网上可下载的模型的原始网格顶点数量一般都在 5000 以上，面片数量在 15000 以上。为了简化后面的计算，首先对原始网格进行一定的简化。简化方法采用的是 Garland（Sigraph97）提出的基于二次误差函数的边收缩简化方法。

边收缩简化原理如下：

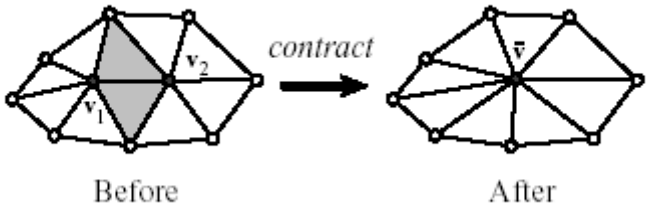


图 1. 边收缩简化原理

如图一所示，边收缩方法将一条边的两个顶点合并成一个顶点，并将相关边、面的拓扑结构合并。一次合并操作能删除一个顶点、两个面、三条边。显然，边的收缩顺序和收缩一条边后剩下的顶点的位置是保证简化后的模型和简化前模型几何形状尽量一致的关键因素。

Garland 提出了如下原则来确定边收缩顺序和边收缩后剩下顶点的位置：某条边收缩后，收缩所产生的点到这条边的两个顶点所邻接的面的距离平方和最小；并以距离平方和这个量作为确定边收缩顺序的标准，距离平方和越小，越早进行收缩。Garland 称这个距离平方和为二次误差函数。

二次误差函数和新顶点的位置用矩阵表示如下：

$$\Delta v = \sum \left(p^T v\right)^2 = \sum v^T \left(pp^T\right) v \qquad (1)$$

其中，$p = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$，$p$ 代表平面方程 $ax + by + cd + e = 0$，并且满足 $a^2 + b^2 + c^2 = 1$

$$K_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \qquad (2)$$

欲使（1）式最小，（1）式对 x, y, z 的偏导值应该为 0。

对（1）式求偏导，得：

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{bmatrix} \bar{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (3)$$

由（3）式，可以解得原始模型中每一条边收缩后，所产生的新顶点的位置 $\bar{v}$，将这一顶点位置 $\bar{v}$ 带入（1）式，即可得到每一条边的二次误差函数，并将其作为每一条边的权重。将所有边的权重进行排序后，按从小到大的顺序插入队列，并按出对的顺序对模型进行简化。

在队列中的边出对后，进行简化前，判断对出对的边进行收缩后，是否会引起模型的歧义现象：如产生悬边、悬点。如果产生歧义现象，则不对该边进行收缩，将该边的权加上一个较大权重后，重新插入队列中。

如果收缩后无歧义，则进行收缩操作，相应改变与之相邻的点、边、面的拓扑关系。重新计算所有方程更改了的面所相邻的边的权重，并重新进行插入排序。

按如上方法操作后，可以在简化过程中，较好地保持原模型的几何形状。

2．2 Floater 参数化

Floater 参数化[Floater 97]本身的目的是将一个曲面三角网格映射到一二维平面区域内，在[Praun 01]文中，除继续将这种方法用于二维参数化外，亦将其推广至将曲面顶点映射在用基网格标定的 N 维区域内，做为后面 Patch 划分的基础条件。

Floater 参数化思想如下：

参数化目标：曲面网格上原顶点集为：$X=\{(x_i,y_i,z_i)\}$，将其映射到一二维区域后，其映射目标点集为：$U=\{(u_i,v_i)\}$。

且满足：$u_i=\sum_{j=1}^{n}\lambda_{ij}u_j$ (4)

其中 $\lambda_{ij}>0$，$\sum_{j=1}^{n}\lambda_{ij}=1$。关键问题是求 λ_{ij}。

方法：如果每个点的度都是 3，那么显然可以用 barycentric 坐标实现参数化。假设 pi 点的三条连边的对应顶点分别是：p1，p2，p3。

则 $\lambda_{i1}=\frac{area(p_ip_2p_3)}{area(p_1p_2p_3)}$，$\lambda_{i2}=\frac{area(p_1p_ip_3)}{area(p_1p_2p_3)}$，$\lambda_{i3}=\frac{area(p_1p_2p_i)}{area(p_1p_2p_3)}$

如果节点的度大于 d_i>3，如下图所示：

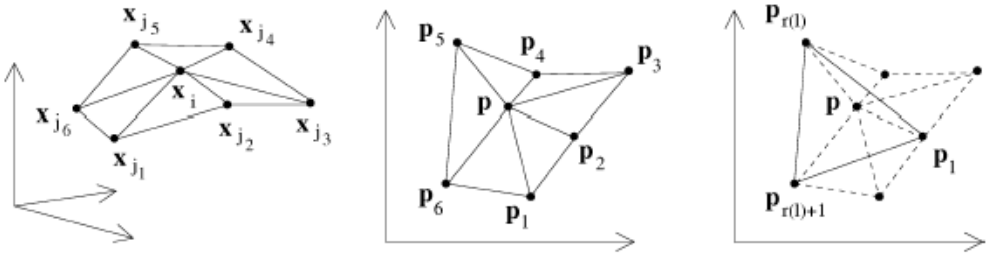


图 2

则 pl 与 p 点连线与 p 点周边多边形交线的两个顶点 pr(l)，pr(l)+1。P 点可以用 p1，pr(l)，pr(l)+1 三个点表示。

可以写成：

$$p=\delta_1p_1+\delta_2p_{r(l)}+\delta_3p_{r(l)+1}$$

对每个 l，p 点都可以写成：

$p=\sum_{k=1}^{d_i}\mu_{kl}p_k$，其中 $\mu_{il}=\delta_1$，$\mu_{ir(l)}=\delta_2$，$\mu_{i3}=\delta_{r(l)+1}$，其余为 0

$$\lambda_{ik} = \frac{1}{d_i}\left(\sum_{l=1}^{d_i}\mu_{k,l}\right)。$$

这样，参数λ_{ik}确定了，将这些参数代入（4）式中，就可以得到所有曲面网格内所有非边界点的方程表达式。如曲面网格上所有边界点的二维坐标已经确定，那么此方程的系数矩阵是一个弱对角占优矩阵，可以用迭代方法进行求解。

剩下的另一个问题是如何确定边界点在二维或 N 维的坐标。

N 维参数化比较简单，因为共有 N 个基点，所以可以分别把 N 个基点的坐标分别取为：(1,0,0,……,0)，(0,1,0,0,……,0)，……..，(0,0,……0,1)。并将这 N 个点作为边界点。

二维边界的选定在本次实验中是按曲面边界上每条直边长度在曲边总长中所占的比例，在二维区域内边界上进行分配的。

2．3 网格重建

在网格上按基网格的拓扑结构 tracing 出网格上的分片结构后，每一片（patch）对应一个三角形拓扑结构，每一个 Patch 中包含若干个原模型中的 Face。可以将每个 patch 的边界映射到一个平面正三角形上，同时用 Floater 参数化方法的方法将 patch 中的所有 faces 的顶点映射到平面三角形内部。

图 4 是豹子上的一个 patch 映射到平面正三角形上的结果。

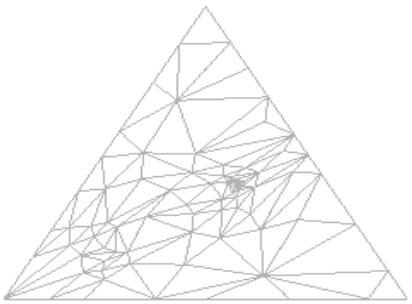


图 3. patch 平面参数化到正三角形

这时的参数化的正三角形结果，同时包含二维的参数化结果和三维原始信息。对平面参数化结果按输入的采样密度，在平面域上进行规则的网格划分。划分结果如下：

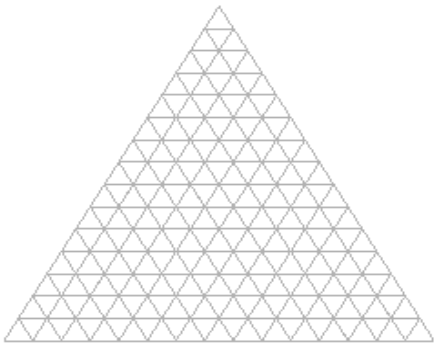


图 4. 对平面参数正三角形进行规则采样

显然，二维参数化结果和二维重采样结果可在平面域内建立一一映射。每一个重采样的顶点必然包含在二维参数化结果的一个小三角形内（或小三角形边界上）。这个重采样顶点可以用小三角形的三个顶点的凸组合表示。

$p=\sum_{i=1}^{3}\alpha_i p_i$ ， α_i 为三角形的面积坐标，亦可用 Floater 参数化方法确定。

确定了平面上的凸组合系数后，可将重采样顶点的三维坐标中的表示为：

$\boldsymbol{P}=\sum_{i=1}^{3}\alpha_i \boldsymbol{P}_i$ ， $\boldsymbol{P}$ 为重采样顶点的三维表示， $\boldsymbol{P}_i$ 为三角形三个顶点的三维坐标。

对每个 Patch 进行重采样后，获得三维数据后，将所有 patch 中的面片集合并，即得到对整个模型的重构结果。

2．4 形状渐变

这是一致化网格划分的一个典型应用。即让一个模型光滑地过渡到另一个模型。这里采用的是最简单的线性插值的方法，即将两个模型对应顶点的三维坐标按照帧数进行线性插值。

公式如下：

$$v_i=\frac{i}{n+1}(v_0+v_n)$$

v_o 、 v_n 分别为始帧和末帧相应节点的三维坐标。 v_i 为中间产生的第 i 帧的对应顶点的三维坐标。

三．系统设计和数据结构

本程序框架用 VC++6.0 向导生成，用 OpenGL 进行显示。

在本程序中，担负主要功能的主要有以下几个类：

CMesh、CBaseMesh：保存模型和模型转换中所需要的所有数据。

Simplify：简化模型

Param：对程序实行 N 维和 2 维 Floater 参数化

CRemesh：对网格进行一致性网格划分

这个程序中所使用的数据结构是标准的 DCEL 结构，这里不再多说。点、边、面之间的关系用指针连接。所使用的类有：CVertex（顶点），CEdge（半边），CNDEdge（边），CFace（面），CMesh（网格）。

另外，因为基网格与网格有一些不同的性质，因此从 CMesh 类上又派生产生了 CBaseMesh 类。在 CBaseMesh 中增加了 dq_Patch，保存基网格上 Patch 的队列；dq_VtxMap，保存基网格边上顶点和原网格的对应关系。P_Mesh，指向该模型网格的指针。

四．测试结果

4．1 网格简化结果

按 Garland 方法进行简化，可以在简化过程中，较好地保持原模型的几何形状，甚至可以将一亏格为 0 的拓扑结构甚至简化为一四面体。

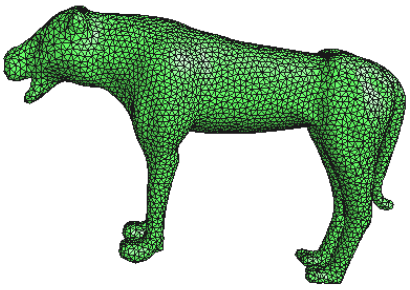


图 5. 5000 个顶点的豹子

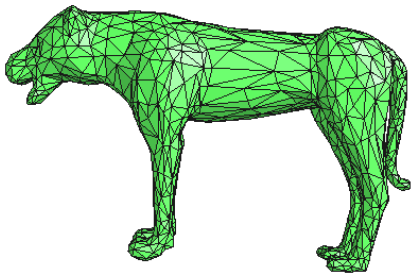


图 6. 1200 个顶点的豹子

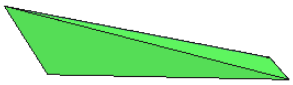


图 7. 豹子简化为 4 面体

4．2 模型一致化网格重建结果

三角恐龙和豹子原模型的顶点数均为 1200，但点的连接关系不一致。一致化重构后网格的顶点数为 2927，点与点之间的连接关系完全相同。

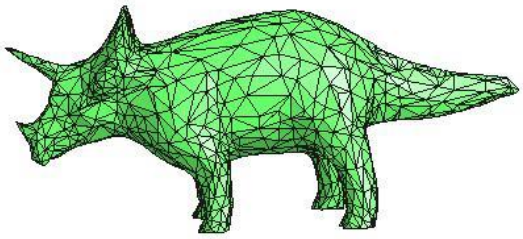


图 8.原三角恐龙模型网格

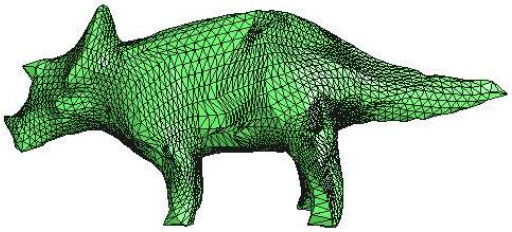


图 9.重构后的三角恐龙模型网格

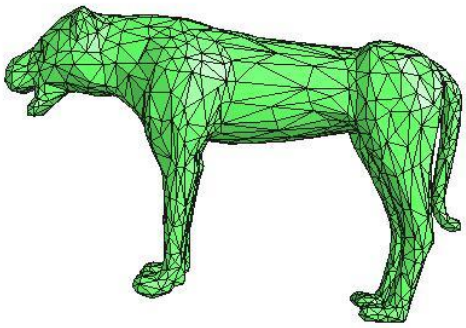


图 10.原豹子模型网格

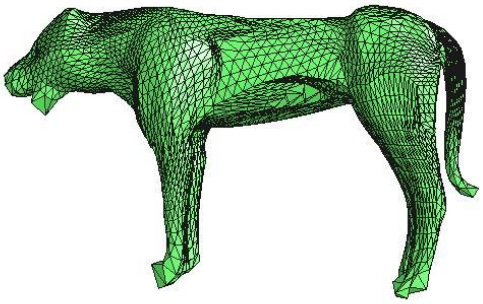
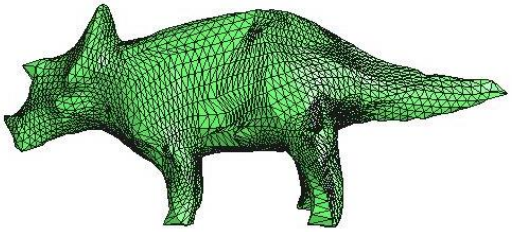
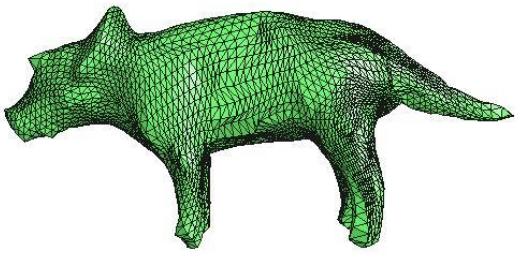
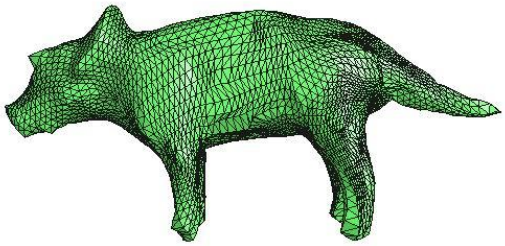


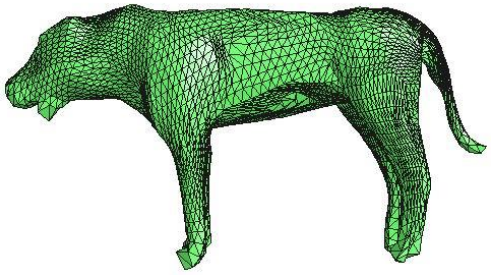
图 11.重构后的豹子模型网格

4．3 三角恐龙到豹子的渐变效果









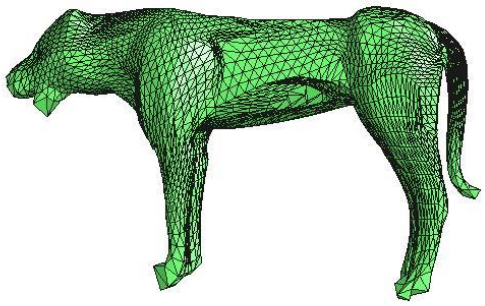


图 12-16 三角恐龙到豹子的渐变效果

从以上结果可以看出，由于一致化网格划分方法带来的好处，虽然线性插值的方法非常简单，但最后的效果还是可以接受的。

五．现存问题

从现在的结果来看，重构后的网格在曲率相近的地方网格划分并不均匀，并有细节丢失。

重构后网格在曲率相近的地方网格划分并不均匀的原因是因为获得的基网格面片大小、各方向尺寸相差较大。在每个面片上每个方向作同密度的采样，因而采样得到的重构网格不均匀。因此为了获得比较均匀的网格，还要在基点选取、基网格的划分上下更多工夫，以获得大小大致一致的基网格划分。

细节丢失也和基网格划分有关系。对于细节越丰富的地方，基网格应该划得尽量小一些，以在相同的采样密度下，保留住所需要的细节。

六．其它问题

在打包上载的文件中，除源程序外，还有一个名为 Huma.exe 的 release 版执行程序。测试 Simplify 和 Consistent Remesh 命令所需要的输入文件分别在\chetah 和\hema 目录下；测试 Shape Blending 命令所需要的文件在\Shape Blending 目录下。所有功能都测试通过。

五、总结

本次课程设计有非常大的工作量，处理理论上较为困难，实现上也又很多具体困难。我们小组团结合作，经过分析和不断探索，最终实现了初步的效果。由于没有好的模型网格和基网格，我们效果并不是那么好看，可我们认为，我们实现应该实现的内容。总的来说，我们实现了大约 4 篇文章的内容。

通过实验，对 DCEL 结构，模型简化，同拓扑，参数化，重构等许多问题有了深入的认识，受益很大。

六、參考文獻：

[Floater 97] FLOATER, M. S. *Parameterization and Smooth Approximation of Surface Triangulations.* Computer Aided Geometric Design 14 (1997), 231–250.

[Guskov 00] GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRODER, P. *Normal Meshes.* Proceedings of SIGGRAPH 2000 (2000), 95–102.

[Hoppe 96] HOPPE, H. *Progressive meshes.* Proceedings of SIGGRAPH 1996 (1996),.,pages 99–108, Aug. 1996.

[Garland 97] GARLAND, M., AND HECKBERT, P., S. *Surface Simplification Using Quadric Error Metrics.* Proceedings of SIGGRAPH 2001 (1997).

[Praun 01] PRAUN, E., SWELDENS, W., AND SCHRODER, P.. *Consistent Mesh Parameterizations.* Proceedings of SIGGRAPH 2001 (2001), 95-102