基于 Web 墨卡托投影的地图算法研究与实现

李长春[a], 蔡伯根[a], 上官伟[a,b], 王 剑[a,b]
(北京交通大学 a. 电子信息工程学院; b. 轨道交通控制与安全国家重点实验室, 北京 100044)

摘 要: 针对 Web 墨卡托投影地图用于地理信息系统开发时其第三方应用程序接口仅支持浏览器/服务器模式的问题,采用墨卡托投影式的变形并结合数据分片技术,以 Google 地图为例提出一种 Web 墨卡托投影地图用于客户端/服务器模式的算法。实验结果表明,该算法有效解决了 Web 墨卡托地图不能用于客户端/服务器模式的问题。

关键词: 地图投影; Web 墨卡托; 地理信息系统; Google 地图
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2012)12-4793-04
doi:10.3969/j.issn.1001-3695.2012.12.103

Research and implementation of map algorithm based on Web Mercator

LI Chang-chun[a], CAI Bai-gen[a], SHANGGUAN Wei[a,b], WANG Jian[a,b]
(*a. School of Electronics & Information Engineering, b. State Key Laboratory of Rail Traffic Control & Safety, Beijing Jiaotong University, Beijing* 100044, *China*)

Abstract: The third-party application development interface of Web Mercator projection map is generally only supports browser/server mode. This paper used Mercator projection formula combined with data fragmentation technology to proposed a Web Mercator projection map algorithm for the client/server mode, and used experiments to prove the effectiveness of the algorithm.
Key words: map projection; Web Mercator; GIS; Google map

基于 Web 墨卡托投影的地图因其丰富的资源和良好的用户体验而受到广泛关注,第三方应用程序接口的开放进一步加快了其在地理信息系统中的应用。但是基于 Web 墨卡托投影地图的第三方应用程序接口通常只能支持浏览器/服务器模式,这一缺陷为客户端/服务器模式的应用带来了极大的困难。目前,在客户端/服务器模式应用中,整合基于 Web 墨卡托投影地图主要有两种方案:a)使用 Web 插件,通过插件使用第三方应用程序接口对基于 Web 墨卡托投影的地图进行间接控制;b)通过创建与第三方应用程序接口兼容的新接口来支持客户端/服务器模式[1]。这两种方案都存在不足,前者应用不够灵活,难以满足定制的需求;后者工作量较大,而且仍然受到第三方应用程序接口的限制。因此,应用于客户端/服务器模式的 Web 墨卡托地图算法研究受到关注。现在的研究多从 Web 墨卡托投影原理和图片数据结构入手寻求解决方案[2,3],目前还缺少完整的易于实现的研究成果。本文重点研究 Web 墨卡托投影原理并基于数据分片技术设计一种基于 Web 墨卡托投影的地图算法,能够有效实现基于 Web 墨卡托投影的地图数据直接应用于客户端/服务器模式。

1 地图投影模型

地图投影是指按照一定的数学法则,把地球表面的地理信息映射到平面地图的理论和方法[4]。为了保证地理信息从球面映射到平面后仍然能够被充分表达和理解,地图投影要遵循一定的规则,如面积不变、距离不变、角度不变等。

1.1 墨卡托投影

墨卡托投影是角度不变的投影方式,又称为等角正切圆柱投影。假设有一圆柱体切于赤道。取本初子午线与赤道交点的投影为坐标原点,赤道的投影为横坐标 x 轴,本初子午线的投影为纵坐标 y 轴,构成墨卡托平面直角坐标系。取地球椭球体的长轴为 a,短轴为 b,如图 1 所示。根据等角条件推算出的墨卡托投影公式[5]为

$$\begin{cases} x = a \times \theta \\ y = a \times \left[\ln \tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right) + \frac{e}{2} \ln\left(\frac{1 - e \times \sin\varphi}{1 + e \times \sin\varphi}\right) \right] \end{cases} \tag{1}$$

其中:$\theta(-\pi, +\pi)$ 为经度,且东经取正值,西经取负值;$\varphi(-\pi/2, +\pi/2)$ 为纬度,且北纬取正值,南纬取负值;$e = \sqrt{a^2 - b^2}/a^2$ 为地球椭球体第一偏心率。

由墨卡托投影式(式(1))可知,墨卡托投影将经纬度坐标(θ,φ)映射为平面坐标(x,y)。

1.2 Web 墨卡托投影

墨卡托投影的等角特性保证了对象的形状不会变形,同时保证了方向和相互位置的准确性。Web 墨卡托投影与常规墨卡托投影有一个重要区别:Web 墨卡托投影把地球假设为球体而不是椭球体。这种假设主要是为了计算简单和实现方便。理论精度差别在 0.33% 之内[6,7]。当比例尺很大,地物很详细时,其差别可以忽略不计。

考虑把地球椭球体假设为正球体,取地球半径 $R = 6378137.0$ m,有 $a = b = R$,第一偏心率 $e = 0$。因此墨卡托投影

收稿日期: 2012-05-09; **修回日期**: 2012-06-18

作者简介:李长春(1987-),男,新疆阿勒泰人,硕士研究生,主要研究方向为数字化无线技术应用(10125058@bjtu.edu.cn);蔡伯根(1966-),男,教授,博士,主要研究方向为智能交通工程;上官伟(1979-),男,副教授,博士,主要研究方向为交通信息工程及控制;王剑(1978-),男,副教授,博士,主要研究方向为交通信息工程及控制.

式可简化为

$$\begin{cases} x = a \times \theta \\ y = a \times \ln \tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right) \end{cases} \tag{2}$$

Google 地图是采用 Web 墨卡托投影的典型实例。自推出以来,它以免费的方式向全球用户发布了大量的电子地图和卫星影像数据。本文以 Google 地图数据为例讨论算法的设计与实现。

2 地图数据分片技术

Google 地图分为普通地图、卫星地图和混合地图三类。向用户提供的地图数据以常见的栅格图像显示。本文主要研究卫星地图和普通地图。根据缩放的详细程度,其地图提供 0 ~ 17 共 18 个不同的缩放等级。缩放等级越高,图像显示越详细。不管是何种地图数据都采用图像分片技术,将各个缩放等级的全球数据分割成大小为 256 × 256 像素、格式为 PNG 或 JPG 的小图片保存起来[8]。为了实现连续快速地遍历和索引这些切片数据,其地图数据采用四叉树方式对小图片进行编码。把树的根对应于全球影像,树叶对应于各单个影像,所有的节点都有 4 个子节点,这样就形成 Google 地图的四叉树。下面以卫星影像数据为例介绍。

卫星影像数据参照金字塔模式按照不同的缩放等级分别存储,采用 qrst 四个字母按照四叉树模式对小图片进行编码[9],如图 2 所示。

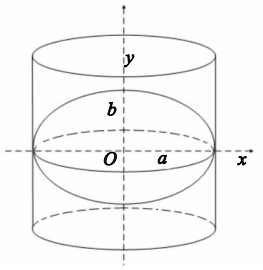


图1 墨卡托投影模型

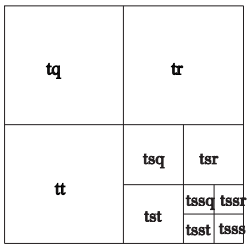


图2 qrst四叉树编码示意图

卫星影像数据每个切片都有一个对应的位置参数 t,t 值的长度减 1 为缩放等级。由图 2 可知,当缩放等级为 0 级时,全球就为一个 256 × 256 的图片,其位置参数 t = t;当缩放等级为 1 时,全球的图片分裂为 4 块,每块的位置参数分别是 t = tq,t = tr,t = ts 和 t = tt。依此类推,每放大一倍,每一块小切片都分裂为四块,卫星图片从左上到左下顺时针按 qrst 编号,分裂后的图块编码为分裂前的图块编码加上新生成小图块的编码。表 1 给出缩放等级、图幅、图块数量、分辨率之间的关系。

表 1 缩放等级、图幅、图块数量、分辨率的关系

缩放等级	图幅/像素	图块数量	分辨率/m/像素
0	256	4^0	156 543.04
1	512	4^1	78 271.52
2	1 024	4^2	39 135.76
⋮	⋮	⋮	⋮
n	256×2^n	4^n	$2\pi R/(256 \times 2^n)$

Google 设置 khm0、khm1、khm2、khm3 四个服务器存储卫星影像数据。用户可以通过与每个切片对应的(形如 http://khm0.google.com/kh/v = 106&t =)URL 访问服务器并获取相应的切片数据。其中,参数 v 表示数据版本,参数 t 是上述 qrst 编码后对应的位置参数。图 3 给出一个符合 qrst 四叉树编码规则的卫星影像数据的拼接图。由图 3 可知,最右下角的切片编码为 tsss,缩放等级为编码长度 4 - 1 = 3 级,根据表 1,此时的地图分片为 64 个 256 × 256 像素的小切片。

3 算法设计

如果根据上述墨卡托投影公式和卫星影像数据的编码规则计算出 t 值,获得对应切片的 URL,然后按照 URL 访问 Google 服务器获取图片,并按照一定的拼图算法就可以完成地图数据的下载和应用。

3.1 卫星地图算法

为了得到位置参数 t 值,需要将地理坐标系下的经纬度 (θ,φ) 按照式(2)转换为投影坐标系下的 (x,y)[10]。而实际上 Google 地图最终是按像素切片来存储和显示的,所以这里的 (x,y) 还需要转换为投影像素坐标系下的 $(\bar{x},\bar{y})$。由表 1 可得

$$\begin{cases} \frac{x}{\bar{x}} = \frac{2\pi R}{2^n} \\ \frac{y}{\bar{y}} = \frac{2\pi R}{2^n} \end{cases} \tag{3}$$

联立式(3)(2),可以得到

$$\begin{cases} \bar{x} = \frac{\theta}{2\pi} \times 2^n \\ \bar{y} = \frac{\ln \tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)}{2\pi} \times 2^n \end{cases} \tag{4}$$

投影坐标系以赤道和本初子午线投影的交点作为坐标原点,而像素坐标系则以图片左上角作为坐标原点。由表 1,某一缩放等级下的图块数量为 4^n,即在横轴和纵轴方向各为 2^n 块,$\bar{x}$ 和 $\bar{y}$ 值均取对应的整数值,其范围是 $(0,2^n - 1)$。因此,从地理坐标系下的经纬度 (θ,φ) 转换为像素坐标系下 $(\bar{x},\bar{y})$ 的公式为

$$\begin{cases} \bar{x} = \frac{(\pi + \theta)}{2\pi} \times 2^n \\ \bar{y} = \left[\frac{\ln \tan\left(\frac{\pi}{4} - \frac{\varphi}{2}\right)}{2\pi} + 0.5\right] \times 2^n \end{cases} \tag{5}$$

其中:$\theta(-\pi, +\pi)$ 为经度,且东经取正值,西经取负值;$\varphi(-\pi/2, +\pi/2)$ 为纬度,且北纬取正值,南纬取负值;n 为缩放等级。通过式(5)可以计算得到 $(\bar{x},\bar{y})$ 坐标值。根据式(5)可以看到,对于同一 (θ,φ),决定其 $(\bar{x},\bar{y})$ 坐标的实际上是缩放等级 n 的值。根据 qrst 四叉树编码规则,可以得到一维 t 值和二维 $\bar{x}$、$\bar{y}$ 值存在如图 4 所示的关系。

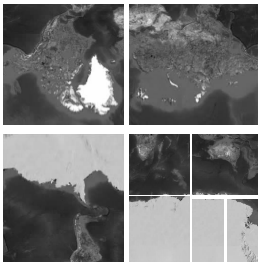


图3 卫星影像数据拼接示意图

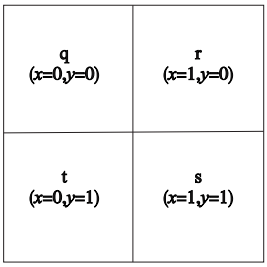


图4 qrst编码与x、y值的关系

根据图 4 所示,Google 卫星影像的全球影像即四叉树的根节点,总是从索引“t”开始,相应的缩放层级为第 0 级。把此层级划分为 4 个象限,每一个象限为子节点,对应下一层级影像。4 个象限的二进制象限索引分别为“00”“10”“11”和“01”。在上一层级二进制索引的后面添加对应位置的象限索引得到本

层级影像的二进制索引。按照同样的螺旋机制,得到全部影像的二进制索引。因此,可以建立 t 值与($\bar{x}$、$\bar{y}$)坐标之间的转换关系,即将$\bar{x}$、$\bar{y}$转换为二进制,按照图 4 所示的规律,按位组合即可得到位置参数 t 的值。例如,取 $\theta = 116.3364°$,$\varphi = 39.9478°$,$n = 4$,按照式(5)计算得到$\bar{x} = 13$,$\bar{y} = 6$。其二进制形式为$\bar{x} = 1101$、$\bar{y} = 0110$。将$\bar{x} = 1101$,$\bar{y} = 0110$ 按位组合,得到 t = tstr。按照上面的讨论,实现(θ,φ,n)到 t 值的转换流程如图 5 所示。

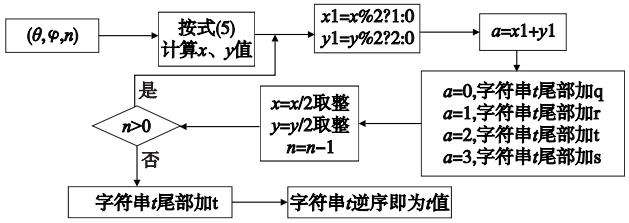


图5　(θ,φ,n)到t值的转换流程

3.2　拼图算法

按照 3.1 节介绍的算法能够获取任意图片数据资源。对于用户有意义的是按照一定的拼图算法把这些孤立的图片资源拼接成自己需要的地图。一种可行的方案是,以用户关心的经纬度坐标和缩放等级按照上述算法求出 t 值,保存其对应的图片数据并作为地图中心,然后按照某种算法获取中心周围的图片数据资源保存并拼接成用户希望的地图。

qrst 的编码规则使得每块切片的分布具有一定的规律。假设按照所关心的经纬度和缩放等级 n 值得到的图块为 x,拼接一幅以 x 为中心的 3 × 3 大小的地图。由图 6 可以发现以下规律:s 的左侧和右侧总是 t,上侧和下侧总是 r;同理,t 的左侧和右侧总是 s,上侧和下侧总是 q;q 的左侧和右侧总是 r,上侧和下侧总是 t;r 的左侧和右侧总是 q,上侧和下侧总是 s。无论缩放等级如何变化,始终遵遁这一规律。按照这个规律,可以 x 为中心,向左索引一次,获取中心影像西邻的切片数据;再向上索引一次,获取中心西邻切片的北邻切片数据,即确定 3 × 3 表的左上角。然后以左上角 t 值为基准,保存本列 3 个切片数据,再保存后两列的切片数据,最后把这 9 个切片数据按顺序拼接,就是希望得到的地图。图 7 给出按照上述拼图算法拼接 3 × 3 切片地图的流程。

qq	qr	rq	rr
qt	qs	rt	rs
tq	tr	sq	sr
tt	ts	st	ss

图6　qrst的分布规律

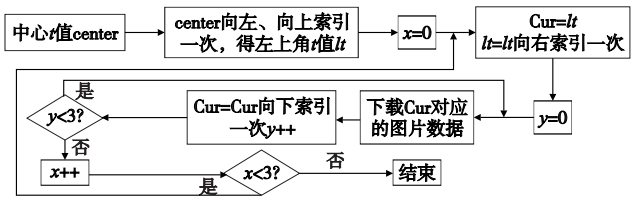


图7　3 × 3拼图流程

3.3　普通地图算法

普通地图与卫星地图 URL 不同。Google 设置 mt0、mt1、mt2、mt3 四个服务器存储普通地图的切片数据。通过形如 http://mt0.google.com/vt/lyrs=m@174000000&hl=zh-CN&src=app&x=&y=&z=&s=G 的 URL 可以访问相应的地图切片。其中决定位置的参数只有 x、y、z,这三个参数的作用与卫星影像数据 URL 中的位置参数 t 是一样的。前面已经讨论了如何根据(θ,φ)和缩放等级 n 来计算 x、y 和 z 三个参数的值。考虑到 x、y、z 参数不利于利用已有的拼图算法,故普通地图算法以卫星地图算法为基础,将 t 值转换为 x、y、z 参数进行地图数据下载,然后根据拼图算法进行地图的拼接。

3.4　地图缓存机制

每次都从 Google 服务器下载地图数据既增加网络流量的开支,同时也使地图显示的速度大大降低。为提高地图显示的速度,可采用缓存机制。将所访问过的图块保存到本地硬盘上或数据库中,并根据缩放等级建立索引。索引和图片所对应的经纬度信息存在对应关系。这样,在用户请求地图数据时:首先根据经纬度信息查询本地硬盘或数据库,如果存在对应的图片数据,则直接调用而不必通过 Google 服务器进行下载;如果没有对应的图片数据再通过 Google 服务器进行下载、拼接,并将下载到的数据建立索引后保存。通过缓存机制,地图显示和刷新的速率大大提高。表 2 是有缓存机制和无缓存机制地图显示消耗的时间与 CPU 占用率情况。

表 2　地图显示消耗时间与 CPU 占用率

缩放等级	无缓存消耗时间/s	有缓存消耗时间/s	无缓存 CPU 占用率/%	有缓存 CPU 占用率/%
4	2.80	0.50	25	5
5	2.75	0.41	26	3
6	2.60	0.50	25	4
7	2.45	0.53	26	3
8	2.55	0.53	28	4
9	2.68	0.47	27	5
10	2.80	0.58	28	3
11	2.82	0.55	29	2
12	2.59	0.45	28	3
13	2.56	0.59	26	4
14	2.67	0.52	25	4
15	2.80	0.58	26	3
16	2.75	0.43	25	5
17	2.79	0.57	24	3

图 8 和 9 是有缓存机制与无缓存机制地图显示时的消耗时间和 CPU 资源情况的实验对比。由图分析实验结果如下:没有缓存机制的情况,每次显示地图所需的数据都要从服务器下载,每一层级显示大约会消耗 2.7 s,占用 25% 左右的 CPU 资源,另外显示效率还要取决于当前的网络状况,如果网络状况不好,相应的资源消耗还会有所提升;当增加缓存机制后,地图显示的数据来源会首先检测本地数据,只有没有本地数据,即初次访问某一层级地图数据时才会通过服务器下载,此时每一层级显示大约只消耗 0.5 s 和 5% 的 CPU 资源,有效地提高了地图显示的效率。

4　算法实例

按照以上分析,图 10 给出用 Visual C ++ 6.0 实现的地图实例,图 10(a)为普通地图,地图中心经纬度坐标为(116.3364°,39.9478°),缩放等级 $n = 17$;(b)为卫星地图,地图中心

经纬度坐标为(116.3364°,39.9478°),缩放等级 $n=17$。

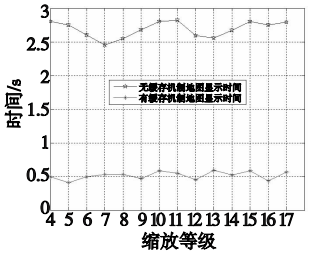


图8 有缓存机制与无缓存机制地图显示消耗时间对比

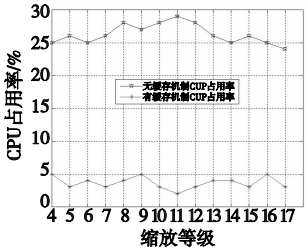
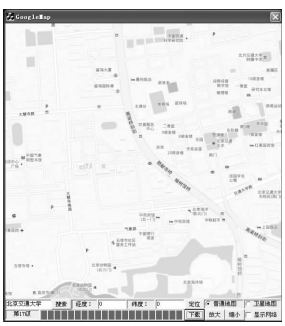


图9 有缓存机制与无缓存机制地图显示CPU占用率对比



(a)普通地图

(b)卫星地图

图10 地图算法实例

两个算法实例中,最后呈现出来的地图实际都是由 9 个地图切片拼接而成的。对于卫星地图,根据给定的中心经纬度坐标和缩放等级,由图 5 所示的流程取得 t 值,从而计算出对应切片的 URL 地址,再按照图 7 所示的 3 × 3 拼图流程,访问服务器下载 9 个数据切片并拼接显示。对于普通地图,其实现方式与卫星地图的区别仅在于位置参数 t 到 x、y、z 的转换。通过实例,可以看出算法并不依赖第三方应用程序接口,算法的实现过程也相对简单,开发人员能够借助算法根据实际经纬度坐标控制地图数据,有效地实现了基于 Web 墨卡托投影的地图在客户端/服务器模式下的应用。算法的不足之处在于,为了提高显示效率而采用的缓存机制在一定程度上增加了存储空间的压力,但是在存储介质容量增大、价格降低的现状下,这一不足在实际开发过程中能够被用户接受。

5 结束语

本文系统地讨论了 Web 墨卡托投影的原理,以推导的变形投影公式为基础,基于地图数据分片技术设计一种基于 Web 墨卡托投影地图下载、拼接、显示和存储的算法,并对算法效率进行仿真验证。算法实例表明,本文研究的算法为基于 Web 墨卡托投影的地图应用于客户端/服务器模式提供了有效的解决方案。

参考文献:

[1] LIU Zao, PIERCE M E, FOX G C. Implementing a caching and tiling map server: a Web 2.0 case study[C]//Proc of International Symposium on Collaborative Technologies and Systems. 2007: 247-256.

[2] LI Hai-ting, FEI Li-fan, WANG Hui-bing. An efficient mechanism for organizing and indexing tile caches on map server[C]//Proc of International Conference on Information Engineering and Computer Science. 2009: 1-5.

[3] 崔金红, 王旭. Google 地图算法研究及实现[J]. 计算机科学, 2007, 34(11): 193-195.

[4] 孙达, 蒲英霞. 地图投影[M]. 南京: 南京大学出版社, 2005.

[5] 任留成, 杨晓梅. 空间墨卡托投影研究[J]. 测绘学报, 2003, 32(1): 78-81.

[6] ENRIQUEZ C. Accuracy of the coefficient expansion of the transverse Mercator projection[J]. International Journal of Geographical Information Science, 2004, 18(6): 559-576.

[7] MERCEDES B S. Simple and highly accurate formulas for the computation of transverse Mercator coordinates from longitude and isometric latitude[J]. Journal of Geodesy, 2009, 83(1): 1-12.

[8] LI Shao-mei, KAN Ying-hong, LIU Hai-yan. Image-based mapping using google earth images[C]//Proc of the 2nd IEEE International Conference on Advanced Computer Control. 2010: 282-285.

[9] 游兰, 彭庆喜. 基于 Google Maps API 的地图解析研究与实现[J]. 湖北大学学报, 2010, 32(2): 161-164.

[10] 夏兰芳, 胡鹏, 黄梦龙. 地图投影解析变换的数值实现方法[J]. 测绘科学, 2007, 32(3): 69-71.

(上接第 4757 页)

[2] FENG Jun, HORACE H S. A multi-resolution statistical deformable model (MISTO) for soft-tissue organ reconstruction[J]. Pattern Recognition, 2009, 42(7): 1543-1558.

[3] 张彦飞. 基于分区的统计颅面重建算法研究与实现[D]. 西安: 西北大学, 2010.

[4] CHEN Xiao-bai, GOLOVINSKIY A, FUNKHOUSER T. A benchmark for 3D mesh segmentation[J]. Journal of ACM Trans on Graphics, 2009, 28(3): 73.

[5] SHAPIRA L, SHAMIR A, COHEN-OR D. Consistent mesh partitioning and skeletonisation using the shape diameter function[J]. The Visual Computer: International Journal of Computer Graphics, 2008, 24(4): 249-259.

[6] ATTENE M, FFALCIDIENO B, SPAGNUOLO M. Hierarchical mesh segmentation based on fitting primitives[J]. The Visual Computer, 2006, 22(3): 181-193.

[7] TAX D, DUIN R. Support vector domain description[J]. Pattern Recognition Letters, 1999, 20(11-13): 1191-1199.

[8] ITTI L, KOCH C, NIEBUR E. A model of saliency based visual attention for rapid scene analysis[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 1998, 20(11): 1254-1259.

[9] RUSINKIEWICZ S. Estimating curvatures and their derivatives on triangle meshes[C]//Proc of International Symposium on 3D Data Processing, Visualization, and Transmission. 2004.

[10] XU Cheng-hua, TAN Tie-niu, WANG Yun-hong, *et al*. Combining local features for robust nose location in 3D facial data[J]. Pattern Recognition Letters, 2006, 27(13): 1487-1494.

[11] WANG Duo, CLARK B, JIAO Xiang-min. An analysis and comparison of parameterization-based computation of differential quantities for discrete surfaces[J]. Computer Aided Geometric Design, 2009, 26(5): 510-527.

[12] PARK S Y, SUBBARAO M. An accurate and fast point-to-point registration technique[J]. Pattern Recognition Letters, 2003, 24(16): 2967-2976.

[13] ZHANG Yan-fei, ZHOU Ming-quan, GEN Guo-hua, *et al*. Face appearance reconstruction based on a regional statistical craniofacial model(RCSM)[C]//Proc of International Conference on Pattern Recognition. 2010: 1670-1673.