N 维 Hilbert 曲线生成算法

李晨阳 段雄文 冯玉才
(华中科技大学计算机学院, 武汉 430074)

摘 要 Hilbert 曲线描述了一种多维空间与 1 维空间一一映射的方法,在图像处理、多维数据索引等领域有着重要的地位。但因为高维 Hilbert 曲线的复杂性,对高维 Hilbert 的相关算法研究很少。提出了产生 *N* 维 Hilbert 曲线的一个新算法。该算法基于静态演化规则,自底向上地分析 *N* 维 Hilbert 曲线编码规律,实现 *N* 维 Hilbert 曲线的编码生成。与现有的算法相比,本文算法易于实现。实验结果表明,该算法具有更好的计算性能。

关键词 算法 编码 *N* 维 Hilbert 曲线 空间填充曲线
中图法分类号: TP301.6 **文献标识码**: A **文章编号**: 1006-8961(2006)08-1068-08

Algorithm for Generating *N*-dimensional Hilbert Curve

LI Chen-yang, DUAN Xiong-wen, FENG Yu-cai
(*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract The Hilbert curve is a way of mapping the multidimensional space into the one-dimensional space. Such mappings are of interest in a number of application domains including image processing and the indexing of multidimensional data. However, little has been discussed on its high dimensional algorithms due to the complexity. In this paper, a novel algorithm is presented for generating an *N*-dimensional Hilbert curve, which analyzes a Hilbert curve from bottom to top, based on a static evolvement rule table. The experimental results show that our method is easier to implement and faster in computation than other methods.

Keywords algorithm, encoding, *N*-dimensional hilbert curve, space-filling curve

1 引 言

自 1890 年,意大利数学家 Peano 提出一族通过一个空间所有的点的曲线[1]以来,许多研究者都开始在这个问题上进行研究。这种曲线也被叫做 Peano 曲线或者是空间填充曲线 。这种映射关系被应用到许多的领域,包括图像的处理[2,3],最近还被应用到多维数据的索引里[4,5]。在这些空间填充曲线里,Z-order 曲线[1]与 Hilbert 曲线[6]是最有名的,因为 Z-order 曲线最易编码而 Hilbert 曲线有最好的聚集性[7]。在许多的领域,空间填充曲线的应用狭隘地被限制成 Z-order 曲线的应用,原因在于Z-order 的简单性。

关于 Hilbert 曲线编码生成,有两种实现方法:一个是表驱动方法,另一个是计算的方法。表驱动方法通过扫描代码扫描列表来实现曲线生成。Fish[8]给出了一个迭代的表驱动版本使得 1 维到 2 维的映射得以执行。Cole[9,10]给出了由 2 维向 1 维转化的逆向的映射表驱动版本。Jin 和 Mellor-Crummey[11]提出了一个空间填充曲线产生的框架来有效地生成空间填充曲线。表驱动的一个最大缺点就是它的空间复杂度很高。计算的方法通过一对一的计算来实现映射。Butz[12]的算法计算与曲线上任意一个点对应的坐标。Faloutsos 和 Roseman[13]给出了一个非迭代的方法,通过分析 Z-order 与 Hilbert 的关系来实现这个映射。Liu 和 Schrack[14]提出了一个由 2 维坐标到 1 维值的计算方法。以上

收稿日期:2005-06-01;**改回日期**:2005-09-06

第一作者简介:李晨阳(1975 ~),男,讲师,2005 年于华中科技大学计算机学院获得计算机与理论专业博士学位。主要研究方向为数据库与多媒体技术。E-mail :tolcy@ hotmail. com

算法大多都局限在可视化的 2 维或 3 维坐标系中。由于 N 维 Hilbert 曲线的复杂性,N 维 Hilbert 曲线编码问题一直是个难点。

近年来,一些针对 N 维 Hilbert 曲线的算法被提出,最具有代表性的是由 Kamata Richard 与 Yukihiro 提出的算法[15]。基于归纳方式,Kamata 提出了一套基于硬件实现 N 维 Hilbert 曲线的生成算法,其算法采用自顶向下分解的策略,逐层分析确定下一层细化的构造形态,根据分析结果动态计算或从终端表里的提取对应 Hilbert 单元进行逐层的细化扩展。通过这个过程的反复,来产生一个 Hilbert 曲线扫描序列。Kamata 算法的计算量包含动态确定每层单元的形态,以及获取所需的单元编码两个部分。这两部分的计算量都是随着 Hilbert 曲线的维数和代数增加而呈指数上升的。第 2 部分的计算有两种方式获取:一种是动态计算,另一种是从预处理生成的终端表中提取。预先生成终端表虽然可以减少一部分计算,但是,终端表的空间复杂度随着维数的提高而急剧上升(5 维约占用 4K 内存,10 维约占用 70M 内存)。综上所述 Kamata 算法具有较大的时空复杂度,因此限制了该算法的软件实现效率,也影响了 N 维 Hilbert 曲线编码的实用性。为此,在对 N 维 Hilbert 曲线编码特征进行深入分析的基础上,提出了一套全新的自底向上的 N 维 Hilbert 曲线编码算法,即从基本单元开始按固定的规则进行迭代变换,实现 N 维 Hilbert 曲线的生成。该算法避开了 Kamata 算法中两大计算中的第 1 个部分,即动态确定每层单元的构造形态。试验对比表明,该算法提高了 N 维 Hilbert 曲线编码的效率,对高维 Hilbert 曲线编码具有明显优势。

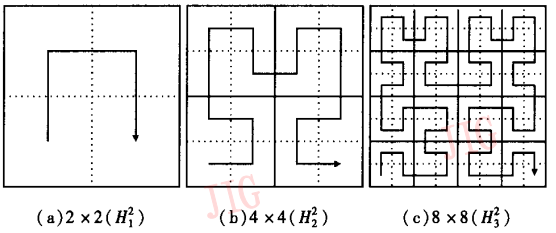
2　分析 Hilbert 曲线

2.1　Hilbert 曲线的定义

首先,用 $\mathbf{R}^N$ 来表示 N 维空间。用 $X_n,\cdots,X_2,X_1$ 来表示 N 维空间 $\mathbf{R}^N$ 的 N 个维。把从 N 维空间坐标转换成的 1 维空间坐标值叫做 Hilbert 码,记为 H_{code}。约定若一条 Hilbert 曲线可以填充一个 $2^m\times 2^m\times\cdots\times 2^m(2^{mN})$ 的 N 维空间,则称该 Hilbert 曲线为一个 N 维 m 代 Hilbert 曲线,用 H_m^N 表示。例如图 1 表示 H_1^2,H_2^2 和 H_3^2。

定义 1　(N 维 Hilbert 单元或细胞:C^N)

当 $m=1$ 时,N 维的 Hilbert 曲线就是一个 N 维 Hilbert 单元或称为 N 维 Hilbert 细胞。用 C^N 表示。

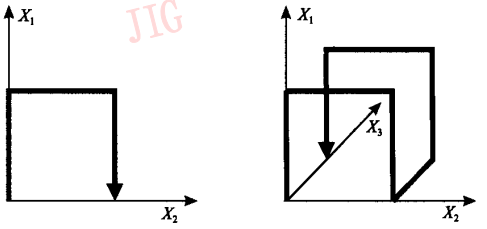


(a)$2\times2(H_1^2)$　(b)$4\times4(H_2^2)$　(c)$8\times8(H_3^2)$

图 1　Hilbert 曲线

Fig. 1　Hilbert curves

图 2(a)是一个 2 维的 Hilbert 单元(C^2),图 2(b)是一个 3 维的 Hilbert 单元(C^3)。



(a) 2D 单元　(b) 3D 单元

图 2　2 维与 3 维 Hilbert 单元

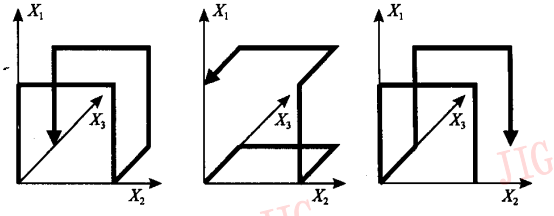
Fig. 2　Examples of 2D and 3D Hilbert cells

定义 2　(N 维 Hilbert 基因)

一个 N 维 Hilbert 基因是一系列坐标变换指令信息列表,它控制如何由 H_{m-1}^N 生成 H_m^N。

坐标变换包含交换("↔")、求反("′")两类。

图 3(a)是一个原始的状态,图 3(b)是通过对 C^3 进行了坐标交换变换后的状态,图 3(c)是通过将 C^3 进行求反变换后得到的状态。



(a) C^3　(b) $X_1\leftrightarrow X_3$ of C^3　(c) X_2' of C^3

图 3　坐标变换的例子

Fig. 3　Examples of coordinates transformation

用 $G[i]$来描述基因列表。这里的 i 表示在 1 维 Hilbert 顺序里的顺序号(即构成 H_m^N 里的 H_{m-1}^N 的位置顺序)。此外用 $H_m^N[i]$来表示 H_m^N 按照基因指令转换

后的结果。表 1 是一个 3 维的 Hilbert 基因列表。

表 1 $G[i]$：3 维 Hilbert 基因列表

Tab.1 $G[i]$:3D Hilbert gene list

位置	交换	求反
0	$X_1 \leftrightarrow X_3$	-
1	$X_2 \leftrightarrow X_3$	-
2	-	-
3	$X_1 \leftrightarrow X_3$	X'_1, X'_3
4	$X_1 \leftrightarrow X_3$	-
5	-	-
6	$X_2 \leftrightarrow X_3$	X'_2, X'_3
7	$X_1 \leftrightarrow X_3$	X'_1, X'_3

注：“-”表示没有转换(“-” denote no transformation)

$G[i]$表示出如何通过坐标的变换来实现 $H_m^3 \rightarrow H_m^3[i]$ 的转换。例如，通过 $X_1 \leftrightarrow X_3$ 实现了 $H_1^3 \rightarrow H_1^3[0]$ 的转换，如图 3(b)所示。这是本文新算法的思想基础的关键点。

2.2 N 维 Hilbert 曲线的生成

从上面的分析可以看出：C^N 是 Hilbert 曲线最基本的元素，就好像一个生物体的细胞一样。$G[i]$ 则是定义了如何通过 C^N 变换来合成一个 N 维 Hilbert 曲线，这好比生物的基因一样。所以，给出一个 C^N 与 $G[i]$，可以决定整个 Hilbert 曲线的细节。下面用 3 维 Hilbert 曲线作为例子来说明一个 Hilbert 曲线的生成过程。

图 4 显示了 8 个 $H_1^3[i]$ 是如何构成 H_2^3 的。

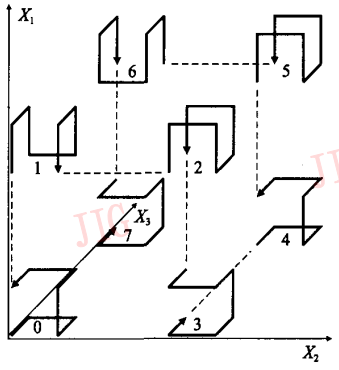


图 4 $H_1^3(C^3) \rightarrow H_2^3$ 按照 $G[i]$

Fig.4 $H_1^3(C_3) \rightarrow H_2^3$ according to $G[i]$

由图 4 所示，能很直观地理解 H_1^3 生成 H_2^3 的过程：通过 $X_1 \leftrightarrow X_3$ 转换生成 $H_1^3[0]$；通过 $X_2 \leftrightarrow X_3$ 转换生成 $H_1^3[1]$；通过直接复制生成 $H_1^3[2]$；通过 $X_1 \leftrightarrow X_3$、X'_1、X'_3 转换生成 $H_1^3[3]$；通过 $X_1 \leftrightarrow X_3$ 转换生成 $H_1^3[4]$；通过直接复制生成 $H_1^3[5]$；通过 $X_2 \leftrightarrow X_3$、X'_2、X'_3 转换生成 $H_1^3[6]$；通过 $X_1 \leftrightarrow X_3$、X'_1、X'_3 转换生成 $H_1^3[7]$；对生成的 $H_1^3[i]$ ($i=0,1\cdots,7$) 按下标顺序进行拼接，这样实现 3 维二代 Hilbert 曲线 H_2^3 的生成。这是一个可迭代的过程，将 H_2^3 作为初始的元素，重复上面的步骤，可以生成 H_3^3。从这个例子，可以得出在给出 C^N 和 $G[i]$ 的情况下，如何由 H_{m-1}^N 生成 H_m^N 的过程。

3 预处理阶段：产生 C^N 与 $G[i]$

在这一节里，提出 Hilbert 单元与 Hilbert 基因的产生算法，它们是编码的基础。预处理部分只需在 Hilbert 曲线编码前执行一次就行了。

3.1 生成 C^N

这里讨论生成 N 维 Hilbert 单元的算法的细节。算法从编码、解码两个不同的方面对 Hilbert 进行了描述，基于归纳给出了 Forward 编码和 Backward 编码两种算法。表 2、表 3 所示是两种不同编码方式的 3 维单元编码表。

表 2 3 维 Forward 扫描单元

Tab.2 3D Forward scan cell

X_3	X_2	X_1	H_{code}
0	0	0	000
0	0	1	001
0	1	0	011
0	1	1	010
1	0	0	111
1	0	1	110
1	1	0	100
1	1	1	101

表 3 3 维 Backward 扫描单元

Tab.3 3D Backward scan cell

H_{code}	X_3	X_2	X_1
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

Forward 编码算法可以如下描述:

$$\begin{cases} C^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ C^n = Merge\left(\begin{matrix} 0 \oplus C^{n-1} \\ 1 \oplus Reverse(C^{n-1}) \end{matrix} \right) \quad n = 2,3,\cdots \end{cases} \tag{1}$$

其中,*Reverse*( )表示按位求反,*Merge* 表示按顺序合并结果,⊕表示串联。

Backward 算法如下描述:

$$\begin{cases} C^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ C^n = Merge\left(\begin{matrix} 0 \oplus C^{n-1} \\ 1 \oplus Reverse(X_{n-1} \text{ of } C^{n-1}) \end{matrix} \right) \quad n = 2,3,\cdots \end{cases} \tag{2}$$

Forward 编码算法可以高效地实现从 $\mathbf{R}^N$ 向 $\mathbf{R}^1$ 转换。Backward 编码算法可以高效地实现 $\mathbf{R}^1$ 向 $\mathbf{R}^N$ 转换。Forward 编码算法空间效率高,但在计算机上难于表达,所以在实现时,以 Backward 编码算法为主。在 Backward 编码算法的情况下,一个单元用了一个 1 维的数组表达:

$$C^N[H_{code}] = X_n \cdots X_1$$

H_{code}是在 Hilbert 曲线的 1 维坐标值,$X_n \cdots X_1$ 是在 $\mathbf{R}^N$ 的坐标值。举例来说,表 3 可以表示为 $C^3[0]=000, C^3[1]=001, C^3[2]=011, C^3[3]=010, C^3[4]=110, C^3[5]=111, C^3[6]=101, C^3[7]=100$。

3.2　产生 *N* 维 Hilbert 基因

Hilbert 曲线最重要的特性就是自相似性。通过研究发现,分析隐藏在 *N* 维 Hilbert 单元里的信息,可以得到一个 *N* 维 Hilbert 曲线的构成信息:基因。这需要分两步来实现:第 1 步是确定 $H_1^N[i]$ 的入口与出口;第 2 步就是通过分析入口与出口来产生它的基因列表 $G[i]$。

3.2.1　$H_1^N[i]$的入口与出口的确定

用 $X_n \cdots X_1$ 表示一个 $\mathbf{R}^N$ 坐标值,例如 $\mathbf{R}^3$ 的坐标值表示为 $X_3X_2X_1$。'*i*'表示在 R^1 空间的 H_{code}。

先给出确定 $H_1^N[i]$入口、出口点算法:

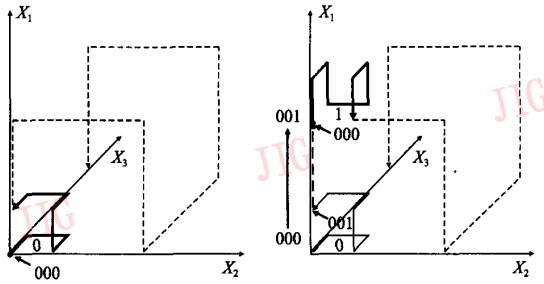
```
Confirm_Entry_Exit(input i)
{
  if (i<2^N/2){
    if (i==0){
      H1^N[i][0]=C^N[0];
      H1^N[i][1]=H1^N[i][0]∧(C^N[0]∧C^N[1]);
    }
    else{
      H1^N[i][0]=H1^N[i-1][1]∧(C^N[i-1]∧C^N[i]);
      if(H1^N[i][0]&(C^N[i]∧C^N[i+1])==C^N[i]&
        (C^N[i]∧C^N[i+1])){
        H1^N[i][1]=H1^N[i][0]∧(C^N[i]∧C^N[i+1]);
      }
      else{
        H1^N[i][1] has N-1 alternative options;
      }
    }
  else{
    if(i==2^N/2){
      H1^N[i][0]=H1^N[2^N-i-1][1]∧(0x00000001<<n-1);
      H1^N[i][1]=H1^N[i][0]∧(H1^N[2^N-i-1][0]∧H1^N[2^N-
                 i-1][1]);
    }
    else{
      H1^N[i][0]=H1^N[i-1][1]∧(H1^N[2^N-i][0]∧H1^N[2^N-
                 i-1][1]);
      H1^N[i][1]=H1^N[i][0]∧(H1^N[2^N-i-1][0]∧H1^N[2^N-
                 i-1][1]);
    }
  }
}
```

说明:$H_1^N[i][0]$是 $H_1^N[i]$的入口,$H_1^N[i][1]$是 $H_1^N[i]$的出口。符号'∧'表示 *XOR*(0∧1=1;0∧0=0;1∧1=0),符号'&'表示 *AND*(0&1=0;0&0=0;1&1=1)。

入口确定分两种情况:

情况 1　$H_1^N[0]$的入口点的坐标值一定是与 *N* 维单元码的入口点相同的。如图 5(a)所示,$C^3[0]$坐标值为 000,则 $H_1^3[0]$的入口坐标值也为 000。

情况 2　$H_1^N[i-1]$和 $H_1^N[i]$的连接方向是与 $C^N[i-1]\rightarrow C^N[i]$方向一致的,并且 $H_1^N[i]$的入口



(a) 确定 $H_1^3[0]$的入口　　(b) 确定 $H_1^3[1]$的入口

图 5　$H_1^N[i]$入口点的确定

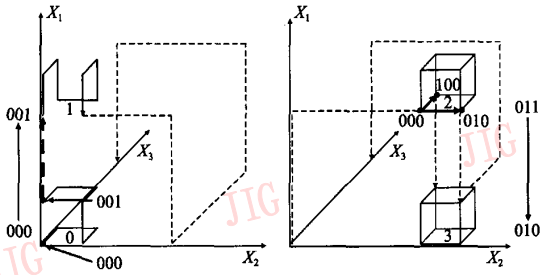
Fig. 5　Confirming the entry point of $H_1^N[i]$

的坐标值必须确保 $H_1^N[i]$ 和 $H_1^N[i-1]$ 的可连接性，所以由 $H_1^N[i-1]$ 的出口坐标到 $H_1^N[i]$ 的入口坐标中发生变化的一维，必须与 $C^N[i-1]\rightarrow C^N[i]$ 所发生变化的那一维是相同的。如图 5(b)所示：000→001 表示 $C^3[0]\rightarrow C^3[1]$ 走向，它是在 X_1 轴方向的变化，所以 $H_1^3[0]$ 的出口坐标→$H_1^3[1]$ 的入口坐标变化必须也是在 X_1 轴方向的，即 001→000。

出口点确定也分两种情况。

$H_1^N[i]$ 和 $H_1^N[i+1]$ 的连接方向必须与 $C^N[i]\rightarrow C^N[i+1]$ 的方向一致，并且，$H_1^N[i]$ 出口的坐标值必须确保 $H_1^N[i]$ 与 $H_1^N[i+1]$ 之间的可连接性。

情况 1　$H_1^N[i]$ 的入口点不能直接连到 $H_1^N[i+1]$，则 $H_1^N[i]$ 的入口到出口的走向一定与 $C^N[i]\rightarrow C^N[i+1]$ 的走向一致。如图 6(a)，$H_1^3[0]$ 的入口点坐标是 000，无法直接与 $H_1^3[1]$ 进行连接，所以，$H_1^3[0]$ 对出口的要求就是必须使得出口点可以连接 $H_1^3[1]$，所以只要保证入口到出口的变化与 $C^N[0]\rightarrow C^N[1]$ 的走向一致，就可以保证连接性了，所以是 000→001。



(a) 确定 $H_1^3[0]$ 的出口　　(b) 确定 $H_1^3[1]$ 的出口

图 6　$H_1^N[i]$ 的出口点的确定

Fig. 6　Confirming the exit point of $H_1^N[i]$

情况 2　$H_1^N[i]$ 的入口点可以直接连到 $H_1^N[i+1]$，则 $H_1^N[i]$ 的入口到出口的走向是除 $C^N[i]\rightarrow C^N[i+1]$ 发生变化的维方向以外的其他 $N-1$ 种选择。如图 6(b)，$H_1^3[2]$ 的入口点(000)可以直接连接 $H_1^3[3]$，而 $C^N[2]\rightarrow C^N[3]$ 的变化是在 X_1 轴方向的(011→010)，所以其出口点只能是 100 或 010 两种选择。

Hilbert 曲线具有对称性的特征，所以只需分析一半单元的出入口点，剩下的根据对称性就可以得到。

3.2.2　产生 N 维 Hilbert 曲线的基因列表

现在已经可以确定 $H_1^N[i]\ (i=0,\cdots,2^N-1)$ 的入口与出口，下一步，给出一个通过分析入口与出口来确定基因列表的算法。这里用 $G[i][0]$ 来表示交换，用 $G[i][1]$ 来表示求反。

交换的变换算法可以用下面的式子表示：

$$G[i][0]=(C^N[0]\wedge C^N[2^N-1])\wedge(H_1^N[i][0]\wedge H_1^N[i][1])$$

例如，C^3 的入口→出口的变化发生在 X_3 轴方向，因为 C^3 的入口($C^3[0]$)的坐标值是 000，C^3 的出口($C^3[7]$)坐标值是 100，如图 3(a)。$H_1^N[0]$ 的入口→出口的变化发生在 X_1 轴方向，因为 $H_1^N[0]$ 的入口的坐标值是 000，$H_1^N[0]$ 的出口的坐标值是 001，如图 3(b)。这样交换变换就一定是在 X_3 与 X_1 间发生了($G[0][0]=101$)。

求反的变换算法可以用下面的式子表示：

$$G[i][1]=C^N[0]\wedge H_1^N[i][0]$$

例如，C^3 的入口($C^3[0]$)是 000，$H_1^3[3]$ 的入口是 101。这样，求反变换一定发生在 X_3 和 X_1 之间($G[3][1]=101$)。

这样 3 维 Hilbert 基因列表就可以如下表示：$G[0][0]=101$，$G[0][1]=000$；$G[1][0]=110$，$G[1][1]=000$；$G[2][0]=000$，$G[2][1]=000$；$G[3][0]=101$，$G[3][1]=101$；$G[4][0]=101$，$G[4][1]=000$；$G[5][0]=000$，$G[5][1]=000$；$G[6][0]=110$，$G[6][1]=110$；$G[7][0]=101$，$G[7][1]=101$。

4　Hilbert 曲线快速生成算法

以 3 维 Hilbert 曲线生成为例，给出 Hilbert 曲线的生成算法。

具体的 Hilbert 曲线编码的产生步骤如下：

(1) 根据基因表产生 $H_{m-1}^N\rightarrow H_{m-1}^N[i]$ ($m=2,3,\cdots$)，以 $H_1^3\rightarrow H_1^3[i]$ 为例描述这个过程。

$H_1^3\rightarrow H_1^3[0]$ by $G[0](X_1\leftrightarrow X_3)$：$H_1^3[0]$ = (000, 100, 110, 010, 011, 111, 101, 001)；

$H_1^3\rightarrow H_1^3[1]$ by $G[1](X_2\leftrightarrow X_3)$：$H_1^3[1]$ = (000, 001, 101, 100, 110, 111, 011, 010)；

$H_1^3\rightarrow H_1^3[2]$ by $G[2]$ (replication)：$H_1^3[2]$ = (000, 001, 011, 010, 110, 111, 101, 100)；

$H_1^3\rightarrow H_1^3[3]$ by $G[3](X_1\leftrightarrow X_3; X_1', X_3')$：$H_1^3[3]$ = (101, 001, 011, 111, 110, 010, 000, 100)；

$H_1^3\rightarrow H_1^3[4]$ by $G[4](X_1\leftrightarrow X_3)$：$H_1^3[4]$ = (000, 100, 110, 010, 011, 111, 101, 001)；

$H_1^3\rightarrow H_1^3[5]$ by $G[5]$ (replication)：$H_1^3[5]$ = (000, 001, 011, 010, 110, 111, 101, 100)；

$H_1^3 \rightarrow H_1^3[6]$ by $G[6](X_2 \leftrightarrow X_3; X_2', X_3')$: $H_1^3[6]$ = (110,111,011,010,000,001,101,100);

$H_1^3 \rightarrow H_1^3[7]$ by $G[7](X_1 \leftrightarrow X_3; X_1', X_3')$: $H_1^3[7]$ = (101,001,011,111,110,010,000,100);

(2)从 $H_{m-1}^N[i]$产生 H_m^N,以 $H_1^3[i] \rightarrow H_2^3$ 为例:

H_2^3 = *Merge* (000 ⊕ $H_1^3[0]$, 001 ⊕ $H_1^3[1]$, 011⊕$H_1^3[2]$,010 ⊕ $H_1^3[3]$, 110 ⊕ $H_1^3[4]$, 111 ⊕ $H_1^3[5]$,101⊕$H_1^3[6]$, 100 ⊕ $H_1^3[7]$)　($C^3[i]$ ⊕ $H_1^3[i]$)。例如:000 ⊕ $H_1^3[0]$ = (000 ⊕ 000, 000 ⊕ 100,000⊕110, 000⊕010, 000 ⊕ 011, 000 ⊕ 111, 000⊕101, 000 ⊕ 001) = (000000, 000100, 000110, 000010,000011,000111,000101,000001)。

H_2^3 编码结果如下:

H_2^3 = (000000,000100,000110,000010,000011, 000111, 000101, 000001, 001000, 001001, 001101, 001100, 001110, 001111, 001011, 001010, 011000, 011001, 011011, 011010, 011110, 011111, 011101, 011100, 010101, 010001, 010011, 010111, 010110, 010010, 010000, 010100, 110000, 110100, 110110, 110010, 110011, 110111, 110101, 110001, 111110, 111111, 111011, 111010, 111000, 111001, 111101, 111100, 101110, 101111, 101011, 101010, 101000, 101001, 101101, 101100, 100101, 100001, 100011, 100111, 100110, 100010, 100000, 100100)。

其中,生成的序列码是用 Z-order 码表示的 3 维空间坐标值,例如:$H_2^3[0]$ = 000000, $H_2^3[1]$ = 000100,$H_2^3[0]$ =000110 分别表示 $\mathbf{R}^3$ 空间中的(00,00,00),(01,00,00)和(01,01,00)坐标点。下标是 Hilbert 曲线 $\mathbf{R}^1$ 空间中的序号。重复步骤 1、2,可以产生 N 维 m 代的 Hilbert 曲线的编码扫描表,也就是生成了任意维任意代的 Hilbert 曲线。

5　实验分析

用 C 语言对算法进行实现,试验平台为 P4 1.6G CPU,512M 内存。令 n 为 N 维 Hilbert 单元中点的数量,那么 n 和 N 之间的关系可以表示为 $n = 2^N$。设 m 为 N 维 Hilbert 曲线的代数。

5.1　预处理

这里以 N = 2,3,4,5,10,20 为例,讨论比较算法时空复杂度。

首先讨论算法空间复杂度,表 4 显示了本文算法和 Kamata 的算法在空间性能上的比较。

表 4　空间性能上的比较

Tab. 4　Comparison of storage costs

单位:bytts

N	本文算法	Kamata 算法
2	48	8
3	96	84
4	192	576
5	384	3 840
10	12 288	15 073 280
20	12 582 912	60 473 139 527 680

本文算法预处理部分的空间复杂度主要包含标准单元 C^N 和基因表 $G[i]$的存储空间耗费,其计算表达式为 $3 \times 4 \times 2^N$bytes。与之相比,Kamata 算法则需要 $(N \times U + N^2) \times 2^{2N-4}$ bytes (U 是一个满足 $2^{U-1} \leqslant N2^{N-1} \leqslant 2^U$条件的自然数)[15]。由于,Kamata 算法的预处理部分包含生成全方向单元编码的终端表,所以其空间复杂度随维数提高而呈指数上升。从表 4 的数据可以看出,Kamata 算法随着维数提高到 10 维以上后,空间开销将使得算法无法在内存一级处理。而采用动态计算单元编码时,Kamata 算法效率的优势将完全丧失。这样严重限制了 Kamata 算法的软件实现。因此 Kamata 算法避开了软件的算法实现,而采用了硬件实现的思路。与 Kamata 算法相比,本文算法对高维 Hilbert 预处理的空间复杂度具有明显优势。因此,能够在内存一级实现高维 Hilbert 曲线编码的预处理过程(25 维约占用 400M 内存),从而提高了 N 维 Hilbert 曲线编码的效率。

在算法时间复杂度上,本文算法的开销为 $O(2^N)$。由于存储空间的限制,Kamata 算法无法进行计算机的实现。图 7 显示本文方法预处理时间与维数 N 的关系。

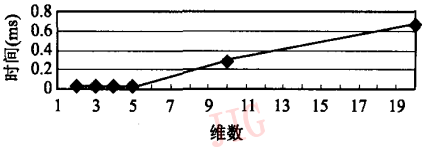


图 7　预处理的时间消耗

Fig. 7　Time costs of pretreatment

从图 7 的数据可以知道,本文算法的预处理时间开销非常小。同时,由于预处理部分在曲线编码生成过程中只需执行一次,因此这样的时间开销可

以忽略。完全能够满足实际的需求。

通过以上讨论,可以发现在预处理部分主要的限制是内存空间的问题。本文算法的内存开销在小于 25 维都是可行的,这远远超过 Kamata 算法,能够满足绝大多数应用的需求。而超过 25 维的应用时,本文算法可以根据需要,从内存扩展到文件系统或者是数据库中。

5.2 Hilbert 曲线编码

采用本文算法和 Kamata 算法,对 $N=3,4,5$ 和 $G=3,4,5$ 的 Hilbert 曲线进行了的生成实现。图 8 显示了本文算法和 Kamata 算法在时间性能上的比较。在 Hilbert 曲线编码算法上,Kamata 算法采用的是自顶向下分解的策略。本文的 Hilbert 曲线编码算法采用的是自底向上迭代拼接的策略。由于采用方法的不同,Kamata 算法在逐级向下扩展的过程中,演化规则需要动态确定;而本文算法的演化规则是固定不变的。这个差异决定了算法效率的差异。Kamata 算法在逐级向下扩展的过程中需要动态确定扩展的细节,即确定内部单元的起终点方向。而这个计算是随着 Hilbert 曲线代数 m 的提高而呈指数提高,其时间复杂度描述为 $O(2^{N\times(m-1)})$。在本文的算法中,起相同作用的基因表,它的生成时间复杂度为 $O(2^N)$。从这可以看出它是与代数 m 无关的,并且只需在预处理阶段执行一次。因此节省了一个复杂度高的计算过程。而在扩展编码变换部分,本文算法和 Kamata 算法的时间复杂度都是 $O(2^{N\times m})$。Kamata 算法试图依靠预处理生成单元的终端表的方法消除扩展编码变换部分,即 $O(2^{N\times m})$ 的计算量。但由于空间开销的原因使该思路并不可行。因此 Kamata 算法的效率优势是理论上的。而本文算法是从根本上避开了一个高时间复杂度的计算,因此在效率上具有明显优势。

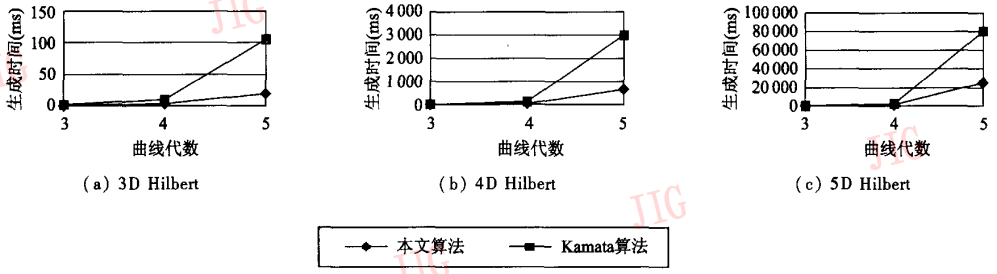


图 8 计算时间的比较

Fig. 8 Comparison of the computation costs

6 结 论

针对 N 维 Hilbert 曲线编码这个传统的难题进行研究,给出了一套全新 N 维 Hilbert 曲线快速编码的算法。基于对 N 维 Hilbert 曲线的深入分析,在自相似特征中归纳出了 Hilbert 曲线自底向上固定方式演化的规律,并研究了获取固定演化规则的分析算法。最后提出了基于该思想的 N 维 Hilbert 曲线编码算法。通过分析和试验,与现有算法相比,该算法对高维 Hilbert 曲线在编码效率上具有优势。此外,建立 Hilbert 一一映射算法将是今后研究的重点。

参考文献(References)

1 Peano G. Sur une courbe qui remplit touteune aire plane [J]. Mathematische Annalen, 1890, **36**:157 ~ 160.

2 Biswas S. One-dimensional B-B polynomial and Hilbert scan for graylevel image coding[J]. Pattern Recognition, 2004, **37**(4): 789 ~ 800.

3 Stevens R J, Lehar A F, Preston F H. Manipulation and presentation of multi-dimensional image data using the peano scan[J]. IEEE Transations on Pattern Analysis and Machine Intelligence, 1983, **5**(9): 520 ~ 526.

4 Chen H, Chang Y. Neighbor-finding based on space-filling curves [J]. Information Systems, 2005, **30**(3): 205 ~ 226.

5 Mokbel M F, Aref W G. Irregularity in multi-dimensional space-filling curves with applications in multimedia databases[A]. In: Proceedings of the 10th ACM SIGMIS Information and Knowledge Management[C], Atlanta, Georgia, USA, 2001: 512 ~ 519.

6 Hilbert D. Über die stetige Abbildung einer Linie auf ein Flächenstück[J]. Mathematische Annalen, 1891, **38**: 459 ~ 460.

7 Moon B, Jagadish H V, Faloutsos C, *et al*. Analysis of the clustering properties of the Hilbert space-filling Curve[J]. IEEE Transactions on Knowledge and Data Engineering, 2001, **13**(1): 124 ~ 141.

8 Fisher A J. A new algorithm for generating Hilbert curves [J]. Software-Practice and Experience, 1986, **16**(1): 5 ~ 12.

9 Cole A J. Direct transformations between sets of integers and Hilbert polygons[J]. International Journal of Computer Mathematics, 1986, **20**(3): 115 ~ 122.

10 Cole A J. Compaction techniques for raster scan graphics using space-filling curves[J]. The Computer Journal, 1987, **30**(1): 87 ~ 92.

11 Jin G, Mellor-Crummey J. A framework for efficient generation of multi-dimensional space-filling curves by recursion [J]. ACM Transactions on Mathematical Software (TOMS) Archive, 2005, **31**(1): 120 ~ 148.

12 Butz A R. Alternative algorithm for Hilbert's space-filling curve[J]. IEEE Transactions on Computers, 1971, **20**(4):424 ~ 426.

13 Faloutsos C, Roseman S. Fractals for secondary key retrieval[A]. In: Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems [C], Philadelphia, Pensylvania, USA, 1989: 247 ~ 252.

14 Liu X, Schrack G. Encoding and decoding the Hilbert order[J]. Software-Practice and Experience, 1996, **26**(12):1335 ~ 1346.

15 Kamata S, Eason R O, Bandou Y. A new algorithm for N-dimensional Hilbert scanning [J]. IEEE Transactions on Image Processing, 1999, **8**(7): 964 ~ 973.