在游戏中利用邻域特性扩展的 kd-tree 及其查找算法

徐建民[1] 李 欢[1] 刘博宁[2]

(河北大学数学与计算机科学学院 保定 071002)[1] (河北大学工商学院 保定 071002)[2]

摘 要 处理场景中数量庞大的各种对象间的交互是游戏的一类主要计算工作。将 kd-tree 用于组织场景,提高了这类计算的效率。传统算法采用树的层次遍历方式进行查找,处理跨节点情况时性能下降明显。提出了邻域特性概念以扩展传统 kd-tree 结构,增添了树节点间的平面邻接关系,且考虑了游戏对 kd-tree 的一些限定,设计了从起始节点向四周扩展的查找算法。经分析与实验证明,新算法比传统算法有约 40% 的性能提升且更稳定。

关键词 邻域特性,kd-tree,查找,场景分割,游戏

中图法分类号 TP391 **文献标识码** A

Neighbor Feature Extended kd-tree and Searching Algorithm for Game

XU Jian-min[1] LI Huan[1] LIU Bo-ning[2]

(College of Mathematics and Computer Science, Hebei University, Baoding 071002, China)[1]

(College of Industrial and Commercial, Hebei University, Baoding 071002, China)[2]

Abstract Processing the interactions among large numbers of objects is the main computation task in game system. Using kd-tree to organize the game scene improves such computation. There's a obvious performance degradation in situation of node-crossings as traditional algorithm uses hierarchically recursive way to search. The concept of neighbor feature was proposed to extend traditional kd-tree structure, so the planar adjacent relationship of hierarchical nodes was added. A new algorithm searching the tree in a 4-sides expanding way from the standing node as the center was devised. The analysis and simulation showed that the new algorithm improves the performance by about 40% and is more stable than the traditional one.

Keywords Neighbor feature, kd-tree, Searching, Scene partitioning, Game

兴趣域(AOI: Area Of Interest) 角色感兴趣的区域。一般定义为以角色为中心的矩形或圆形区域。本文假定 AOI 被定义为左上角坐标为(left, top)和右下角坐标为(right, bottom)的矩形区域。新算法不受限于 AOI 是否圆形或矩形。

树节点(TN: Tree Node) kd-tree 结构中的一个节点,代表场景中一个左上角坐标为(left, top)和右下角坐标为(right, bottom)的矩形子区域。TN_X 表示树节点 X。

叶节点(LN: Leaf Node) 不含孩子的 TN,它关联着其所代表的区域内的事件列表。LN_X 表示叶节点 X。

相关节点(RN: Relative Node) 是一个 LN,且其区域与 AOI 有交集。RN_X 表示相关节点 X。

驻扎节点(SN: Standing Node) 角色当前所处的 RN。SN_X 表示驻扎节点 X。

分割维度(SD: Splitting Dimension) kd-tree 中对区域进行分割的分割面。

X 分割维度(SDX: Splitting Dimension X) 以 x 坐标为依据对区域进行横向分割的 SD。

Y 分割维度(SDY: Splitting Dimension Y) 以 y 坐标为依据对区域进行纵向分割的 SD。

分割位置(SC: Splitting Coordinate) 采用 SDX 时的 x 坐标值或采用 SDY 时的 y 坐标值。

1 前言

游戏系统,特别是服务器中,逻辑处理最繁重的任务就是判断游戏场景中各个对象间的交互情况,例如判断角色能否触发场景中的某事件。在实际中,这表现为触发任务点的任务领取、触发 NPC 对话、触发机关打开暗门等。

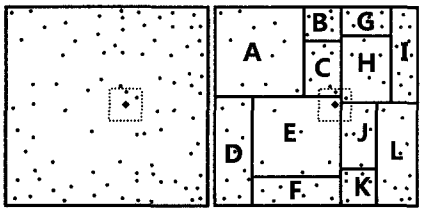
当今游戏,特别是 MMOG(大规模多人在线游戏),设定的事件量非常之大,系统同时服务数十万、百万角色,虽然使用分布式集群技术设计服务器系统,实现负载平衡[6],但每个服务器仍需处理几千角色。在庞大的信息集中,能否高效地查找出角色感兴趣的信息,将影响整个系统性能。

游戏中,角色与事件间的距离决定能否产生交互。而虚拟世界非常辽阔,角色在某时刻仅能感受到世界的一小部分。从而以角色为中心、视距为兴趣半径,便可定义出角色的兴趣域(AOI)[4]。角色仅可能与处于其 AOI 内的事件发生交互,这个现象带有很强的地域性。如图 1(a)所示,小菱形为角色,虚线矩形为角色 AOI,小原点表示事件。文献[5]针对常见的查找算法在此方面的应用做了讨论:用顺序表查找时,往往中标与不中标的比例达到 1:10000,浪费大量查找时间。

到稿日期:2010-04-05 返修日期:2010-07-16 本文受中国博士后科学基金(20070420700)资助。

徐建民(1966-),男,教授,主要研究方向为虚拟现实及信息检索;李 欢(1981-),男,硕士生,主要研究方向为虚拟现实,E-mail:iqxtreme@yahoo.com.cn(通信作者);刘博宁(1982-),女,硕士生,主要研究方向为模式识别。

该文献还指出，游戏中信息与角色的位置是可移动的，用二分查找法会在段内重排序上给系统带来很大的计算负担；再者，由于信息位置的随机性、密度不均匀性，采用哈希表时会带来极大的内存资源浪费，且哈希函数难以制定；由于游戏中的查找大部分属于基于坐标的查找，因此采用了空间二叉树（即 kd-tree，$k=2$）。



(a)　(b)

图 1　(a)未经空间分割的原始场景；(b)经 kd-tree 分割后的场景

但在传统 kd-tree 中进行查找要进行层次遍历，跨越深层叶节点时要访问过多的无关节点，影响了查找效率。本文根据游戏场景分割特点，在提出邻域特性的基础上扩充 kd-tree 结构，设计了更高效、稳定的查找算法。

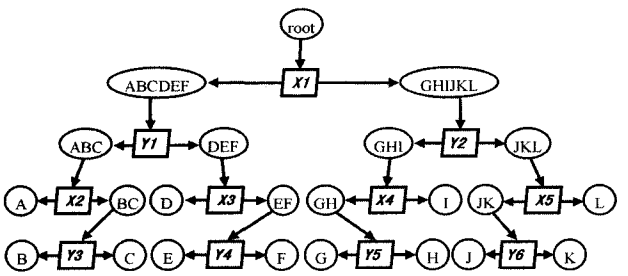
2　传统 kd-tree

2.1　传统 kd-tree 在游戏中的应用

kd-tree 是一种根据 k 维空间中的点集对空间进行分割的数据结构，采用多维索引值进行查找，常用于范围查找和最近邻查找等，是一种特殊的二叉空间分割树[1,2]。

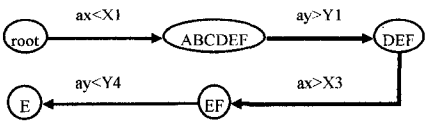
kd-tree 可将一个给定的空间进行多级分割，每一级轮换使用各个维度将空间或子空间一分为二。比如 2d-tree，第一级按 x 维度分割，得到两个子空间，接着对两个子空间分别按 y 维度分割，再到下一级时又按 x 维度分割，最终将给定空间分为若干矩形区域，以每个区域包含的事件数小于一个阈值作为停止分割标志，或者以分割区域尺寸不得小于 AOI 尺寸为停止分割标志（这是游戏场景的分割特点，分割场景的目的在于快速排除无关区域中的对象。若 AOI 尺寸大于分割区域，即可同时跨越 4 个以上区域，则分割将在一定程度上失去意义）。

图 1(a)中包含 84 个事件，限定每个分割区域最多含有 11 个事件，利用 kd-tree 对其分割的结果如图 1(b)所示，树结构如图 2 所示。在此结构下查找可能与角色发生交互的事件的过程为：1)如图 3 所示，根据 kd-tree 特点从根节点逐层向下，通过比较角色坐标分量与节点的 SC，确定 SN 为 LN_E；2)如图 4 所示，从 LN_E 先向上找到完全包含角色 AOI 的节点，再向下遍历树结构，查找与 AOI 相交的节点，最终确定 RN 为 LN_C，LN_E，LN_H，LN_J；3)计算这 4 个 RN 所包含的 22 个事件可否与角色发生交互。kd-tree 将 84 个待测事件筛选为 22 个待测事件，从而极大地提高了系统性能。



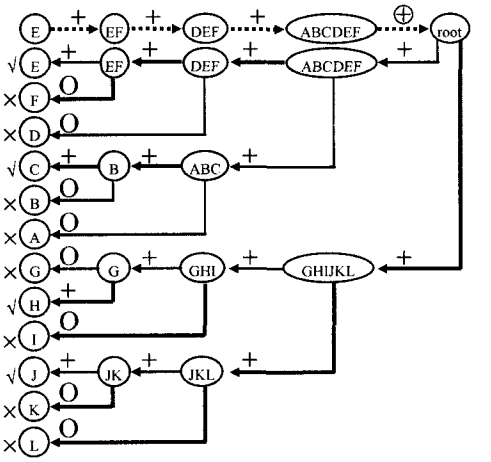
$X1$ 到 $X5$，$Y1$ 到 $Y6$ 表示各个子区域的分割维度，箭头表示关联方向。

图 2　场景的 kd-tree 结构



ax 与 ay 是角色坐标，$X1$，$X3$，$Y1$，$Y4$ 表示对应的 SC，细路径表示选择左子树，粗路径表示选择右子树。

图 3　确定 SN



虚线表示向上遍历，细线表示向下遍历左子树，粗线表示向下遍历右子树，
⊕表示目的节点完全包含 AOI，+表示目的节点与 AOI 相交但不包含，
O表示目的节点与 AOI 无交集，√表示确定该节点为 RN，
×表示确定该节点不为 RN。

图 4　传统的 kd-tree 查找 RN 的过程

2.2　传统 kd-tree 查找算法要点描述

由图 4 可知，首先要从 SN 开始一层层向上遍历到一个包含 AOI 的 TN。此过程中，对遍历到的 TN 进行的主要处理是判断其是否包含 AOI，如伪代码 1 所示。找到包含 AOI 的 TN 后，以其为起始节点，向下遍历以找到所有 RN。此过程中，对遍历到的 TN 进行的主要处理是判断其是否与 AOI 有交集，如伪代码 2 所示，相关数据结构如表 1 和表 2 所列。

伪代码 1　判断一个 TN 是否包含 AOI

```
if(TN. sdL. sc <= AOI. left &&. TN. sdT. sc <= AOI. Top &&.
  TN. sdR. sc >= AOI. right &&. TN. sdB. sc >= AOI. bottom)
{  TN 包含 AOI;  }
else { TN 不包含 AOI;}
```

伪代码 2　判断一个 TN 是否与 AOI 有交集

```
if(AOI. left >= T. sdR. sc || AOI. top >= TN. sdB. sc ||
  AOI. right <= TN. sdL. sc || AOI. bottom <= TN. sdT. sc)
{  AOI 与 TN 无交集;}
else { AOI 与 TN 有交集;}
```

2.3　传统 kd-tree 的缺点

传统 kd-tree 查找算法需要进行层次遍历，在处理跨深层节点情况时，性能下降明显，这是多维树普遍存在的缺点[3]。此种情况正如图 4 所示，为了确定 4 个 RN，传统算法遍历了所有 23 个树节点，且某些节点遍历了两遍。这是因为对场景分割后，传统 kd-tree 结构仅表现出每对被分割出的两个子区域间的邻接关系，即左右孩子的邻接关系。这个邻接关系由分割它们父节点的 SD 维护。如图 1(b)、图 2 所示，LN_E 和 LN_F 邻接，它们的邻接关系由分割它们父节点 TN_{EF} 的分割面 Y4 维护。从图 1(b)还可知，LN_E 与 LN_J 在空间上也属于邻接关系，但未体现在传统 kd-tree 结构中。从 LN_E 可通过 Y4 一步访问到 LN_F，但如图 4 所示，从 LN_E 访问 LN_J 却要

经过 8 步。LN_E 与 LN_J 通过分割面 $X1$ 邻接，虽然 $X1$ 在数据结构上没有直接表现出 LN_E 与 LN_J 的邻接关系，但如果将 $X1$ 看作 LN_E 的右边界，即从 LN_E 直接关联到 $X1$，再基于传统 kd-tree 结构，从 $X1$ 右侧节点 TN_{GHIJKL} 进行查找，则只需 4 步就可到达 LN_J。另外如图 1(b)所示，除 SN_E 外，与 AOI 相交的区域 LN_C，LN_H，LN_J 均通过一分割面与 SN_E 邻接，因此有必要研究基于分割面的邻域特性，以改善传统算法。

3 邻域特性

3.1 边界的定义

包围一个节点区域的 4 个分割面构成这个节点的上下左右边界。如图 1(b)所示，LN_E 的左边界是 $X3$，上边界是 $Y1$，右边界是 $X1$，下边界是 $Y4$。

3.2 邻域的定义

若两个节点的边界中含有共同的分割面，则这两个节点互为邻域。如图 1(b)所示，LN_E 的上边界是 $Y1$，LN_A，LN_C 的下边界也是 $Y1$，所以 LN_A 与 LN_E 互为邻域，LN_C 与 LN_E 互为邻域。

3.3 邻域特性

特性 1　根据游戏场景分割特点，除 SN 外，其它 RN 仅可能是与 SN 互为邻域的 LN。如图 1(右)，除 SN_E 外，其它 RN 仅可能是 LN_D，LN_A，LN_C，LN_H，LN_J，LN_K，LN_F，而不可能是 LN_B，LN_G，LN_I，LN_L。

特性 2　一个 LN 的上邻域一定是作为其上边界的 SDY 的上侧区域中最靠下的 LN。图 1(b)所示，LN_A，LN_C 作为 SN_E 的上邻域，可看作 SN_E 上边界 $Y1$ 上侧区域 TN_{ABC} 中最靠下的两个 LN。同理可推断出左、右、下 3 个邻域的判定特性。

特性 3　若 AOI 与 SN 上边界相交，则上边界中由 SDX 分得的在 AOI 横向范围所处一侧的最靠下的 SN 的邻域必为 RN。如图 1(b)所示，LN_C 处于 SN_E 上边界 $Y1$ 上侧区域 TN_{ABC} 中由 $X2$ 分得的 AOI 横向范围所处一侧的最靠下的 SN_E 的邻域，因此 LN_C 必为 RN。虽然 LN_A 同为最靠下的邻域，但它在 AOI 横向范围另一侧，因此不是 RN。同理可得出 AOI 与 SN 左、右、下边界相交时 RN 的判定特性。

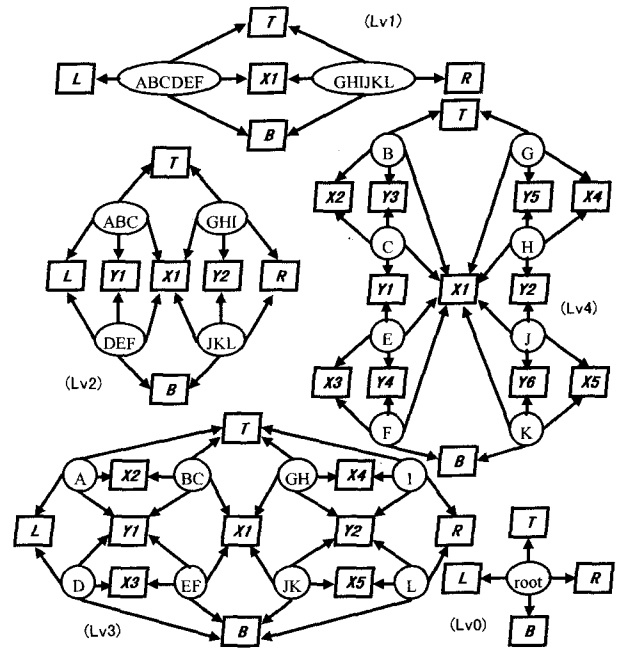
特性 3 中还隐含了一种 AOI 横向范围同时属于某 SDX 两侧时的情况。此情况虽可由特性 3 分为左右两侧解决，但其本身仍突显一个特征，即特性 4。为了与图 1(b)结合说明该特性，特性 4 中以右邻域为例描述。

特性 4　若 AOI 与 SN 右边界相交，且 AOI 纵向范围同时处于右边界右区域节点中某 SDY 两侧，则此 SDY 上区域的最左下 LN 以及该 SDY 下区域的最左上 LN 必为 RN。如图 1(b)所示，AOI 与 SN_E 右边界 $X1$ 右区域节点 TN_{GHIJKL} 中的 $Y2$ 相交，LN_H 是 $Y2$ 上区域最左下的 LN，LN_J 是 $Y2$ 下区域最左上的 LN，因此 LN_H，LN_J 必为 RN。同理可得 AOI 与 SN 左、上、下边界相交时此类情况下 SN 的判定特性。

4 扩展 kd-tree

4.1 扩展的 kd-tree 结构

传统 kd-tree 结构如图 2 所示，每个 TN 关联到自己的 SD，每个 SD 关联左右两个 TN。这种结构并没有考虑到子区域周边的邻接关系。利用本文上一节讨论的邻域特性，为整个场景增设左(L)、上(T)、右(R)、下(B)4 个 SD 作为 TN_{root} 边界，由此，每个区域都有 4 个 SD 作为其左上右下 4 个边界。为每个子区域增加到其 4 个边界 SD 的关联，实现对传统 kd-tree 结构的扩展，在树节点的层次关系基础上添加了节点间的平面邻接关系。扩展结构呈网状，为清晰表达，图 5 中按图 2 所示树的层结构分别给出 5 个层的扩展结构。如此，图 2 和图 5 共同构成了扩展的 kd-tree 结构。



L，T，R，B 表示整个场景的边界，箭头表示关联方向。

图 5　场景的 kd-tree 扩展结构

4.2 数据定义

为描述算法方便，定义树节点(TreeNode)、分割维度(SplitDim)两个结构体如表 1、表 2 所列。

表 1　树节点 TreeNode 数据结构定义

属性名	类型	属性说明
sdMe	SplitDim	分割该节点的 SD，叶节点忽略此属性。
sdL	SplitDim	作为该节点左边界的 SD。
sdT	SplitDim	作为该节点上边界的 SD。
sdR	SplitDim	作为该节点右边界的 SD。
sdB	SplitDim	作为该节点下边界的 SD。
parent	TreeNode	该节点的父节点，根节点忽略此属性。
events	列表	该节点包含的事件，仅对叶节点有效。

表 2　分割维度 SplitDim 数据结构定义

属性名	类型	属性说明
type	枚举	该 SD 的类型，取值为 SDX 或 SDY。
sc	浮点	该 SD 的 SC 数值。
tnL	TreeNode	该 SD 分割空间得到的左树节点(上或左区域)。
tnR	TreeNode	作 SD 分割空间得到的右树节点(下或右区域)。

5 新的在扩展 kd-tree 中的查找算法

5.1 新算法要点描述

首先，SN 肯定是一个 RN；然后，探索 SN 的左、上、右、下 4 个方向的邻域，以找到其它可能的 RN。每个方向上搜索邻域的模式一样，故仅以搜索右邻域来描述算法。算法利用了本文第 3 节给出的 4 个邻域特性。

根据特性1，算法将重点放在 SN 的邻域 LN 上，并且是那些与 AOI 相交的边界所关联的 LN，从而可快速排除大量无关节点；

根据特性2，仅消耗极小计算资源便可找到 SN 的邻域，因为在查找右邻域时，只要遇到 SDX 便可直接选其左节点（即靠左的区域）继续查找，无需相交或包含判定运算；

根据特性3，在搜索右邻域过程中遇到 SDY 时，若 SDY 在 AOI 上方，则选其右节点（即靠下的区域）继续搜索；若 SDY 在 AOI 下方，则选其左节点（即靠上的区域）继续搜索；最后找到的 LN 即为 RN；

根据特性4，在搜索右邻域过程中遇到 SDY 时，若 SDY 与 AOI 相交，接下来的搜索无需任何相交或包含运算，直接找到此 SDY 上区域中（即左节点）最左下方的 LN 和其下区域中（即右节点）最左上方的 LN 即可，这两个 LN 一定是 RN。

特性1体现在伪代码3中，特性2—特性4体现在伪代码4中。

伪代码3　以 SN 为中心向四面探索邻域

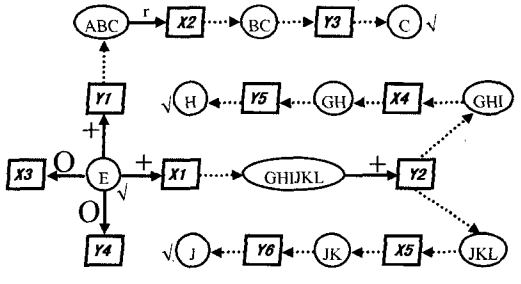
```
if(SN. sdL. sc > AOI. left) 从 SN 左边界 SDX 的左节点展开查找;
if(SN. sdT. sc > AOI. top) 从 SN 上边界 SDY 的左节点展开查找;
if(SN. sdR. sc < AOI. right) 从 SN 右边界 SDX 的右节点展开查找;
if(SN. sdB. sc < AOI. bottom) 从 SN 下边界 SDY 的右节点展开查找;
```

伪代码4　搜索右邻域

```
if(SD. type 是 SDX){ 进入 SD 的左节点进行探索;} //特性 2
else{  //特性 3,SD. type 是 SDY
    if(SD. sc <= AOI. top){ 进入 SD 右节点进行探索;}
    else if(SD. sc >= AOI. bottom){
      进入 SD 左节点进行探索;}
    else{ //特性 4
      取 SD 左节点中最左下方的 LN;
          取 SD 右节点中最左上方的 LN;}
  }
```

5.2　新算法举例

对于图1(b)所示的分割场景，结合图2、图5的树结构，用新的算法查找 RN，过程如图6所示。



虚线表示非计算性探索，实线表示计算性探索，箭头表示探索方向，+表示目的 SD 与 AOI 相交，O 表示目的 SD 与 AOI 不相交，√表示确定该节点为 RN，r 表示 AOI 位于 SD 右节点。

图6　新的 kd-tree 查找 RN 的过程

(1) LN_E 作为 SN 本身就是一个 RN；

(2) LN_E 左边界 $X3$ 不与 AOI 相交，不用搜索左邻域；

(3) LN_E 上边界 $Y1$ 与 AOI 相交，进而搜索 $Y1$ 上区域 TN_{ABC}；TN_{ABC} 采用 $X2$，AOI 在 $X2$ 右边，故探索 $X2$ 右区域 TN_{BC}；TN_{BC} 采用 $Y3$，则其靠下的节点 LN_C 即为一个 RN；

(4) LN_E 右边界 $X1$ 与 AOI 相交，进而探索 $X1$ 右区域 TN_{GHIJKL}；TN_{GHIJKL} 采用 $Y2$，AOI 与 $Y2$ 相交，可直接探索到 $Y2$ 上区域 TN_{GHI} 最左下的 LN 以及 $Y2$ 下区域最左上的 LN 作为 RN 即可，即 LN_H 和 LN_J；

(5) LN_E 下边界 $Y4$ 不与 AOI 相交，不用搜索下邻域。

6　算法分析比较

6.1　性能提升分析

新算法与传统算法相比较，有3个方面的提升：

提升1　分析伪代码1和伪代码2可知，传统算法在向上或下遍历时，对每个遍历到的 TN 最少进行1次比较运算，最多4次，平均2.5次。新算法除开始时如伪代码3所示，需要4次比较运算，以确定是否向某个方向进行搜索，此后对于每个 TN 最少进行1次比较运算，最多3次，平均2次，少于传统算法。

提升2　传统算法仅依照树的层次结构遍历查找 RN，有时需要遍历大量无关 TN，特别是 AOI 跨越层级深的 LN 时。新算法基于分割维度，充分利用各个区域间的邻接特性，以 SN 为中心向四面扩展进行 RN 的查找，比传统算法更为直接地找到 RN，过程中经历的无关 TN 要少于传统算法，特别是在 AOI 跨越层级比较深的 LN 时，会少很多。

提升3　在新算法要点描述的特性4的情形下，虽然访问 TN，但不需要进行任何相交或包含判定运算，如图6虚线所示，这相当于进一步减少了 TN 的遍历量。

6.2　时间消耗分析

设传统算法的消耗为 $Cost_1$，计算方法如式(1)所示。其中，N_1 是遍历过的 TN 数量，S_1 是遍历单个节点时执行的比较运算的次数。设新算法的消耗为 $Cost_2$，计算方法如式(2)所示。其中，N_2 是查找过的使用了比较运算的 TN 数量，S_2 是查找单个节点时执行的比较运算的次数，常量4是查找之初用于判断方向取舍的比较运算的使用次数。

$$Cost_1 = N_1 \times S_1 \tag{1}$$

$$Cost_2 = 4 + N_2 \times S_2 \tag{2}$$

根据新算法比传统算法在前述的3个效率提升可知，S_2 小于 S_1，N_2 小于且可能远小于 N_1，常量4在 N_2 比较大时可以忽略，因此 $Cost_2$ 小于且可能远小于 $Cost_1$。本文没有找到可定量分析 $Cost_1$ 和 $Cost_2$ 的方法，但通过实验结果发现新算法确实比传统算法有很大的效率提升。

6.3　空间消耗分析

扩展的 kd-tree 结构在每个节点上增加了到4个边界的关联，比传统结构多占用的存储空间为 $(N \times R \times 4)$，其中 N 是节点数量，R 是一个关联占用的存储空间（仿真实验中使用C语言的指针实现关联）。

7　仿真实验

7.1　实验环境的设定

为了对两种算法得出公平的实验数据，在实验环境上需保证以下几个方面：

(1)创建一个不再修改的坐标集文件，其中存储着随机产生的20万个二维坐标；

(2)从坐标集文件中第一个坐标向后顺序读取 N_e 个坐标作为放置 N_e 个事件的位置；

(3)从坐标集文件中最后一个坐标向前顺序读取 N_a 个坐标作为创建 N_a 个角色的位置；

(4)每个角色的移动速度都是 S，AOI 都是以角色为中心、边长为 R 的矩形；

(5)玩家总是会尽量与事件交互，为了能有效地模拟真实玩家，在文献[7]的基础上，设计角色的移动模式为：初始状态下，每个角色以自己被创建的顺序号 m 来标定第(m mod N_e)个被创建的事件的位置为目的地；到达目的地后再以第((m+1) mod N_e)个被创建的事件的位置作为新的目的地；如此不断地移向新的目的地(事件点)；

(6)两个测试程序仅仅在 *RN* 的查找模块用不同的代码实现，其它部分一样，保证空间分割一致，采用相同的 *SN* 定位方法，最终突出两种 *RN* 查找算法的性能差异；

(7)两个测试程序分别在同一台机器上运行主循环 L 次，每次主循环中调用一次 *RN* 查找模块。

7.2 测试结果包含的数据项及细节设定

单步比较指令使用量记录(CMP)：记录测试程序每次循环中 *RN* 查找模块使用的比较指令数量。为此，*RN* 查找模块中每一个比较指令都设置了计数器。

单步走访节点量记录(VISIT)：记录测试程序每次循环中 *RN* 查找模块走访的节点数量。为此，*RN* 查找模块中每个节点访问部分都设置了计数器。

单步 *RN* 获取量记录(RN)：记录测试程序每次循环中 *RN* 查找模块查找到的 *RN* 数量。两种算法在此项目上数值应该一致，说明新算法不会漏查或多查 *RN*，证明了新算法的正确性。

单步事件点测试量(VERT)：记录测试程序每次循环中计算角色与事件间距的次数。两种算法在此项目上数值应该一致，说明新算法与传统算法会产生相同的游戏逻辑，证明了新算法的正确性。

单步 CPU 周期耗时记录(CPU)：记录测试程序每次循环中 *RN* 查找模块消耗的 CPU 指令周期数。对于 CPU 指令周期数的获取采用文献[8]中所述的方法。

RN 查找总耗时：测试程序从开始到结束，*RN* 查找模块消耗的时间总量。

8 实验结果

实验设定的世界为当前主流网游单台服务器处理的世界尺寸[9]，为 2000m 见方(即 400 万 m^2)的正方形，事件数量 N_e 为 50000，角色 AOI 边长 R 为 10m，移动速度 S 为 2m/步，总测试步数 L 为 10000。

先用两种算法对单个用户进行测试，结果如表 3 所列。*RN* 项和 *VERT* 项的数值在两种算法中是完全一致的，这说明新算法在正确性上与传统算法一致，保证了获取的其他项目数据都是有效的。两种算法在其它项上中值相同或相差很小，是因为对于单个角色而言，大部分时间处于非跨区状态，这些值的明显对比如表 4 所列。此外，均方差显示新算法比传统算法要稳定很多。最后，总耗时以最直观的方式表明，新算法比传统算法的性能小幅提高了 9.08%。

表 3 单角色测试结果

算法	中值		均方差		总耗时(ms)	
	传统	新	传统	新	传统	新
VISIT	2	2	12.6	3.7		
RN	1	1	0.74	0.74		
VERT	25	25	18.2	18.2	7390	6719
CMP	8	4	36.6	1.6		
CPU	464	400	797.8	286.6		

然后，保证同样的世界设定，在 5000 角色的情况下对两种算法进行测试，结果如表 4 所列。大量角色存在下，会频繁出现跨区的情况，因此各个项目在中值上有了非常大的差别，表明新算法比传统算法在实际情况下有更大的优越性。虽然表 4 的均方差比表 3 中的大，但新算法仍表现得比传统算法稳定得多。总耗时方面，新算法比传统算法提高了 38.13%，效果显著。

表 4 5000 角色测试结果

算法	中值		均方差		总耗时(ms)	
	传统	新	传统	新	传统	新
VISIT	46335	19587	905.3	262.9	20281	12547
CMP	147899	23538	264.8	121.2		
CPU	1633100	738680	144470	71803		

最后，仍旧保证同样的世界设定，对传统算法进行了角色数为 2800 和 2500 的测试，并与新算法在 5000 角色时取得的耗时进行对比。结果如表 5 所列，解释为一台游戏服务器采用传统算法可以正常负载用户数 2500 到 2800，而采用新算法，可以正常负载的用户数提升为 5000。

表 5 负载能力对比

	新算法	传统算法	传统算法
角色数	5000	2800	2500
总耗时(ms)	12547	12641	12172

结束语 本文提出并利用邻域特性扩展 *kd*-tree 结构，在此基础上设计了以驻扎节点为中心向四周扩展的查找算法，将树中各节点间的层次关系转换为平面邻接关系。相比传统算法采用的对树结构进行层次遍历的方式，新算法能更直接地找到相关节点，更少地经历无关节点。同时，在经历单个节点时的实现细节方面，新算法比传统算法使用更少的比较运算。此外，传统算法在处理角色兴趣域跨越深层子树节点时性能降低最严重，而新算法充分考虑到游戏中角色的兴趣域不超过最小分割区域这一限定，在算法设计中进一步减少了这种情形下比较运算的使用量，反而可能达到最高性能(如图 6 虚线所示)。通过对新算法和传统算法在原理和关键实现细节上的分析以及仿真实验，证明新算法比传统算法拥有更好的表现，在本文设定的实验环境下，性能提升 38.13%，且均方差显示新算法更加稳定。

参 考 文 献

[1] Bentley J L. Multidimensional Binary Search Trees Used for Associative Searching[J]. Communications of the ACM, 1975, 18(9): 509-517

[2] http://en.wikipedia.org/wiki/Kd-tree

[3] Kubica J M, Masiero J, Moore A, et al. Variable KD-Tree Algorithms for Spatial Pattern Search and Discovery[C]// Neural Information Processing Systems. Dec. 2005

[4] Smed J, Kaukoranta T, Hakonen H. Aspects of Networking in

Multiplayer Computer Games[J]. The Electronic Library, 2002, 20(2): 87-97

[5] 张渊. 空间二叉树排序查找算法及其在网络游戏中的应用[J]. 计算机应用, 2007, 27: 356-359

[6] Assiotis M, Tzanov V. A Distributed Architecture for MMORPG[C]// Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games. Singapore, October 2006

[7] Bettstetter C, Hartenstein H, Xavier Pérez-Costa. Stochastic Properties of The Random Waypoint Mobility Model: Epoch Length, Direction Distribution, and Cell Change Rate[C]// Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems. USA, September 2002

[8] Yuan Feng. Windows Graphics Programming: Win32 GDI and DirectDraw[M]. Hewlett-Packard Professional Books, 2001

[9] Hsiao T Y, Yuan S M. Practical Middleware for Massively Multiplayer Online Games[J]. IEEE Internet Computing, 2005, 9(5): 47-54

(上接第 253 页)

数的个数，N_c 是类数。

3.1 在数据集 CMU PIE 上的实验

CMU PIE 人脸数据库共包含 68 人在不同的姿态和光照条件下的共 41368 幅图片，包含各种光照和表情的变化。实验中选取了所有 68 人在 5 种近正面姿态(C05, C07, C09, C27, C29)下的一种近正面姿态(C27)，每个人 49 幅图片。包含各种光照和表情的变化，实验从每人的图像中随机抽取 30 幅作为训练样本，其余的作为测试样本。为得到随机的训练集、测试集划分，实验重复了 10 轮。每轮实验中，又从 30 幅训练图像中随机选取 $L(L=2,3)$ 幅作为有标签样本，其余的作为无标签样本。对每个 L 值实验又重复了 10 次。最终的识别率取其平均。当 α 分别取 0.01, 0.02, 0.04, 0.06, 0.08 时，最优识别率的变化曲线如图 2 所示。

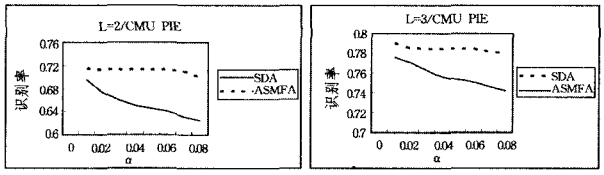


图 2 最优识别率随 α 变化曲线

SDA 算法随着 α 的不同取值识别率变化非常明显，而 ARSMFA 算法对 α 具有鲁棒性。当 α 取 0.01 时，最优识别率随特征维数变化曲线如图 3(a)、(b)所示。

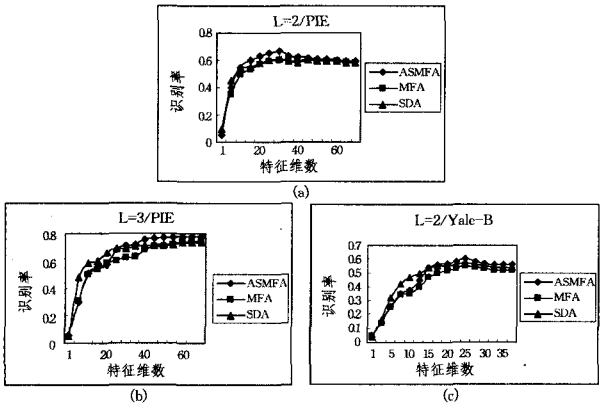


图 3 最优识别率随特征维数变化曲线

3.2 在数据集 Yale-B 上的实验

Yale-B 人脸数据库共包含 28 个人在 9 种姿态、64 种光照下的 16128 幅人脸图像，每人 576 幅图片。从数据库中选择 10 人，每人选择 100 幅图像。实验从每人 100 幅图像中随机抽取 30 幅作为训练样本，其余的作为测试样本，为得到随机的训练集、测试集划分，实验重复了 10 轮。每轮实验中，又从 30 幅训练图像中随机选取 $L(L=2,3)$ 幅作为有标签样本，其余的作为无标签样本，对每个 L 值实验又重复了 10 次。最优识别率随特征维数变化曲线如图 3(c)所示。

由实验结果可知，ASMFA 算法的识别性能超过 MFA 算法，说明了无标签数据能够被用来提高人脸识别算法的性能。ASMFA 算法的识别性能也超过 SDA，说明了基于原空间和期望的低维空间融合所建立的正则化项能够提高识别性能。SDA 随着 α 的不同取值发生了显著的变化，新算法对 α 的取值有很好的鲁棒性。在 Yale-B 和 CMU PIE 人脸数据库的实验表明，ASMFA 能够充分利用样本的类别信息和样本原空间与期望的低维空间的局部几何结构，可以得到比 MFA 和 LDA 算法更优的性能。

结束语 人脸图像是高维数据，获得足够的带标签的训练样本是不可能的。当每类训练样本很少时，进行人脸识别不能得到理想的识别效果。随着数字图像技术的发展，收集大量的无标签的人脸图像却是可能的，因此，人脸识别可以看作是一个半监督学习问题。研究表明人脸数据处在一个具有较低维数的流形上。传统算法在原空间中定义正则化项，而本文在半监督思想基础上探索出一种新的正则化项的创建方法，这种将半监督的思想与流形学习结合用于人脸识别中取得了很好的效果。

参 考 文 献

[1] 谢永华，严云洋，杨静宇. 分块 PCA 鉴别特征抽取能力的分析研究[J]. 计算机科学, 2006, 33(3): 155-159

[2] Sets D L, Weng J. Using Discriminant Eigenfeatures for image Retrieval[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 1996, 18(8): 831-836

[3] He Xiaofei, Yan Shuicheng, Hu Yuxiao, et al. Face Recognition Using Laplacianfaces[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2005, 27(3): 328-340

[4] Yan Shuicheng, Xu Dong, Zhang Benyu, et al. Graph Embedding and Extensions: A General Framework for Dimensionality Reduction[J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2007, 29(1): 40-51

[5] Chen H T, Chang H W, Liu T L. Local discriminant embedding and its variants[J]. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, 2(1): 846-853

[6] Cai Deng, He XiaoF, Han J W. Semi-supervised discriminant analysis[C]// Proc. of the IEEE Int'l Conf. on Computer Vision. Rio de Janeiro, 2007

[7] Xu Dong, Yan Shui-cheng. Semi-supervised Bilinear Subspace Bilinear Subspace Learning[J]. IEEE Trans. on Image Processing, 2009, 19(7): 1671-1676