计算几何实验报告

Shared Camera Control 近似算法的实现

肖晶	2004310456	xiaojing00@mails.tsinghua.edu.cn	51537911
丁力	2004310451	tomding00@mails.tsinghua.edu.cn	51537944
孙欢	2004210966	sunhuan00@mails.tsinghua.edu.cn	62775281

2005 年 1 月 3 日

目录

图表目录

Shared Camera Control 近似算法的实现

——计算几何实验报告

小组成员

肖晶	2004310456	xiaojing00@mails.tsinghua.edu.cn	51537911
丁力	2004310451	tomding00@mails.tsinghua.edu.cn	51537944
孙欢	2004210966	sunhuan00@mails.tsinghua.edu.cn	62775281

2005-1-3

问题背景

随着网上摄像机的商业化，可以用它们在网络上转播公众们关注的影像，例如体育比赛，音乐演唱会，救援行动，空间站等等。大量的用户通过 Internet 连接到这些网站，用户可以在线控制摄像机的镜头位置，倾斜角度，拉近和拉远镜头。要解决的问题是如何找出满足众多用户需求的最佳摄像机参数。

目前已有的网上摄像机的解决方案是，采用时间片轮转的策略，每个用户在他的时间片中可以完全控制摄像机。这种做法有很多缺点，在任一时间里，事实上只有一个用户处于活动状态；有可能被某些用户进行恶意的操作；如果用户很多，那么等待属于自己的时间片的时间会很长，很少有用户会有足够的耐心。

比较理想的解决方案是：系统能够同时接受所有用户输入他们感兴趣的参数，通过计算选取最佳的参数，尽可能好的满足他们的要求。

我们的工作

我们的工作所要解决的是实现名为 Shared Camera Control 的算法，即对用户输入的期望摄像机镜头在全画面中的位置及大小，计算出最佳的摄像机镜头在全画面中的位置及大小。面临的输入是一些与坐标轴平行的矩形表示期望的镜头边框（注意在实际应用中，这些矩形的长宽比应是固定的，即摄像机镜头的长宽比，但在实验中没有必要规定这个比例）。输出是固定长宽比的一个最佳镜头矩形。程序中根据实际需要将摄像机镜头的长与宽离散化到一定的区间上，对每一组离散值计算最优镜头矩形位置，这我们称之为 Fixed Camera Algorithm。最后取所有离散值上的总的最优一组解输出。

假设给出的输入集 $R=\{R_1,R_2,...,R_n\}$ 为一组与坐标轴平行的矩形，要计算最佳矩形 C_{opt}，采用全局满意函数

$$S(C)=S_R(C)=\sum_{i=1}^{n}S_i(C)$$

其中 $S_i(C)$ 是对单个矩形的满意函数，比较 R_i 与 C 的贴近程度。贴近程度有多种定义方式，我们使用称之为**最大相交覆盖**的函数：

$$S_i(C) = IOM(C, R_i) = \frac{Area(C \cap R_i)}{\max(Area(C), Area(R_i))}$$

使用这个评价函数的优点在于，不仅考虑了摄像机矩形与用户矩形相交的面积大小，而且考虑了摄像机和用户矩形的面积大小关系。假设单纯考虑覆盖的用户矩形面积，那么直接使用覆盖全画面的摄像机矩形就可以覆盖所有的用户矩形，但是这样并不能使用户满意，因为用户选择一块较小的区域意味着他们想看到更多的细节。

算法与数据结构

整体算法概述

我们先对整个算法做一个整体的概述。首先我们根据问题的实际情况作出离散化镜头的处理，把原来的 Shared Camera Control 问题简化为 Fixed Camera Algorithm 的问题。接着对于 Fixed Camera Algorithm 问题我们对其作垂直分解，通过一些带权矩形的集合来近似原来的满意函数。然后我们对这些矩形进行排序等预处理来生成一颗经过修改的线段树，并以此来实现扫描线算法求出全局的最优摄像头位置。流程图示如下

- 镜头离散化
- 对每一个用户矩形的满意函数作垂直分解（实现近似处理）
- 对所有的矩形进行多路排序
- 用扫描线算法（使用线段树的数据结构）求出全局满意函数的解

算法与数据结构的实现

1. 问题简化和分析

由于摄像头的镜头调整不是连续的，所以把它离散化。一个特定的镜头大小对应着一个特定摄像头矩形的大小，这是因为摄像头矩形的长宽比已经固定了。因此，对于一个固定的镜头大小，ShareCam 问题就简化为寻找一个固定大小的摄像头矩形的问题了。我们把镜头大小离散化成 m 个不同的值，这样就对应了 m 个不同大小的摄像头矩形，但是它们的长宽比都是相同的。接着，对于每一个固定大小的摄像头矩形都分别独立地求出它们的最佳位置从而使得相应的 $S_R(C_i)$ 为最大，这样便得到了 m 个候选的矩形。然后再从中挑选出一个使得 $S_R(C_i)$ 最大的那个最优的矩形。

由此，现在我们可以假定摄像头矩形的大小是固定的。这样摄像头矩形就只由其中心点 p 在平面上的坐标来决定了，表示为 C(p)。对于某一个给定的用户矩形 R_i，目标函数 $S_i(\bullet)$ 可以看作是一个定义在二维平面上的二元函数：

$$S_i(p) = \delta_i Area(C(p) \cap R_i)$$

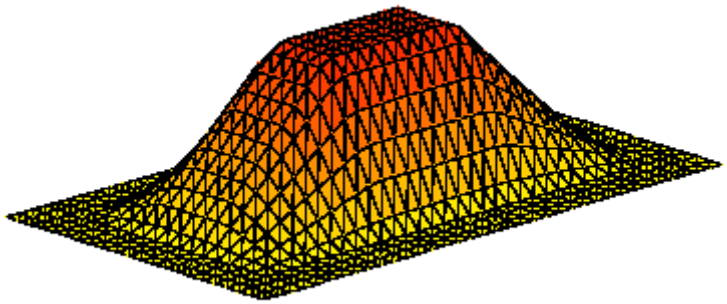


图 1

其中 $\delta_i = 1/(\max Area(C(p)) \cap Area(R_i))$ 为常数，S_i 的图像如图 1 所示。使得 $C(p) \cap R_i \neq \varnothing$ 的矩形中心点 p 的坐标轨迹为一个矩形，表示为 $D(R_i)$。根据 $S_i(p)$ 的不同我们可以把 $D(R_i)$ 分解成 9 个不同的部分：

- 中央部分
  即 $C(p) \subseteq R_i$ 或者 $R_i \subseteq C(p)$ 的区域
- 四个侧边部分
  即 C(p)有两个角在 R_i 中的区域或者 R_i 有两个角在 C(p)中的区域
- 四个拐角部分
  即 C(p)有一个角在 R_i 中的区域或者 R_i 有一个角在 C(p)中的区域

2. 近似算法与垂直分解的实现

1) 概述

首先，对于每一个用户矩形 R_i，我们构造一些带有权值的矩形来 ε 近似函数 $S_i(p)$。ε 近似的定义如下：

a) 函数 $g(\square)$ $\varepsilon - approximate$ 于函数 $f(\square)$，如果

- 对于所有的 $x \in \square^2$，$g(x) < f(x)$
- 对于所有的 $x \in \square^2$，令 $c = \max_{p \in \square^2} f(p)$，当 $f(x) \geq (\varepsilon / 5)c$ 时有 $(1-\varepsilon) f(x) \leq g(x)$.

b) 对于平面上一组带权矩形集合 R，令 $W_R(\square)$ 为相应的权值函数，该函数返回平面上每个点的累积权值（包围该点的所有矩形）。那么若 $W_R(\square)$ $\varepsilon - approximate$ 于函数 $f(\square)$，则带权矩形集合 R $\varepsilon - approximate$ 于函数 $f(\square)$。

$S_i(p)$ 的 ε 近似可以通过将 $D(R_i)$ 划分为一些矩形的集合 $\Re_i$，并且设置这些矩形的权值为 $S_i(p)$ 在这些矩形区域上的最小值。形象化的描述如下：我们将 $S_i(p)$ 的图像用水平的平面在不同的高度切割成很多层，并且将这些层在 xy 平面上投影，对于任何两个连续的层之间的部分我们用一些与轴向平行的矩形来近似，每个矩形的权值就是这层的高度。

然后我们将所有的 $\Re_i$ 合并为一个集合 $\Re$，然后我们在整个平面上找出被矩形覆盖的权值最高的点，这个点就是我们需要求出的摄像头矩形的中心位置的近似解。

2) ε **近似** $S_i(p)$

论文中原来使用 vertical decomposition 的方法将 $S_i(p)$ 划分成很多矩形，这种划分方法需要 $O(n\log n)$ 的时间复杂度，而且实现起来也需要比较复杂的数据结构。我们采用了新的算法，时间复杂度降为 $O(n)$，并且很容易用程序实现，不需要复杂的数据结构。

- $D(R_i)$ 的区域划分

设用户矩形 C(p) 的左上角坐标为（xleft,ytop），右下角坐标为（xright,ybottom），摄像头矩形的长宽分别为 c 和 d。令 a=xright-xleft，b=ytop-ybottom，则 $D(R_i)$ 中的区域可以划分如下（如图 2）：

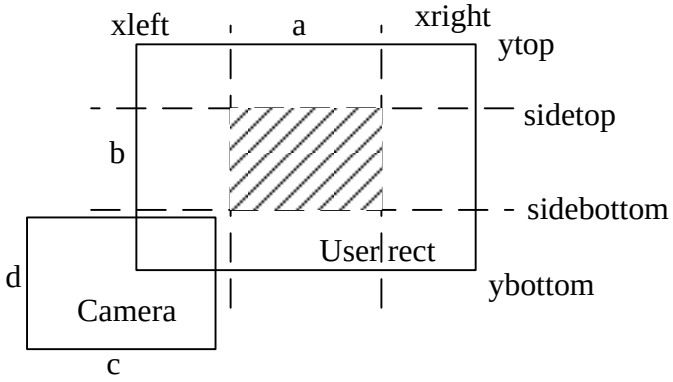


图 2

左下角的拐角区域的范围为：

$$y\text{范围}=\begin{cases}[ybottom-\frac{d}{2},ybottom+b-\frac{d}{2}](b\le d)\\[ybottom-\frac{d}{2},ybottom+\frac{d}{2}](b>d)\end{cases}$$

$$x\text{范围}=\begin{cases}[xleft-\frac{c}{2},xleft+a-\frac{c}{2}](a\le c)\\[xleft-\frac{c}{2},xleft+\frac{c}{2}](a>c)\end{cases}$$

其他的三个区域由对称性可以很容易得到。

左侧边的区域的范围如下：

$$y\text{范围}=\begin{cases}[ybottom+b-\frac{d}{2},ytop-b+\frac{d}{2}](b\le d)\\[ybottom+\frac{d}{2},ytop-\frac{d}{2}](b>d)\end{cases}$$

$$x\text{范围}=\begin{cases}[xleft-\frac{c}{2},xright](a\le \frac{c}{2})\\[xleft-\frac{c}{2},xleft+\frac{c}{2}](a>\frac{c}{2})\end{cases}$$

由对称性右侧边同理可得。

下侧边的区域的范围如下：

$$y\text{范围}=\begin{cases}[ybottom-\dfrac{d}{2},ytop](b\le\dfrac{d}{2})\\[ybottom-\dfrac{d}{2},ybottom+\dfrac{d}{2}](b>\dfrac{d}{2})\end{cases}$$

$$x\text{范围}=\begin{cases}[xleft+a-\dfrac{c}{2},xright-a+\dfrac{c}{2}](a\le c)\\[xleft+\dfrac{c}{2},xright-\dfrac{c}{2}](a>c)\end{cases}$$

由对称性上侧边同理可得。

- 中央区域的矩形划分

中央区域上的$S_i(p)$值都相同，所以只用一个矩形就可以精确表示。这个矩形的左上角坐标为(sideleft,sidetop)，右下角坐标为(sideright,sidebottom)。其中

$$sideleft=\begin{cases}xleft+\dfrac{c}{2}(a\ge c)\\xright-\dfrac{c}{2}(a<c)\end{cases}$$

$$sideright=\begin{cases}xright-\dfrac{c}{2}(a\ge c)\\xleft+\dfrac{c}{2}(a<c)\end{cases}$$

$$sidetop=\begin{cases}ytop-\dfrac{d}{2}(b\ge d)\\ybottom+\dfrac{d}{2}(b<d)\end{cases}$$

$$sidebottom=\begin{cases}ybottom+\dfrac{d}{2}(b\ge d)\\ytop-\dfrac{d}{2}(b<d)\end{cases}$$

这个矩形的权值为$S_i(\max)=\delta_i\times\min\{a,c\}\times\min\{b,d\}$，即为两个矩形处在最大覆盖的情况下，也就是$S_i(C)$的最大值。

- 拐角区域的矩形划分

设摄像头矩形中心点 p 的坐标为(x,y)，则目标函数为

$$S_i=\delta_i(x+\frac{c}{2}-xleft)(y+\frac{d}{2}-ybottom)$$

我们在拐角区域建立一个新的坐标系$x'y'$，所以 p 点在新坐标系中的坐标为(x',y')，其中

$$\begin{cases} x' = x + \dfrac{c}{2} - xleft \\ y' = y + \dfrac{d}{2} - ybottom \end{cases}$$

所以 $S_i = \delta_i x' y' \le S_i(\max)$。

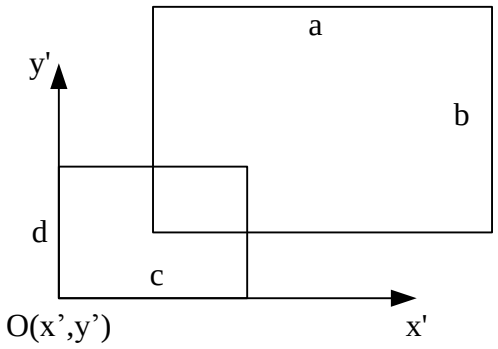


图 3

对于 j=0,1,...,M，令 $\gamma_j = \{p \in D(R) \mid S_i$，其中 $\beta_j = \min\{S_i(p) \mid p \in \gamma_j\}$，$M = \left\lfloor \log_{1+\varepsilon}\left(\frac{50}{\varepsilon}\right) \right\rfloor$。所以，$\beta_0 < \beta_1 < \cdots < \beta_{M-1} < \beta_M$，$\beta_0 = \dfrac{\varepsilon}{50} S_i(\max)$，$\beta_{M-1} < S_i(\max)$，$\beta_M = S_i(\max)$。曲线 γ_j 就是 $S_i(\bullet)$ 的图像在高度为 β_j 处的等高线在水平面上的投影，M 就是这些等高线的条数。

曲线 γ_j 实际上就是一条双曲线，我们令 P_j 为曲线 γ_j 和 γ_{j+1} 之间的最小的环节多边形。对于两个相邻的环节多边形，可以对它们之间的区域进行垂直分解，这样就得到了一些小矩形，它们的权值为他们所经过的最低的那条等高线的值。（原文中使用点定位的方法进行矩形分解，我们通过几何计算，找出这些矩形的关系，从而引入新的算法进行分解）

下面我们分析环节多边形 P_j 的几何特点。对于某一个给定的 j，假定 $\beta_{j+1} < S_i(\max$。定义环节多边形 $P_j = p_1 p_1' p_2 p_2' \cdots p_{m_j-1} p_{m_j-1}' p_{m_j}$，对于每一个 p_k 和 p_k' 按照如下来计算：

定义 $l_x = \min$，$l_y = \min\{b, d\}$。令 $x_{j,1} = (1+\varepsilon)^{j+1} \dfrac{\varepsilon}{50} l_x$，$p_1 = ((1+\varepsilon)^{j+1} \dfrac{\varepsilon}{50} l_x, l_y) \in \gamma_{j+1}$ 为 P_j 的第一个顶点。P_j 上奇数顶点都在双曲线 γ_{j+1} 上，偶数顶点都在双曲线 γ_j 上。每一个奇数顶点 p_k 是通过从偶数顶点 p_{k-1}' 做水平射线向右与曲线 γ_{j+1} 相交得到的，而每一个偶数顶点 p_k' 是通过从奇数顶点 p_k 向下做垂直射线与

曲线 γ_j 相交得到的。由此， p_k 点的坐标为 $\left(x_{j,k},\frac{\beta_{j+1}}{x_{j,k}}\right)$， p_k' 点的坐标为 $\left(x_{j,k},\frac{\beta_j}{x_{j,k}}\right)$。

经过计算得到：

$$p_k=\left(x_{j,k},\frac{\beta_{j+1}}{x_{j,k}}\right)=\left((1+\varepsilon)^{k-1}\beta_{j+1},\frac{1}{(1+\varepsilon)^{k-1}}\right)$$

$$=\left((1+\varepsilon)^{k+j}\frac{\varepsilon}{50}l_x,\frac{1}{(1+\varepsilon)^{k-1}}\right)$$

$$p_k'=\left(x_{j,k},\frac{\beta_j}{x_{j,k}}\right)=\left((1+\varepsilon)^{k+j}\frac{\varepsilon}{50}l_x,\frac{1}{(1+\varepsilon)^{k}}\right)$$

注意到以下两个重要的结论：

- ✓ $x_{j,k+1}=x_{j+1,k}\quad(0\le j\le M-1,1\le k\le m_j)$
- ✓ $y_{j,k}'=\frac{\beta_j}{x_{j,k}}=\frac{\beta_j}{x_{j-1,k+1}}=y_{j-1,k+1}=y_{j-1,k}'$

由此得出一个*重要的结论*， P_j 的奇数顶点与 P_{j+1} 的偶数顶点相重合， P_j 的偶数顶点与 P_{j-1} 的奇数顶点相重合。（如图 4 所示）

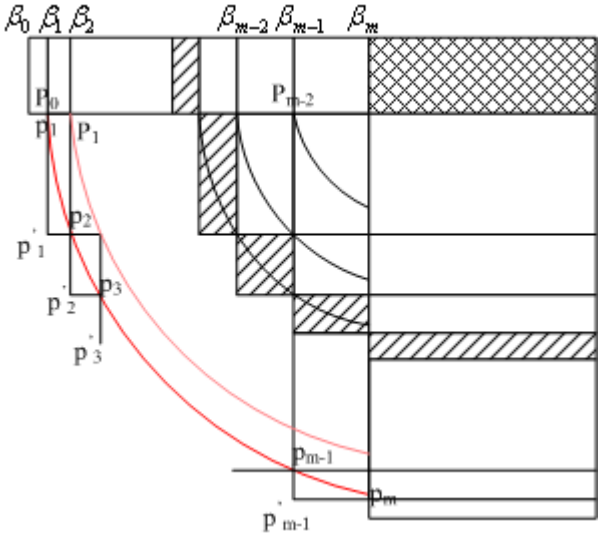


图 4

斜线阴影部分的矩形具有相同的权值

所以，只需要计算出每一个 $x_{j,1}$ 的坐标就可以得到所有的带权矩形了。这样我们就可以用迭代的方法在现行时间内求出拐角区域所有带权矩形。

- ➢ 左侧边区域的矩形划分

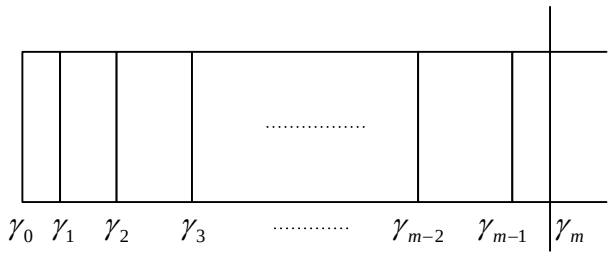


图 5

左侧边区域的划分只需要对 x 方向进行。如图 5 所示，

$\gamma_j = \min\left\{ (1+\varepsilon)^j \frac{\varepsilon}{50} S_i \quad (\max), \right\}$，$M = \left\lceil \log_{1+\varepsilon}\left(\frac{50}{\varepsilon}\right) \right\rceil$，$x_j' = \frac{\gamma_j}{S_i(\max)} l_x$

所以，对拐角区域的垂直划分线与对左侧边的垂直划分线重合。

- 下侧边区域的矩形划分

下侧边区域的划分只需要对 y 方向进行就可以了，如图所示，

$\gamma_j = \min\left\{ (1+\varepsilon)^j \frac{\varepsilon}{50} S_i(\max), S_i(\max) \right\}$ ， $M = \left\lceil \log_{1+\varepsilon}\left(\frac{50}{\varepsilon}\right) \right\rceil$ ，

$y_j' = \frac{\gamma_j}{S_i(\max)} l_y$，由此可以看出它与拐角的区域的水平划分并不重合，一般来说互相交错，两两相间。但是容易证明，将此划分用与拐角的水平划分相重合的水平划分来代替时最多是 $(1+2\varepsilon)$ 的近似，所以为了简便，就用与拐角区域的水平划分相重合的水平划分来进行矩形分割。这样做不但简化了矩形分割的过程也简少了后面 segment tree 过程的存储空间（减少了一半的存储空间）。

3) ε 近似 $S_R(\bullet)$

将每一个用户矩形经过垂直分解所得的各个 $\Re_i$ 求并得到 $\Re$，由原论文中的证明可以知道这是全局函数 $S_R(\bullet)$ 的一个近似。

4) 数据结构

由于我们已经能够精确的计算分解后每一个带权矩形的位置和权值，并且我们也能事先清楚的知道分解后各个矩形的几何位置关系。所以我们采用了如下的二维线性表的结构来存储这些矩形，并籍此表明它们的几何位置关系。

首先用一维数组存放有多少列矩形队列，数组的每一个元素为矩形队列的指针。每一列对应于处于同一条扫描线上所有矩形。用链表存储处于每一列的矩形。这样既存储了所有的分解后的矩形，又通过数组的位置关系反映了这些矩形的几何位置关系，处于同一个数组元素中的所有矩形表示它们具有相同的水平坐标。如下图所示。

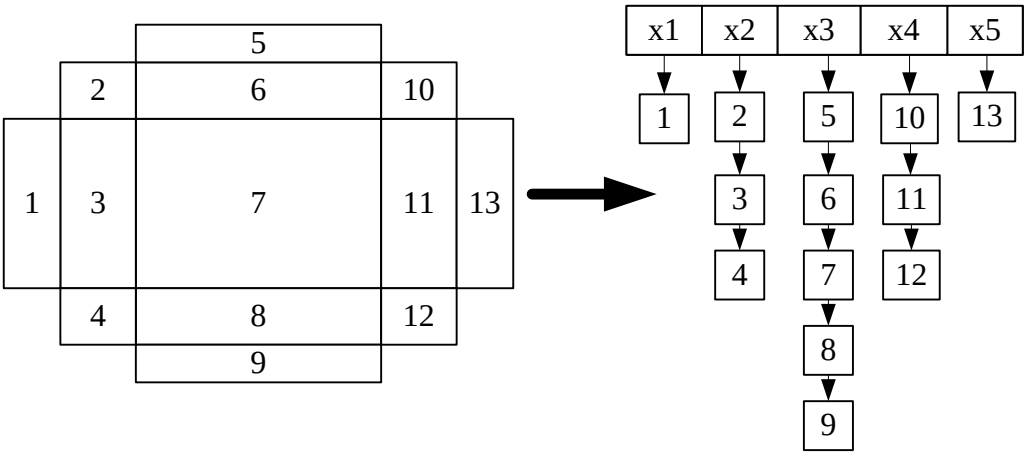


图 6

3. 扫描线算法及其实现

寻找最优 Camera 大小和位置的问题可以分解为两步骤进行。首先，在 Camera 大小固定的情况下寻找最优的位置；其次，随着 Camera 大小的不断变化而得到一系列的局部最优值，从中再选取全局极优值。

固定 Camera 大小的最优位置的选取。由于 Camera 大小已经确定，所以这时问题集中在 Camera 中心位置的确定上。在经过对 User Rectangle 的剖分之后，整个平面被离散成一个一个矩形区域，每个区域都有自己的权值（反映 Camera 中心在此处的优劣程度）。因此，整个平面具有最优性的性质，即每一阶段的最优值仅与当前状态有关，与之前状态和之后状态都无关。基于这样的性质，这里使用了扫描线的算法。

- 扫描的顺序是从屏幕的左侧到右侧（沿 x 轴 $-\infty \rightarrow +\infty$ 方向）。
- 扫描线是垂直线（Vertical Line），平行于 y 轴的直线。
- 扫描线状态，在某一时刻扫描线的状态表示了其经过的矩形的信息。

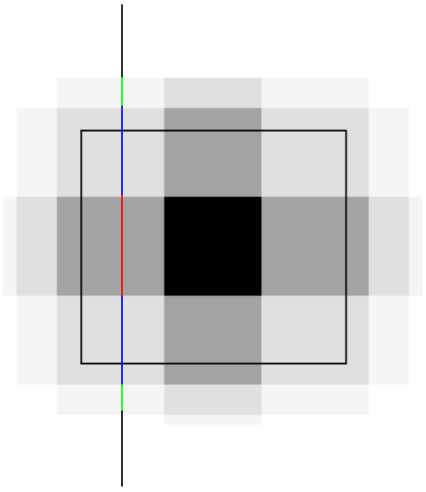


图 7

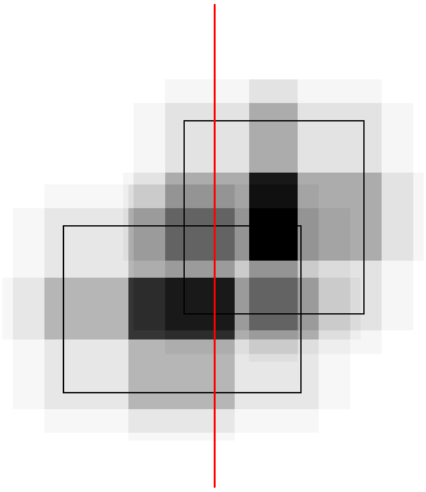


图 8

图 7 中是扫描线在 $x = x_0$ 处的情况，其中黑色线框表示原先的 User Rectangle，灰色色块表示剖分后的矩形，一条竖直的线表示扫描线，此时扫描线可由 7 段权值来表示。

扫描线从另一个角度说是若干条有权值的线段的重叠。如图 8 所示，当有多个 User Rectangle 时，扫描线上的权值将随着小矩形的重叠而叠加，产生更多的权值段。

- 扫描线的事件序列(x_Event)，将平面中每一个小矩形的左边和右边都当作扫描线的一个事件点。

在任何两个事件点中间，扫描线的状态是始终保持不变的。在事件点时刻，对扫描线的状态只有两种操作：遇到矩形的左边，从这个时刻起扫描线将经过该矩形，此时在扫描线状态中插入该矩形信息；遇到矩形的右边，从这个时刻起扫描线将不再经过该矩形，在扫描线状态中删除对应矩形信息。

在每一时刻的扫描线上总有一个最优区间，该区间拥有最大的权值，如图 a 中的红色区间。每经过一次事件点，扫描线上的权值区间就发生变化，从而产生新的最优值区间。在经历完所有的事件之后，就能够在所有时刻的最优值中找到整体的最优值。

算法流程：

```
Interval SweepForFixedCamera(rect rects //Vertical Decomposition 后的矩形)
{
  Interval ReturnValue;
  Status SweepLine;
  Initialize(SweepLine);     //初始化 Sweep Line 状态
  Foreach x_Event do
  {
     Foreach rects do
       If (x_Event.x == rects.left) //  矩形左边事件
            InsertRect(SweepLine, rects);
       Else
          If (x_Event.x == rects.right) //  矩形右边事件
             DeleteRect(SweepLine, rects);
     Interval temp = FindMaxInterval(SweepLine); //返回该时刻扫描线最优区间
     If (temp . value > ReturnValue . value)
```

```
                ReturnValue = temp;
        }
    }
```

4. 线段树数据结构的使用和实现

为了能够在特定时刻在扫描线上寻找到最优线段区间，使用基于 Segment Tree 结构的处理方法。

在进行扫描线算法之前，全部的矩形位置和权值都是已知的，此时根据 y 方向建立一颗一维的线段树 T。在所有矩形的上下两边投影在 y 方向上是一系列的点，$I=\{y_{\min},y_1,y_2,y_3,\cdots,y_n,y_{\max}\}$，$y_1 \sim y_n$ 是投影后得到的点，而 $y_{\min},y_{\max}$ 分别代表场景区域的最小值和最大值。

- 线段树的主体结构为一颗平衡二分查找树 T。I 中的所有区间导出了一组基本区间，与一般意义上的线段树不同，这里的 Int(v) 是一个半闭半开的区间。$\mathrm{Int}(v_0)=[y_{\min},y_1)$； $\mathrm{Int}(v_1)=[y_1,y_2)$ ..... $\mathrm{Int}(v_n)=[y_n,y_{\max})$。
  而 T 中的叶子就按照某种次序，分别对应于这些基本区间——最左侧的叶子对应于最左端的基本区间，其余依次类推，与叶子 u 相对应的基本区间记作 Int(u)
- T 中的每一个内部节点 v，都对应于由（至少两段）基本区间合并而成的某段区间，就是在以 v 为根节点的子树中，所有叶子 u 对应的基本区间 Int(u)的并集。
- 在 T 中的任一叶子 v 处，都存放了一段区间 Int(v)。

下图中是与三段线段对应的一棵线段树。在线段树中，每一条线段会被分成若干段存储在树中。若将线段拆分成基本区间的话，最坏情况下将会占用 O(n)的空间，导致 m 条线段占用 O（mn）的空间。若将线段存放在叶子和内部节点上，则会占用很少的空间，每条线段在每一层最多在 2 个节点上出现，也就是所有的线段将只占用 O(mlogn)的空间。

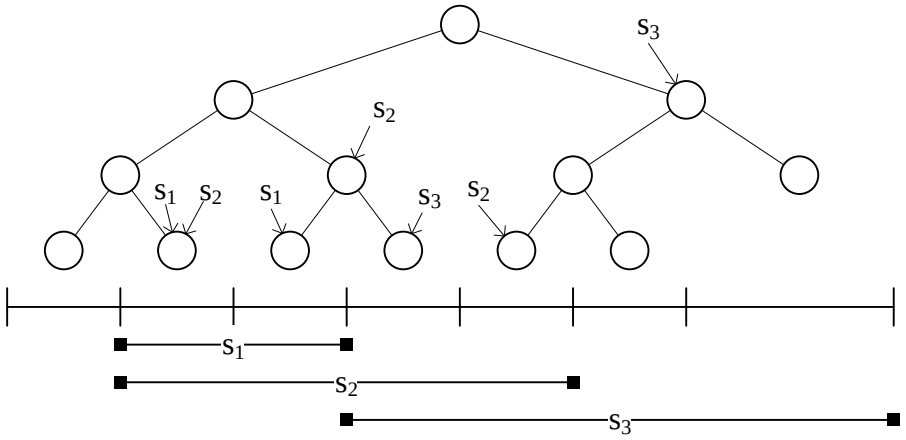


图 9

由于每段线段的权值不同，因此在线段树的每一个节点处都需要有一个数 Wi 记录当前节点对应线段的权值，同样这个权值是放在线段对应区间的最低共同祖先处。如上图中 S2 线段的存储位置。

在一般意义下的线段树中，每条线段需要在每一个记录节点上存储一遍。但是在 Shared Camera Control 问题中，我们并不需要对线段进行存贮，我们关心的问题仅仅是线段的权值和其对应的当前区域$[y_{\min},y_{\max})$中的那段区域。因此在每个节点 v 处（包括叶子节点）仅仅需要记录节点对应区间的权值的叠加值 Weight、该区间内最优权值 BestWeight 和其对

应的区间 BestInt(v)。实际上，整个 Segment Tree 仅仅需要 O(2n-1) = O(n)的空间。

在 Segment Tree 中添加/删除一条线段只需要 O(logn)时间，因为每条线段最多被存储在 O（logn）个节点上。要得到$[y_{\min}, y_{\max})$中的最优权值和对应区间，最简单的方法是遍历一遍 Segment Tree，查找权值最大的区间并记录。而在扫描线算法中，线段是逐条加入或者删除的，遍历树来查找最优权值必需在所有操作结束后才能进行。我们讨论后认为，可以在线段进行插入和删除的时候同时完成最优权值（和对应区间）的查找。

算法流程

```
BestWeight InsertSegment (Segment s, SegmentTreeNode * node)
{
    If (Interal(node) ∈ Interval(s))
    {
        node -> Weight += s.value;
        node -> BestWeight += s.value;
        return node -> BestWeight;
    }
    LeftBest = InsertSegment(s , node->lch);
    RightBest = InsertSegment(s , node->rch);
    Switch{
        Case LeftBest > RightBest:
          node -> BestWeight = LeftBest + node -> Weight;
          node->BestInt = node->lch->BestInt;
        Case RightBest > LeftBest:
          node -> BestWeight = RightBest + node -> Weight;
          node->BestInt = node->rch->BestInt;
        Case RightBest == LeftBest:
          node -> BestWeight = LeftBest + node -> Weight;
          node->BestInt = node->lch->BestInt ∪ node->rch->BestInt;
    }
    return node -> BestWeight;
}
```

综上所述，事先建立 Segment Tree 需要 O(nlogn)的时间和 O(n)的空间；扫描过程中每添加或者删除一条线段需要 O（logn）时间。

5. 多路排序的实现（kmerge）

从上面的介绍中可以看到，不论是在确定事件序列还是在建立 Segment Tree 的时候都需要进行排序，排序的对象分别是，所有矩形在 y 方向上的投影和所有矩形左右边在 x 方向上的投影。数据的个数为 N=m*n（每个 User Rectangle 被分解成$(2m+1)^2$个矩形，一共有 n 个 User Rectangle）。若使用快速排序，则需要 O(NlogN)时间。由 Vertical Decomposition 的修改算法我们可以得到这样的结果，每一个 User Rectangle 被分解后的矩形在 x，y 方向上的投影都可以按顺序得到。于是问题转化为 n 个有序数列的合并问题。

算法：

设 n 个有序数列 Ai 是按照升序排列，card(Ai)=ni。

首先建立一个最小二叉堆，叶子节点数为 n'（$n' = 2^p \geq n > 2^{p-1}$），堆的高度为 log(N')。第 1~n 个叶子节点为每个序列的首元素。其余叶子节点值赋为最大值。

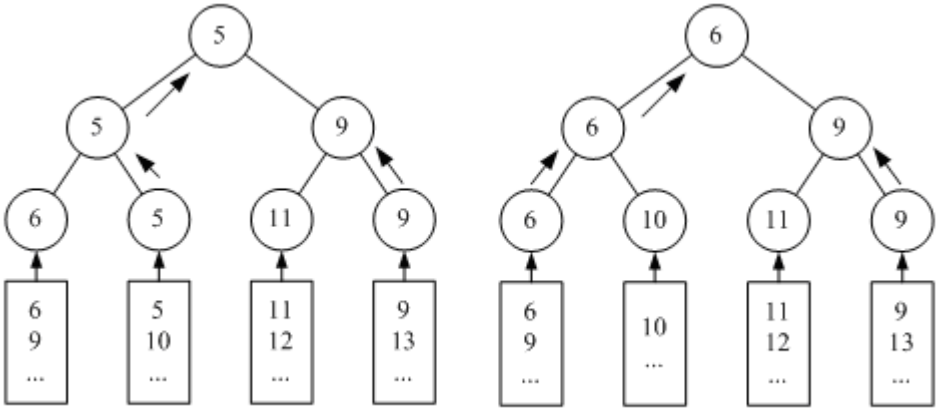


图 10

每次从堆顶取走一个元素，从相应的序列中填补一个数，修改堆的情况。如图 10 左图中的 5 被取走后 10 被填补进堆中，堆变为图 10 右图的情况。直到所有序列都变为空集。

数据结构：

堆得实现采用数组的方式，如图 11，编号为 i 的元素父节点为$\left\lfloor \frac{i}{2} \right\rfloor$，左孩子为 2i，右孩子为 2i+1。数组的每一个元素的结构如下：

```
Struct ArrayElement{
    double Value;          //节点值
    int ChosenChild;       //较小的孩子节点
};
```

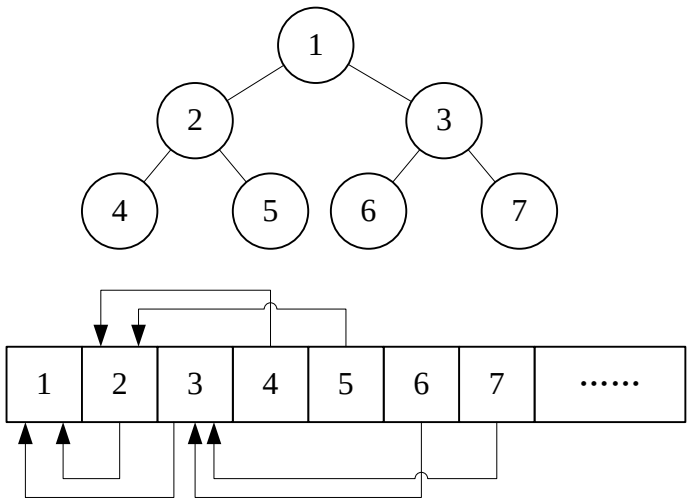


图 11

综上，多有序数列的排序需要 O(Nlogn)的时间和 O（n）的空间。

6. 随机生成矩形的算法

这是我们在测试算法正确性时使用到的算法。

我们没有按照均匀分布统一的生成所有的用户矩形，因为如果这样生成的话目标函数将在整个空间上趋于一个相同的值，和真实的情况不相符。在真实情况中，用户总是对一个画面中的某几个精彩的部分感兴趣，所以我们采用随机种子的方法来生成用户矩形。

我们首先让用户选定一些种子点，这些种子点代表感兴趣的位置。然后我们在这些种子点周围随机的生成矩形。具体的算法如下：首先给定一个比较小的概率 e，每一次生成矩形的时候，以概率 e 在整个空间上均匀的随机生成矩形，以概率(1-e)均匀的选定某一个种子，然后在种子周围以正态分布生成矩形，矩形的大小和位置都按照正态分布来生成，位置坐标的期望为种子的坐标。由于每一个种子代表画面上不同的部分，而画面上每一个不同的精彩部分用户会有不同的兴趣，有的希望仔细地查看细节，有的希望粗略的观看整体，所以我们对每一个种子都在一定范围内随机生成了正态分布的标准差，当在此种子周围生成矩形时就按照这个标准差所代表的正态分布进行产生。这样不同的种子就具有了不同的标准差，周围产生的矩形分布也各不相同。由此便可以很好的模拟真实的情况。接下来便是如何产生正态分布的问题。

由于 C 语言中只能生成均匀分布的随机数，所以我们按照如下的方法来生成正态分布的随机数。假设 x_1 和 x_2 为 $(0,1)$ 之间的均匀分布，我们所要生成的正态分布 y 的期望和标准差分别为 μ 和 σ，则

$$y = \sigma\sqrt{-2\ln x_1}\sin 2\pi x_2 + \mu$$

可以由联合分布的性质推导出上面的公式，此处从略。

系统结构和设计

1. 开发环境

硬件：

Intel P4 1.6G + 512 RAM

软件：

Windows 2003 + Visual C++ 7.0

2. 编译与链接

工程中使用了 MFC 自带的链接库，不需要特殊操作。

3. 总体框架

本程序严格依照 MFC Single Document 的 Doc – View 体系结构设计。一切数据（有 user rectangles, random seeds, brute force result）保存在 CSharedCameraControlDoc 中，所有界面控制在 CSharedCameraControlView 里。

主要分成三大模块：

1．系统框架及演示

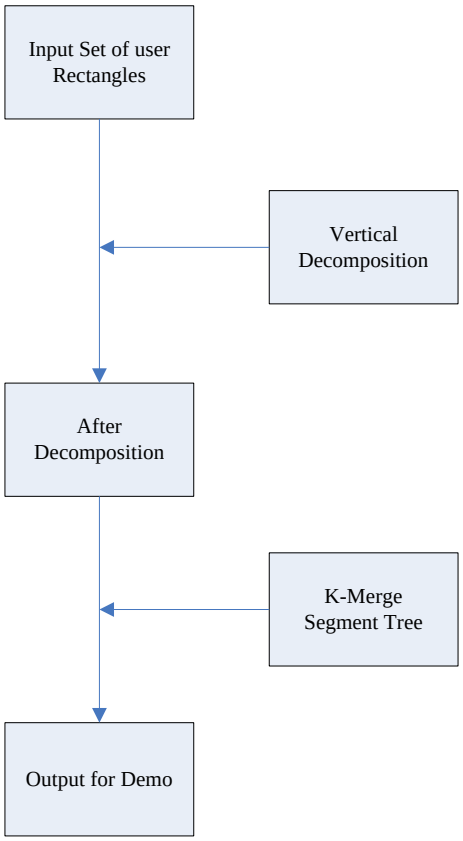
整个程序框架，负责数据的输入及输出，与其余两大模块的连接，所有演示的操作。

2．Vertical Decomposition

对系统输入的矩形集及相关参数，作 Vertical Decomposition，返回结果给系统。

3．K-Merge & Segment Tree。

对系统提供的Vertical Decomposition之后的矩形集，作 K 路归并，建立 Segment Tree，计算评价函数获得结果返回给系统。



4. 主要数据结构

矩形类型定义：

```
class CRectAngle
{
public:
	CRectAngle(void);
	~CRectAngle(void);
	// return the square of the rectangle
	double Square(void);
	// convert the double value to the int value (as a CRect class)
	CRect ToCRect(void);
	// another constructor
	CRectAngle(double dXLeft, double dXRight, double dYTop, double dYBottom,double
value);
	// set the gray value
	void SetGrayValue(unsigned int nValue);

	// left x
	double m_dXLeft;
	// right x
	double m_dXRight;
	// top y
```

```
    double m_dYTop;
    // bottom y
    double m_dYBottom;
    // the gray value, from 0 to 255
    unsigned int m_nGrayValue;
    // value in double
    double m_dValue;
};
```

注：为了适应精度运算的要求，使用浮点坐标。

扫描线类型定义：

```
class CSweepLine
{
public:
    CSweepLine(void);
    ~CSweepLine(void);
    // clear
    void Clear(void);

public:
    // current x-coordinate
    int m_nXCoord;
    // the local best area bottom
    int m_nYBottom;
    // the local best area top
    int m_nYTop;
    // the start x-coordinate
    int m_nXStart;
    // the end x-coordinate
    int m_nXStop;
};
```

注：其中，m_nXStart 与 m_nXStop 为扫描线结构数组中标记某一段局部最优区域起止位置的，与单条扫描线并无关系，这里把他们合并起来了。这里的扫描线演示只是为了形象地说明局部最优与全局最优的解，并非严格依照扫描过程边扫描边显示，而是先使用了 Fixed Camera Algorithm 计算完毕之后，保存相应结果，利用这里的 SweepLine 演示出来而已。

结果与分析

1. 退化和临界情况测试

当矩形退化到一条边时，如 test1~test3 所示，甚至当矩形退化到一个点时，程序能够及时报错。

2. 压力测试

为了测试程序的鲁棒性，我们还进行了大数据量的测试，测试了 1000 个 User Rectangle 的情况（当然 epsilon 最好不要取得太低，推荐使用 2，否则程序会需要很长时间才能得到结果）。

3. 数据结果对比

以下是运行多次的平均结果（Levels = 1）：

Number of users	Epsilon	Run time (ms)
10	0.2	250
10	0.5	30
50	0.2	1312
50	0.5	231
100	0.5	511
100	1.0	130
200	1.0	360
200	1.5	160
200	2.0	90
400	1.0	2684
400	1.5	411
400	2.0	180
600	1.5	1833
600	2.0	931
800	2.0	1813
1000	2.0	1982

图中带蓝色标记的是我们认为对于所给 users 数目，比较合适的 epsilon 数值，这样的运行时间可以接受，并且精度尽可能高（epsilon 尽量小）。

测试中发现我们的算法是对输入敏感的，即 users 数相同时，不同的输入对时间的影响非常大，例如在 users = 1000, Epsilon = 2.0 的情况下，Run time 可相差一倍（见下图）。表格中我们取的是随机生成的输入，运算 10 次的平均结果。

Number	Run time (ms)
1	1291
2	1382
3	2293
4	2844
5	2163
6	1332
7	2053
8	2003
9	2264
10	2193

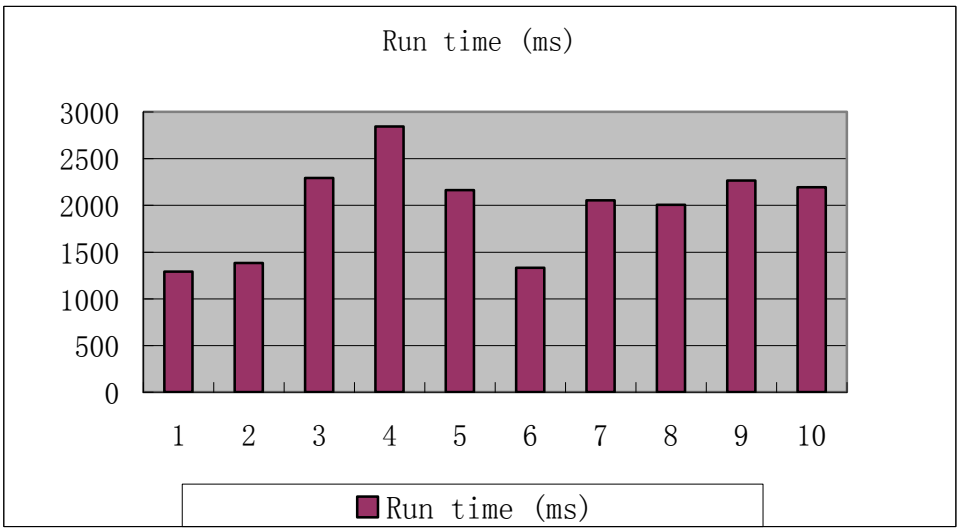


图 12

在 Epsilon = 2.0，Levels = 1 的情况下，我们与论文中的结果有如下比较：

Number of users	Song at al. (ms)	On the paper (ms)	Ours (ms)
200	15	125	90
400	32	204	180
600	125	562	931
800	157	781	1813
1000	281	1578	1982

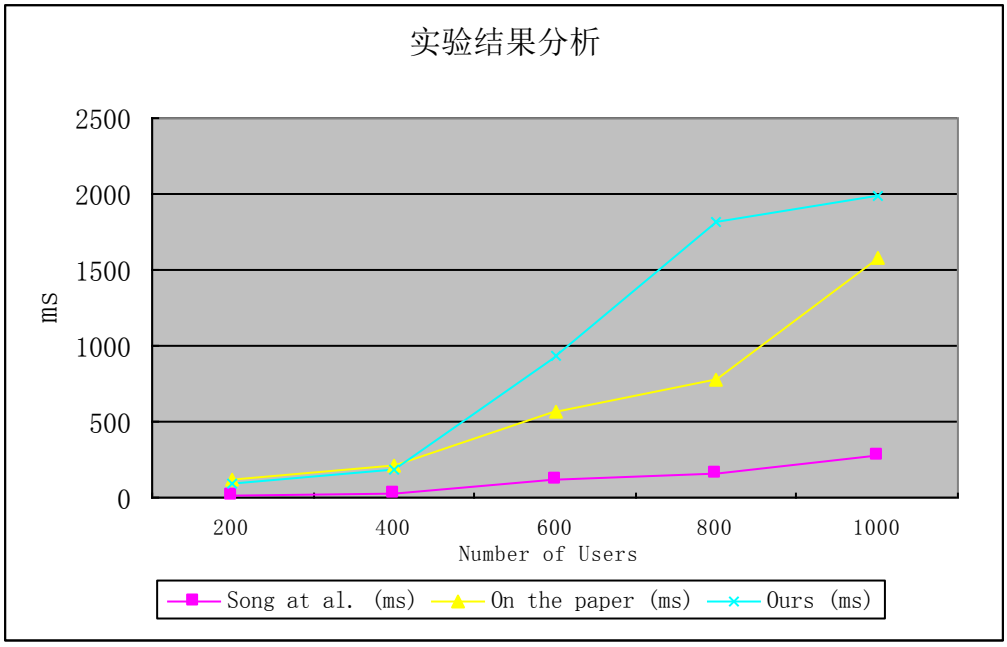


图 13

需要注意的是，表格中的 Song at al.，On the paper 是 Sariel Har-peled 等人在[1]中所给出的结果，他们所用的硬件环境是 Dual Intel Xeon 2.4G，2G RAM，软件环境为 VC++ / Windows。而我们的硬件环境是 Intel P4 1.6G，512 RAM，软件环境为 VC++ / Windows。从表格中可以看出在 users 较少（200，400）的情况下，我们算法的速度比[1]中的要快，但是在 users 数目比较多（600 以上）之后速度就明显比[1]中要慢。users = 600, 800 的时

候得到的数据约是[1]中的两倍，到了 users = 1000 时又差不多。这应该是由内存原因造成的。

结论：我们很好的实现了[1]中提出的 Shared Camera Control 的近似算法，并且做出了一些有效的改进。

4. 对实验结果的理论分析

通过扫描线演示可以发现，在扫描线的扫描过程中，最大值的位置会出现上下振荡的情况（如图），下面我们来说明这一情况出现的原因及其正确性。

考虑图 14 中所示的两个矩形，它们矩形分解后位置关系如图所示。各个矩形的中心标出了其权值（没有叠加的时候）。阴影部分为在此扫描线上权值最大的部分。下面我们来说明满足什么情况的时候会出现如图所示的阴影部分上下振荡出现的情况。

首先容易知道每一条扫描线上权值最大的区域只可能出现在表上记号的两个条带上。考虑矩形区域 1，其权值的总和为 $A^* + \frac{B^*}{1+\varepsilon}$，矩形区域 $1'$，其权值总和为 $B^* + \frac{A^*}{1+\varepsilon}$。矩形区域 2 的权值为 $\frac{A^*}{1+\varepsilon} + \frac{B^*}{1+\varepsilon}$，矩形区域 $2'$ 的权值为 $\frac{A^*}{(1+\varepsilon)^2} + B^*$。

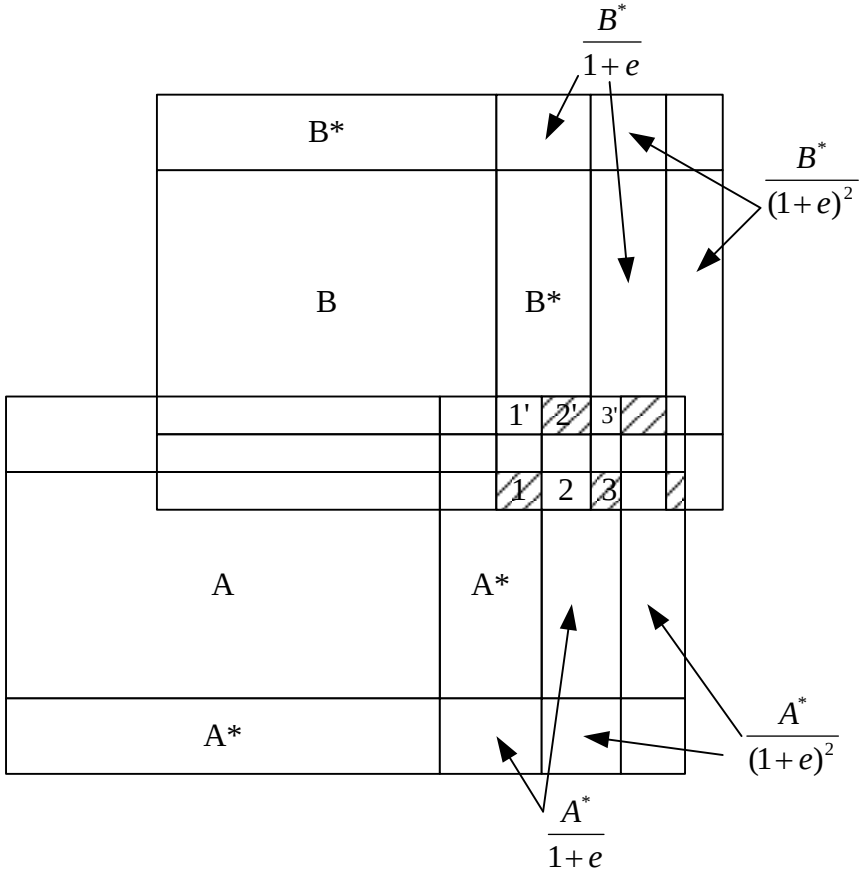


图 14

按照图中的阴影部分分布要求

$$A^* + \frac{B^*}{1+\varepsilon} > B^* + \frac{A^*}{1+\varepsilon}$$

$$\frac{A^*}{(1+\varepsilon)^2} + B^* > \frac{A^*}{1+\varepsilon} + \frac{B^*}{1+\varepsilon}$$

所以 $A^* > B^* > \frac{A^*}{1+\varepsilon}$，同理可以证明，只要满足这个条件就会产生后面的震荡分布。

对于两个不同的用户矩形，当它们的位置靠得比较近，产生上述重叠区域并且矩形的权值又正好满足 $A^* > B^* > \frac{A^*}{1+\varepsilon}$ 的关系时就会使最大值产生震荡的分布。

修改与展望

对检查中发现的问题的改进

在检查实验的过程中，邓俊辉老师指出我们的程序中存在以下几点问题：

a) 扫描线演示，应当能够设定扫描速度。
b) 扫描过程中，应该能够随时终止，不必每次等待至扫描完成。
c) 还存在一些 warning 没有消除。
d) 在扫描的过程中不应当能够选择"保存"的选项。
e) 设置参数时可以选取消，但是程序无法正常运行了。
f) 由于运算使用的坐标系与屏幕坐标系的 Y 轴方向相反，所以使用一组输入数据后，窗口的大小不能进行拖动，否则会造成显示上的错误。

以上 bug 在这次的版本中都更正了。

进一步的工作

1．算法方面

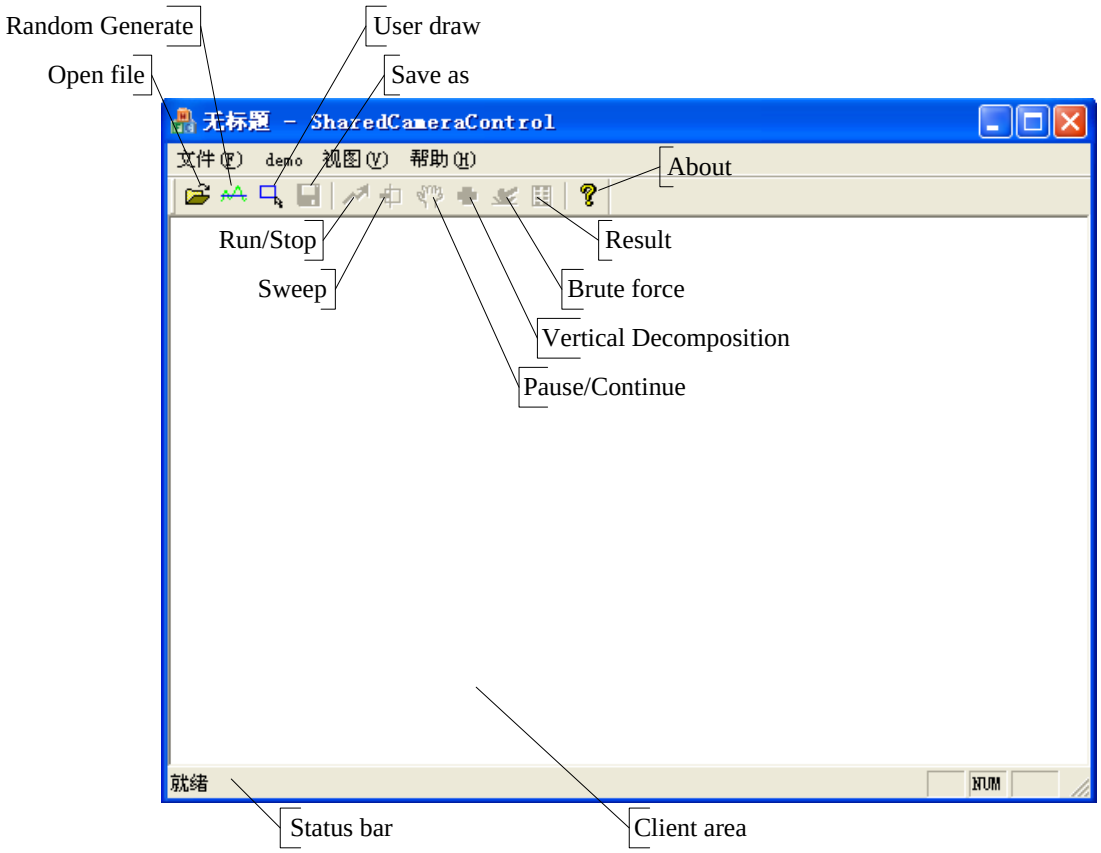
a) 除了 epsilon 近似算法之外，[1]中还提出了准确的算法，该算法难度较大。

2．界面方面

a) 现在的扫描线演示过程中，画面抖动太严重，比较刺眼。
b) 用户绘制矩形的过程中，看不到正在画的那个矩形框，只能通过坐标判断。

用户手册

1. 界面一览



注：所有工具栏中的图标都与菜单中的某项对应，可以从菜单选择，也可以点击工具栏中的图标。

2. 使用说明

1．第一步输入矩形，可以有 3 种来源方式：

a) 给定种子点随机生成。

首先设置要生成的矩形数目（1 ~ 1000），宽与长的比（0 ~ 2.0，若设为 0 则表示不必按照固定比例），近似值 epsilon（0.2 ~ 2.0），摄像机长（10 ~ 客户区长），宽（10 ~ 客户区宽），欲比较的 Level 数。

然后用左键点选若干种子，右键点表示输入结束，这时就在所给的种子周围生成了指定数目的矩形。

b) 用户手动画出矩形。

首先设置近似值 epsilon（0.2 ~ 2.0），摄像机长（10 ~ 客户区长），宽（10 ~ 客户区宽），欲比较的 Level 数。

然后在客户区用鼠标拖出矩形，同样用右键点表示输入结束。

c) 从文件读取。

首先选择要读入的文件（*.rec，实际上是使用本程序保存功能存的文件）。

然后设置近似值 epsilon（0.2 ~ 2.0），摄像机长（10 ~ 客户区长），宽（10 ~ 客户区宽），欲比较的 Level 数。

2．第二步可以对所给输入进行运算，在菜单->Demo 中，共有以下几项功能：

a) Run

顾名思义，运行整个 Shared Camera Algorithm，即使用不同的 Level 下的摄像机镜头大小运行 Fixed Camera Algorithm 得到多个全局最优解，最后取一个总的最大解作为结果。

b) Sweep

显示对当前 Camera 大小所做的 Fixed Camera Algorithm，扫描的整个过程。可设置隔多少毫秒（25 ~ 500）水平前进 1 个像素。扫描过程中绿色的是扫描线，暗红色是同一扫描线上的局部最优区域。扫描结束后标出全局最优区域（蓝色），所选择的 Camera 框（绿色）及 Camera 中心十字叉（红色）。

扫描过程中再点 Sweep 可中止扫描。

c) Pause

扫描过程中点 Pause 可以暂停扫描，再点则继续。

d) Show Decomposition

对输入矩形显示我们所做的 Decomposition 后的结果，这是一个算法示意，所以一般画一个矩形调用此功能。

e) Brute Force

对输入矩形做 Decomposition 后，使用 Brute Force 方法计算评价函数，画出评价值的灰度图。这个结果可以作为我们的近似算法的验证。灰度越高的区域表示 Value 值（匹配度）越大。

f) Result

做完 Run 或 Sweep 操作后，可以选这个选项察看结果。包括所选择的 Camera 大小，中心位置，Value，以及运行时间（仅指 Run 操作）。

3．其余功能

a) Save as

保存当前所有的输入矩形到文件（*.rec）。在扫描过程中不能保存。

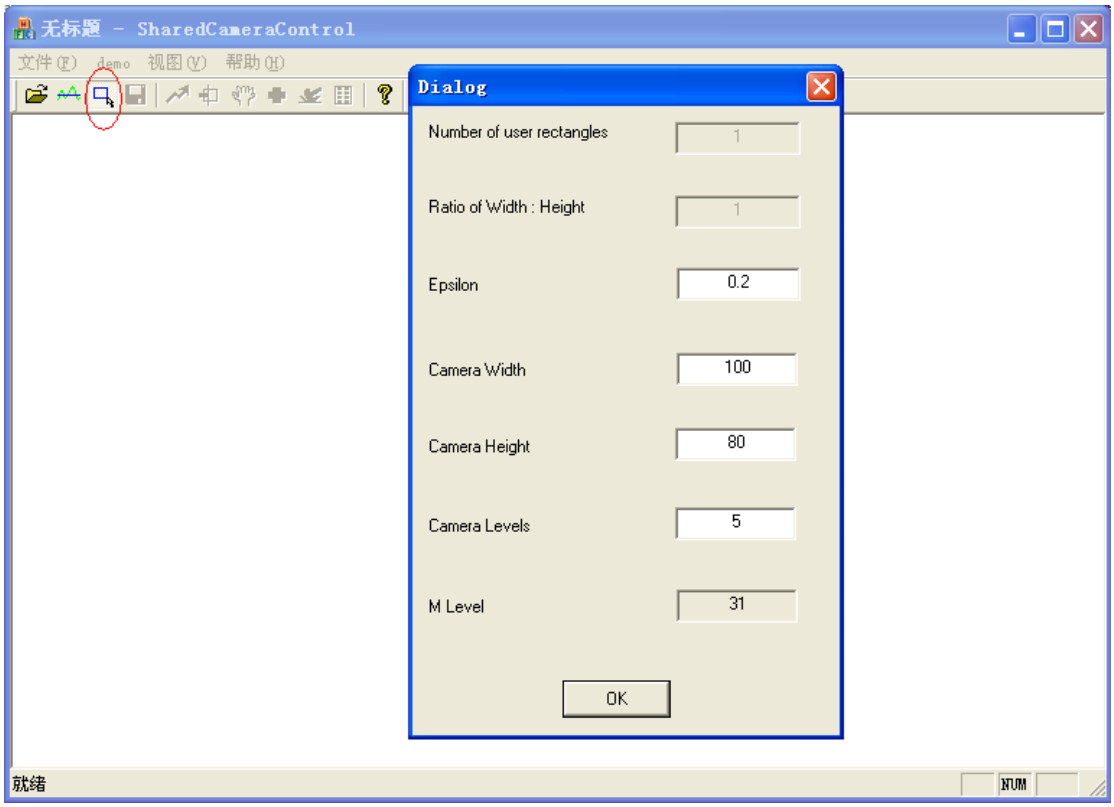
b) Misc

某些图标有 Pushed，Poped 两种状态，代表两个不同的操作。如果有不清楚的可以随时看状态栏的提示。

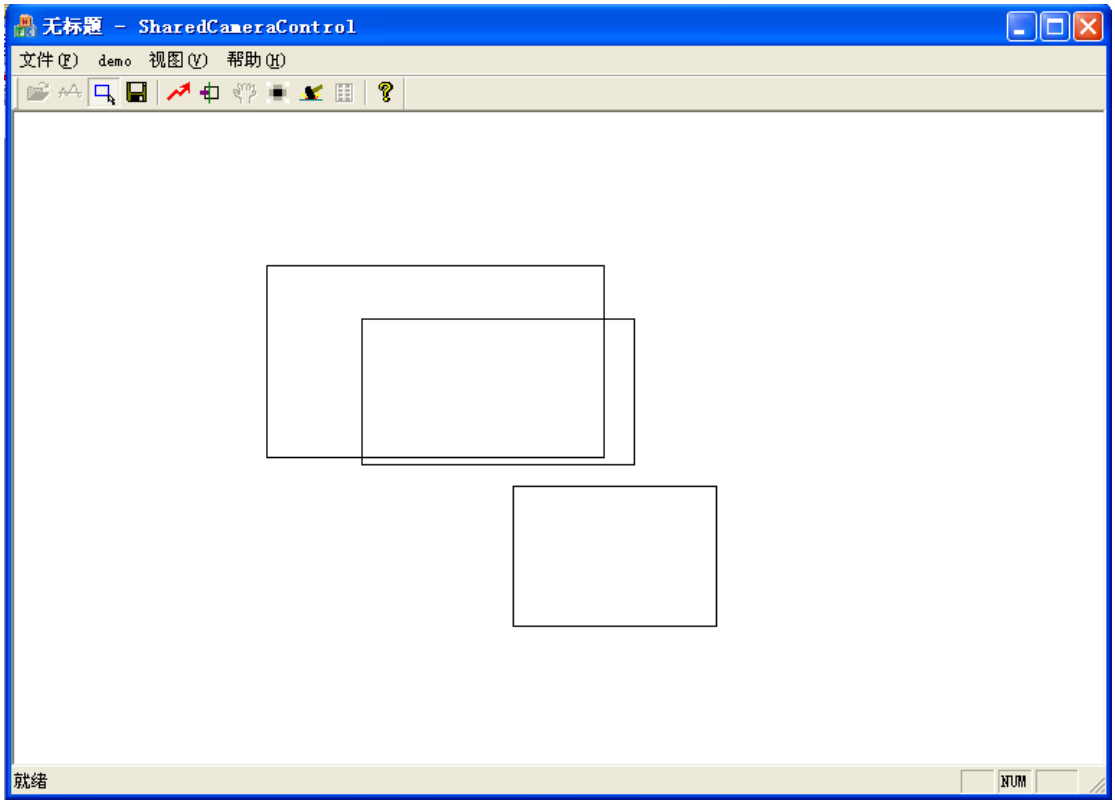
3. 示例

以下是用户输入的一个小例子，演示使用过程：

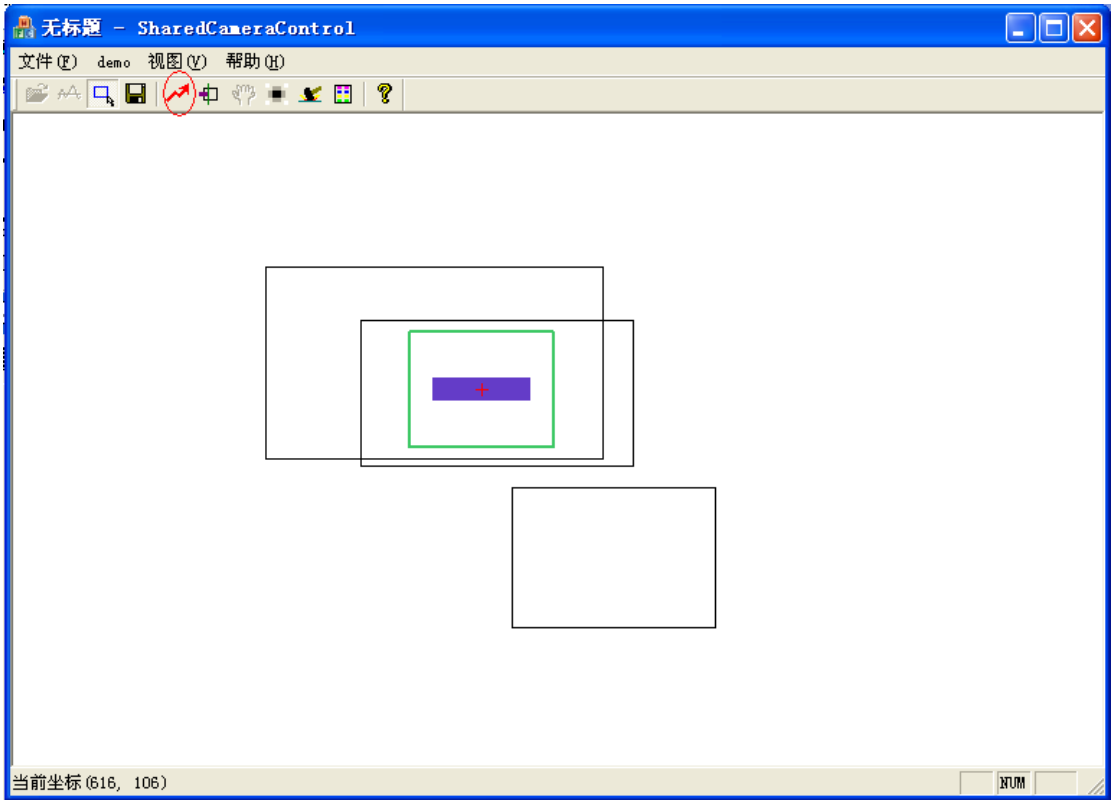
1．选择用户输入，设置参数。



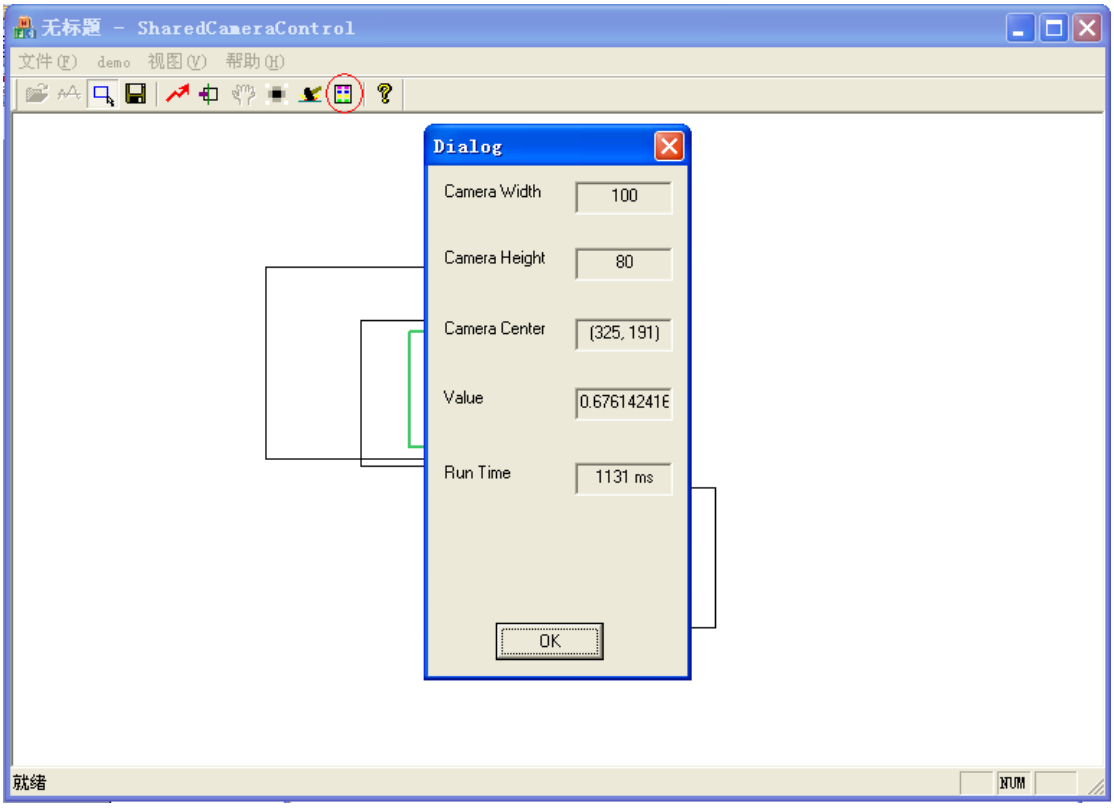
2. 拖出矩形，右键表示输入结束



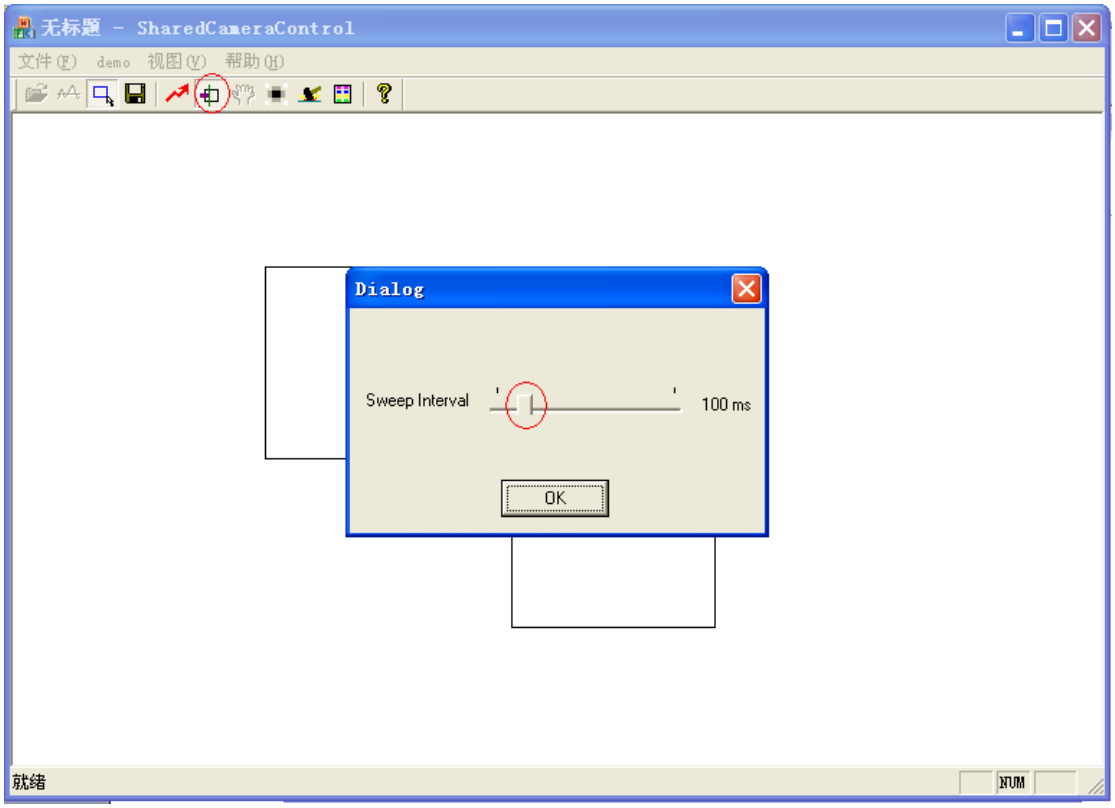
3. Run



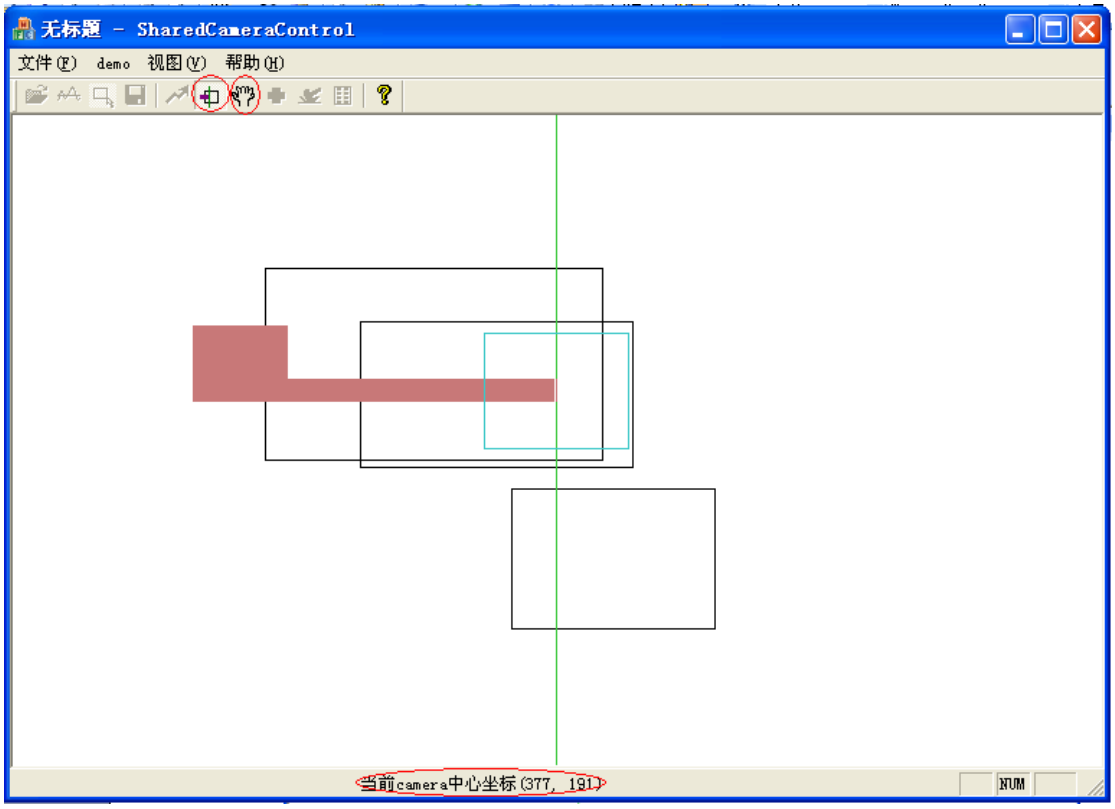
4. 察看结果



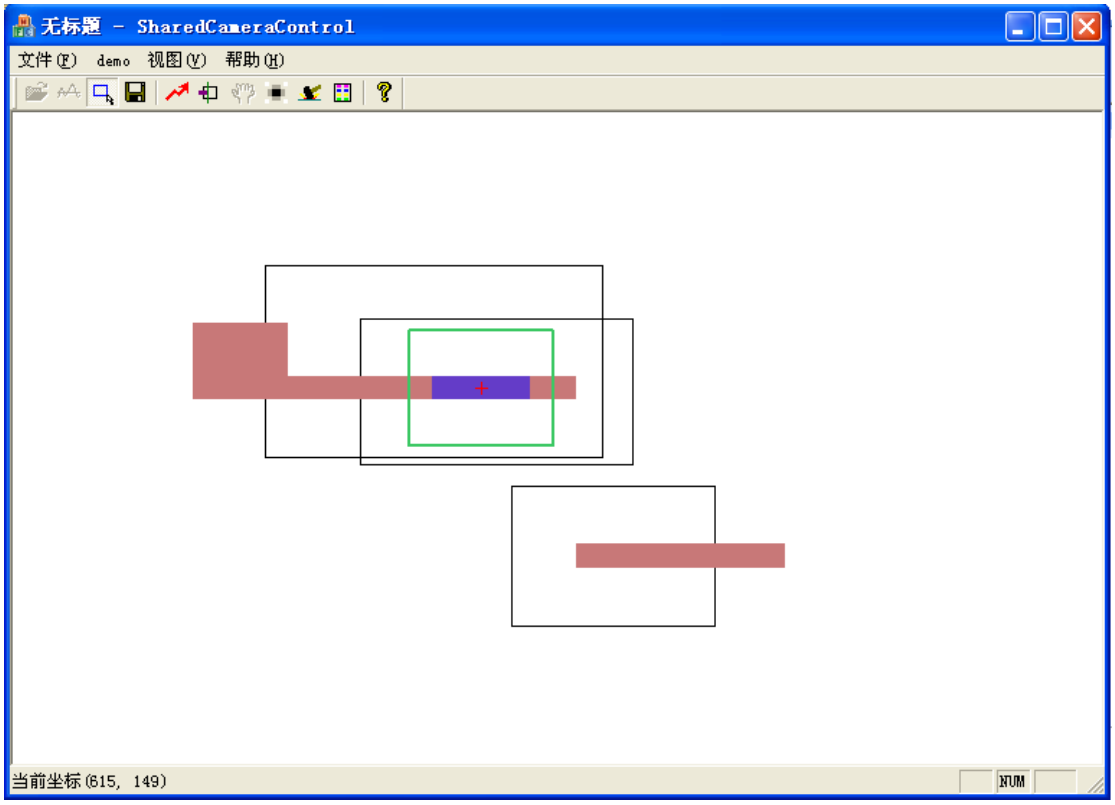
5. Sweep 设置



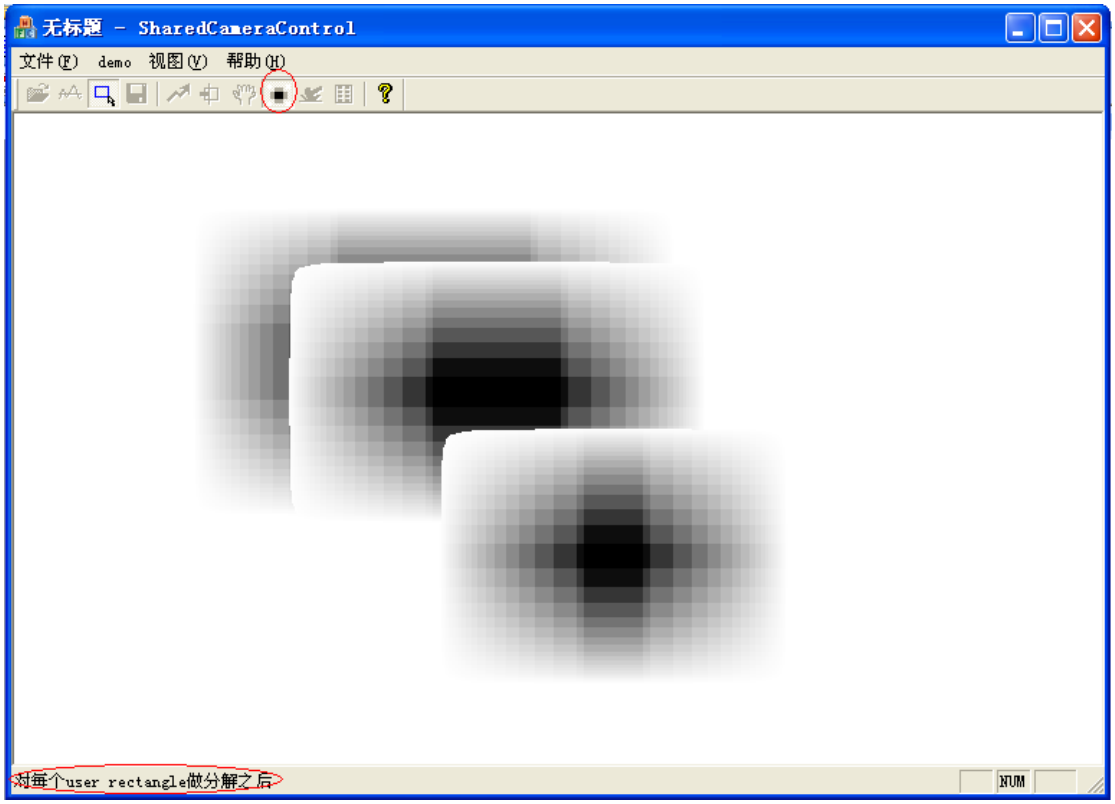
6. Sweeping



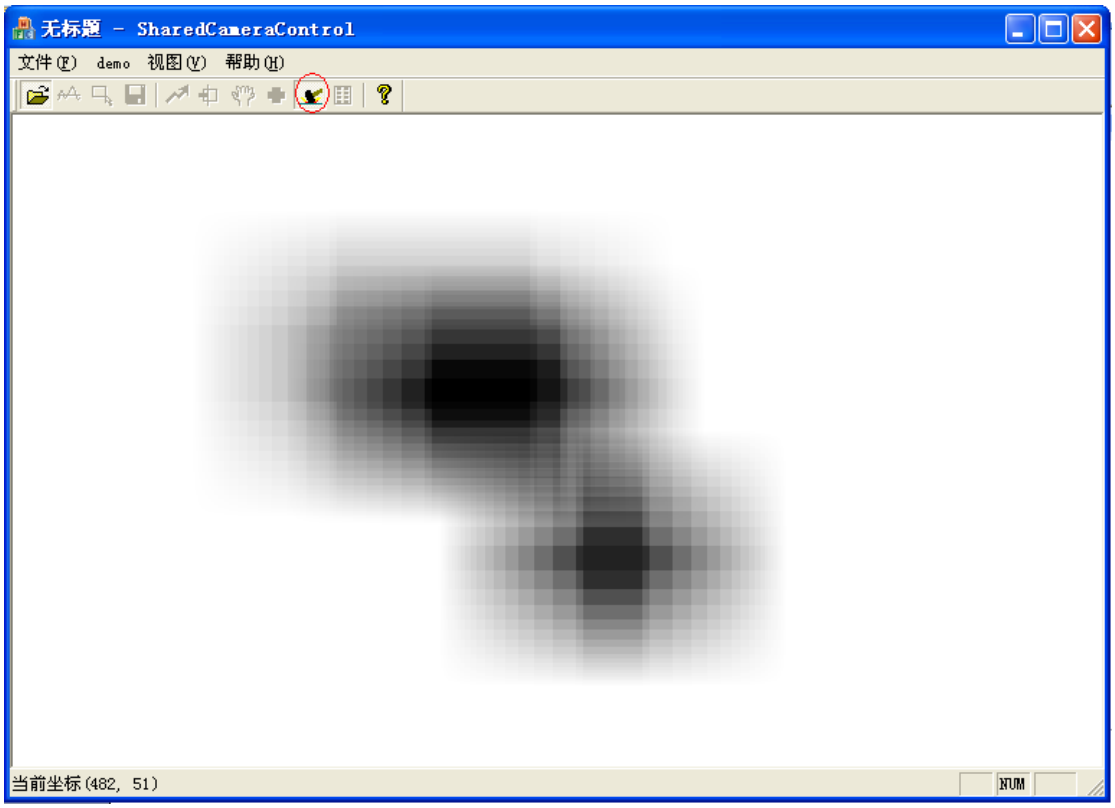
7. Sweep 结束



8. 显示 vertical decomposition



9. 显示 brute force



参考文献

1. Efficient algorithms for shared camera control
Sariel Har-Peled, Vladlen Koltun, Dezhen Song, Ken Goldber
Proceedings of the nineteenth conference on computational geometry (June 2003)

2. Computation Geometry – Algorithms and Applications
Mark de berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf

3. Kinetic data structures — a state of the art report.
L. J. Guibas.
In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, Workshop on Algorithmic Foundations of Robotics, pages 191–209. 1998.

4. Maintenance of configurations in the plane
M. H. Overmars and J. van Leeuwen.
J. Comput. Syst. Sci.,23:166–204, 1981.