Z 缓冲消隐算法的改进

简学东 陆 玲 莫桂花
(东华理工学院 江西 抚州 344000)

摘 要 Z 缓冲算法是一种比较简单的消隐算法,Z 缓冲区的大小与屏幕的视图区大小相关,一般情况下设置成视图区的大小,因此该算法只适用于固定大小投影,即通用性差,另外当图形较小时又浪费空间。本算法通过动态生成相应 Z 缓存,从而适用于任何大小的图形,即通用性强,同时减小内存。

关键词 Z 缓冲算法 通用性 减小内存

AN IMPROVEMENT TO Z-BUFFER HIDDEN SURFACE REMOVAL ALGORITHM

Jian Xuedong Lu Ling Mo Guihua
(*East China Institute of Technology, Fuzhou 344000, Jiangxi, China*)

Abstract Although z-buffer method is a simple hidden surface removal algorithm, it needs considerable systemic resources and only suits a fixed image. As an improvement to Z-buffer method, it dynamically create the relative memory so that it can suit arbitrary image, at the same time, it can also decrease memory.

Keywords Z-buffer algorithm Generality Decrease memory

0 引 言

造型是计算机三维图形处理的基础,而消隐则是三维造型的关键。所谓消隐就是隐藏从当前观察点看不见的三维模型表面,即面消隐。其中面消隐的方法有:区域排序算法、Z 缓冲消隐算法(Z-buffer)[1]、线射踪迹算法[2]、扫描线算法[3]。其中 Z 缓冲消隐算法是最简单的消除隐藏面算法之一。Z 缓冲算法的最大缺点是:两个缓存数组占用的存储单元太多,而且事先要设置一定的缓冲大小,但是实际中要处理的图形的大小是未知的,一般情况下设置成视图区的大小,如果图形比视图区大时就会造成部分图形被裁剪,即通用性差,如果图形比视图区小时又浪费存储空间。本算法通过动态生成相应 Z 缓存,从而适用于任何大小的图形,即通用性强,同时减小内存。本算法通过 Delphi 实现。

1 通用 Z 缓冲区算法的原理

依据 Z 缓冲算法的原理可知:Z 缓冲区中的每个单元与视图区上的每个像素是一一对应的。如果 Z 缓冲区中的每个单元的位置通过某种关系使之与视图区上相应像素相联系时,就可以适用于任何大小的图形。通过观察投影图可知,Z 缓冲区的每个单元的位置与视图区上相应像素的关系如下:点 Z(x,y)相对应 Z 缓冲单元为 Zbuffer [y-TopLeftY ,x-TopLeftX],其中 TopLeftX 为投影图被一个矩形刚好所包含时该矩形左上角的横坐标,TopLeftY 为相应的纵坐标,其中矩形代表缓冲区的大小,如图 1 所示。

当图形消隐时,视图的大小为:宽度等于 TopLeftX + 矩形的宽度,高度等于 TopLeftY + 矩形的高度,即视图大小为[TopLeftY + 矩形的高度,TopLeftX + 矩形的宽度],由此可知,对于任何大小的图形,本算法都适用。从图 1 可知,如果 Z 缓冲区的大小设置为上述矩形的大小时,就可以减小 Z 缓存空间。

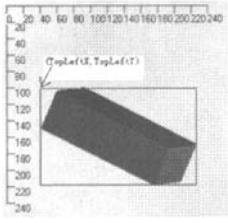

图 1 投影图

2 消隐算法

(1) 设置 Z 缓冲区的大小

根据所要投影图形与其投影方向的关系,计算出该图形投影到平面上的可见面,并计算出刚好能够包含该可见面的矩形,其中矩形的长度和宽度分别代表缓冲区的长度和宽度。

(2) 设置 Z 缓冲区的开始位置

计算上述矩形左上角的横坐标 TopLeftX,纵坐标 TopLeftY。

(3) 三维图形上的点与 Z 缓冲区相应单元的关系

对于图形上任意一点的深度 Z 值,表示为 Z(x,y),其中 x 为横坐标,y 为纵坐标。依据投影关系可知,Z(x,y)应与 Zbuffer

收稿日期:2006 - 10 - 30。简学东,硕士生,主研领域:计算机图形图像处理和数据库应用。

[y-TopLeftY ,x-TopLeftX]相比较(假设采用右手坐标系),当 Z(x,y) > Zbuffer [y-TopLeftY ,x-TopLeftX]时,表明该点在前面,视图区上该位置的颜色应设置为该点的颜色,并把 Zbuffer [y-TopLeftY ,x-TopLeftX]改为 Z(x,y)的值;当 Z(x,y) < Zbuffer [y-TopLeftY ,x-TopLeftX]时,则说明对应像素已经显示了对象上一个点的颜色,该点要比对应的点更接近观察点,所以 Z 缓冲区相应单元的值均不改变。

(4) 实现方法

本程序算法如下:

```
通用 Z 缓冲区算法()
begin
  计算刚好能包含图形的矩形的宽度(Width)和高度(Height)
  计算矩形的左上角横坐标 TopLeftX,纵坐标 TopLeftY;
  设置 z 缓冲区的大小为 Zbuffer[Height,Width];
  设置颜色 Colorbuffer 缓冲区的大小为 Colorbuffer[Height,Width];
  Zbuffer 深度缓存全部设置为最小值;
  Colorbuffer 缓存全部设置为背景颜色;
  for(图形的每个面)
    begin
      for(上述面所包含的每个像素(x,y) )
        begin
          计算该像素的深度值 Z(x,y);
          If (Z(x,y)大于 Zbuffer [y-TopLeftY ,x-TopLeftX]的值)
          Begin
            Zbuffer[y-TopLeftY ,x-TopLeftX]:= Z(x,y);
           Colorbuffer[y-TopLeftY ,x-TopLeftX]处的颜色改为该面的
           颜色;
        End;//if
      End;//for
    End;//for
  End;// 通用 Z 缓冲区算法()
```

3 结果与讨论

动态改变 Z 缓冲算法是在经典 Z 缓冲算法的基础上改进而成,该算法能解决经典 Z 缓冲算法通用性差的缺点,同时能有效减小 Z 缓冲空间,适用于任何大小的图形。为了进一步说明问题,下面分别列出两种算法的测试(默认视图大小:1024 * 768,多边形 60000 个,电脑配置:P4 2.66G,内存 512MB),测试效果如表 1。

表 1 两种算法效果比较

多边形大小 \ 算法	经典 Z 缓冲算法			改进 Z 缓冲算法		
	占用内存(KB)	是否正确显示	运行时间(MS)	占用内存(KB)	是否正确显示	运行时间(MS)
100 * 100	1536	√	2.57	19KB	√	2.78
200 * 200	1536	√	11.68	78KB	√	12.50
500 * 500	1536	√	25.17	488KB	√	27.09
1000 * 1000	-	×	-	1953KB	√	92.92
2000 * 2000	-	×	-	7812KB	√	417.92

参 考 文 献

[1] 陆玲,杨勇. 计算机图形学[M]. 北京:科学出版社,2006.

[2] 潘云鹤. 计算机图形学-原理、方法及应用. 2001.

[3] 生滨,李东,徐学敢. Z 缓冲区消隐算法的改进[J]. 计算机工程与应用,2001(23).

(上接第 142 页)

```
Object[ ] paramvalue = new Object[paramlength];
    paramvector.addElement(mapreg,map(elementtype,value));
    parameters[i] = ((Bean) paramvector.elementAt(i)).type;
    paramvalue[i] = ((Bean) (paramvector.elementAt(i))).value;
```

(2) RPC 服务调用

远程服务器在 2048 端口监听到达的 RPC 调用,当 RPC 调用消息到达服务器时,服务器分析出调用的服务对象、方法和参数,调用相应的对象的方法,然后生成应答数据。其核心代码如下:

```
Class Server =
  Class.forName((String)(vectors.elementAt(0))).trim());
Class param[ ] = null;
Constructor con = Server,getDeclaredConstructor(param);
object values[ ] = null;
Object ServerObj = con.newInstance(values);
int paramlength = vectors.size( ) -2;
Class Parameters[ ] = new Class[paramlength];
for(int i =0;i <paramlength,i ++ ){
Parameters[i] = ((Bean)vectors.elementAt(i +2)).type;
}
Method meth = Server.getDeclaredMethod((String)
  (vectors.elementAt(1))).trim( ),Parameters);
Object paravalue[ ] = new Object[paramlength];
for(int i =0;i <paramlength;i ++ ){
  paravalue[i] = ((Bean) (vectors.elementAt(i +2))).value;
}
Class returnclass = meth.getReturnType( );
Vector responsevector = (Vector) meth.invoke((ServerClass) ServerObj,
paravalue);
String returnclassname = (String) responsevector.elementAt(0);
Class classinreturn = Class.forName(returnclassname);
Field[ ] fields = classinreturn.getDeclaredFields( );
int fieldnum = fields.length;
    if(
      fields[j].getType( ).toString( ).equal
  s("class java.lang.String")
)
String temp = (String)
responsevector.elementAt(elementlocate)
```

以上的程序代码为 XML 数据编码实现和 RPC 调用的核心代码。

4 结束语

本文实现了基于 SOAP 协议的 XSOAP 系统。该系统使用 JAVA 开发语言,实现了对分布式计算的支持。由于对该系统的核心代码拥有版权,可以很好的在它的基础上实现二次开发,嵌入新的功能。该系统可以应用于各种分布式计算和 Web 服务系统,具有极大的应用价值和实际意义。

参 考 文 献

[1] W3C. SOAP:Simple Object Access Protocol Specification 1.1.2000.

[2] Whitehead E,Murala M. XML Media Types. RFC2376,1998.

[3] SOAP Toolkit 2.0 Documentation[EB/OC],http://www.microsoft.com/soap,2001-4-10.