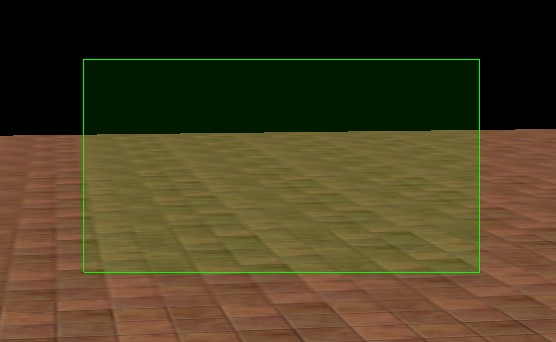
几次看见有人问框选物体的做法，之前斑竹也介绍过，用画的框生成的视椎，用经典图形学的视锥裁剪就能做到。  
视锥裁剪资料学习:<http://www.linuxgraphics.cn/graphics/opengl_view_frustum_culling.html>  
  
在unity里实现，其实很简单，因为有两个前提：  
1.画的方框始终是在屏幕空间进行的，而屏幕空间其实就是摄像机的视锥空间的投射了,不需要另外计算视锥。  
2.unity摄像机内建的功能，可以方便的把屏幕坐标和世界坐标互换。  
  
这样最简单的画框选物体就简化成了：  
1.用GL在屏幕空间动态画框。  
2.把备选对象的transform.position用camera.WorldToScreenPoiont变换成屏幕坐标。  
3.判断这些position的点是否落在画的方框之内，如果是就把对象切换到画边框的层，呈现选中效果。  
  
**一.屏幕动态画框**  
这个过程相对简单 就是在鼠标按下的时候记下鼠标位置，然后在鼠标移动时在当前鼠标位置和按下的位置之间画一个方形就行了。

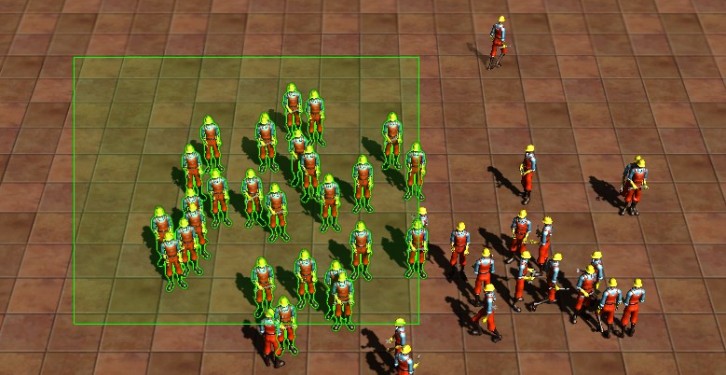
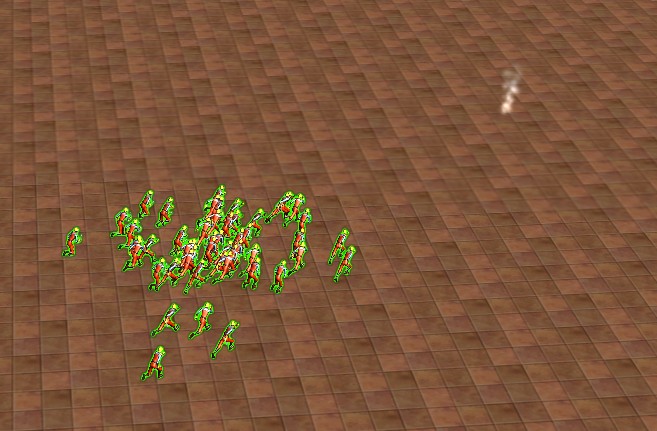
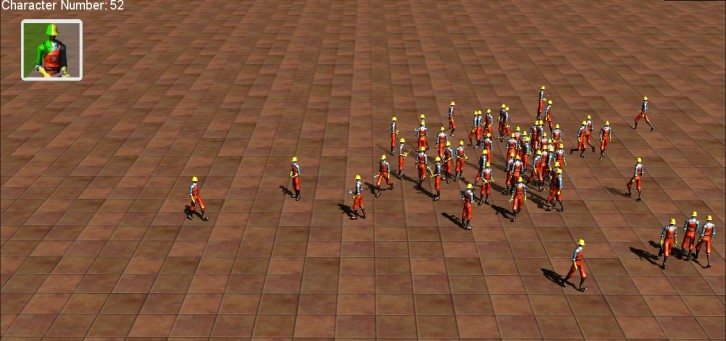
[复制代码](http://blog.csdn.net/xv_ly15/article/details/8563068)

1. using UnityEngine;
2. using System.Collections;
3. public class DrawRectangle : MonoBehaviour {
4. public Color rectColor = Color.green;
6. private Vector3 start = Vector3.zero;//记下鼠标按下位置
7. private Material rectMat = null;//画线的材质 不设定系统会用当前材质画线 结果不可控
8. private bool drawRectangle = false;//是否开始画线标志
10. // Use this for initialization
11. void Start () {
12. rectMat =  new Material( "Shader \"Lines/Colored Blended\" {" +
13. "SubShader { Pass { " +
14. "    Blend SrcAlpha OneMinusSrcAlpha " +
15. "    ZWrite Off Cull Off Fog { Mode Off } " +
16. "    BindChannels {" +
17. "      Bind \"vertex\", vertex Bind \"color\", color }" +
18. "} } }" );//生成画线的材质
19. rectMat.hideFlags = HideFlags.HideAndDontSave;
20. rectMat.shader.hideFlags = HideFlags.HideAndDontSave;
21. }
22. void Update () {
23. if(Input.GetMouseButtonDown(0)){
24. drawRectangle = true;//如果鼠标左键按下 设置开始画线标志
25. start = Input.mousePosition;//记录按下位置
26. }else if (Input.GetMouseButtonUp(0)){
27. drawRectangle = false;//如果鼠标左键放开 结束画线
28. }
29. }
31. void OnPostRender() {//画线这种操作推荐在OnPostRender（）里进行 而不是直接放在Update，所以需要标志来开启
32. if(drawRectangle){
33. Vector3 end = Input.mousePosition;//鼠标当前位置
34. GL.PushMatrix();//保存摄像机变换矩阵
36. if (! rectMat)
37. return;
39. rectMat.SetPass( 0 );
40. GL.LoadPixelMatrix();//设置用屏幕坐标绘图
41. GL.Begin(GL.QUADS);
42. GL.Color( new Color(rectColor.r,rectColor.g,rectColor.b,0.1f) );//设置颜色和透明度，方框内部透明
43. GL.Vertex3( start.x,start.y,0);
44. GL.Vertex3( end.x,start.y,0);
45. GL.Vertex3( end.x,end.y,0 );
46. GL.Vertex3( start.x,end.y,0 );
47. GL.End();
48. GL.Begin(GL.LINES);
49. GL.Color(rectColor);//设置方框的边框颜色 边框不透明
50. GL.Vertex3( start.x,start.y,0);
51. GL.Vertex3( end.x,start.y,0);
52. GL.Vertex3( end.x,start.y,0);
53. GL.Vertex3( end.x,end.y,0 );
54. GL.Vertex3( end.x,end.y,0 );
55. GL.Vertex3( start.x,end.y,0 );
56. GL.Vertex3( start.x,end.y,0 );
57. GL.Vertex3(start.x,start.y,0);
58. GL.End();
59. GL.PopMatrix();//恢复摄像机投影矩阵
60. }
61. }
62. }

效果如图:  
  
注意GL绘图都是每帧进行的，所以不需要清除，直接不绘制方框就消失了。   
  
二.判断物体是否选中  
有了方框，要判断物体是否在方框内，按照经典的数学算法可以根据直线方程和点的坐标计算判断点在线的左边 右边 还是线上  
  
空间平面方程可表示为：  
Ax+By+Cz=0  
  
对于点（x1, y1, z1），有  
若 Ax1+By1+Cz1 = 0,则点在平面上；  
若 Ax1+By1+Cz1 < 0,则点在平面的一侧；  
若 Ax1+By1+Cz1 > 0,则点在平面的另一侧；  
  
但是在这里，因为都是水平、垂直的线，我们并不需要考虑画斜线、曲线框选物体........,所以只需要简单的比大小就行了，土了那么一点但其实效率更高，毕竟作游戏不是算法比赛，怎么简单高效怎么用。  
如果:物体的屏幕position.x >方框左下角.x && 物体的屏幕position.y >方框左下角.y && 物体的屏幕position.x <方框右上角.x && 物体的屏幕position.y <方框右上角.y, 那么这个物体就是在框选范围里了。至于z的判定，直接选择摄像机的near和far距离就行了，小于near大于far都无视。因为屏幕空间是左下角为0,0，判断是否在方框内要注意这个前提。  
有Unity内置的世界坐标->屏幕坐标转换，其实根本不需要什么复杂的裁剪算法，也不需要搞一大堆参考物体，往外发射一堆射线什么的，一次坐标转换加6个判断条件的一条if语句，就完成全部选择逻辑了，简单高效。

[复制代码](http://blog.csdn.net/xv_ly15/article/details/8563068)

1. void checkSelection(Vector3 start,Vector3 end){
2. Vector3 p1 = Vector3.zero;
3. Vector3 p2 = Vector3.zero;
4. if(start.x>end.x){//这些判断是用来确保p1的xy坐标小于p2的xy坐标，因为画的框不见得就是左下到右上这个方向的
5. p1.x = end.x;
6. p2.x = start.x;
7. }
8. else{
9. p1.x = start.x;
10. p2.x = end.x;
11. }
12. if(start.y>end.y){
13. p1.y = end.y;
14. p2.y = start.y;
15. }
16. else{
17. p1.y = start.y;
18. p2.y = end.y;
19. }
20. foreach(GameObject obj in characters){//把可选择的对象保存在characters数组里
21. Vector3 location = camera.WorldToScreenPoint(obj.transform.position);//把对象的position转换成屏幕坐标
22. if(location.x<p1.x || location.x>p2.x ||location.y<p1.y ||location.y>p2.y
23. ||location.z<camera.nearClipPlane || location.z > camera.farClipPlane)//z方向就用摄像机的设定值，看不见的也不需要选择了
24. {
25. disselecting(obj);//上面的条件是筛选 不在选择范围内的对象，然后进行取消选择操作，比如把物体放到default层，就不显示轮廓线了
26. }
27. else
28. {
29. selecting(obj);//否则就进行选中操作，比如把物体放到画轮廓线的层去
30. }
31. }
32. }

效果如图:  
  
  
注意这种判定是以物体的position点为基准，如果物体中心不在几何中心可能会有奇怪的效果，但对作游戏来说模型中心在几何中心是基本要求，所以应该不是问题。当然也可以按照部分顶点相交来选中，但是那样程序就会复杂很多，要先解构物体的mesh顶点，然后把顶点变换到屏幕坐标，然后判定是否在方框范围内。实际上对大多数游戏来说划过中心点算作选中条件完全可以满足，足够有说服力了。  
  
附上一个自己作的小Demo，综合了之前的显示边缘光、轮廓效果，按钮遮罩效果，加上框选物体的效果。   
鼠标左键按下画框 右键双击地面移动选中的群体 按住滚轮移动鼠标拖动地图，按住右键旋转鼠标转动视角，滚动滚轮缩放视角。  
语音借用的是那啥3的人类语音，你懂的。  
  
选择：  
  
  
双击右键移动目标，目标用一组粒子显示，到达目标或者目标改变时自动消失  
  
  
可以同时移动几组，有点像1代的C&C......  
  
  
点按钮生成新的人物，选中按delete删除人物  
  
  
鼠标划过显示边缘光，单击目标或者画框选中目标显示轮廓线  
  
  
选中的目标会始终对着镜头，未选中的目标会随机行动  
  
  
  
Unity的便利性显现无疑，基本无需复杂的几何、线代知识，也没有繁琐的外围代码，用几句判断语句搞定框选。加上所见即所得的集成开发环境，写到这样程度的Demo实际大概也就花了几个小时的人时，真正框选部分不超过1小时，有点像游戏开发的卡拉OK，优势确实很明显。

Have fun.

http://blog.csdn.net/xv\_ly15/article/details/8563068