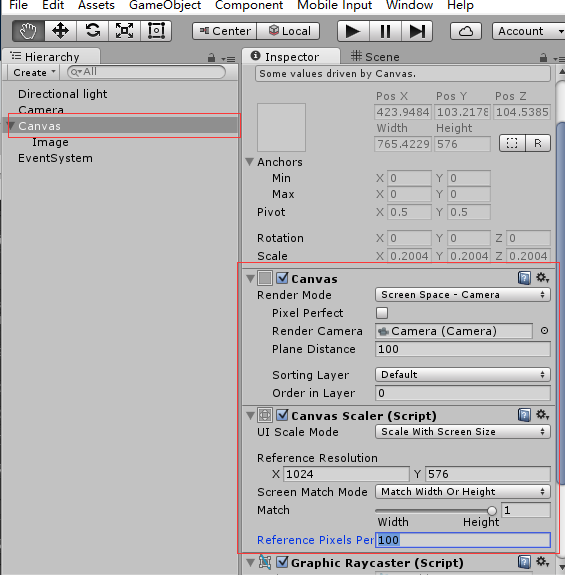
# Unity-UGUI学习之一 Canvas(画布)

1. **什么是画布，作用是什么**

Canvas画布负责UI组件的布局以及渲染，所有的UI组件元素必须作为Canvas的子节点。所以我们所有的UI组件，都是挂在Canvas画布下面。

当我们在新建的Unity工程下面放入UI选项里面任何一个空间，都会默认生成Canvas以及EventSystem。原因就是因为所有的UI组件元素必须作为Canvas的子节点。EventSystem处理UI的事件系统，比如触碰等。

1. **主要属性设置说明**

**为什么Canvas的Rect Transform无法被编辑？**

因为Canvas的大小是根据屏幕定的，你给的屏幕多大Canvas就多大，所以无法手动设置。

**Canvas:**

**render Mode：渲染模式**

1. Screen Space - Overlay：纯2D效果，无需绑定相机，始终显示在屏幕最前方。(不适用，有可能特效什么的需要显示UI前面)，此时Scene View是世界空间的、Canvas是屏幕空间的，两者处于不同空间。
2. Screen Space - Camera：绑定到指定摄像机。

Plane Distance 和相机的距离

Sorting Layer

Order in Layer

1. World Space：3DUI，存在3D空间中的UI,比如可以放人物的血条，称号等。

**Pixel Perfect：是否以像素的方式来显示UI**

将图像尽量和屏幕上的的像素贴合，对于静止的图像／文字会更清晰。如果UI组件是会动的 选这个会使得动画效果变得一顿一顿的（贴合像素的原因），而且因为要额外计算如何贴合所以性能也会有损耗，所以一般不打勾。不过我打钩的时候，也没看到文字变得更清晰~

**Canvas Scaler：**

**UI Scale Mode：UI缩放模式**

1. Constant Pixel Size：像素大小始终不变，即一个100\*100的图片在任何的分辨率下都占用100\*100的像素。移动端项目开发不合适，不同移动端下，分辨率不同，图片需要根据相应规则进行等比例缩放。
2. Scale With Screen Size：不关心图片的实际像素大小，而只关心Width及Height值，Reference Resolution这个值的高度如果是1000，那么100高度的图片在任何分辨率下都只占用屏幕1/10的尺寸。一般移动端会使用这种方式，因为移动端分辨率差异较大。
3. Constant Physical Size：根据物理单位来进行缩放。

**Reference Resolution：参考分辨率，定义制作UI组件时用的分辨率**

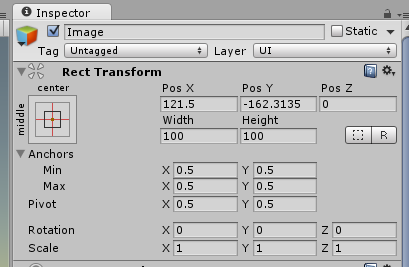
**Screen Match Mode：屏幕匹配模式**

1. Match Width Or Height：根据参考分辨率的高或宽，来缩放UI元素。高度或者宽度比例不变，对UI进行伸缩。
2. Expland：宽度和高度比例不变，分辨率设置不会小于Canvas设置的分辨率，也就是Canvas的Width和Height会刚刚好包围住Canvas设置的分辨率。
3. Shrink：宽度和高度比例不变，分辨率不会大于Canvas设置的分辨率，也就是Canvas的Width和Height会刚刚好被Canvas设置的分辨率包围住。

# Unity-UGUI学习之二 UI组件Rect Transform与Anchor

* 1. **Rect Transform**

Unity3D的基本元件叫GameObject，所有的组件都包含它。而所有的GameObject都必须要携带一个Transform组件，且该组件无法移除，那么作为UI显示的GameObject则不是携带Transform，而是使用Unity3D专门为UI组件设计的Rect Transform组件，如下：



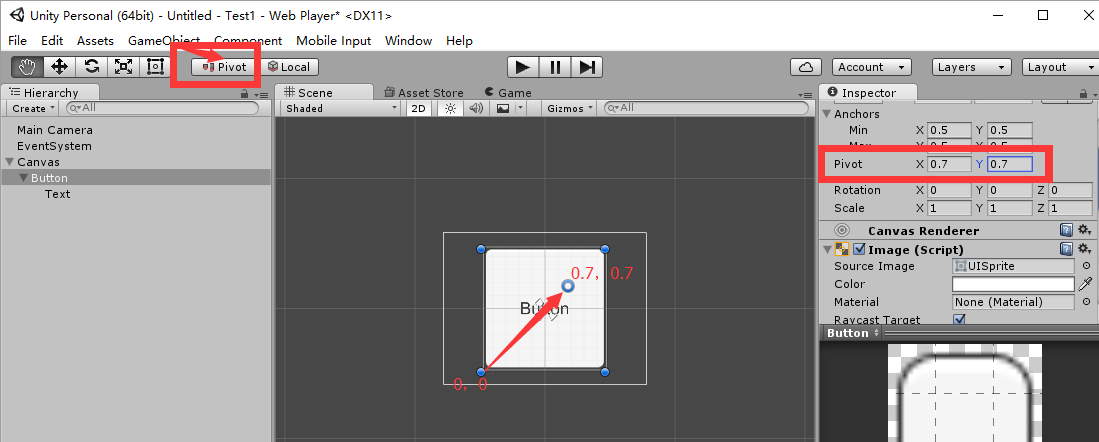
Pos X, Pos Y ，PosZ：以像素为单位，是Pivot和Anchors计算出的数值。负数不显示。

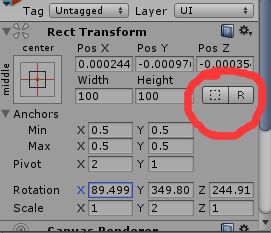
Width,Height: 以像素为单位，是UI的的尺寸大小。

Rotation:旋转。

Scale:缩放，负数会翻转。

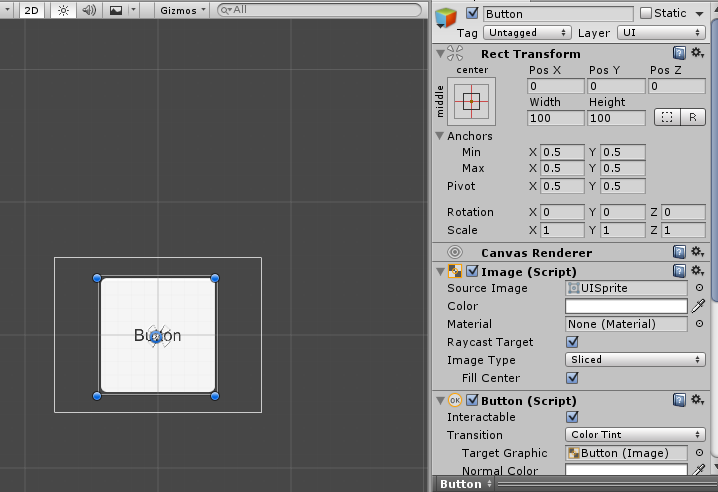
Pivot: 中心点,一个相对于本身左下角（0，0），右上角（1，1）的坐标，切换到Pivot按钮可以见，如下图：



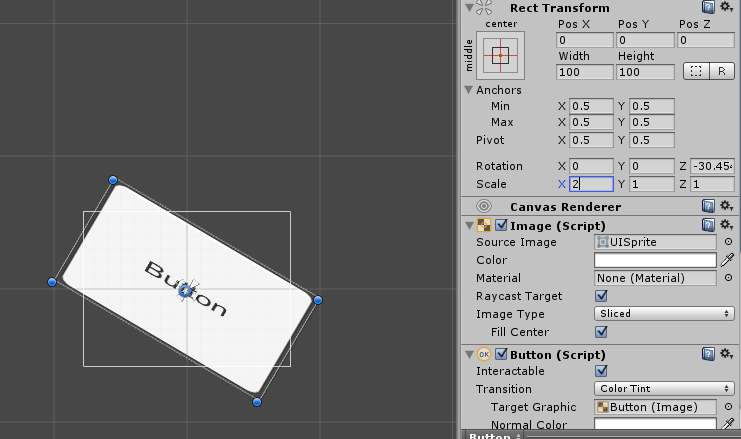


•Blueprint：在此模式下，物体会显示0度和scale等于1的状态，可以对这个零状态进行操作。图如下：

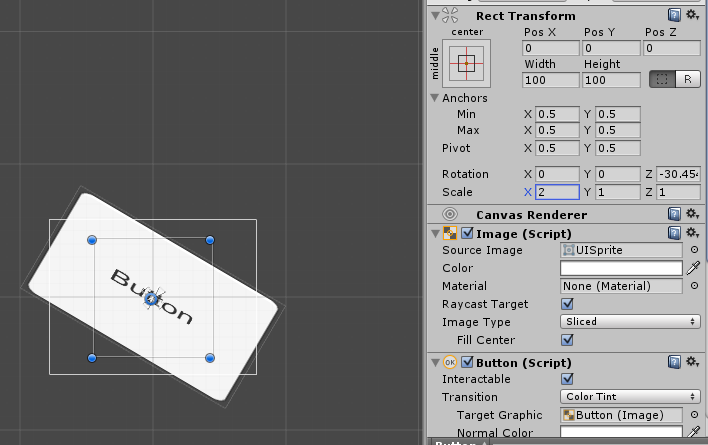
初始图：



旋转并且缩放过：



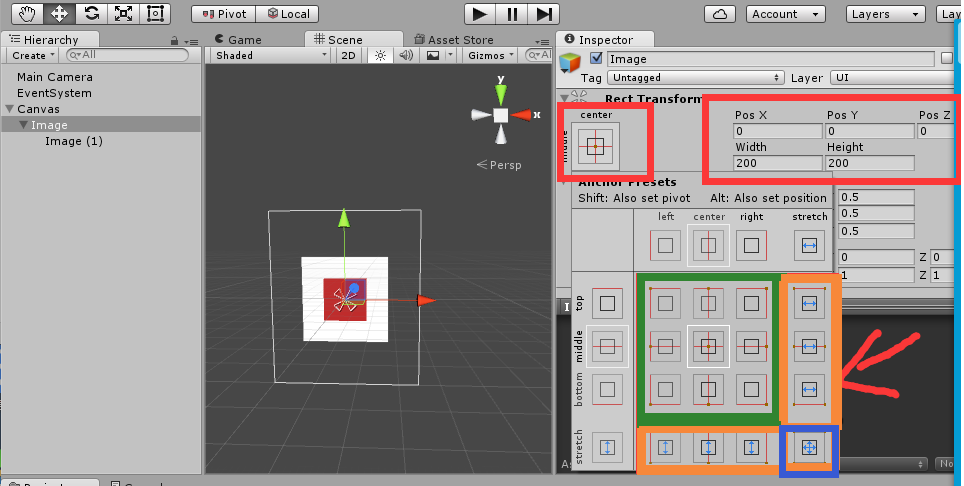
Blueprint状态下：



•Raw Edit：在此模式下，调整物体的Pivot和Anchor不会改变物体的尺寸和物体相对于父节点的位置。

不过经过测试，还是会有一点点的位置变化，应该是unity的bug。

* 1. **Anchors锚点，锚线，锚框**



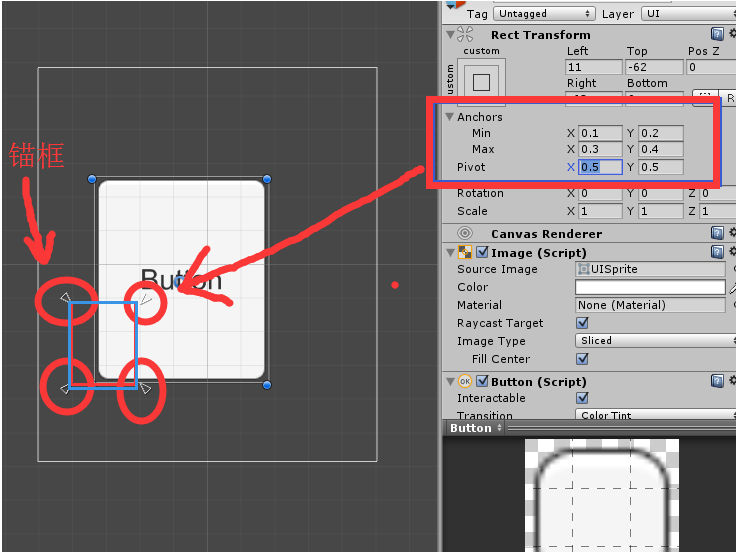
Anchors和Pivot对应，Pivot是对相对于组件本身的一个坐标点。而Anchors是对于组件的Parent的一个Anchors。

锚点：绿色框中，两条相交直线，产生一个点。

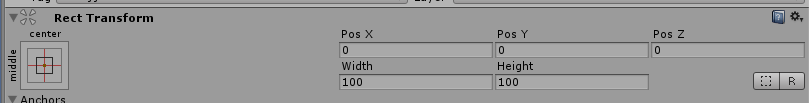
锚线：橙色框中，一条直线。

锚框：蓝色框中，两条平行直线组成的一个矩形。 虽然图上没画两条直线出来。

可以看四个点（MinX, MinY）, (MinX, MaxY）, (MaxX, MinY）, (MaxX, MaxY），因为0.1，0.2，0.3，0.4是我们定制的，所以16种图案里面找不到相匹配的，就会显示空白无红线。



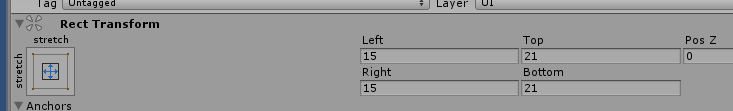
其中锚点属于**非stretch**，也就是不会随着parent的拉伸而拉伸。



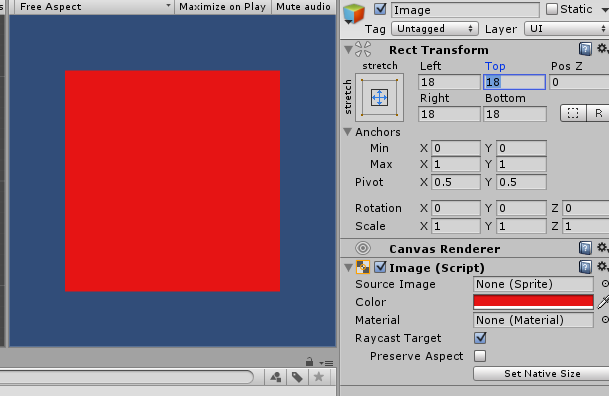
这五个数值上面已经解释过了。

锚边，锚框都属于**stretch**，也就是会被拉伸，并且Rect Transform的属性名称会变化，分两种如下：

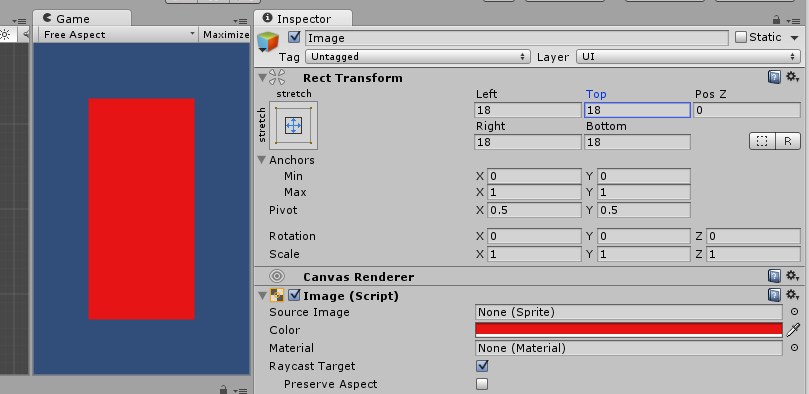
一种是：



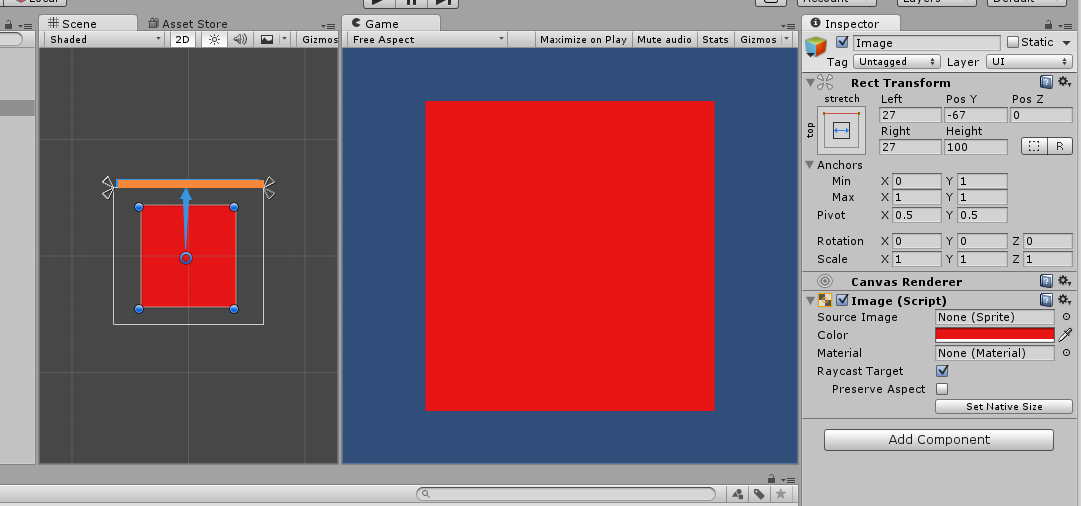
Pos X、Pos Y、Width和Height改变为Left、Top、Right及Bottom，也就是组件的四条边相对于组件Anchors锚框的四条边的绝对位置。



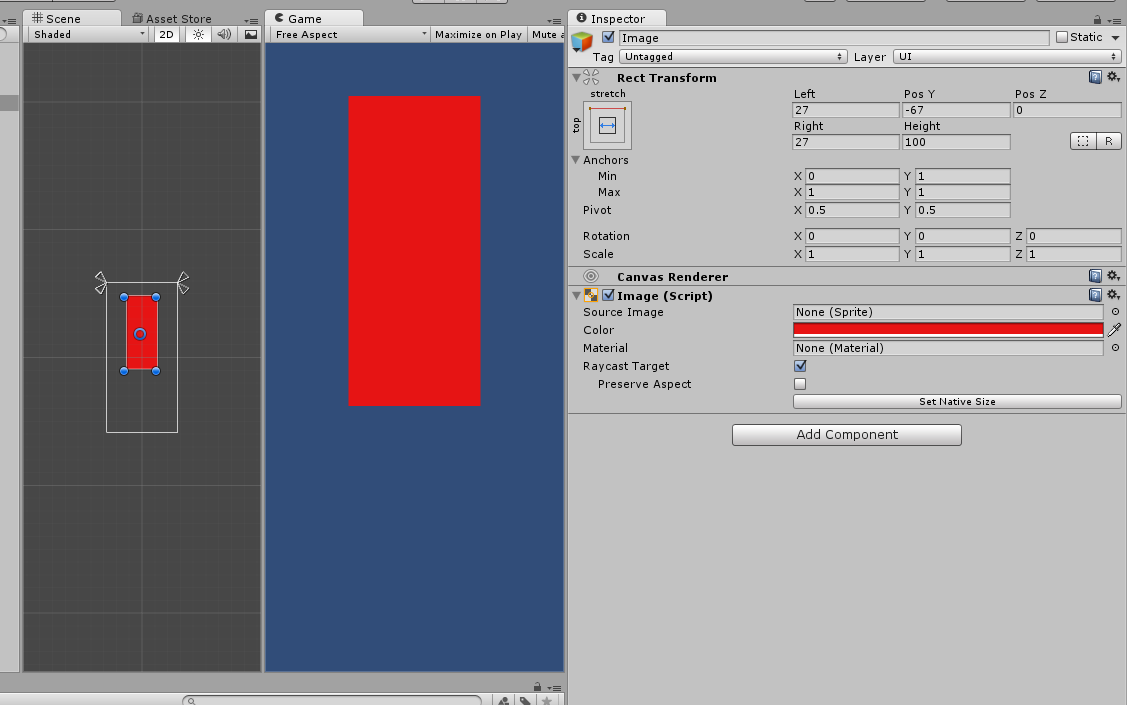
左右上下都是距离Anchors锚框18像素。这时候我拉扯parent，等到左右上下都是距离Anchors锚框依然是18像素，得到如下图示：



还有一种是:

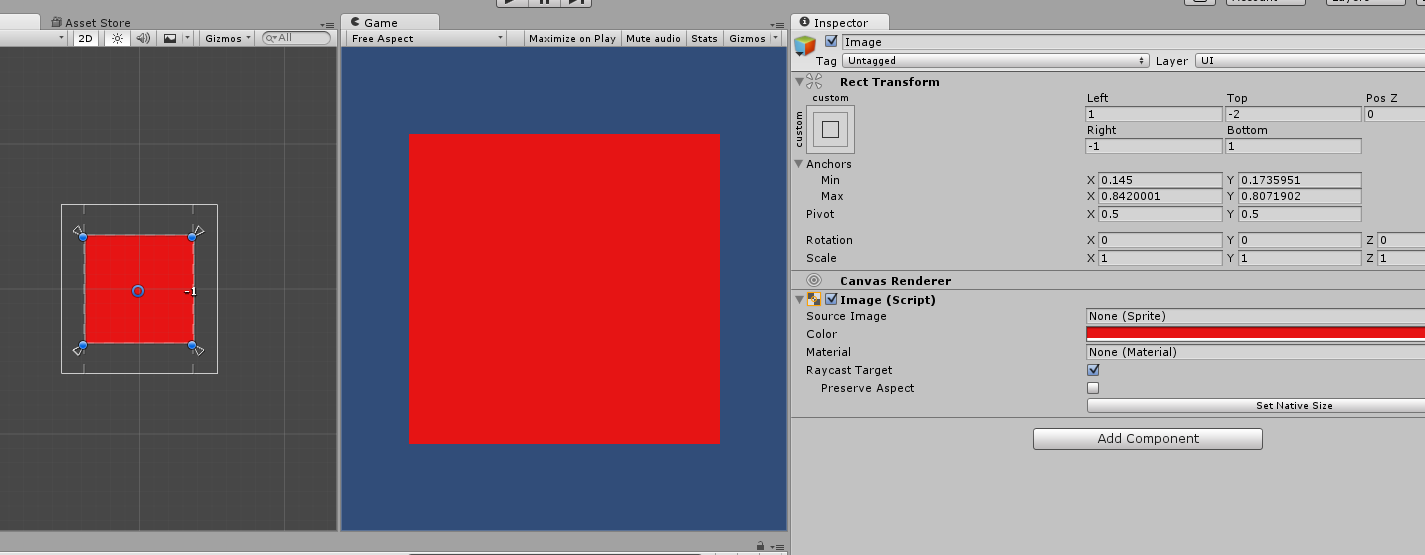


Pos X、Pos Y、Width和Height和Left、Top、Right及Bottom交叉在一起的。比如上图，那么PosY的值，就是锚线到Pivot的距离，这一段距离不会因为拉伸而改变，Heigh定义了组件的长度不会被改变。

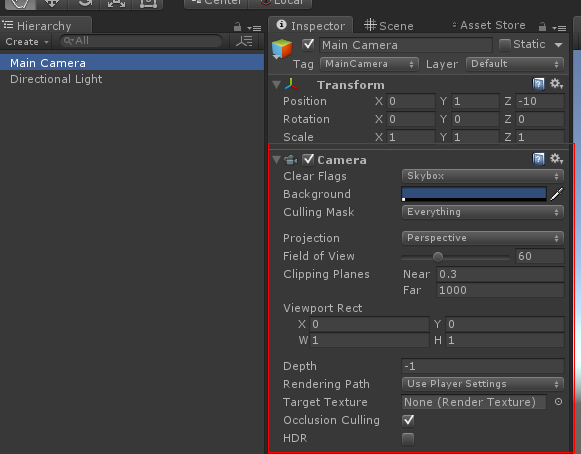


最后讲下自定义的Anchors。比如我们想设置某个UI的尺寸始终和Parent UI的尺寸保持一定的比例，比如无论Parent UI多大，这个子UI都占其Parent面板的比例不变。

很简单，只要把Anchors的四个点拖成这个UI大小即可，这样就可以和Parent等比例了。



# Unity-UGUI学习之三 Camera(相机)



**1. Clear Flags:清除标记。**

**Skybox**：天空盒，默认模式。在屏幕中的空白部分将显示当前摄像机的天空盒。如果当前摄像机没有设置天空盒，会默认用Background色。

**Solid Color**：该模式下，屏幕上的空白部分将显示当前摄像机的background色。

**Depth only**：仅清除深度缓存。该模式用于游戏对象不希望被裁剪的情况。那么不希望被裁剪UI的Depth要比之前显示的UI高。比如 A相机深度最大，并且设置Depth only。还设置了Skybox清除标识的相机B，B的深度低于A，那么B先被绘制出来，接着A因为消除了深度缓存，那么在A前面绘制出来的那个的深度将会被忽略，和B出现在一个界面上。也就是说A不透明的部分将被绘制完全覆盖任何已绘制的事物。

**Dont Clear**：不清除。该模式不清除任何颜色或深度缓存。其结果是，每一帧渲染的结果叠加在下一帧之上,造成涂片效果。这不是用于游戏的典型方式。

**2. Background：背景。**

设置背景颜色。在镜头中的所有元素渲染完成且没有指定skybox的情况下，将设置的颜色应用到屏幕的空白处。

**3. Culling Mask：剔除遮罩。**

选择所要显示的layer。每个组件都有一个Layer的标识。

**4.Projection 投射，投影。**

**正交Orthographic(无消失点投影)**

因为它不会根据距离收缩。所以如果你如果你画一个固定大小的物体在视点前面，同时画一个同样大小的物体在第一个物体的远后方，你无法说那个物体是第一个。因为两个都是一样的大小，根距离无关。他们不会随着距离而收缩。做2D界面的比较合适

**透视Perspective(有消失点投影)**

远小近大，高个子站在100米以外，他甚至还没有你的拇指大。他看上去会随着距离而缩小，但是我们实际上都知道，它依然是个高个子。这种效果叫做透视。如果UI想做成3D效果，可以使用这种模式。

**5. Clipping Planes： 剪裁平面。**

摄像机开始渲染与停止渲染之间的距离。

**Near：近点。**摄像机开始渲染的最近的点。

**Far：远点。**摄像机开始渲染的最远的点。

这个要和挂了camera的canvas联系在一起。canvas的plane distance是离camera的距离，plane distance在Near到Far之间，UI才会被渲染到。

**6.ViewPort rect标准视图矩形。**

用四个数值来控制摄像机的视图将绘制在屏幕的位置和大小，使用的是屏幕坐标系，数值在0~1之间。坐标系原点在左下角。

X,Y表示屏幕可视的位置 W,H表示这个屏幕可视的大小，比如x = y =0.25.w = h = 0.5,则屏幕可视区域就变成之前1/4，处于中间。

**7. Depth：深度。**

用于控制摄像机的渲染顺序，较大值的摄像机将被渲染在较小值的摄像机之上。

**8. Rendering Path：渲染路径。**

用于指定摄像机的渲染方法。

**Use Player Settings：**使用Project Settings-->Player中的设置。

**Vertex Lit：**顶点光照（Vertex Lit） 是最低保真度的光照、不支持实时阴影的渲染路径。最好是用于旧机器或受限制的移动平台上。

**Forward：**正向渲染。正向渲染一个基于着色器的渲染路径。它支持逐像素计算光照（包括法线贴图和灯光Cookies）和来自一个平行光的实时阴影。

**Deferred Lighting**：延迟光照。它需要一定水平的硬件支持，所以一般不会选择这个。

**9. 高动态范围（HDR）图像。**

HDR使颜色更鲜明，像素更清晰。使用浮点缓冲区（渲染很慢并且需要更多显存），不是所有的硬件都支持。所以使用默认关闭状态

**10. Target Texture: 目标纹理。**

用于将摄像机视图输出并渲染到屏幕。可以创建一个渲染纹理应用给相机，然后相机视图渲染到RenderTexture，可以用作视频监控。

**11.**[**Occlusion Culling**](http://blog.csdn.net/pizi0475/article/details/12883401)**：遮挡剔除。**

当一个物体被其他物体遮挡住而不在摄像机的可视范围内时不对其进行渲染。

# Unity-UGUI学习之四 屏幕自适应与UI制作讨论

**2. 覆盖住整个屏幕的背景图做法讨论：**

（1）裁剪做法 比较推荐的做法是做一张较大的背景图，背景图上下左右边的内容允许被裁剪。

（2）伸缩做法 根据当前机型的分辨率，对背景图进行自适应伸缩（脚本ScreenAdaptateBackground）

（3）其他做法

**3.简单的UI动画表现**

比如UI匀速，变速的直线移动，可以由程序控制，只要在UI脚本指定一个开始位置，和一个结束位置即可，程序控制UI的位移。

**1.市场调查：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 分辨率 | 宽/高 比 | 系统 | 市场占有比例 |  |
| 960\*640 | 1.5 | IOS | iPhone5(s)：分辨率1136\*640，高宽比1.775  iPhone6：分辨率1334\*750，高宽比1.778  iPhone6+：分辨率1920\*1080，高宽比1.778  iPad分辨率1024\*768  高宽比1.333 | 主流的是1.778  16:9 |
| 1136\*640 | 1.775 |
| 1024\*768 | 1.333 |
| 2048\*1536 | 1.333 |
| 2732\*2048 | 1.333 |
| 1136\*640 | 1.775 |
| 1334\*750 | 1.778 |
| 1920\*1080 | 1.778 |
| 480\*320 | 1.5 |
| 800\*480 | 1.667 | Android | 11% | 1.778的占有率高达76%  16:9 |
| 854\*480 | 1.778 | 15% |
| 1280\*720 | 1.778 | 39% |
| 1280\*800 | 1.6 | <3% |
| 960\*640 | 1.5 | 8% |
| 960\*540 | 1.778 | <3% |
| 1184\*720 | 1.644 | <3% |
| 1920\*1080 | 1.778 | 22% |
| 其他 | 其他 | <10% |

**2.我们选择多少分辨率 进行设计。**

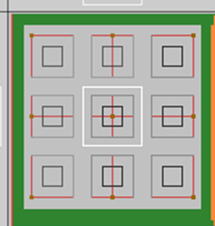
从手机屏幕比例来看，标准16:9比例的屏幕最受用户欢迎，2015-2016年度其关注比例达到87.91%，接近九成，成为用户关注的绝对主流。15:9与16:10比例屏幕的手机关注比例在一成左右。

也就是说16:9的比例几乎占据了所有人的心，所以我们选用的比例会是16:9（1024\*576），这样就算是2K屏幕，看起来也不会太模糊，资源也不会太大。

**3.我们怎么进行屏幕进行适应（方案/算法）**

**不同分辨率，各个子组件布局方式：**

UI组件的布局根据UIGUI那一套，四边中点的停靠，四个角落顶点停靠，一直处在中间的（除了这九个，锚点位置还可以自己设定，原理和这个一样，就是锚点到pivot的距离不变）



只要用上这个机制，UGUI都会很好的根据组件的Parent来进行组件停靠。这个机制让UI组件在各个不同分辨率下能按照设计者的规定有序的布置UI组件。

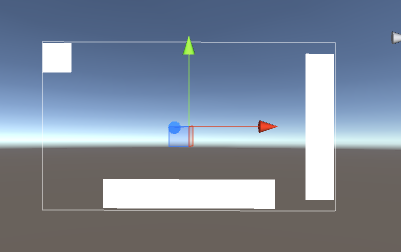
**屏幕自适应原理：**

因为移动端分辨率差异较大，所以我们的Canvas画布的设置如下：

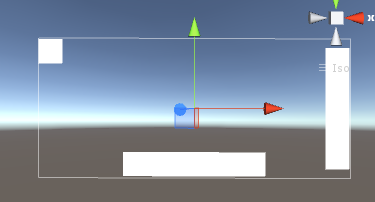
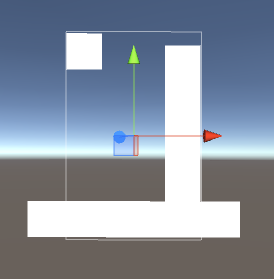
Scale With Screen Size：不关心图片的实际像素大小，而只关心Width及Height值，Reference Resolution这个值的高度如果是1000，那么100高度的图片在任何分辨率下都只占用屏幕1/10的尺寸。

Match Width Or Height：根据参考分辨率的高或宽，来缩放UI元素。高度或者宽度比例不变，对UI进行伸缩。

正常分辨率下：

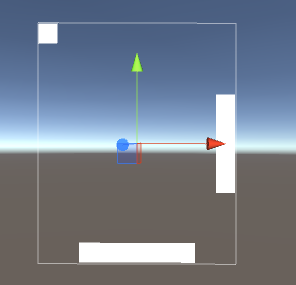


如果我们只选取高度自适应，Match Width Or Height.Match = 1

，

因为UI只知道我要进行高度自适应，不知道宽度是否会失去视野。所以加了个脚本，可以根据手机分辨率和定义的默认分辨率1024\*576计算出Match Width Or Height.Match等于0 或者等于 1。

这样就能控制宽度或者高度来进行自适应，能完美的把整个UI组件都显示出来，不会出现丢失现象。如图：

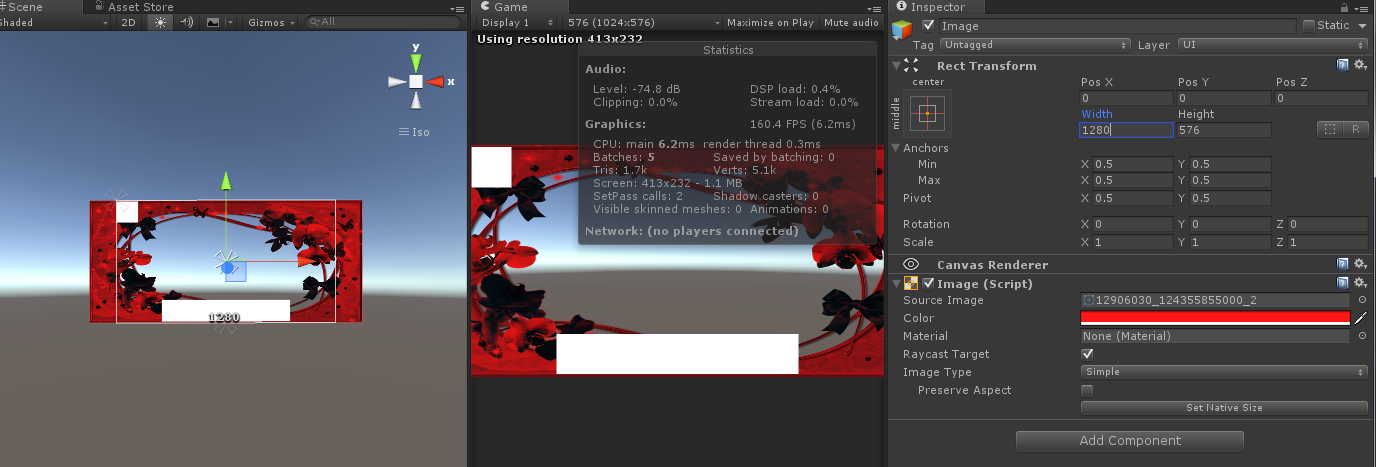


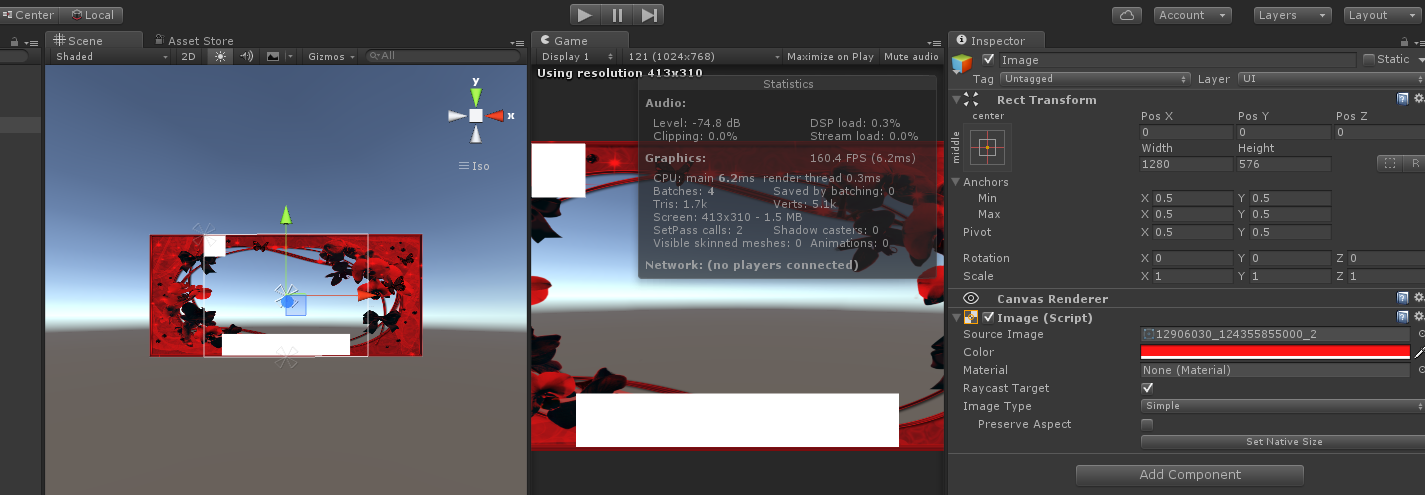
上面的两个点解决了UI布局自适应，这时候还有一个问题，就是战斗背景图如何做到充满整个画布。

**不同分辨率，背景框充满整个画布：**

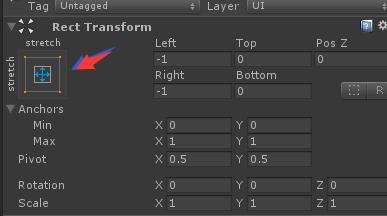
可以根据需求合理使用下面两种做法

1.裁剪做法做法是做一张长度比较长的背景图，背景图左右边的内容允许被裁剪。大概是1092\*576（防止有手机的比例大于16:9=1.778，听说有的机子的比例是1.8多的），这样就可以了，观察下面两张图：





2.伸缩做法 ，只要这个就可以了，当然你的背景图可以配合九宫格来做。



**4.各个分辨率下使用这套方案下的效果。**

|  |  |  |
| --- | --- | --- |
| 分辨率 | 效果 | 能/否 接受 |
| 4K | 标清 | 极为少数的手机，能接受。 |
| 2K | 高清 | 能 |
| 1K | 超清 | 能 |
| 1K以下 | 超清 | 能 |