计算机系科协暑培 5.0

Unity UI 与材质

王溢

清华大学计算机科学与技术系

2024 年 8 月 11 日

UI(User Interface) 是 Unity 中非常重要的一个概念，其主要是指显示给用户的一些界面，常见的有按钮、图片、文本、下拉框、文本输入框、滑动条等等

图 1: 某二字游戏中的一些可交互 UI

与 GameObject 的不同点

UI 与 GameObject 看起来非常相似，但是其实二者在逻辑、行为、控制等方面都有很大的不同点。

以《原神》为例，其中的 GameObject 包括但不限于人物、地图、怪物等等，而 UI 则是各种各样的按钮、血条、物品信息、角色对话框等等。

可以简单理解为，GameObject 是整个游戏世界的组成部分，UI 是为了玩家更轻松的进行游戏而人为添加的部分。

UI 都是二维的

UI 的坐标系统

在 UI 的世界中，我们使用一个名为 Rect Transfrom(矩形 Transfrom) 的组件来确定一个 UI 的坐标。每一个 UI 都对应一个矩形框，而这个坐标是相对于其父 UI 的矩形框来确定的。所有的 UI 都有一个 Canvas 作为其祖先 UI (后面会详细讲)

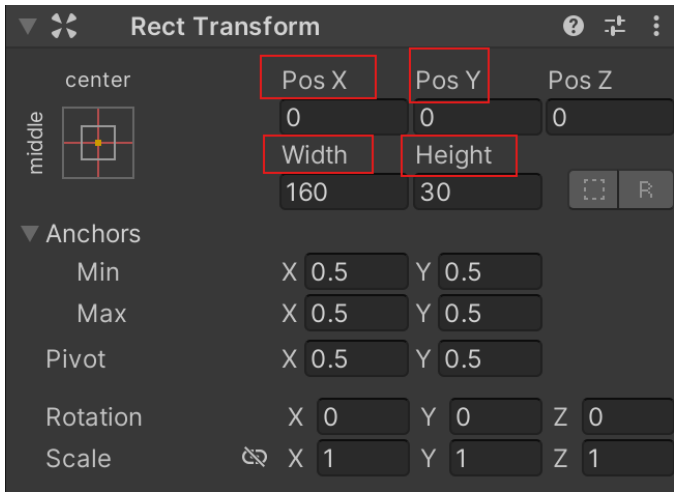


图 2: Rect Transform

锚点和中心点

UI 的坐标是由它和它的父 UI 二者的矩形框的相对值来确定的，因此我们需要使用中心点 (pivot)和锚点 (anchor)来确定“相对”的依据。

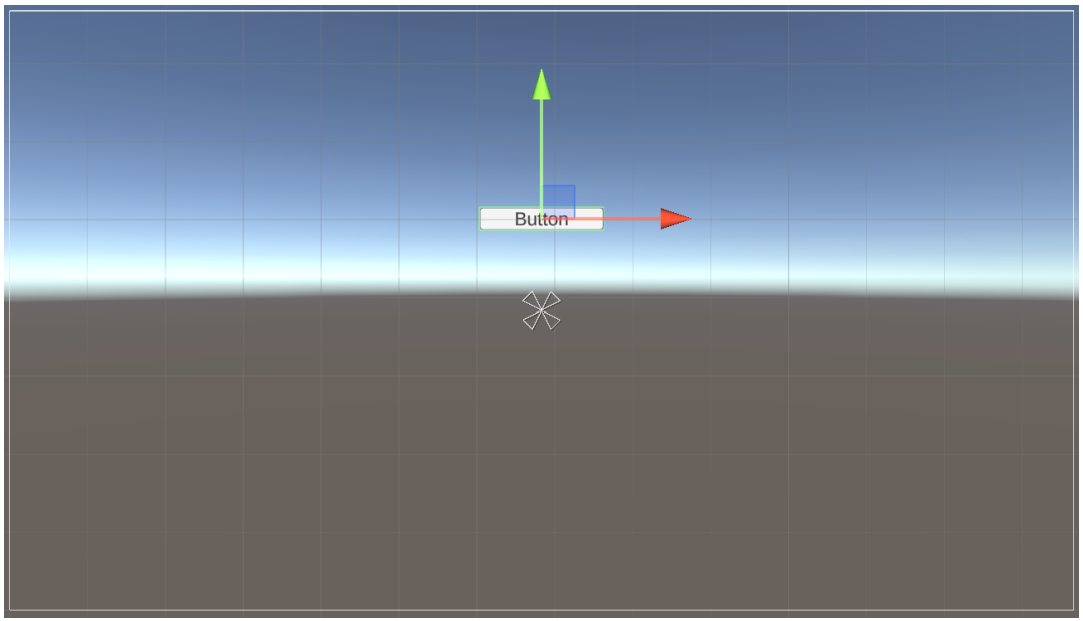


图 3: 锚点与中心点示意图

中心点

中心点相对来说是一个比较简单的概念，其是指我们希望以这个 UI 矩形框的何处为中心，左下角为 $(0,0)$，右上角为 $(1,1)$，按照比例进行缩放

当我们在 Hierarchy 中双击一个 UI 的时候会在其中心点处生成一个坐标轴

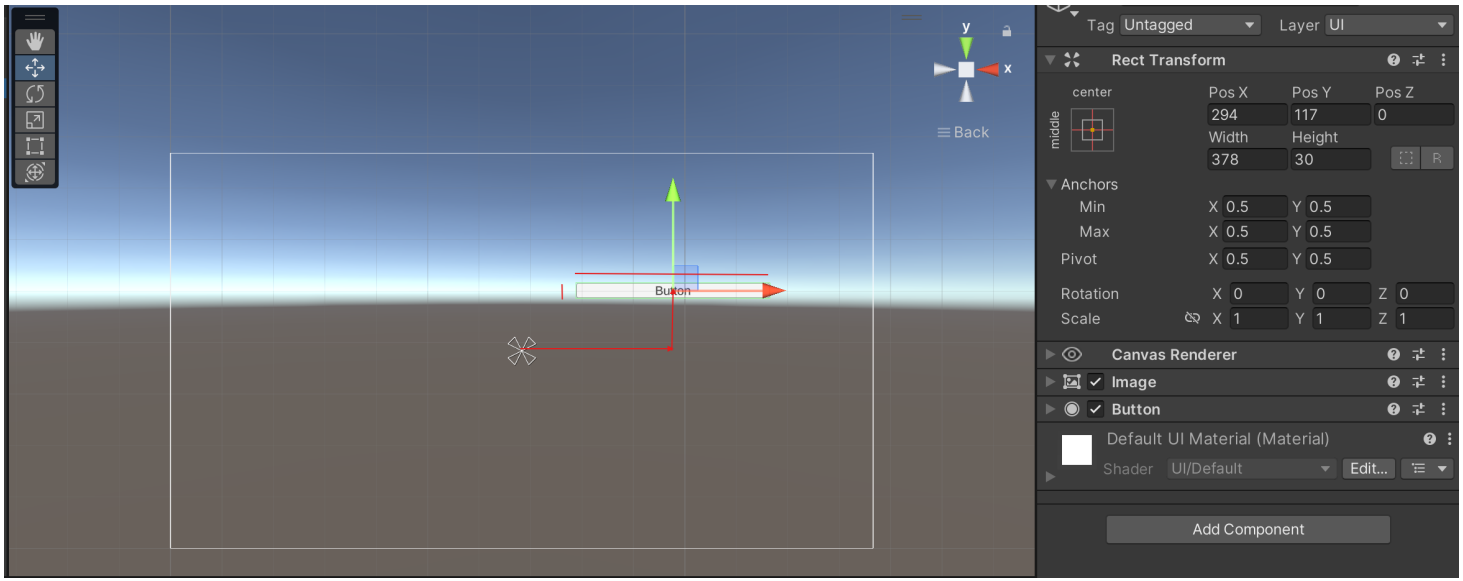
锚点

一个 UI 的锚点是用于根据其父 UI 来确定其自身的矩形框的一组点，父 UI 的左下角为 $(0,0)$，右上角为 $(1,1)$，锚点可以在 Inspector 中快速设置

按照这四个 ▽ 的重合性，UI 绘制方式有所不同：

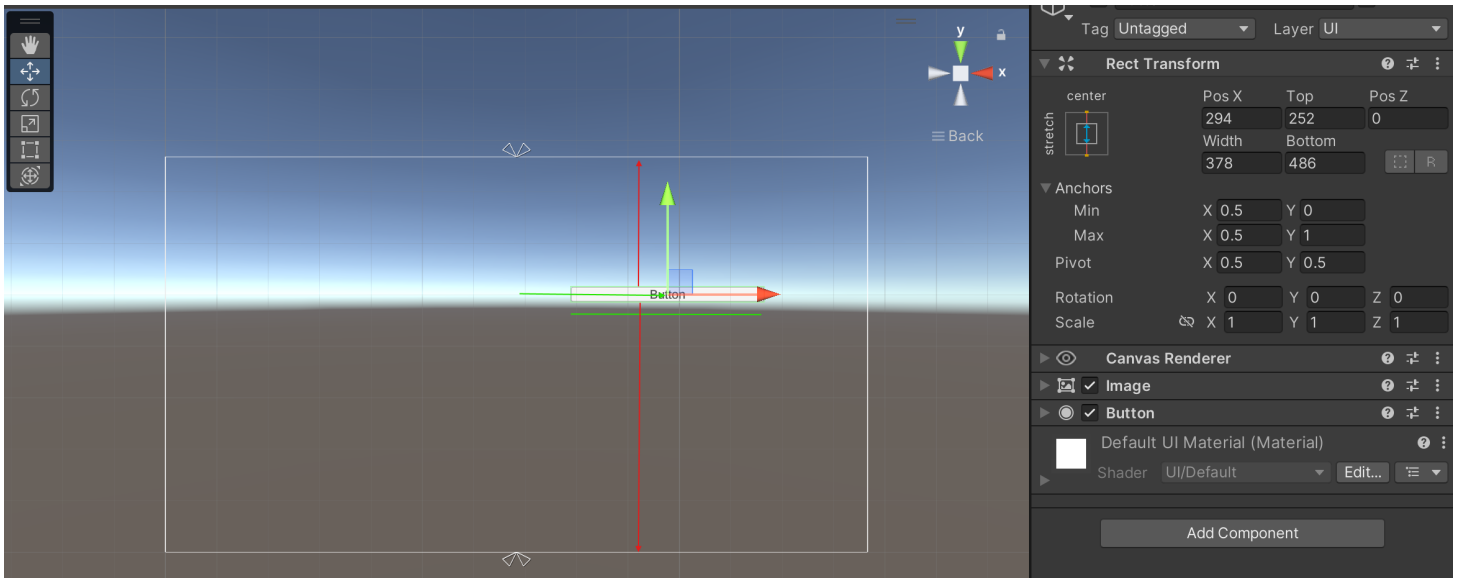
- 当四者重合在一点的时候，使用该点与子 UI 的中心点的距离、子组件的宽高来确定
- 当四者重合在一个方向上的时候：
  - 锚点重合的方向，利用与子 UI 的中心点的距离以及这个方向上的长度来确定
  - 锚点不重合的方向，利用子 UI 在对应方向上边界与父 UI 对应方向上边界的距离确定
- 当四者都不重合的时候：利用四个方向上子 UI 与父 UI 边界的距离差来确定

锚点全部重合



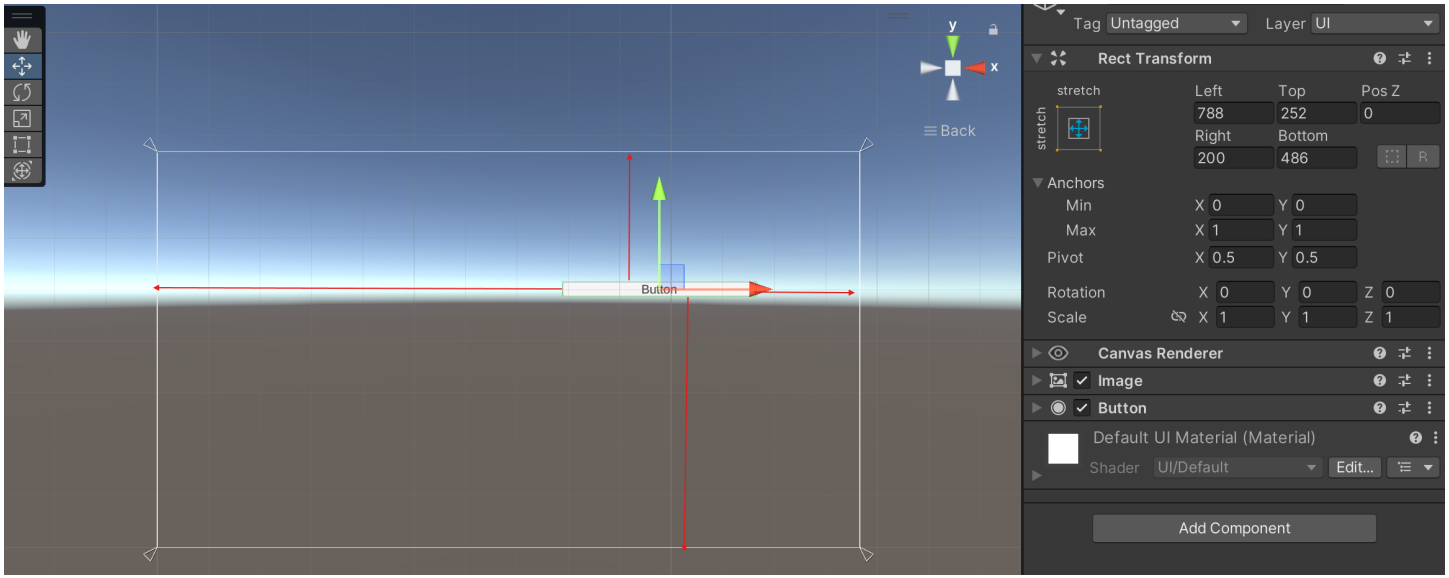
如上图，当四个锚点重合的时候，我们利用的是锚点与中心点之间的距离与这个子 UI 的宽高来确定矩形框，也即 Inspector 中的posX，posY，Width，Height 来确定

锚点在一个方向重合



如上图，当四个锚点在水平重合的时候，利用的是到中心点到水平方向的距离posX、水平方向长度Width、子 UI 上下边界与锚点线距离Top 与Bottom

锚点全都不重合



如上图，当四个锚点都不重合的时候，我们利用的是子 UI 之间四个边界与锚点线之间的距离Top，Bottom，Left，Right 来确定子 UI

常用 API

下面介绍一些常用的 API，在代码中的访问方式为`obj.GetComponent<RectTransform>().xxx`

- `anchorMin/anchorMax`: 对应的锚点值
- `offsetMin/offsetMax`: 矩形框的左下/右上角相对左下/右上锚点的偏移量
- `anchorPosition`: 支点相对于锚点的位置
- `rect`: 该 RectTransform 对应的矩形，包括左下角相对支点的位置、宽高，只读属性
- `sizeDelta`: 锚点重合时即为该方向长度，不重合时为该方向上矩形与其父 RectTransform 的长度差

为什么要这么麻烦

为什么这么麻烦呢？一定要用相对位置吗？

一定！因为 UI 是呈现给用户看的，而不同的用户其设备可能不同，而且即使是相同的设备也会存在不同的显示方式，例如全屏、横屏等，因此我们的实现需要有足够的适配性，保证在各种分辨率的设备下能够呈现出相同的效果，不然非常容易出现换个屏幕 UI 就突然消失了……

Canvas 与 EventSystem

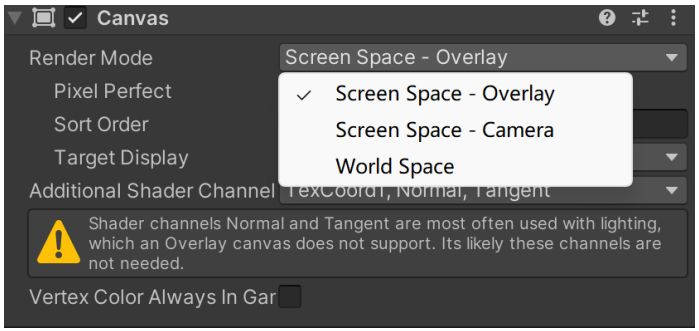
Canvas(画布) 与 EventSystem(事件系统) 是 UI 中最基础但是最重要的两个对象：

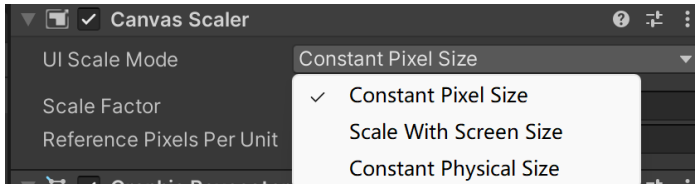
- Canvas 是 UI 显示的基础，所有的 UI 都必须是某一个 Canvas 的子对象才能显示在屏幕上
- EventSystem 是 UI 控制的基础，所有与 UI 有关的事件（例如点击、拖动、滑动等）都是由它控制，一个场景应当只有一个 EventSystem

创建普通 UI 对象的时候会自动创建一个 Canvas 与 EventSystem

Canvas 的渲染

Canvas 的渲染主要有两个组件 Canvas 与 Canvas Scaler 来控制





Canvas 渲染顺序

通常情况下，游戏中的 UI 之间不可避免的会发生重叠，这个时候 Unity 需要一个标准来决定重叠时候的渲染顺序：

1. 渲染模式 Overlay > Camera > World Space
2. Overlay 模式下，Sort Order 大者优先
3. Camera 模式下，相机 Depth > 相机 Sorting Layer > Order in Layer
4. World Space 模式下，UI 以 3D 物体的形式呈现出来
5. 同一 Canvas 下，在 Hierarchy 中越靠后越上层，可以在代码中通过`setSiblingIndex` 来修改

动态组件——以 Button 为例

动态 UI 是指可以与之交互的 UI，交互的逻辑是：

1. 用户的行为会被 EventSystem 捕捉到，并且传递给逻辑层
2. 逻辑层调用对应的事件，执行用户规定的一系列操作

注意到 UI 并没有像 GameObject 那样的`OnMouseDown` 一类的函数，这个时候应该怎么办呢？

动态组件——以 Button 为例

以 Button 为例，我们期待用户点击按钮的时候做到一些炫酷的操作~~抽卡！~~，只需要：

1. 创建一个脚本，实现所需要的功能，并设置成 public
2. 创建一个空物体，并添加脚本作为组件
3. 在 Button 组件中，向 OnClick 事件中添加该物体，并选取对应函数

当然也可以在代码中通过`button.onClick.AddListener` 来添加

其它动态 UI

常用的动态 UI 及其监听的值还有：

- Toggle：勾选框，监听是否勾选
- Slider：滑动条，监听滑块的值，通常用于确定物品的值
- Scrollbar：类似 Slider，通常用于滑动显示
- Scroll View：可滑动区域，监听滑到的区域
- Dropdown：复选框，监听选择项
- Input Field：输入框，监听输入值与焦点变化

当然，开发者可以通过向一个部件添加 EventTrigger，来自定义其监听的事件

常用静态 UI

常用的静态 UI 包括：

- Image：显示 Sprite，显示选项更多，原理复杂
- Raw Image：直接显示图片原始像素，性能更优
- Text：显示文本内容
- Panel：弹窗（本质就是类型为 Background 的 Image）通常用于背景、进度条
- TextMeshPro(TMP)：终极解决方案！提供了更加强大的渲染方式，但是可能会有兼容性问题

让静态 UI 动起来

如果想要让静态 UI 也去交互怎么办？例如就需要点击一张图片的时候也可以抽卡

往其中加入 Button 组件！

UI 本质上也就是一系列组件的堆叠而已，Rect Transform 决定了 UI 的详细位置信息，其他组件共同决定要往这块区域画什么、怎么交互等等

Unity 中物体的渲染

Unity 以网格的形式渲染所有物件，包括物体与 UI 等等，每一个物体都由大量的面片拼接而成

每一个面片的具体信息被封装在Mesh 类中，物体会通过 MeshFilter 组件（决定形状）来从资源库中读取对应的 Mesh，并交给 MeshReander 组件（决定外观）将其渲染到屏幕上

有关过滤器和渲染器的详细信息请参阅官网

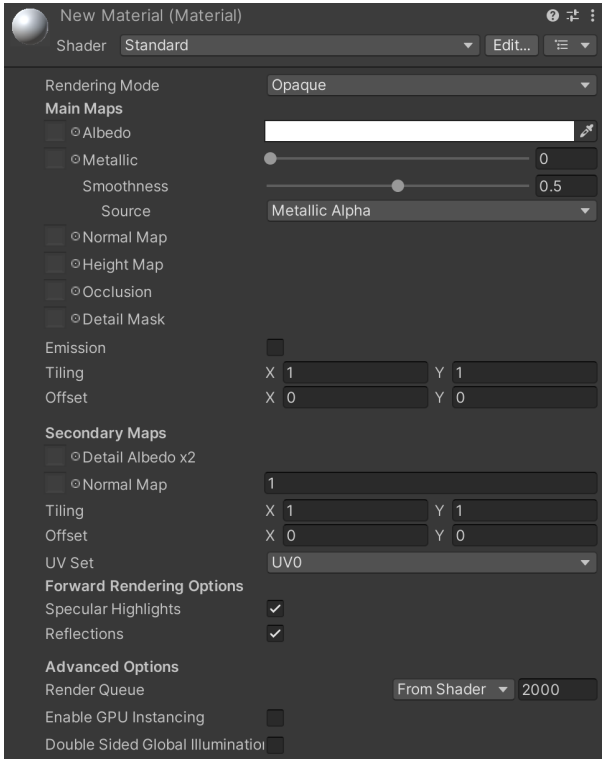
材质球的构成

材质球决定了网格最终渲染出来的状态，由贴图 Texture 与着色器 Shader 二者构成

- Shader 非常复杂！本质是一段程序，拿到所有的输入之后负责通过一系列的手段将整个物件尽可能真实的渲染在屏幕上，不建议在没有计算机图形学前置知识的情况下学习
- Texture 是开发者希望贴上去的图片，后面会细讲

每一个新建的物体都会带有一个默认材质 Default Material，其不能被修改，只能被替换

材质球的属性



- Shader: 选定着色器，Unity 内置了许多种类的着色器，不同类型的着色器可供修改的属性有所不同
- Rendering Mode: 渲染模式，分为不透明、裁剪、渐变与透明

UV 坐标

Unity 中采用的是 UV 坐标贴图的方式，每个物体会有一组 UV 坐标值，被封装在 Mesh 类中，可以通过`mesh.uv` 来访问或修改

其大致的思路是：

- 对于一张二维贴图，将其放在 UV 坐标系中，从左下顺时针开始分别为 $(0,0),(0,1),(1,1),(1,0)$
- 对于一个多边形面片，这个面片每一个顶点在 UV 坐标系中有一个坐标映射
- 通过这些映射可以在贴图上确定唯一的一块区域，这一块区域中的像素通过仿射变换后就得到贴上去的图片

UV 坐标

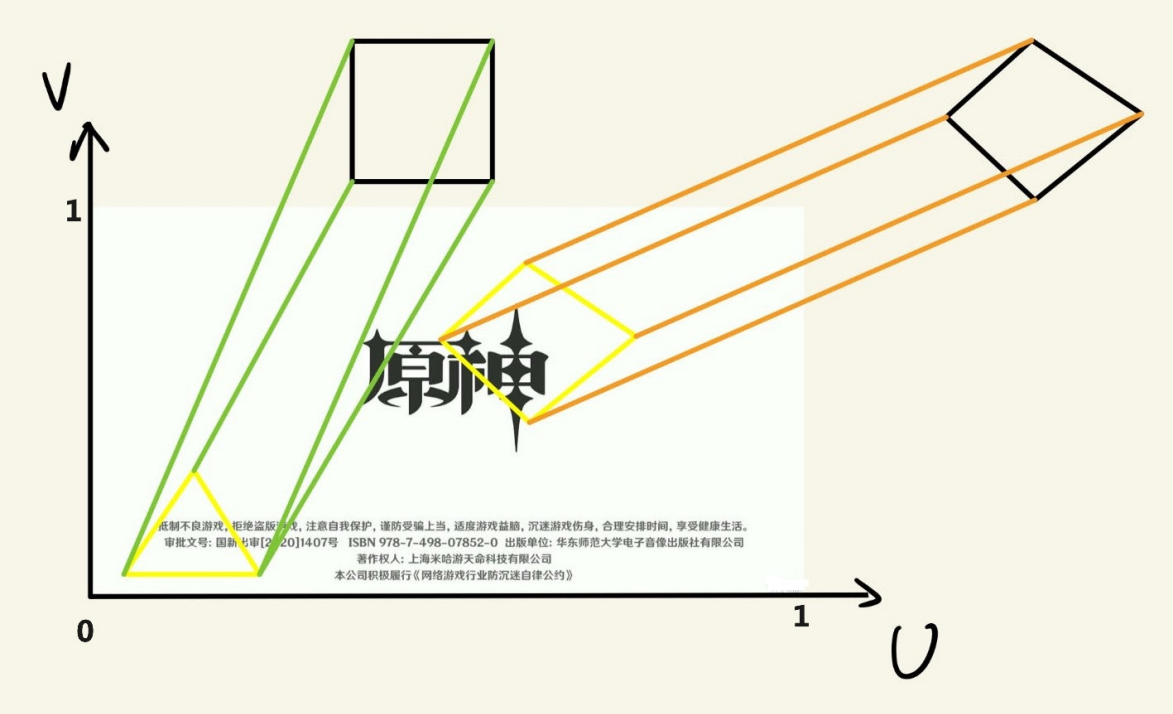


图 4: UV 映射实例

标准着色器下的贴图

分为主贴图与次要贴图两类，简单开发只需要主贴图，次要贴图一般用于添加细节、进行精细控制等，下面介绍主贴图

- Albedo(反照率): 控制材质的基色与纹理贴图，常用于设置整体的外观，当基色为白色的时候不会与贴图进行叠加
- Metalic(金属率): 可以调节材质球的金属性与光滑度
- Normal(法向): 用于增加模型的细节，例如墙面凹凸不平
- Height(高度): 常用于实现一片具有高度差的地形、提升法向贴图的效果
- Occlusion(遮挡): 用于实现阴影效果，确定每一部分应当接受的环境光的强度

Thanks!