Camera in Creator 2.1.2

这篇文章主要介绍如何简单的使用多个摄像机来渲染一个完整3D工程，开发者可以先参考Creator的摄像机说明文档，熟悉Camera的使用方法。

https://docs.cocos.com/creator/2.1/manual/en/render/camera.html

展示效果

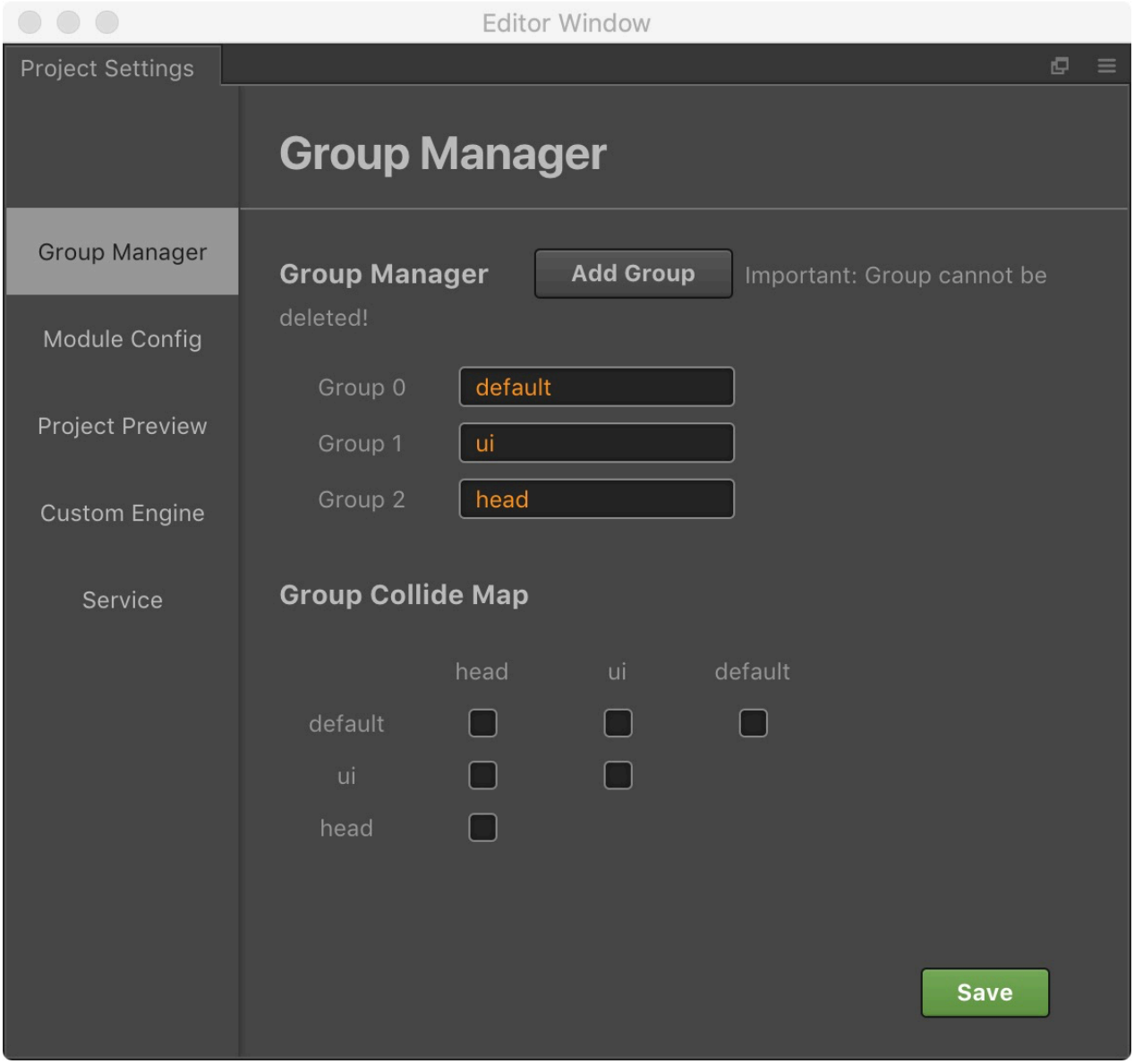
教程提供了非常简单的demo，仅仅实现了下面内容。

1. 3D人物的自由视角控制。
2. 3D人物在UI中的显示。
3. UI界面的显示。

摄像机分组

使用分组来标志场景对象该被哪个摄像机进行渲染。

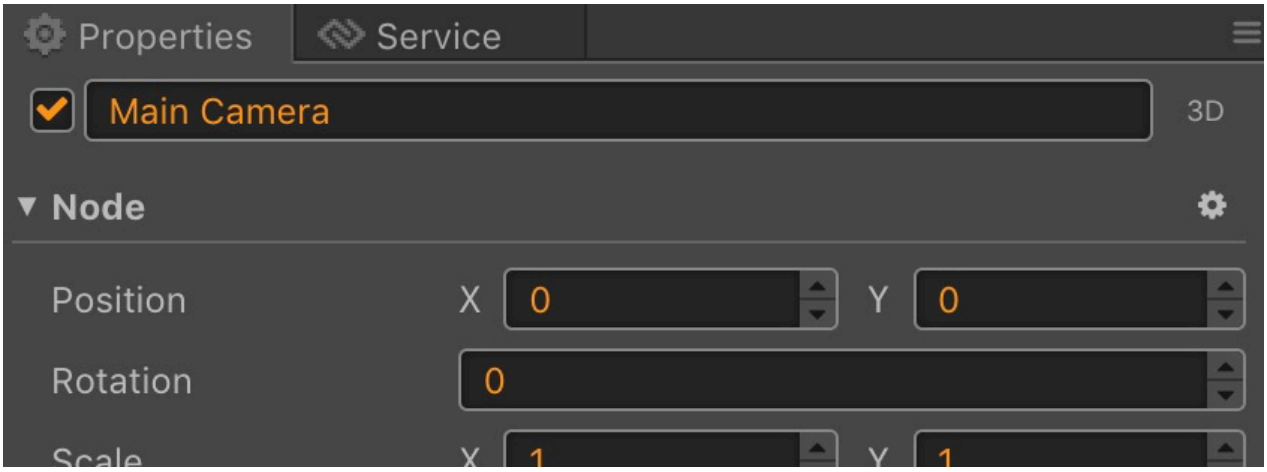
1. default 分组用于3D模型渲染。
2. ui 分组用于界面渲染。
3. head 分组用于将3D模型渲染成2D精灵。

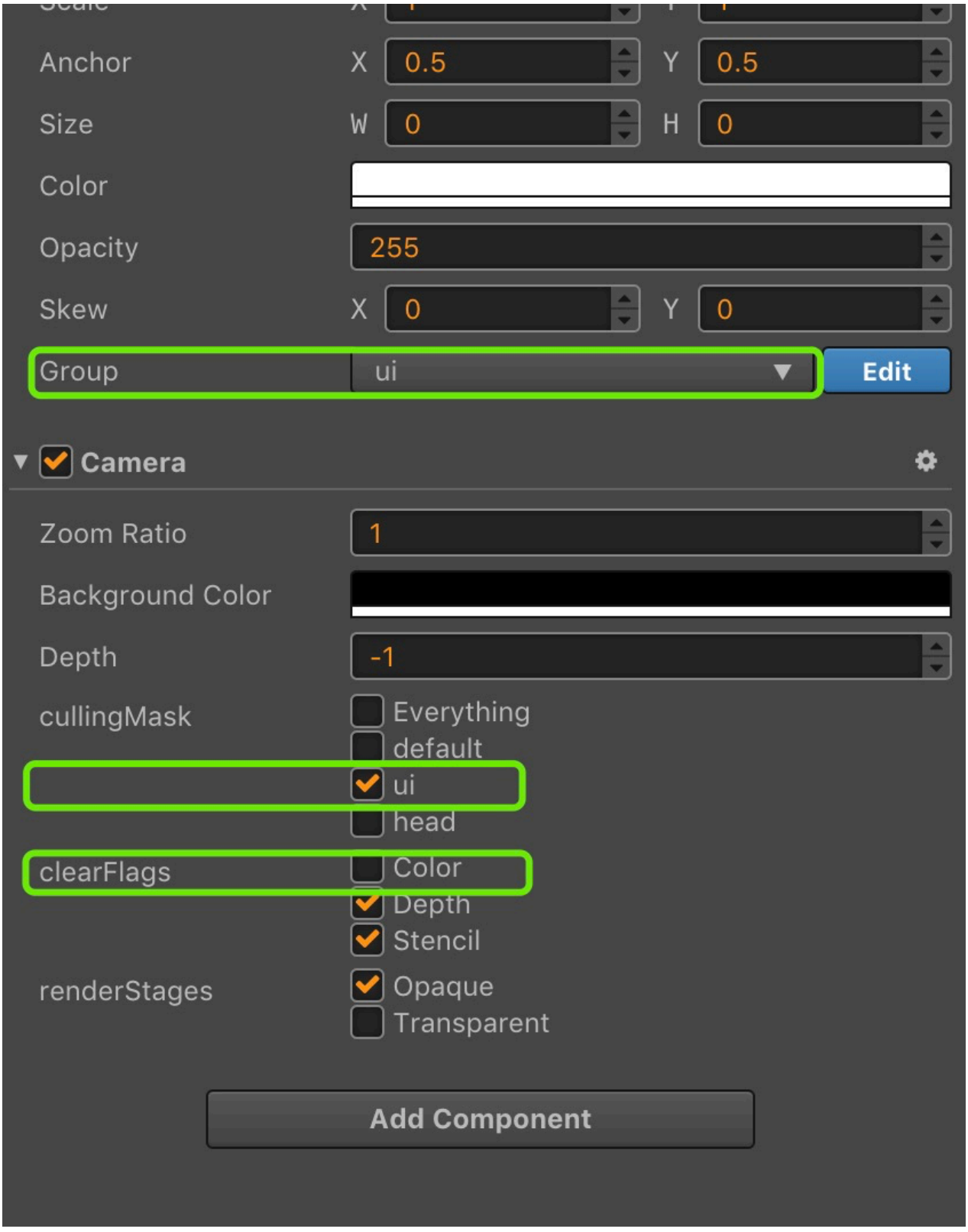


2D摄像机

新建场景，默认会有2D摄像机，使用的是default，记得手动调整成ui分组。

1. 调整Group的值为ui
2. 调整culling mask，勾选ui，标示该camera只会对ui分组的节点对象产生影响
3. 调整 clear flag，由于多个camera同时渲染，clear color的操作只能在一个amera上做处理，这样不会导致某个camera渲染不显示。



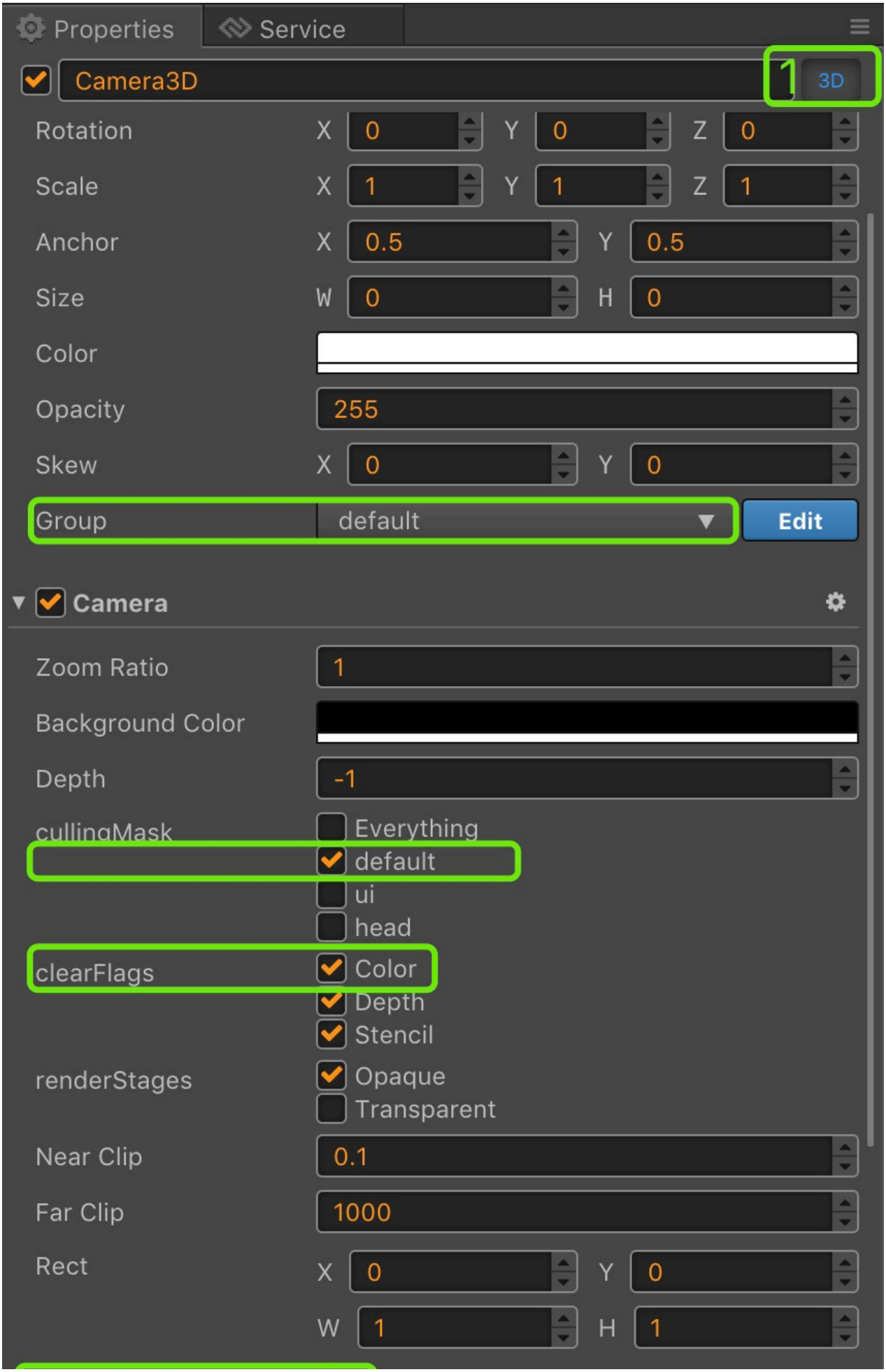


3D摄像机

新建的摄像机，默认是2D配置，我们需要选中3D按钮，使camera工作于3D状态。

1. 激活3D标示
2. 调整Group的值为default
3. 调整culling mask，勾选default，标示该camera只会对default分组的节点对象产生影响
4. 调整 clear flag，由于多个camera同时渲染，clear color的操作只能在一个amera上做处理，这样不会导致某个camera渲染不显示。

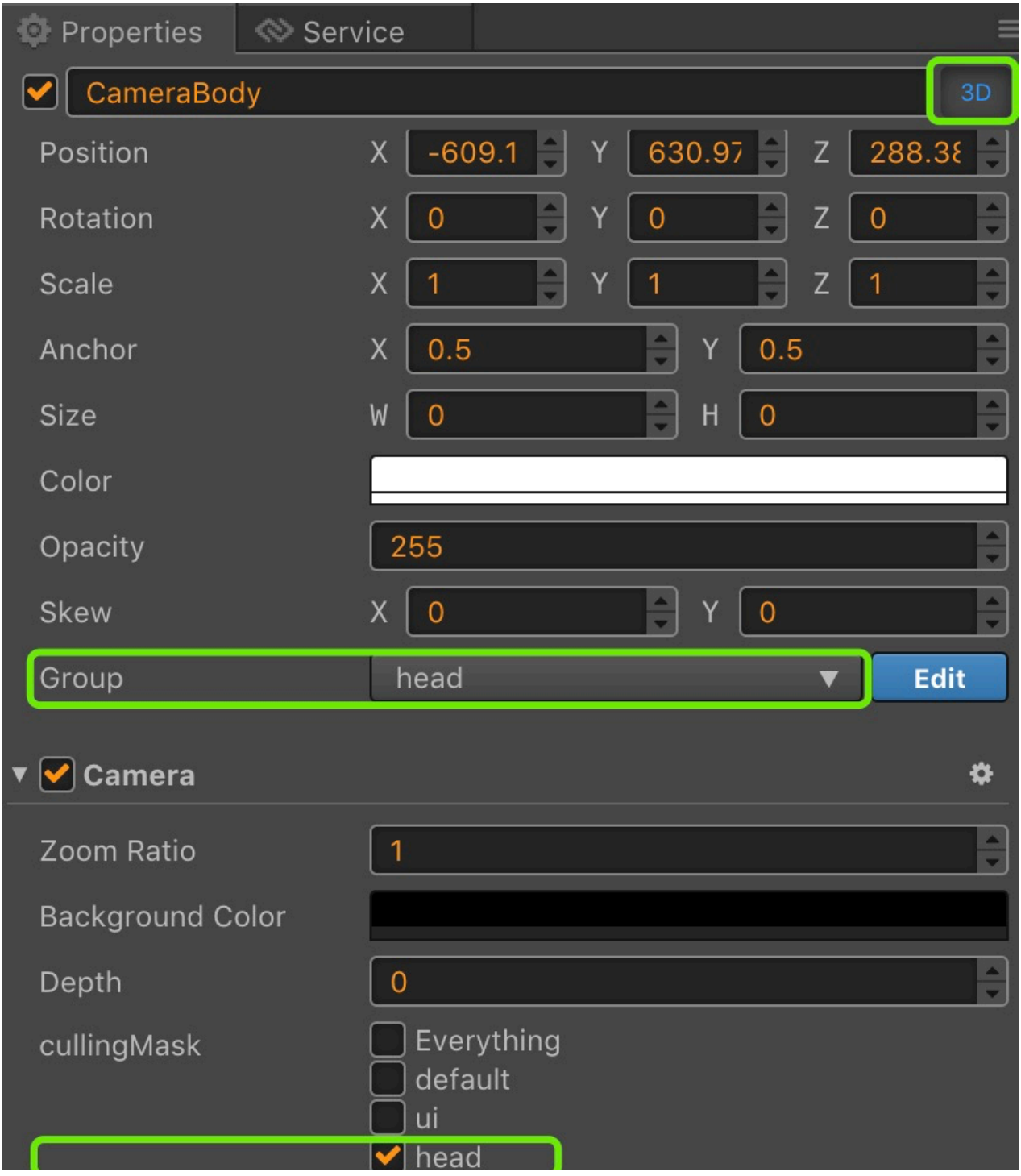
5. 默认使用透视模式

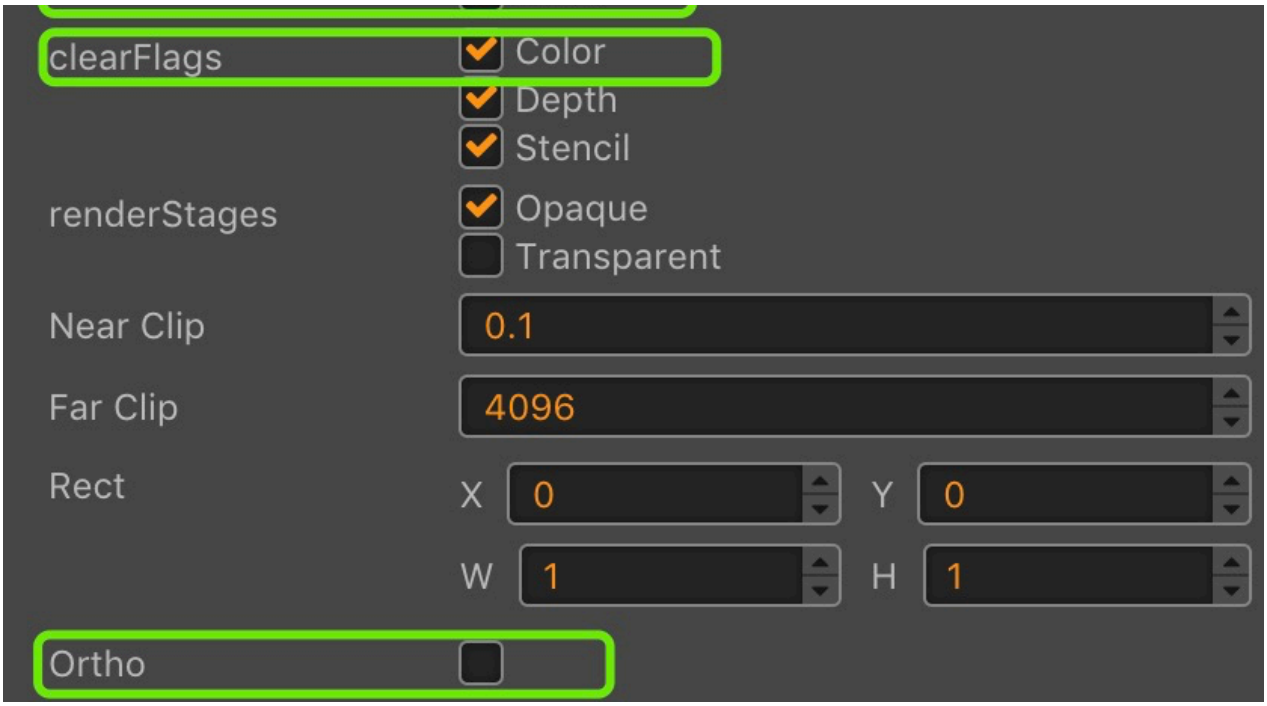

Properties
Service
Camera3D
1
3D
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Anchor X 0.5 Y 0.5
Size W 0 H 0
Color
Opacity 255
Skew X 0 Y 0
Group default
Edit
Camera
Zoom Ratio 1
Background Color
Depth -1
cullingMask
Everything
default
ui
head
clearFlags
Color
Depth
Stencil
renderStages
Opaque
Transparent
Near Clip 0.1
Far Clip 1000
Rect X 0 Y 0
W 1 H 1

3D模型渲染到2D精灵

新建的摄像机，默认是2D配置，我们需要选中3D按钮，使camera工作于3D状态。

1. 激活3D标示
2. 调整Group的值为head
3. 调整culling mask，勾选head，标示该camera只会对head分组的节点对象产生影响
4. 调整 clear flag，勾选clear color 标示，因为我们只会显示模型渲染结果。
5. 默认使用透视模式





下面代码是使用cc.RenderTexture组件，将camera渲染结果保存到纹理，如果期望在编辑器实时预览，可以添加executeInEditMode 标示

```
const { ccclass, property, executeInEditMode } = cc._decorator;

@ccclass
@executeInEditMode
export default class GCCameraRT extends cc.Component {

    @property(cc.Camera)
    cam3D: cc.Camera = null;

    @property(cc.Node)
    world3D: cc.Node = null; // 用于放置3D节点

    @property(cc.Sprite)
    view3D: cc.Sprite = null; // 用于显示

    // LIFE-CYCLE CALLBACKS:

    onLoad() {
        if (!this.cam3D)
            return;

        let texture = new cc.RenderTexture();
        texture.initWithSize(cc.view.getFrameSize().width,
                             cc.view.getFrameSize().height,
                             cc.gfx.RB_FMT_D24S8);

        let spriteFrame = new cc.SpriteFrame();
        spriteFrame.setTexture(texture)
```

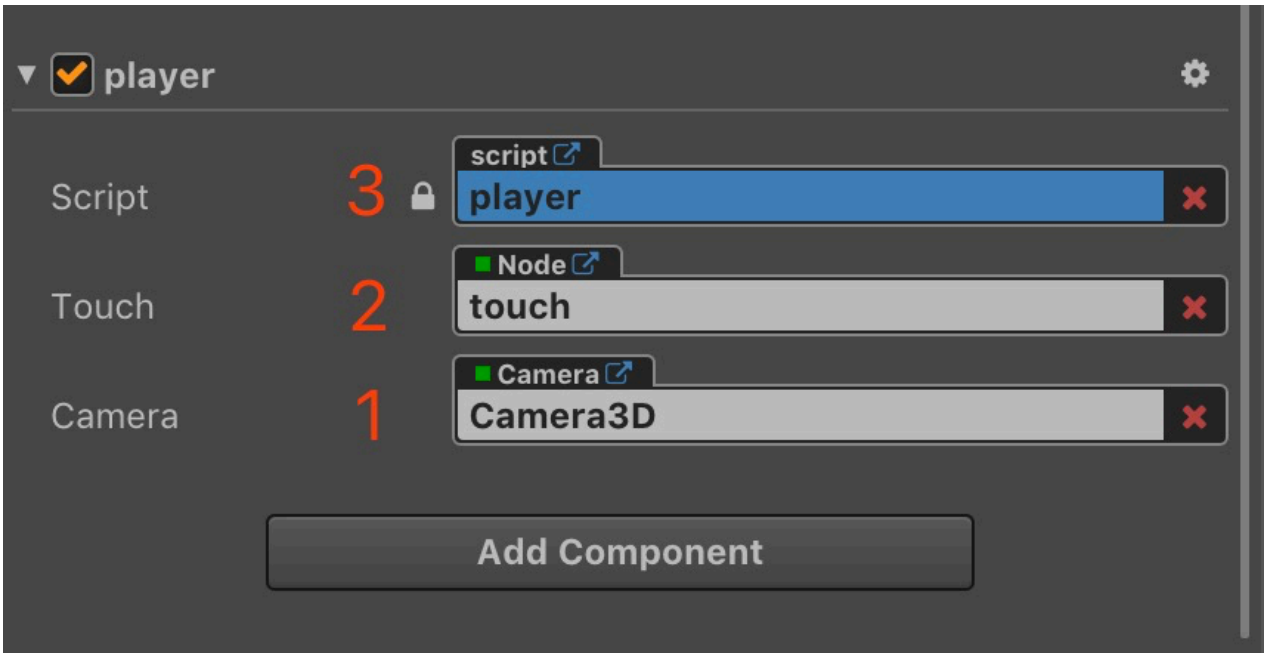
```
        // 绑定纹理到Sprite
        this.view3D.spriteFrame = spriteFrame;

        // camera会自动渲染，这里绑定纹理
        this.cam3D.targetTexture = texture;
    }

    start() {

    }

    update(dt) {
        if (CC_EDITOR && this.cam3D) {
            this.cam3D.render(this.world3D);
        }
    }
}
```

摄像机在3D场景中的使用

在我们这个教程里面，摄像机扮演的是观察者角色，实现以玩家模型为焦点的，全方位旋转的摄像机模式，玩家可以自由移动，摄像机会实现跟随逻辑。

我们在游戏场景中，以人物模型作为世界的焦点，准备实现一下这个内容。

1. 摄像机与人物的跟随绑定
2. 人物镜头的旋转控制
3. 人物模型的移动控制



3D摄像机旋转与位置跟随

1. 第一步我们设置了摄像机的角度

2. 我们根据摄像机设置的角度和摄像机与人物的距离，将摄像机移动到合适的位置
3. 摄像机与人物的位置是实时更新，所以我们将摄像机控制的代码添加到脚本的update函数过程

```
    update (dt) {
        if (!CC_EDITOR) {
            this.onMove();
        }

        // 设置摄像机朝向
        this.camera.node.eulerAngles = new cc.Vec3(this.degreeX, this.degreeY,
0);

        // 节点位置作为原点
        var local = new cc.Vec3(0, 0, 100);
        var newLocal = new cc.Vec3(0, 0, 0);

        // 镜头远近，这里以倍数做简单计算
        local.mul(1, newLocal)
        local = newLocal.clone();

        var outMat = new cc.Mat4(
            1, 0, 0 ,0,
            0, 1, 0, 0,
            0, 0, 1, 0,
            0, 0, 0, 1
        );

        var mat1 = new cc.Mat4(
            1, 0, 0 ,0,
            0, 1, 0, 0,
            0, 0, 1, 0,
            0, 0, 0, 1
        );
        var quaOut =  new cc.Quat;
        quaOut.fromEuler(new cc.Vec3(this.degreeX, this.degreeY, 0))
        mat1.fromQuat(quaOut);

        // 对方向做旋转变化
        local.transformMat4(mat1, newLocal);
        local = newLocal.clone();

        // 加上节点位置 (这里10假设是模型的高度值)
        var nodeLocal = new cc.Vec3(this.node.x, this.node.y + 10,
this.node.z);
        local.add(nodeLocal, newLocal)
        local = newLocal.clone();

        this.camera.node.setPosition(local);
```

```
    }
```

人物模型的移动控制

人物移动的时候，需要保证移动方向不受摄像机自由旋转影响，人物的朝向跟移动方向保持一致

```
onMove () {
  var nowDegree = this.degreeY;
  var find = false;
  this.keymask.forEach(()=>{
    find = true;
  })

  if (!find)
    return;

  if (this.keymask[cc.macro.KEY.w]) {
    nowDegree += 180;
    this.node.eulerAngles = new cc.Vec3(0, this.degreeY, 0);
  }
  else if (this.keymask[cc.macro.KEY.s]) {
    nowDegree += 0;
    this.node.eulerAngles = new cc.Vec3(0, this.degreeY - 180, 0);
  }
  else if (this.keymask[cc.macro.KEY.a]) {
    nowDegree += 270;
    this.node.eulerAngles = new cc.Vec3(0, nowDegree + 180, 0);
  }
  else if (this.keymask[cc.macro.KEY.d]) {
    nowDegree += 90;
    this.node.eulerAngles = new cc.Vec3(0, nowDegree - 180, 0);
  }

  var mat = new cc.Mat4(
    1, 0, 0 ,0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
  );

  var quaOut =  new cc.Quat;
  quaOut.fromEuler(new cc.Vec3(0, nowDegree, 0))
  mat.fromQuat(quaOut);

  var newLocal = new cc.Vec3();
  var local = new cc.Vec3(0, 0, 1);
  local.transformMat4(mat, newLocal);
  this.node.z += newLocal.z;
```

```
    this.node.x += newLocal.x;
}
```