



链滴

cocos creator 官方文档阅读之图形渲染模块

作者: [zhongsheng](#)

原文链接: <https://ld246.com/article/1669708096395>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

渲染管线

cocos creator 3 版本，有两种内置渲染管线，分别是 builtin-forward 前向渲染管线 builtin-deferred 延迟渲染管线

前向渲染管线

执行流程如下图：

对于光照较少的项目，可以使用该渲染管线。

-shadowflow

ShadowFlow 中包含一个 **ShadowStage** 会预先对景中需要投射阴影的物体进行阴影贴图的绘制。至于什么是 shadowflow，这个保留到看完 OpenGL 内容再回来添加上。

-forwardflow

ForwardFlow 包含一个 **ForwardStage**，会对场景所有物体按照 **非透明 > 光照 > 透明 > UI** 的顺序依次进行绘制。计算光照时，每个物体都会与所有光照进行计算确定是否照射到该物体，照射到该物体的光照将会绘制并进行光照计算，目前场景中只支持一个平行光，可接受的最大光照数量为 16。

延迟渲染管线

执行流程如下图：

对于光照数量比较多的项目，可以使用延迟渲染管线，用以缓解光照计算的压力。

-shadowflow-

包含一个 ShadowStage 阶段，预先进行阴影贴图的绘制。

-mainflow

包含了 GBufferStage、LightingStage、BloomStage、PostProcessStage

- GBufferStage**
绘制场景中非透明物体
- LightingStage**
对输出到 GBuffer 中的非透明物体信息进行基于屏幕空间的光照计算，再绘制半透明物体。如果有非透明物体并且设备支持 ComputeShader，还会进行 SSPR 的资源收集与绘制
- BloomStage**
开启了 Bloom 效果，BloomStage 会对已经经过了 LightingStage 处理后的图像进行 Bloom 后处理。
- 项目 > 项目设置 > Macro Configurations**，然后勾选 **ENABLE_BLOOM** 即可开启 Bloom。
- PostProcessStage**
把 BloomStage / LightingStage 输出的全屏图像绘制到主屏幕中，再进行 UI 的绘制。

关于渲染管线，官方介绍比较少，这也许是因为 cocos creator 本身的渲染管线还做得不是很好，不如 unity 做的好。但是关于计算机渲染的知识，还是很有必要花时间去学习的。

相机

通过阅读官方文档，对相机的一些基本属性有了一定的了解。但是这里需要后续补充点有：

<p>**:smiling_imp: 什么是 **透视投影 (PERSPECTIVE) 和 正交影 (ORTHO) </p>
<p>:smiling_imp: 一个节点, 是如何被相机看见的? </p>
<h2 id="光照">光照</h2>
<h2 id="基于物理的光照">基于物理的光照</h2>
<p>Cocos Creator 中采用光学度量单位来描述光源参数。基于光学度量单位, 可以将光的相关参数全部转化为真实世界中的物理值。 </p>
<h3 id="-光学度量单位">:space_invader: 光学度量单位</h3>

光通量 (Luminous Flux)

单位 ****流明 (lm) , 单位时间内光源所发出或者被照物体所接收的总光能。变光源大小不会影响场景照明效果。
亮度 (Luminance)

单位 ****坎德拉每平方米 (cd/m2) , 单位面积光源在给定方向上, 在每单位积内所发出的总光通量。改变光源大小会影响场景照明效果。
照度 (Illuminance)

单位 ****勒克斯 (lux 或 lx) , 每单位面积所接收到的光通量。该值受光的传距离影响, 对于同样光源而言, 当光源的距离为原先的两倍时, 照度减为原先的四分之一, 呈平方反关系。

<h2 id="光源">光源</h2>
<p>cocos 中的光源类型包括这几种: 平行光、球面光、聚光灯、环境光。 </p>
<h3 id="平行光">平行光</h3>
<p>**平行光的光照效果类似于太阳光, 光源与被照射目标的距离是未定义, 因此光照效果不受 **光源位置 和 朝向 的影响。 </p>
<h3 id="球面光">球面光</h3>
<h3 id="聚光灯">聚光灯</h3>
<h3 id="环境光">环境光</h3>
<h2 id="光照贴图">光照贴图</h2>
<p>烘焙系统对稳定光源的静态物体, 受到的光照和阴影进行预先计算, 计算结果存储在张纹理贴图中, 这个贴图就是光照贴图。 </p>
<p>用光照贴图替代实时的光照计算, 可以减少资源消耗, 提高场景运行效率。 </p>
<h2 id="网格">网格</h2>
<p>网格 一般用于绘制 3D 图像。Creator 提供了以下网格渲染器组件来渲染基础网格、蒙皮网格等, 从而将模型绘制显示出来: </p>
<p>MeshRenderer: 网格渲染器组件, 用于渲染基础的模型网格 </p>
<p>SkinnedMeshRenderer: 蒙皮网格渲染器组件, 用于渲染蒙皮模型网格 </p>
<p>SkinnedMeshBatchRenderer: 批量蒙皮网格渲染器组件, 用于将同一个骨骼动画组控制的所有子蒙皮模型合并渲染 </p>
<h2 id="MeshRenderer">MeshRenderer</h2>
<p>Mesh 资源中包含了一组顶点和多组索引。索引指向顶点数组中的顶点, 每三组索引成一个三角形。网格则是由多个三角形组成的, 是 3D 世界中最基本的图元。多个三角形拼接成一个杂的多边形, 多个多边形则拼接成一个 3D 模型。 </p>
<h3 id="网格渲染器合批">网格渲染器合批</h3>
<p>:space_invader: 静态合批 </p>
<p>**目前静态合批方案为运行时静态合批, 通过调用 <code>BatchingUtility.batchStaticModel</code> 可进行静态合批。

该函数接收一个节点, 然后将该节点下的所有 <code>MeshRenderer</code> 里的 <code>Mesh</code> 合并成一个, 并将其挂到另一个节点下。

在合批后, 将无法改变原有的 <code>MeshRenderer</code> 的 <code>transform</code> 但可以改变合批后的根节点的 <code>transform</code>。 </p>

只有满足以下条件的节点才能进行静态合批：

****子节点中只能包含 `MeshRenderer`；**

****子节点下的 `MeshRenderer` 的 `Mesh` 的顶点数据结构须一致；**

****子节点下的 `MeshRenderer` 的材质必须相同。**

:space invader: 动态合批

引擎目前提供 instancing 动态合批功能。

****要开启合批，只需在模型所使用的材质中对应勾选 `USE_INSTANCING` 开关即。**

<blockquote>

注意：目前的合批流程会引入一些限制：

同一批次的透明模型间的绘制顺序无法保证，可能导致混合效果不准确；

合批后没有传递逆转置世界矩阵信息，带有非均一缩放的模型的法线会不准确；

只支持普通 3D 模型和预烘焙骨骼动画控制下的蒙皮模型（实时计算骨骼动画、2D 物、UI、粒子等均不支持动态合批）。

</blockquote>

:space invader: 合批的最佳实践

****通常来说合批系统的使用优先级为：静态合批 > instancing 合批。**

首先要确保材质统一，在这个前提下，如果确定某些模型在游戏周期内完全静止不会变化就可以使用静态合批。

****如果存在大量相同的模型重复绘制，相互间只有相对可控的小差异，就可以使用 instancing 合批。**

SkinnedMeshRenderer

SkinnedMeshBatchRenderer

纹理

纹理是一张可显示的图片，或一段用于计算的中间数据。通过 UV 坐标映射到渲染物的表面，使之效果更为丰富且真实。Cocos 纹理的应用包括以下几种：

用于 2D UI 渲染。

用于 3D 模型渲染，需要在材质中指定纹理贴图资源，才能将其渲染映射到网格表面。纹理贴图还支持在导入图像资源时，将其切换为立方体贴图或者法线贴图。

用于粒子系统，使粒子表现更丰富。与 3D 模型一样，纹理在粒子系统中的应用也依赖于材质。

用于地形渲染。

压缩纹理

这一块比较好理解，直接参考文档，不同平台对应不同的压缩方式，设置好后，会在目标构建过程中对添加了压缩纹理的图片进行压缩。

渲染纹理

****渲染纹理是一张在 GPU 上的纹理，通常会把它设置到相机的 **目标纹理** 上，使相机照射的内容通过离屏的 `framebuffer` 绘制到该纹理上。一般用于制汽车后视镜，动态阴影等功能。**

材质系统

物体与光交互，根据物体表面外观不同，在光照下所表现出来的效果也不同。在游戏中，通过材质描述物体外观，物体在光照情况下所呈现出来的明暗、光点、光反射、光散射等效果都是通过 **着色器 来实现。而材质，则是着色器的数据集（包括纹理贴图，光照算法等）。**

程序化使用材质

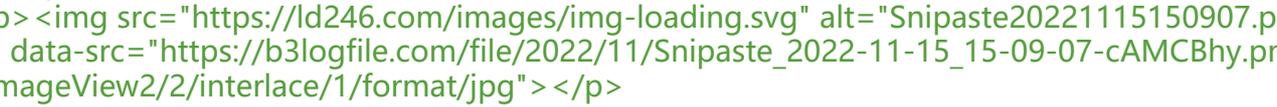
材质 (Material) 资源可以看成是着色器资源 (EffectAsset) 在场景中的资源实例。我们可以从资源管理器中手动创建材质，也可以通过 `IMaterialInfo` 接口，在脚本模块中程序化的创建材质。

内置材质

`cocos` 内置了一些材质，我们在资源管理器中，从 `Internal` 里面，就可以找到

材质系统类图

材质系统控制着每个模型最终的着色流程与顺序，在引擎内相关类间结构如下：



- `Material` 负责 `EffectAsset` 声明的 `Uniform`、宏数据存以及 Shader 使用和管理，这些信息都会以材质资源的可视化属性的形式展示在 `属性检查` 面板中。`Material` 通常是被渲染器组件使用，所有继承自 `RenderableComponent` 的组都是渲染器组件，例如 `MeshRenderer`、`Sprite` 等。
- `EffectAsset` 负责提供属性、宏、Shader 列表定义。每个 `EffectAsset` 最都会被编译成引擎内使用的格式，引擎再根据格式进行解析和应用。所有解析后的 `EffectAsset` 信息会被注册到引擎内的 `ProgramLib` 库里，方便用户直接通过代码获取实际引擎所使用的 `EffectAsset` 源。

着色器

在现代显卡中，若要正确地绘制物体，需要书写基于顶点 (Vertex) 和片元 (Fragment) 的代码片段，这些代码片段称为 Shader。在基于 OpenGL 系列驱动力的硬件设备上，Shader 支持种名为 GLSL (OpenGL Shading Language) 的着色器语言。

`Creator` 封装了基于 GLSL 的着色器 —— `Cocos Effect`

关于着色器的语法学习和编写，这一块在看完 OpenGL 之后再回过头来看。

特效组件

特效组件包括 `广告牌` 和 `线段组件`，可在 `属性检查器` 中点击 `添加组件` `Effects` 进行添加

天空盒

游戏中的天空盒是一个包裹整个场景的立方体，可以很好地渲染并展示整个场景环境在基于 PBR 的工作流中天空盒也可以贡献非常重要的 IBL 环境光照。

全局雾

全局雾用于在游戏中模拟室外环境中的雾效果。在游戏中除了用于雾效表现外，还可用于隐藏摄像机远剪切平面外的模型来提高渲染的性能。

全局雾的类型目前包括 `线性雾`、`指数雾`、`指数平方雾`、`层雾` 四种