# 合成 Compositing

## 简介

着色器的像素处理能力也能够被应用于2D图像，因此可以在原始3D输出的基础上添加后处理，强化游戏的图像效果。

合成是后处理技术的一种，一种简单的合成操作是阿尔法混合Alpha Blending，这种技术主要用于将独立的3D多边形渲染到部分填充的帧缓存中。然而，阿尔法混合的精度受限于定点Fixed-point数学（帧缓存中存储的色彩格式为每通道一个字节），这种限制可能会导致合成的过程中出现严重的错误。

为此，近年来GPU推出了一种浮点Floating-point格式的幕后缓存，允许执行极度复杂的合成效果而不用担心计算过程中数据精度上的损失。浮点缓存的计算不使用硬件中的阿尔法混合模块，而取决于像素着色器流水线——它作为一张纹理被加载到像素着色器流水线中。

## 将输入像素映射到输出像素 Mapping Input to Output Pixels

使用像素着色器处理2D图像时，实际上还是在3D空间中执行的。我们使用多边形直接对齐并覆盖整个屏幕，并将像素着色器程序直接应用到这个表面上。

典型的方法是使用一个简单的屏幕对齐矩形（由两个三角形组成）：定义一个3D单位大小的正方形，使x与y在[-0.5, 0.5]之间取值，z = 0，再使它与屏幕对齐。

根据使用的图形API，也许需要为坐标增加偏移量Offset使图像能够正确地对齐。例如在Direct3D中，你需要在横坐标上增加0.5/imageWidth，纵坐标上增加0.5/imageHeight，因为纹理坐标定位在每个纹理像素的中心点，而不是一个角上。虽然这个偏移量几乎是肉眼无法察觉的，但当我们需要在程序中执行多个图像过程时，若每个过程中都偏移了半个像素，将会导致不希望发生的结果。

## 阿尔法混合 Alpha Blending

阿尔法混合常用于处理半透明物体与其他物体的融合，其数学表达如下：

Alpha = aPixel.w

Result = Alpha \* aPixel + (1.0f - Alpha) \* bPixel

* aPixel与bPixel是需要混合的像素值，其中aPixel在上方

需要注意的是，这个公式成立的条件是aPixel没有预乘（Not Premultiplied）Alpha值，若aPixel已经预乘了Alpha值，则公式如下：

Result = aPixel + (1.0f - aPixel.w) \* bPixel

## 其他合成效果

在阿尔法混合的基础上做出轻微的改动，就可以实现其他的合成效果。

### 内部 In

imageA只有与imageB重叠时才显示。

Result = bPixel.w \* aPixel

### 外部 Out

imageA只有在imageB之外时才显示。

Result = (1.0f - bPixel.w) \* aPixel

### 溶解 Dissolve

混合两幅图像。

Result = lerp(aPixel, bPixel, dissolve)

* lerp(from, to, t)是着色器提供的线性插值函数，其中t在[0.0, 1.0]之间取值，t = 0时，函数返回from，t = 1时，函数返回to
* dissolve代表了aPixel的溶解程度，dissolve = 1时，表示aPixel完全溶解