[TOC]

在shader文件中添加第二个颜色

https://zhuanlan.zhihu.com/p/36635394
https://zhuanlan.zhihu.com/p/36695496
在看了shader编程的第二第三章后,觉的似懂非懂。再上面添加点东西,便于很深入的理解

MyShader.usf

- 我们从目标要求开始,这里我想要使得输出颜色变为两个颜色的和，比如可以让颜色变成R+G,R+B等等
- 这里我们现在usf文件中做如下添加和修改

```
// MyShader.usf
#include "/Engine/Public/Platform.ush"

float4 SimpleColor;
// 增加一个颜色采样的变量
float4 SimpleColor2;
void MainVS(
 in float4 InPosition : ATTRIBUTE0,
 out float4 OutPosition : SV_POSITION
 )
{
 // screenspace position from vb
    OutPosition = InPosition;
}

void MainPS(
    out float4 OutColor : SV_Target0
    )
{
        // 输出颜色变为两种颜色的叠加
    OutColor = (SimpleColor + SimpleColor2);
}
```

MyShaderTest.h

- 因为添加了一个颜色变量，我们得将相关数据传进去。首先想到的是在蓝图中增加一个引脚

```
// MyShaderTest.h
#pragma once

#include "CoreMinimal.h"
#include "UObject/ObjectMacros.h"
#include "Classes/Kismet/BlueprintFunctionLibrary.h"
#include "MyShaderTest.generated.h"
```

```
UCLASS(MinimalAPI, meta = (ScriptName = "TestShaderLibrary"))
class UTestShaderBlueprintLibrary : public UBlueprintFunctionLibrary
{
        GENERATED_UCLASS_BODY()

                UFUNCTION(BlueprintCallable, Category = "ShaderTestPlugin", meta =
(WorldContext = "WorldContextObject"))
                // 在这里增加一个FLinearColor的形参就是增加一个引脚
                static void DrawTestShaderRenderTarget(class
UTextureRenderTarget2D* OutputRenderTarget, AActor* AC, FLinearColor
MyColor,FLinearColor MyColor2);
};
```

MyShaderTest.cpp

DrawTestShaderRenderTarget

- 为此我们需要进入cpp文件中,添加相应的形参。这个函数是在逻辑线程中调用。
- ENQUEUE_RENDER_COMMAND向渲染线程压入一个渲染命令， 调用 DrawTestShaderRenderTarget_RenderThread
- 我们需要在lambda表达式中增加我们需要传的变量给渲染线程。

> DrawTestShaderRenderTarget中主要是对数据的获取和传递,一般不需要修改东西，只需要增加你要传递的各类数据。

```
// MyShaderTest.cpp
void UTestShaderBlueprintLibrary::DrawTestShaderRenderTarget(
        UTextureRenderTarget2D* OutputRenderTarget,
        AActor* Ac,
        FLinearColor MyColor,
        FLinearColor MyColor2
)
{
        check(IsInGameThread());

        if (!OutputRenderTarget)
        {
                return;
        }

        FTextureRenderTargetResource* TextureRenderTargetResource =
OutputRenderTarget->GameThread_GetRenderTargetResource();
        UWorld* World = Ac->GetWorld();
        ERHIFeatureLevel::Type FeatureLevel = World->Scene->GetFeatureLevel();
        FName TextureRenderTargetName = OutputRenderTarget->GetFName();
        ENQUEUE_RENDER_COMMAND(CaptureCommand)(
                [TextureRenderTargetResource, FeatureLevel, MyColor,MyColor2,
TextureRenderTargetName](FRHICommandListImmediate& RHICmdList)
        {
                DrawTestShaderRenderTarget_RenderThread(RHICmdList,
```

```
TextureRenderTargetResource, FeatureLevel, TextureRenderTargetName,
MyColor,MyColor2);
        }
        );

}
```

FMyShaderTest

- 因为给DrawTestShaderRenderTarget_RenderThread添加了一个数据，所以需要修改其函数，但是这个比较复杂我们先看简单的。
- 渲染管线中VS和PS是一定要自己配置的,所以创建自定义的VS和PS,需要从FGlobalShader继承。然后需要使用

```
IMPLEMENT_SHADER_TYPE(, FShaderTestVS,
TEXT("/Plugin/ShadertestPlugin/Private/MyShader.usf"), TEXT("MainVS"), SF_Vertex)
IMPLEMENT_SHADER_TYPE(, FShaderTestPS,
TEXT("/Plugin/ShadertestPlugin/Private/MyShader.usf"), TEXT("MainPS"), SF_Pixel)
```

这个语句来使相关类和Vertex Shader或Pixel Shader文件绑定，这样渲染管线中将会使用我们自定义的定点着色器和像素着色器

- 因为在VS和PS中有些配置是一样的,所以我们可以先从FGlobalShader派生一个FMyShaderTest的类,然后再从FMyShaderTest派生出我们的FShaderTestVS和FShaderTestPS。

```
// MyShaderTest.cpp
class FMyShaderTest : public FGlobalShader
{
public:

        FMyShaderTest() {}

        FMyShaderTest(const ShaderMetaType::CompiledShaderInitializerType&
Initializer)
                : FGlobalShader(Initializer)
        {
                SimpleColorVal.Bind(Initializer.ParameterMap,
TEXT("SimpleColor"));
                SimpleColorVal2.Bind(Initializer.ParameterMap,
TEXT("SimpleColor2"));
        }

        static bool ShouldCache(EShaderPlatform Platform)
        {
                return true;
        }

        static bool ShouldCompilePermutation(const
FGlobalShaderPermutationParameters& Parameters)
```

```
        {
                //return IsFeatureLevelSupported(Parameters.Platform,
ERHIFeatureLevel::SM4);
                return true;
        }

        static void ModifyCompilationEnvironment(const
FGlobalShaderPermutationParameters& Parameters, FShaderCompilerEnvironment&
OutEnvironment)
        {
                FGlobalShader::ModifyCompilationEnvironment(Parameters,
OutEnvironment);
                OutEnvironment.SetDefine(TEXT("TEST_MICRO"), 1);
        }

        void SetParameters(
                FRHICommandListImmediate& RHICmdList,
                const FLinearColor &MyColor,
                const FLinearColor &MyColor2
        )
        {
                SetShaderValue(RHICmdList, GetPixelShader(), SimpleColorVal,
MyColor);
                SetShaderValue(RHICmdList, GetPixelShader(), SimpleColorVal2,
MyColor2);
        }

        virtual bool Serialize(FArchive& Ar) override
        {
                bool bShaderHasOutdatedParameters = FGlobalShader::Serialize(Ar);
                Ar << SimpleColorVal;
                Ar << SimpleColorVal2;
                return bShaderHasOutdatedParameters;
        }

private:

        FShaderParameter SimpleColorVal;
        FShaderParameter SimpleColorVal2;

};
```

其中

```
// MyShaderTest.cpp
SimpleColorVal.Bind(Initializer.ParameterMap, TEXT("SimpleColor"));
SimpleColorVal2.Bind(Initializer.ParameterMap, TEXT("SimpleColor2"));
// MyShader.usf
float4 SimpleColor;
float4 SimpleColor2;
```

- 这是将FMyShaderTest中的私有变量与MyShader.usf中变量进行绑定。
- 但是我们注意到我们还没有将颜色输入到FMyShaderTest中的私有变量。因此我们需要写一个函数用于把我们的颜色信息传到shader里。。

```
// MyShaderTest.cpp
void SetParameters(
        FRHICommandListImmediate& RHICmdList,
        const FLinearColor &MyColor,
        const FLinearColor &MyColor2
)
{
        SetShaderValue(RHICmdList, GetPixelShader(), SimpleColorVal, MyColor);
        SetShaderValue(RHICmdList, GetPixelShader(), SimpleColorVal2, MyColor2);
}
```

- 因为这些参数在PS中使用到，所以使用GetPixelShader()参数

```
        virtual bool Serialize(FArchive& Ar) override
        {
                bool bShaderHasOutdatedParameters = FGlobalShader::Serialize(Ar);
                Ar << SimpleColorVal;
                Ar << SimpleColorVal2;
                return bShaderHasOutdatedParameters;
        }
```

- 虚幻序列化，用于读取磁盘上的渲染数据，这里需要将我们所需要的SimpleColorVal，SimpleColorVal2;

```
static bool ShouldCache(EShaderPlatform Platform)
{
        return true;
}

static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
{
        //return IsFeatureLevelSupported(Parameters.Platform,
ERHIFeatureLevel::SM4);
        return true;
}

static void ModifyCompilationEnvironment(const FGlobalShaderPermutationParameters&
Parameters, FShaderCompilerEnvironment& OutEnvironment)
{
        FGlobalShader::ModifyCompilationEnvironment(Parameters, OutEnvironment);
        OutEnvironment.SetDefine(TEXT("TEST_MICRO"), 1);
}
```

- 剩下这些可以暂时不用管。

VS 和 PS

```
class FShaderTestVS : public FMyShaderTest
{
	DECLARE_SHADER_TYPE(FShaderTestVS, Global);

public:
	FShaderTestVS() {}

	FShaderTestVS(const ShaderMetaType::CompiledShaderInitializerType&
Initializer)
		: FMyShaderTest(Initializer)
	{

	}
};


class FShaderTestPS : public FMyShaderTest
{
	DECLARE_SHADER_TYPE(FShaderTestPS, Global);

public:
	FShaderTestPS() {}

	FShaderTestPS(const ShaderMetaType::CompiledShaderInitializerType&
Initializer)
		: FMyShaderTest(Initializer)
	{

	}
};

IMPLEMENT_SHADER_TYPE(, FShaderTestVS,
TEXT("/Plugin/ShadertestPlugin/Private/MyShader.usf"), TEXT("MainVS"), SF_Vertex)
IMPLEMENT_SHADER_TYPE(, FShaderTestPS,
TEXT("/Plugin/ShadertestPlugin/Private/MyShader.usf"), TEXT("MainPS"), SF_Pixel)
```

- 这里从FMyShaderTest派生两个类分别利用IMPLEMENT_SHADER_TYPE宏指定为VS和PS

```
DECLARE_SHADER_TYPE(FShaderTestVS, Global);
```

- 该宏用于把该Shader加入全局shadermap中，在运行前会将shadermap中的所以shader进行编译。

DrawTestShaderRenderTarget_RenderThread

```
static void DrawTestShaderRenderTarget_RenderThread(
        FRHICommandListImmediate& RHICmdList,
        FTextureRenderTargetResource* OutputRenderTargetResource,
        ERHIFeatureLevel::Type FeatureLevel,
        FName TextureRenderTargetName,
        FLinearColor MyColor,
        FLinearColor MyColor2
)
{
        check(IsInRenderingThread());

#if WANTS_DRAW_MESH_EVENTS
        FString EventName;
        TextureRenderTargetName.ToString(EventName);
        SCOPED_DRAW_EVENTF(RHICmdList, SceneCapture, TEXT("ShaderTest %s"),
*EventName);
#else
        SCOPED_DRAW_EVENT(RHICmdList,
DrawUVDisplacementToRenderTarget_RenderThread);
#endif

        //设置渲染目标
        SetRenderTarget(
                RHICmdList,
                OutputRenderTargetResource->GetRenderTargetTexture(),
                FTextureRHIRef(),
                ESimpleRenderTargetMode::EUninitializedColorAndDepth,
                FExclusiveDepthStencil::DepthNop_StencilNop
        );

        //设置视口
        //FIntPoint DrawTargetResolution(OutputRenderTargetResource->GetSizeX(),
OutputRenderTargetResource->GetSizeY());
        //RHICmdList.SetViewport(0, 0, 0.0f, DrawTargetResolution.X,
DrawTargetResolution.Y, 1.0f);

        TShaderMap<FGlobalShaderType>* GlobalShaderMap =
GetGlobalShaderMap(FeatureLevel);
        TShaderMapRef<FShaderTestVS> VertexShader(GlobalShaderMap);
        TShaderMapRef<FShaderTestPS> PixelShader(GlobalShaderMap);

        // Set the graphic pipeline state.
        FGraphicsPipelineStateInitializer GraphicsPSOInit;
        RHICmdList.ApplyCachedRenderTargets(GraphicsPSOInit);
        GraphicsPSOInit.DepthStencilState = TStaticDepthStencilState<false,
CF_Always>::GetRHI();
        GraphicsPSOInit.BlendState = TStaticBlendState<>::GetRHI();
        GraphicsPSOInit.RasterizerState = TStaticRasterizerState<>::GetRHI();
        GraphicsPSOInit.PrimitiveType = PT_TriangleList;
        GraphicsPSOInit.BoundShaderState.VertexDeclarationRHI =
GetVertexDeclarationFVector4();
        GraphicsPSOInit.BoundShaderState.VertexShaderRHI =
GETSAFERHISHADER_VERTEX(*VertexShader);
```

```
        GraphicsPSOInit.BoundShaderState.PixelShaderRHI =
GETSAFERHISHADER_PIXEL(*PixelShader);
        SetGraphicsPipelineState(RHICmdList, GraphicsPSOInit);

        //RHICmdList.SetViewport(0, 0, 0.0f, DrawTargetResolution.X,
DrawTargetResolution.Y, 1.0f);
        PixelShader->SetParameters(RHICmdList, MyColor,MyColor2);

        // Draw grid.
        //uint32 PrimitiveCount = 2;
        //RHICmdList.DrawPrimitive(PT_TriangleList, 0, PrimitiveCount, 1);
        FVector4 Vertices[4];
        Vertices[0].Set(-1.0f, 1.0f, 0, 1.0f);
        Vertices[1].Set(1.0f, 1.0f, 0, 1.0f);
        Vertices[2].Set(-1.0f, -1.0f, 0, 1.0f);
        Vertices[3].Set(1.0f, -1.0f, 0, 1.0f);
        static const uint16 Indices[6] =
        {
                0, 1, 2,
                2, 1, 3
        };
        //DrawPrimitiveUP(RHICmdList, PT_TriangleStrip, 2, Vertices,
sizeof(Vertices[0]));
        DrawIndexedPrimitiveUP(
                RHICmdList,
                PT_TriangleList,
                0,
                ARRAY_COUNT(Vertices),
                2,
                Indices,
                sizeof(Indices[0]),
                Vertices,
                sizeof(Vertices[0])
        );

        // Resolve render target.
        RHICmdList.CopyToResolveTarget(
                OutputRenderTargetResource->GetRenderTargetTexture(),
                OutputRenderTargetResource->TextureRHI,
                false, FResolveParams());
}
```

- 这是渲染线程中执行的函数
- 这里只要在形参中添加MyColor2，以及SetParameters中添加MyColor2