MME 中文参考手册

MikuMikuEffect ver0.37 Reference

2020-6-9

不吃鱼的喵酱

MME 参考手册更新日志

0.1.0.0 (2010/9/18) 初版

0.2.0.0 (2010/12/12) MME Ver0.20

・添加了 OFFSCREENRENDERTARGET 语义

・拓展了 CONTROLOBJECT 语义可以取得的信息

・缓和来了 CONTROLOBJECT 语义引用的物体的渲染顺序的制约

・添加了 EDGECOLOR 语义

・修正了 VIEWPORTPIXELSIZE 语义中的错误描述

・修正了一部分用词

0.2.2.0 (2010/12/16) MME Ver0.22

・改变了对于 RENDERCOLORTARGET 和 OFFSCREENRENDERTARGET 语义的 Miplevels 的设定方法

0.2.3.0 (2010/12/20) MME Ver0.23

・添加了对于 CONTROLOBJECT 语义的补充

0.2.4.0 (2011/02/09) MME Ver0.24

・添加了 CONTROLOBJECT 语义可以指定的特殊物体名"self"

・添加了 TEXTUREVALUE 语义

0.2.6.0 (2010/02/21) MME Ver0.26

・修正了关于 Draw=Geometry 命令的记述

0.2.7.0 (2011/05/22) MME Ver0.27

・添加了_INDEX 语义

・添加了 VertexCount 变量和 SubsetCount 变量

・添加了 opadd 变量

・添加了 TEXTUREVALUE 语义的补充

0.2.8.0 (2012/03/26) MME Ver0.28

・修正了一部分关于 CONTROLOBJECT 语义的描述

0.3.0.0 (2012/09/19) MME Ver0.30

・添加了 OFFSCREENRENDERTARGET 语义可以指定的 DefaultEffect 的特殊效果名"main_default"

0.3.3.0 (2013/02/13) MME Ver0.33

・添加了用于纹理的材质变形的语义（ADDINGTEXTURE 等）

・添加了 PMX 模型的副 Tex 相关的内容（UseSphereMap, use_spheremap, use_subtexture）

・添加了 MATERIALTOONTEXTURE 语义
・添加了 GROUNDSHADOWCOLOR 语义
・添加了 MME_MIPMAP 宏

注意

・本文档仅对 MMEffect 可以识别的语义以及注解进行说明。对于效果文件更详细的说明请参考以下链接：

效果文件格式：https://docs.microsoft.com/zh-cn/windows/win32/direct3d9/dx9-graphics-reference-effects-file-format

HLSL 参考：https://docs.microsoft.com/zh-cn/windows/win32/direct3dhlsl/dx-graphics-hlsl-reference

・语义和注解的方案的制定参考了 NVIDIA 的 SAS。

https://www.nvidia.com/en-us/drivers/using-sas/

然而，并不保证 FX Composer 用的效果文件可以在 MME 中运行。

目录

1. 前言

（注：本节并非来自原始参考文档，而是译者所写。）

MMD 基于 DirectX 9 开发，对于渲染的自由度非常低。MME 是在此基础上对于 MMD 所进行的二次开发，并以插件的形式存在。

MME 所做的最大贡献就是将 MMD 中封闭的渲染变量暴露给用户，使得用户可以通过自己编写效果文件的方式来控制渲染的细节，并以此实现自己想要的效果。这当然大幅提升了 MMD 的渲染表现力，但同时，MME 的使用一方面受到 DirectX 9 的限制，另一方面受到 MMD 本身的限制。

为了使用 DirectX 9 的特性，用户必须学习 HLSL，即高级着色器语言。这是一门由微软开发的、与 C 语言语法颇为相似的着色器语言。用户最高可以使用到 Shader Model 3.0（DirectX 9.0c）的特性（注：MMD 使用的 DirectX 具体版本号不明，或许无法使用 Shader Model 3.0。但是 Shader Model 2.0 是经过确认可以使用的）。当然，我们都知道，这个大版本还没有诸如次序无关透明度、曲面细分、几何着色器、计算着色器之类的特性，所以不要期待这些功能。

此外，MMD 通过 MME 暴露给着色器的信息也十分有限。虽然一般对于渲染来说是绰绰有余了，但是想进行一些特定效果的计算，比如全局光照，或者以碰撞触发某些效果，依旧是十分难以实现的。此外，由于 MMD 本身不可编程，某些效果只能妥协。

尽管如此，掌握 MME 的效果文件的编写方法依旧是学习渲染技术的一种十分有效的方式。正是由于其在提供一定的自由度的同时具有诸多限制，才会让学习者产生查找资料克服这些限制的动力，从而更加深入地研究渲染技术。

阅读本文档前，读者最好已经具有 HLSL 基础，以及一部分计算机图形学基础。本文档对于大部分专业名词以及渲染原理没有进行任何解释，推荐没有基础的读者优先阅读第 6 节。

感谢 B 站水零枫飞大佬对翻译提供的帮助。

2. technique 与 pass

2.1. 构成

效果文件是由 technique 与 pass 按照层次关系构成的，就像下面这样:

```
/*********************************/
变量声明 1
变量声明 2
...

technique Tech1 {
    pass Pass1 {
        VertexShader = …
        PixelShader  = …
    }
    pass Pass2 {
        VertexShader = …
        PixelShader  = …
    }
    …
}

technique Tec2 {
    pass Pass1 {
        VertexShader = …
        PixelShader  = …
    }
    pass Pass2{
        VertexShader = …
        PixelShader  = …
    }
    …
}
/*********************************/
```

其中，technique 是由一个或以上的 pass 构成的。如果一个 technique 包含多个 pass，那么就意味着这个 technique 使用了多次渲染技术，每次渲染使用其中的一个 pass。

对于 technique 和 pass 可以记录一种被称作注解的设定值。注解是写在

technique 和 pass 后面的尖括号里的内容，如下所示。需要注意的是，注释不区分大小写。

```
/*****************************************************/
technique Tech1 < string Subset = "1-6,8"; > {
    pass Pass1 < string Script = "Draw=Buffer;"; > {
        ...
    }
    pass Pass2 {
        ...
    }
}
/*****************************************************/
```

另外，对于 technique 和 pass 可以指定一种名为 Script 的特殊注解。详情参考第 4 节。

2.1.1. technique 的注解

technique 的注解指定了使用这个 technique 的物体需要满足的条件。这个条件可以包括以下内容:

- 物体的子集的编号（约等于物体的材质编号）
- 被渲染的部分（物体本身/影子/轮廓/描绘本影用的 Z 缓冲）
- 渲染项开关（纹理、spa、toon 渲染 ON/OFF）

由于条件匹配将按照被声明的顺序进行，所以当复数个 technique 的条件都匹配通过时，仅有最先声明的 technique 会被执行。当全部的 technique 都不匹配时，将会使用 MMD 的标准着色器。

2.1.1.1.注解

2.1.1.1.1. string Subset

指定适用的子集编号。如果物体是 PMD 模型，则指定的是模型材质编号。如果省略此注解，则对全部的子集均适用。

可以使用逗号隔开来指定复数编号，如"0,3,5"。

可以使用连字符来指定编号范围，如"6-10"。

可以仅指定开始编号来包含从这以后的全部编号，如"12-"。

例：`string Subset = "0-6,8";`

2.1.1.1.2. string MMDPass

指定适用的渲染过程。值可以为以下任意一个（以下列出的为 MMD 的渲染过程）

"object"：物体本身（本影关闭）

"zplot"：本影用的 Z 缓冲

"object-ss"：物体本身（本影开启）

"shadow"：阴影（不是指本影，而是指影子）

"edge"：轮廓（仅限 PMD 模型）

省略此注解将会使用默认值"object"。

例：`string MMDPass = "object";`

2.1.1.1.3. bool UseTexture

指定是否按照纹理（漫反射贴图）来过滤被渲染的物体。如果设定为 true，那么将仅匹配包含纹理的子集；如果设定为 false，那么将仅匹配不包含纹理的子集；如果不写，则不考虑是否包含纹理。

例：`bool UseTexture = true;`

2.1.1.1.4. bool UseSphereMap

指定是否按照 spa 来过滤被渲染的物体。如果设定为 true，那么将仅匹配包含 spa 的子集(对于 PMX 模型而言，指定为副 Tex 的 spa 也适用)；如果设定为 false，那么将仅匹配不包含 spa 的子集；如果不写，则不考虑是否包含 spa。

例：`bool UseSphereMap = false;`

2.1.1.1.5. bool UseToon

指定是否按照 toon 来过滤被渲染的物体。如果设定为 true，那么将仅匹配使用 toon 渲染的子集(即模型本身)；如果设定为 false，那么将仅匹配不使用 toon 渲染的子集（即附件）；如果不写，则不考虑是否使用 toon 渲染。

例：`bool UseToon = true;`

2.1.1.2.例子

```
//开启本影的场合，子集 0-6 与 8 使用 Tech1，子集 7 与 9 以后使用 Tech2
//本影关闭的场合使用 Tech3

technique Tech1 <
    string MMDPass = "object_ss";
    string Subset = "0-6,8";
> {
    ...
}

technique Tech2 <
    string MMDPass = "object_ss";
    string Subset = "7,9-";
> {
    ...
}

technique Tech3 <
    string MMDPass = "object";
> {
    ...
}
```

2.1.1.3.补充

无效的 technique 将被排除在外。参考：https://docs.microsoft.com/zh-cn/windows/win32/direct3d9/validation

UseTexture，UseSphereMap，UseToon 在 MMDPass="object"，"object_ss"之外的 technique 中不会正常工作。

3. 变量的语义与注解

接下来将要对 MME 使用的效果文件的变量的语义和注解进行说明。我们可以使用语义和注解来辅助声明变量。通过解读变量的声明，系统能获取渲染中所需的各种信息。

变量的声明遵循以下形式：

```
类型名 变量名 : 语义名 <类型名 注解 1 = 值; 类型名 注解 2 = 值; … >;
```

根据语义不同，也有不指定注解的情况。值得注意的是，语义和注解均不区分大小写。

3.1. 几何变换

参考：https://docs.microsoft.com/zh-cn/windows/win32/direct3d9/transforms

3.1.1. 顶点坐标变换

顶点坐标变换使用的矩阵。坐标变换分为世界坐标系变换、视图变换和投影变换。使用以下 6 个语义将可以直接获得对应的变换矩阵（类型是 float4x4）：

- WORLD：世界坐标系变换矩阵
- VIEW：视图变换矩阵
- PROJECTION：投影变换矩阵
- WORLDVIEW：世界坐标系变换矩阵乘视图变换矩阵
- VIEWPROJECTION：视图变换矩阵乘投影变换矩阵
- WORLDVIEWPROJECTION: 世界坐标系变换矩阵乘视图变换矩阵乘投影变换矩阵

你可以在各个语义的末尾加上"INVERSE"来得到逆矩阵，如"WORLDINVERSE"；你可以在各个语义的末尾加上"TRANSPOSE"来得到转置矩阵，如"WORLDTRANSPOSE"；你可以在各个语义的末尾加上"INVERSETRANSPOSE"来得到转置矩阵的逆矩阵。

3.1.1.1.注解

3.1.1.1.1. string Object（可省略）

视图变换的视点是可以指定的，值为"Camera"或者"Light"。默认为"Camera"。通常情况下，使用摄像机作为视点的场合会将值指定为"Camera"。本影、影子等场合会将值指定为"Light"。

3.1.1.2.例子

```
float4x4 WorldMatrix : WORLD ;
float4x4 WorldViewProjMatrix : WORLDVIEWPROJECTION ;
float4x4 LightViewMatrix : VIEW < string Object = "Light"; > ;
float4x4 WorldInvMatrix : WORLDINVERSE ;
float4x4 WorldViewProjTransMatrix : WORLDVIEWPROJECTIONTRANSPOSE;
```

3.1.1.3.补充

注解 Object 设定为"Light"时所取得的矩阵与 MMD 处理本影所使用的矩阵是相关联的。也就是说，如果你在显示->显示本影中关闭了本影，那么将无法取得正确的矩阵。

3.2. 光源与材质

参考：https://docs.microsoft.com/zh-cn/windows/win32/direct3d9/lights-and-materials

3.2.1. 颜色

3D 物体的材质颜色或者光源颜色。语义可以使用以下 8 个值：

- DIFFUSE：漫反射颜色
- AMBIENT：环境光颜色
- EMISSIVE：自发光颜色
- SPECULAR：镜面反射颜色
- SPECULARPOWER：镜面反射强度
- TOONCOLOR：toon 色
- EDGECOLOR：轮廓颜色
- GROUNDSHADOWCOLOR：地面影颜色

对于 PMD 模型，可以使用 toon 贴图的左下角的颜色来设置 TOONCOLOR。至于类型，仅有"SPECULARPOWER"的类型是 float，其余均为 float3 或 float4。颜色有 4 个分量（R，G，B，Alpha），float3 会省略 Alpha 值。

3.2.1.1.注解

3.2.1.1.1. string Object（必须）

指定取得光源颜色或者物体的材质颜色。可以指定为"Light"或者"Geometry"。想要取得物体的材质颜色应使用"Geometry"，想要取得光源颜色应使用"Light"。对于"SPECULARPOWER"、"EMISSIVE"和"TOONCOLOR"，由于光源没有这些变量，所以不能指定为"Light"。

3.2.1.2.例子

```
float4 MaterialDiffuse  : DIFFUSE  < string Object = "Geometry"; >;
float3 MaterialAmbient  : AMBIENT  < string Object = "Geometry"; >;
float3 MaterialEmmisive : EMISSIVE < string Object = "Geometry"; >;
float3 MaterialSpecular : SPECULAR < string Object = "Geometry"; >;
float  SpecularPower    : SPECULARPOWER < string Object = "Geometry"; >
;
float3 MaterialToon     : TOONCOLOR;
float3 EdgeColor        : EDGECOLOR;
float3 LightDiffuse     : DIFFUSE   < string Object = "Light"; >;
float3 LightAmbient     : AMBIENT   < string Object = "Light"; >;
float3 LightSpecular    : SPECULAR  < string Object = "Light"; >;
static float4 DiffuseColor = MaterialDiffuse * float4(LightDiffuse, 1.0
f);
static float3 AmbientColor = MaterialAmbient * LightAmbient + MaterialE
mmisive;
static float3 SpecularColor = MaterialSpecular * LightSpecular;
float4 GroundShadowColor : GROUNDSHADOWCOLOR;
```

3.2.1.3.补充

当且仅当 technique 含有 MMDPass="edge"注解时，轮廓颜色可以被正确地获得；当且仅当 technique 含有 MMDPass="shadow"注解时，地面影颜色可以被正确地获得。然而，当 technique 含有 MMDPass="zplot"或 MMDPass="edge"注解时，其他各项颜色均不能被正确地获得。

3.2.2. 空间位置

光源与摄像机在世界空间中的位置与朝向。类型为 float3 或 float4。语义可以使用以下 2 个值:

- POSITION
- DIRECTION

3.2.2.1. 注解

3.2.2.1.1. string Object（必须）

指定获取光源或者摄像机的坐标。可以指定为"Light"或者"Camera"。

3.2.2.2. 例子

```
float3 LightDirection : DIRECTION < string Object = "Light"; >;
float3 CameraPosition : POSITION  < string Object = "Camera"; >;
```

3.2.2.3. 补充

MMD 的光源为方向光，即，光源位置为光源方向的反方向的无限远点。

3.2.3. 材质纹理

材质中设置的纹理、spa 和 toon 贴图。语义可以指定为以下 3 个值:

- MATERIALTEXTURE
- MATERIALSPHEREMAP
- MATERIALTOONTEXTURE

3.2.3.1. 注解

无。

3.2.3.2. 例子

```
texture ObjectTexture : MATERIALTEXTURE;
sampler ObjTexSampler = sampler_state
{
    texture = <ObjectTexture>;
    MINFILTER = LINEAR;
    MAGFILTER = LINEAR;
    MIPFILTER = LINEAR;
    ADDRESSU  = WRAP;
    ADDRESSV  = WRAP;
};
// 可以参考 tex2D(ObjTexSampler, float2(x,y))的内容。

texture ObjectSphereMap : MATERIALSPHEREMAP;
sampler ObjSphSampler = sampler_state
{
    texture = <ObjectSphereMap>;
    MINFILTER = LINEAR;
    MAGFILTER = LINEAR;
    MIPFILTER = LINEAR;
    ADDRESSU  = WRAP;
    ADDRESSV  = WRAP;
};
// 可以参考 tex2D(ObjSphSampler, float2(x,y)) 的内容

texture ObjectToonTexture : MATERIALTOONTEXTURE;
sampler ObjToonSampler = sampler_state
```

```
{
    texture = <ObjectSphereMap>;
    MINFILTER = LINEAR;
    MAGFILTER = LINEAR;
    MIPFILTER = NONE;
    ADDRESSU  = CLAMP;
    ADDRESSV  = CLAMP;
};
// 可以参考 tex2D(ObjToonSampler, float2(x,y)) 的内容
```

3.2.3.3. 补充

当 technique 含有 MMDPass="zplot"或 MMDPass="edge"注解时，此项不能被正确地获得。此外，当 technique 含有 UseToon=false 注解时，toon 贴图不能被正确地获得。对于 PMD 模型而言，不使用 toon 的材质的 toon 贴图是纯白色的。

3.2.4. 附加纹理

针对 PMX 模型的顶点动画（比如表情）所使用的纹理与 spa，可以是加算或者乘算。语义可以使用以下 4 个值：

- ADDINGTEXTURE
- MULTIPLYINGTEXTURE
- ADDINGSPHERETEXTURE
- MULTIPLYINGSPHERETEXTURE

3.2.4.1. 注解

无。

3.2.4.2. 例子

```
float4   TextureAddValue   : ADDINGTEXTURE;
float4   TextureMulValue   : MULTIPLYINGTEXTURE;
float4   SphereAddValue    : ADDINGSPHERETEXTURE;
float4   SphereMulValue    : MULTIPLYINGSPHERETEXTURE;
```

3.2.4.3. 补充

当 technique 含有 MMDPass="object_ss"注解时，这些值才能被正确地获得。此外，当 technique 含有 UseToon=false 注解时，加算值全部为 0，乘算值全部为 1。

3.3. 屏幕信息

3.3.1. 屏幕尺寸

渲染的目标屏幕的尺寸。类型是 float2，单位是像素。此值为 MMD 的预览窗口大小或者离屏渲染的屏幕大小。即使通过 Scripts 的 RenderColorTarget 命令变更了渲染目标，此值也不会改变。此语义可以使用以下值：

- VIEWPORTPIXELSIZE

3.3.1.1. 注解

无。

3.3.1.2. 例子

```
float2 ScreenSize : VIEWPORTPIXELSIZE;
```

3.3.1.3. 补充

依赖 Viewport 的 Width 和 Height。

3.4. 时间

3.4.1. 经过时间

时间信息。类型是 float，单位是秒。语义可以使用以下 2 个值：

- TIME
- ELAPSEDTIME

"TIME"是从第 0 帧开始的播放时间。举例来说，第 0 帧的值为 0.0（秒），第 45 帧的值为 1.5（秒）。"ELAPSEDTIME"是距离上一次渲染的间隔时间。举例来说，以 60FPS 导出 AVI 的话，"ELAPSEDTIME"的值恒为 1/60。

3.4.1.1. 注解

3.4.1.1.1. bool SyncInEditMode（可省略）

指定是否在 MMD 的编辑模式下同步时间。值可以指定为 true 或者 false，默认为 false。

由于在 MMD 的编辑模式中，帧的播放是停止的，而 TIME 是与帧的播放相关的，因此使用了 TIME 的效果也会停止变化。若将此注解指定为 false，那么 TIME 和 ELAPSEDTIME 的值将使用系统的时间，因此即使在编辑模式中，使用了这两个值的效果也不会停止变化。

3.4.1.2. 例子

```
float ftime : TIME <bool SyncInEditMode=true;>;
float elapsed_time : ELAPSEDTIME;
static float fps = 1.0 / elapsed_time;
```

3.4.1.3. 补充

若将 SyncInEditMode 指定为 true，那么在编辑模式下，根据用户在时间轴上的移动帧的操作，ELAPSEDTIME 的值可以为 0，甚至是负值。

3.5. 鼠标

3.5.1. 位置

鼠标现在的位置。类型为 float。此语义可以使用以下值

- MOUSEPOSITION

MMD 的渲染范围的中心点坐标是(0, 0)，左下角是(-1, -1)，右上角是(1, 1)。这个 XY 坐标的取值方法和经过投影变换后的顶点坐标的取值方法是一致的。

3.5.1.1. 注解

无。

3.5.2. 按键

与鼠标按键相关的信息。类型为 float4。语义可以使用以下 3 个值:

- LEFTMOUSEDOWN
- MIDDLEMOUSEDOWN
- RIGHTMOUSEDOWN

取得的值由以下 4 个值构成:

- 最后一次按下此按键时鼠标的坐标（x 和 y）
- 现在此按键是否被按下（0 或 1）
- 最后一次按下此按键时的 TIME 的值（秒）

此外，鼠标坐标的获取方法与 MOUSEPOSITION 的相同。

3.5.2.1. 注解

无。

3.5.2.2. 例子

```
float4 mouse_down : LEFTMOUSEDOWN;
static float2 pos = mouse_down.xy;
static bool is_pressed = (mouse_down.z != 0);
```

3.6. 控制物体

3.6.1. 物体

取得被指定的物体的坐标和世界坐标系变换矩阵。如果希望从 MMD 中控制着色器内变量的值, 那么可以考虑使用这个语义。类型可以是 bool, float, float3, float4 或者 float4x4。此语义可以使用以下值:

- CONTROLOBJECT

根据声明的变量类型不同，所能取得的信息也有差异。具体如下:

- bool：被指定的物体当前是否显示
- float：被指定的物体的缩放值
- float3，float4：被指定的物体的坐标（offset）
- float4x4：被指定的物体的世界坐标系变换矩阵

此外，item 注解如果被指定为一些特殊的字符串，也可以取得上述以外的其他信息。

3.6.1.1.注解

3.6.1.1.1. string name（必须）

指定物体的文件名（不含文件夹路径）。如果被指定为特殊文件名"(self)"，则物体为在 MME 界面中指定使用此效果文件的物体。如果被指定为特殊文件名"(OffscreenOwner)"，则物体为离屏渲染物体（仅在离屏渲染时可用）。这里的离屏渲染物体指的是通过 OFFSCREENRENDERTARGET 声明的效果所分配的物体。

3.6.1.1.2. string item（可省略）

当你想让这个物体取一些特殊值时可以使用此注解。以下是可以使用的值:

- 骨骼名称: PMD 模型中指定骨骼的坐标以及世界坐标系变换矩阵。其值可以是 float3，float4 或 float4x4
- 表情名称：PMD 文件中指定表情的值。类型是 float。
- "X"：附件面板中 X 的值。类型为 float。

- "Y"：附件面板中 Y 的值。类型为 float。
- "Z：附件面板中 Z 的值。类型为 float。
- "XYZ"：附件面板中 X、Y、Z 的值。类型为 float3。
- "Rx"：附件面板中 Rx 的值。类型为 float。（注 1）
- "Ry"：附件面板中 Ry 的值。类型为 float。
- "Rz：附件面板中 Rz 的值。类型为 float。
- "Rxyz"：附件面板中 Rx、Ry、Rz 的值。类型为 float3。
- "Si"：附件面板中 Si 的值。类型为 float。（注 2）
- "Tr"：附件面板中 Tr 的值。类型为 float。

注 1：得到的值是面板上的值转化为弧度制表示的。

注 2：得到的值是面板上的值乘 10。

3.6.1.2. 例子

```
//获取"stage01.x"是否显示
bool flag : CONTROLOBJECT < string name = "stage01.x"; >;

//获取"negi.x"缩放值
float scaling : CONTROLOBJECT < string name = "negi.x"; >;

//获取"negi.x"的绕 X 轴旋转的弧度
float rot_x : CONTROLOBJECT < string name = "negi.x"; string item = "Rx
"; >;

//获取"negi.x"的绕 X 轴旋转的角度
float rot_x_rad : CONTROLOBJECT < string name = "negi.x"; string ite
m = "Rx"; >;
static float rot_x = rot_x_rad * 180 / 3.14159265;

//获取"弱音ハク.pmd"的【ポニテＩＫ】骨的坐标
float3 pos : CONTROLOBJECT < string name = "弱音ハク.pmd"; string ite
m = "ポニテＩＫ"; >;

//获取"弱音ハク.pmd"的【まばたき】表情的值
float morph : CONTROLOBJECT < string name = "弱音ハク.pmd"; string ite
m = "まばたき"; >;
```

3.6.1.3. 补充

指定的文件名的物体不存在的情况下，会使用以下默认值：

附件不存在的情况：

缩放值：10

offset 值：(0, 0, 0, 1)

世界坐标系变换矩阵：缩放矩阵（xyz 各十倍）

特殊 item 值：0

PMD 文件不存在的情况：

缩放值：1

offset 值：(0, 0, 0, 1)

世界坐标系变换矩阵：单位矩阵

骨骼坐标：(0, 0, 0, 1)

骨骼变换矩阵：单位矩阵

表情值：0

如果被指定的物体存在复数个（名称相同的），将按照以下顺序进行匹配：

(1) 比参照源更早被渲染的物体中最近的那一个

(2) 最后被渲染的那一个

对于一个顶点都没有的物体（可能仅包含骨骼），即使将其指定为参照先，也无法获取坐标。

3.7. 纹理相关

3.7.1. 通常纹理

直接生成的纹理。类型为 texture，texture2D，texture3D，textureCUBE 中的一个。即使指定了 RENDERCOLORTARGET, RENDERDEPTHSTENCILTARGET, ANIMATEDTEXTURE 以外的语义也不会生效。直接生成的纹理，根据设定于其上的 sampler 的不同，可以使用诸如 tex2D(s, t)之类的函数进行采样。

3.7.1.1. 注解

3.7.1.1.1. string ResourceType

指定纹理的种类。可以设置为"2D"，"3D"，"CUBE"，但是不能指定为与纹理本身种类相矛盾的类型。纹理类型是"texture"，而 sampler 想要设定为"2D"以外的情况下，这个注解不能省略。

3.7.1.1.2. string ResourceName

指定纹理的图像文件。可以使用的图片格式包括 bmp、dds、dib、jpg、png 和 tga。如果使用相对路径来指定，那么起始目录为效果文件所在的文件夹。

3.7.1.1.3. int Width

指定纹理的宽度。

3.7.1.1.4. int Height

指定纹理的高度。

3.7.1.1.5. int Depth

指定纹理的深度。仅有在使用立体材质的情况下才需要指定材质深度。不能和 Dimensions、ViewportRatio 一起指定。(Width,Height,Depth)，Dimensions，ViewportRatio 全部都没有指定的情况下，纹理大小为 64 像素。如果指定了 ResourceName，那么纹理尺寸将从图像文件自动获得。

3.7.1.1.6. int2(or int3) Dimensions

指定纹理的宽度、高度和深度，单位是像素。仅有在使用立体材质的情况下才需要指定材质深度。不能和 ViewportRatio，Width，Height，Depth 同时指定。

3.7.1.1.7. float2 ViewportRatio

以比值的形式来指定纹理的尺寸（相对于渲染屏幕）。比如你想生成一个和渲染屏幕大小相同的纹理，那就使用"float2 ViewportRatio = {1.0, 1.0};"；如果你想生成两倍于屏幕大小的纹理，那就使用"float2 ViewportRatio = {2.0, 2.0};"。

不能和 Dimensions，Width，Height，Depth 同时指定。

3.7.1.1.8. string Format

指定材质的格式。省略时默认使用"A8R8G8B8"。如果指定了 ResourceName，就会自动从图片文件获得，并且忽略此注解。能够指定的格式参考 D3DFORMAT（https://docs.microsoft.com/zh-cn/windows/win32/direct3d9/d3dformat）。

"A8R8G8B8"、"FMT_A8R8G8B8"、"D3DFMT_A8R8G8B8"中的任何一种写法都可以。

3.7.1.1.9. int Miplevels

指定多级渐远纹理的级别。省略或设为 0 的情况下会生成完全的多级渐远纹理，设定为 1 则不生成多级渐远纹理。

3.7.1.1.10. int Levels

Miplevels 的别名。

3.7.1.2. 例子

```
texture negi_tex < string ResourceName = "negi.bmp"; >;
sampler TexSampler = sampler_state {
    texture = <negi_tex>;
};

texture2D map_tex <
    string ResourceName = "map.png";
    int Miplevels = 1;
    int Width = 64;
    int Height = 64;
>;
```

3.7.2. 帧缓冲

生成颜色缓冲或深度缓冲，以供着色器使用。类型为 texture 或 texture2D。语义可以指定为以下 2 个值:

- RENDERCOLORTARGET
- RENDERDEPTHSTENCILTARGET

通过 RENDERCOLORTARGET 声明的颜色缓冲可以被 Script 注解中的 RenderColorTarget 所引用，渲染结束后可以像正常纹理一样使用 tex2D()函数进行采样；通过 RENDERDEPTHSTENCILTARGET 声明的深度缓冲可以被 Script 注解中 RenderDepthStencilTarget 所引用，但是与 RENDERCOLORTARGET 不同的是，在渲染结束后将不能对其进行采样。

3.7.2.1.注解

3.7.2.1.1. int Width

3.7.2.1.2. int Height

3.7.2.1.3. int Depth

3.7.2.1.4. int2(or int3) Dimensions

3.7.2.1.5. float2 ViewportRatio

与通常纹理的相关注解含义相同。需要注意的是，在 ViewportRatio 缺省的情况下将会使用"float2 ViewportRatio = {1.0, 1.0};"作为变量。

3.7.2.1.6. string Format

指定纹理的格式。RENDERCOLORTARGET 默认为"A8R8G8B8"，RENDERDEPTHSTENCILTARGET 默认为"D24S8"。

3.7.2.1.7. int Miplevels

指定多级渐远纹理的级别。可以设定为 1 或者 0。设为 0 的情况下会生成完全的多级渐远纹理，设定为 1 则不生成多级渐远纹理。默认值为 1。RENDERDEPTHSTENCILTARGET 不可指定此项。

3.7.2.1.8. int Levels

Miplevels 的别名。RENDERDEPTHSTENCILTARGET 不可指定此项。

3.7.2.2.例子

```
texture2D ScnMap : RENDERCOLORTARGET <
    float2 ViewPortRatio = {1.0,1.0};
    int MipLevels = 1;
    string Format = "A8R8G8B8" ;
>;
sampler2D ScnSamp = sampler_state {
    texture = <ScnMap>;
};

technique Tech <
    string Script = "RenderColorTarget0=ScnMap;
    ..."
>;


texture2D DepthBuffer : RENDERDEPTHSTENCILTARGET <
    float2 ViewPortRatio = {2.0,2.0};
    string Format = "D24S8";
>;

technique Tech <
    string Script = "RenderDepthStencilTarget=DepthBuffer;
    ..."
>;
```

3.7.3. 动画纹理

生成动画纹理。类型为 texture 或 texture2D。语义可以使用以下值：

- ANIMATEDTEXTURE

默认情况下，动画纹理是和帧时间联动的。当然，它也可以和诸如控制对象之类的其它变量联动。

3.7.3.1.注解

3.7.3.1.1. string ResourceName（必须）

指定动画文件作为动画纹理的来源。可以使用的格式包括 gif（动画 gif）和 png（APNG）。

3.7.3.1.2. float Offset（可省略）

推迟动画的开始时间（单位：秒）例如，如果设定为 2.5，则动画会被 2.5 秒再开始。默认值是 0.0。

3.7.3.1.3. float Speed（可省略）

指定动画的播放速度。例如，如果设定为 2.0，那么动画将以两倍速播放。默认值是 1.0。

3.7.3.1.4. string SeekVariable（可省略）

当你希望使用时间以外的值来控制动画播放时，可以设置此注解。如果你指定了一个变量名字，那么动画将会和其联动。默认情况下联动的变量是 TIME<SyncInEditMode=true>。

3.7.3.2. 例子

```
// 动画播放与物体 seek.x 的缩放值联动

float atime: ControlObject < string Name = "seek.x"; >;

texture AnimeTex : ANIMATEDTEXTURE <
    string ResourceName = "anime.png";
    string SeekVariable="atime";
>;
```

3.7.3.3. 补充

如果 MMD 渲染的 FPS 低于动画纹理的 FPS，则动画会丢帧。对于 APNG 而言，姑且是可以播放以 GB 为单位的巨大动画文件的。

3.7.4. 离屏渲染

生成离屏渲染物体。类型为 texture 或 texture2D。语义可以使用以下值:

- OFFSCREENRENDERTARGET

如果生成了离屏渲染物体，则会自动按照指定条件在其上渲染全部帧的所有物体。渲染结果同样可以使用 tex2D()函数进行采样。

3.7.4.1.注解

3.7.4.1.1. int Width

3.7.4.1.2. int Height

3.7.4.1.3. int Depth

3.7.4.1.4. int2(or int3) Dimensions

3.7.4.1.5. float2 ViewportRatio

与通常纹理的相关注解含义相同。需要注意的是，在 ViewportRatio 缺省的情况下将会使用"float2 ViewportRatio = {1.0, 1.0};"作为变量。

3.7.4.1.6. string Format

指定纹理的格式。默认值为"A8R8G8B8"。

3.7.4.1.7. int Miplevels

指定多级渐远纹理的级别。可以设定为 1 或者 0。设为 0 的情况下会生成完全的多级渐远纹理，设定为 1 则不生成多级渐远纹理。默认值为 1。

3.7.4.1.8. int Levels

Miplevels 的别名。

3.7.4.1.9. float4 ClearColor

设定屏幕初始颜色。当开始渲染一帧时，系统会使用此值初始化整个屏幕。

3.7.4.1.10. float ClearDepth

设定深度缓冲初始值。当开始渲染一帧时，系统会使用此值初始化整个深度缓冲。

3.7.4.1.11. bool AntiAlias

指定是否开启抗锯齿。默认值为 false。开启抗锯齿的情况下会大量消耗显存，所以需要特别留意纹理尺寸。

3.7.4.1.12. string Description

指定离屏渲染物体的描述。此处设定的字符串会显示在分配效果的 GUI 的对话框上。

3.7.4.1.13. string DefaultEffect

指定离屏渲染物体所使用的效果文件的分配方法。如以下方式：

"(物体文件名)=(效果文件名);"

如果需要切换物体文件所使用的效果文件名，则需要声明多次。在这种情况下，系统会按照声明的顺序对物体文件名进行比较，并且调用第一个完全匹配的效果。

例：`string DefaultEffect = "self=hide; Mirror*.x=hide; *=MirrorObject.fx;";`

物体文件名中可以包含诸如*和?之类的通配符。如果使用特殊物体名 self，则此物体为持有离屏渲染物体效果的物体。如果使用相对路径指定效果文件名，则起始路径为此效果文件的路径。此外，还可以指定特殊的效果文件名"none"和"hide"。它们分别表示【无效果】和【不显示】。如果效果文件名指定为"main_default"，则会和主屏幕的默认动作一样，以物体文件的路径为起始路径来自动进行 fx 文件和 emd 文件的分配。

3.7.4.2.例子

```
texture MirrorRT: OFFSCREENRENDERTARGET <
    string Description = "OffScreen RenderTarget for Mirror.fx";
    int Width = 256;
    int Height = 256;
    float4 ClearColor = { 1, 1, 1, 1 };
    float ClearDepth = 1.0;
    bool AntiAlias = true;
    string DefaultEffect =
        "self = hide;"
        "Mirror*.x = hide;"
        "*=MirrorObject.fx;";
>;
```

3.7.5. 纹素

获取指定纹理的纹素并且放置在数组里。即使是在 VTF（Vertex Texture Fetching）不对应的情况下也可以在顶点着色器中获取到纹理的值（有限制，参考补充内容）。类型为 float4 的二维数组（[列长度][行长度]）或者一维数组。语义可以指定为以下值:

- TEXTUREVALUE

3.7.5.1. 注解

3.7.5.1.1. string TextureName（必须）

所使用的纹理的变量名。

3.7.5.2. 例子

```
float4 ParticleBaseArray[TEX_HEIGHT][TEX_WIDTH] : TEXTUREVALUE <
    string TextureName = "ParticleBaseTex";
>;
float4 ParticleBaseArray2[TEX_HEIGHT] : TEXTUREVALUE <
    string TextureName = "ParticleBaseTex2";
>;

float4 color1 = ParticleBaseArray[v][u];
float3 color2 = ParticleBaseArray2[idx].rgb;
```

3.7.5.3. 补充

因为要通过常量寄存器传值，所以可以引用的纹素的最大数量在 200 左右。如果设定的尺寸和图像尺寸不同，那么不能保证取得正确的纹素。特别需要注意的是，在一些环境中，如果设定的尺寸不是 2 的整数幂，那么数组尺寸会被自动扩大到最接近的 2 的整数幂。此外，纹理是在每一帧开始渲染是获得的。如果在渲染途中被更新了，那么直到下一帧开始渲染之前，数组的值都不会改变。

3.8. 效果文件

3.8.1. 版本

指定 SAS（Standard Annotations and Semantics）版本。此外，也用来指定一些与效果文件全局相关的注解。变量名必须为"Script"，类型为 float，值为 0.8（即版本编号）。可以使用以下语义：

- STANDARDSGLOBAL

3.8.1.1. 注解

3.8.1.1.1. string ScriptOutput（可省略）

不能指定为"color"以外的值。默认值为"color"。

3.8.1.1.2. string ScriptClass（可省略）

指定效果文件的目的（是用来渲染什么的）。可以指定为以下值：

- "object"：渲染物体（默认值）
- "scene"：渲染帧缓冲
- "sceneorobject"：以上两者

基本上来说，用来渲染一般物体的效果文件应指定为"object"，用来渲染预处理和后处理的效果文件应指定为"scene"。值指定为"object"的情况下，在 pass 的 Script 注解中不能声明 Draw=Buffer。值指定为"scene"的情况下，在 pass 的 Script 注解中不能声明 Draw=Geometry。值指定为"sceneorobject"则没有限制。

3.8.1.1.3. string ScriptOrder（可以省略）

指定效果文件被执行的时机。可以指定为以下值：

- "standard"：渲染物体时（默认值）
- "preprocess"：渲染物体之前（预处理用）
- "postprocess"：渲染物体之后（后处理用）

注：正确来说，postprocess 的预处理阶段（从 technique 的第一行 Script 开始，

直到"ScriptExternal=Color"为止）要比 preprocess 执行得更早。

3.8.1.1.4. string Script（可以省略）

指定要被使用的 technique 的检索顺序。通常情况下是根据效果文件中声明的顺序检索的（如 2.1 中所述）。但是使用这条注解可以显式指定检索顺序。值可以按照以下的方式设置：

"Technique=Technique ? technique 名 1 : technique 名 2: ~ ;"

例：`string Script = "Technique=Technique?SimplePS:TexturedPS:SimpleQuadraticPS:TexturedQuadraticPS;";`

此外，当你使用的 technique 只有一条的情况下，你也可以使用以下形式进行指定：

```
string Script = "Technique=MainTech;";
```

3.8.1.2.例子

```
//普通的效果文件的情况
float Script : STANDARDSGLOBAL <
    string ScriptOutput = "color";
    string ScriptClass = "object";
    string ScriptOrder = "standard";
> = 0.8;

//后处理效果的情况
float Script : STANDARDSGLOBAL <
    string ScriptOutput = "color";
    string ScriptClass = "scene";
    string ScriptOrder = "postprocess";
> = 0.8;
```

3.9. 特殊变量

3.9.1. 自动赋值的变量

以下列出的变量，在不指定语义的情况下会被自动赋值：

- parthf（bool 型）：对应本影的 mode1 和 mode2（false 为 mode1）
- spadd（bool 型）：spa 的合成方式（true 为加算）
- transp（bool 型）：半透明标志（显示->半透明化）
- use_texture（bool 型）：渲染中的材质如果使用纹理则为 true
- use_spheremap（bool 型）：渲染中的材质如果使用 spa 则为 true（包含 PMX 模型使用副 Tex 的情况）
- use_subtexture（bool 型）：PMX 模型使用副 Tex 的情况下为 true
- use_toon（bool 型）：渲染中的材质如果使用 toon 则为 true。PMD 模型恒为 true
- opadd（bool 型）：渲染中的物体使用加算合成时为 true
- VertexCount（int 型）：物体的顶点数
- SubsetCount（int 型）：物体的子集数量

3.9.1.1.例子

```
bool parthf;
bool use_texture;
bool use_toon;
int VertexCount;
```

3.9.1.2.补充

MMDPass="object","object_ss"以外的情况下，上述变量不保证正确。

3.10. 顶点着色器语义

3.10.1. 顶点下标

为了获得顶点的下标所使用的语义。顶点着色器的输入变量之一。类型为 int。可以使用以下语义：

- _INDEX

3.10.1.1. 例子

```
VS_OUTPUT Basic_VS(float4 Pos : POSITION, int index: _INDEX) {
    VS_OUTPUT Out;
    Out.Pos = mul( Pos, WorldViewProjMatrix );

    float f = (float)index/VertexCount;
    Out.Color = float4(f,f,f,1);

    return Out;
}
```

3.10.1.2. 补充

在需要把顶点下标转化为 float 型的情况下，顶点的下标如果超过了 2^24=16777216 将无法保证获取正确的值。

3.11. 宏

3.11.1. MME_MIPMAP

在启用了多级渐远纹理的情况下，这个宏会被声明。

3.11.1.1. 例子

```
sampler ObjTexSampler = sampler_state {
   texture = <ObjectTexture>;
#ifdef MME_MIPMAP
    MIPFILTER = LINEAR;
#endif
};
```

4. Script 注解（脚本）

对于 technique 和 pass 可以指定名为 Script 的特殊的注解。如：

```
technique Technique名 < string Script = "命令; 命令; ..." ; > { ... }
pass Pass名 < string Script = "命令; 命令; ..." ; > { ... }
```

执行时，这些命令会按照声明的顺序逐行执行。通过使用 Script 可以实现渲染对象的变更、清楚以及 pass 的循环处理等功能。如果不进行后处理等特殊效果的话，通常是不需要使用 Script 的。在 technique 的 Script 注解被省略的情况下，会单纯地按照 pass 被声明的顺序执行。此外，在 pass 的 Script 被省略的情况下，"Draw=Geometry"会作为被执行的 Script。

4.1. 命令

4.1.1. 颜色缓冲

指定要使用的颜色缓冲。可以使用以下命令：

- RenderColorTarget=(纹理名或空白)
- RenderColorTarget0=(纹理名或空白)
- RenderColorTarget1=(纹理名或空白)
- RenderColorTarget2=(纹理名或空白)
- RenderColorTarget3=(纹理名或空白)

其中，RenderColorTarget 是 RenderColorTarget0 的别名。通常会和 RenderDepthStencilTarget 一起使用。此外，RenderColorTarget1~3 不能被单独使用，必须和 RenderColorTarget0 一起使用。参数为通过 RENDERCOLORTARGET 语义声明了的变量名。需要重置为默认渲染目标时，参数应为空白。最后，设定完成的渲染目标直到 technique 结束之前都会保持不变。

4.1.2. 深度缓冲

指定要使用的深度缓冲（Z 缓冲）。可以使用以下命令：

- RenderDepthStencilTarget=(纹理名或空白)

通常和 RenderColorTarget0 一起使用。参数为通过 RENDERDEPTHSTENCILTARGET 语义声明了的变量名。需要重置为默认深度

缓冲时，参数应为空白。

4.1.3. 清除颜色

指定要使用的清除颜色（并不会立即进行初始化）。可以使用以下命令：

- ClearSetColor=(变量名)

参数为一个 float4 类型的变量名。设定了以后，这个变量的值将会被设置为渲染目标的清除颜色。

4.1.4. 清除深度

指定要使用的清除深度（并不会立即进行初始化）。可以使用以下命令：

- ClearSetDepth=(变量名)

参数为一个 float 类型的变量名。设定了以后，这个变量的值将会被设置为渲染目标的清除深度。

4.1.5. 执行初始化

执行初始化操作（使用清除颜色或清除深度）。可以使用以下命令：

- Clear=Color/Depth

4.1.6. 后处理

渲染其他的物体。这个命令无法在 technique 的 Script 上使用。这个命令只能在后处理（即使用 STANDARDSGLOBAL 语义声明的变量的注解中通过 ScriptOrder 指定的效果）中执行。可以使用以下命令：

- ScriptExternal=Color

通常，后处理是在输入的纹理设定为渲染目标后，再向其上渲染其它物体的流程。在渲染其他物体的时候，这条命令会被执行。（参考第 5 节中的后处理）。后处理在 technique 的 Script 执行之后必定只执行一次。

4.1.7. Pass

指定要执行的 pass。可以使用以下命令：

- Pass=(pass 名)

这条命令只能在 technique 的 Script 中使用。在 technique 使用了 Script 的情况下，如果不显式声明这条命令，那么所有的 pass 都不会被执行。

4.1.8. 循环

以指定的次数循环执行 Script 的一部分。可以使用以下命令:

- LoopByCount=(变量名)
- LoopEnd=

这条命令只能在 technique 的 Script 中使用。参数为数值型（int，bool，float）变量名。从 LoopByCount 到 LoopEnd 之间的命令将会被反复执行被设定的变量的值的次数。循环可以嵌套。

在以下的例子中，p0 在被执行了三次后，p1 才会被执行。

```
/*****************************/
int Count = 3;

technique TShader <
    string Script =
        "LoopByCount=Count;"
        "Pass=p0;"
        "LoopEnd=;"
        "Pass=p1;";
> {
/*****************************/
```

4.1.9. 循环计数

将循环中的循环计数器的值改变为指定的变量的值。这条命令只能在循环中使用。可以使用以下命令:

- LoopGetIndex=(变量名)

4.1.10. 渲染类型

渲染物体或者与渲染对象尺寸大小一致的一个长方形。可以使用以下命令:

- Draw=Geometry/Buffer

这条命令不能在 pass 的 Script 中使用。pass 的 Script 被省略时自动执行 Draw=Geometry。预处理和后处理时使用 Draw=Buffer。在 STANDARDSGLOBAL 语义的 ScriptClass 设定为"scene"时，Draw=Geometry 不执行；ScriptClass 设定为

"object"时，Draw=Buffer 不执行。

4.1.11. 例子

```
technique TShader <
    /* 如果两个字符串之间只有空白字符的话将会被识别为一个连续的字符串
       所以可以像下面这样分开写 */
    string Script =
        "RenderColorTarget0=RenderTarget;"
        "RenderDepthStencilTarget=DepthBuffer;"
        "ClearSetColor=ClearColor;"
        "ClearSetDepth=ClearDepth;"
        "Clear=Color;"
        "Clear=Depth;"
        "ScriptExternal=color;"
        "Pass=P0;" ;
> {
    pass P0 < string Script= "RenderColorTarget0=; RenderDepthStencilTa
rget=; Draw=Buffer;" ; > {
        ...
    }
}
```

5. Tips

5.1. 使用 MMD 的标准着色器

如果用户希望在一部分物体中使用效果文件的同时，另一部分物体使用 MMD 的标准着色器，那么，对于希望使用 MMD 的标准着色器的渲染流程下，效果文件中可以不指定 technique（参考 2.1 节 technique 的注解）。

```
/***********************************************************/
// 如果效果文件中只有这一个 technique，
// 那么在渲染物体（本影 ON）之外的情况都会使用 MMD 的标准着色器
technique Tech1 < string MMDPass = "object_ss"; > {
    pass Pass1 {
        ...
    }
    pass Pass2 {
        ...
    }
}
/***********************************************************/
```

此外、pass 默认使用的着色器就是 MMD 的标准着色器。如果像下面这样不指定顶点着色器和片段着色器的话，就会使用 MMD 的标准着色器。

```
/***********************************************************/
technique Tech1 < string MMDPass = "object_ss"; > {
    pass Pass1 {
      // VertexShader = xxx
      // PixelShader  = xxx
    }
}
/***********************************************************/
```

5.2. 空的 technique

如果把 technique 置为空的（不包含任何 pass），那么所有使用那个 technique 的物体都不会被渲染。利用好这一特性的话，就可以实现隐藏物体的一部分甚至全部。

```
/***********************************************************/
technique ShadowTech < string MMDPass = "shadow";  > {
```

```
}
/*****************************************************/
```

5.3. 根据物体的有无来启用/停用 pass

按照以下方式书写 Script 的话，就可以实现仅在特定物体的显示状态是 ON 的情况下执行指定的 pass。

```
/*****************************************************/
bool flag : CONTROLOBJECT < string name = "aaa.x"; >;

technique Tech1 <
    string Script =
        "LoopByCount=flag;"
        "Pass=Pass1;"
        "LoopEnd=;"
         ;
> {
    pass Pass1 {
        ...
    }
}
/*****************************************************/
```

5.4. 效果文件间共享变量

通常情况下，不同的效果文件之间的变量是不可共享的。但是，如果声明变量时使用了"shared"关键字，相同名字的变量可以被不同的效果文件共享。

以下的例子中，effect1.fx 生成的纹理可以在 effect2.fx 中调用。为此，两个文件中都需要使用"shared"关键字声明变量。此外，除了变量名，变量类型和语义也需要一致。

```
/************** effect1.fx ******************/

shared texture ShadowBuffer : RENDERCOLORTARGET <
    float2 ViewPortRatio = {2.0,2.0};
    int MipLevels = 1;
    string Format = "A8R8G8B8" ;
>;

/*****************************************/
```

```
/************** effect2.fx ******************/

shared texture ShadowBuffer : RENDERCOLORTARGET;

/*******************************************/
```

5.5. if 语句

为了高速渲染，在书写着色器时应尽量避免使用 if 之类的条件判断。如，相比于获取 use_texture 语义并使用 if 语句在着色器内对纹理有无进行判断，按照纹理有无创建不同的着色器进行渲染会更加高效。

5.6. uniform 关键字

函数的参数使用"uniform"关键字声明的情况下，在编译时会将其作为常数进行赋值。以下例子中，Basic_PS()中的 if 语句会在编译期被去除（因为条件是常量），其所带来的效率影响也将消失（大概）。

```
/*************************************************************/
float4 Basic_PS( VS_OUTPUT IN, uniform bool useTexture ) : COLOR0
{
    float4 Color = IN.Diffuse;
    if ( useTexture ) {
        Color *= tex2D( ObjTexSampler, IN.Tex );
    }
    ...
}

technique TechWithTex {
    pass P1 {
        /* useTexture=true 时被编译的着色器 */
        PixelShader  = compile ps_2_0 Basic_PS(true);
        ...
    }
}

technique TechWithoutTex {
    pass P1 {
        /* useTexture=false 时被编译的着色器 */
        PixelShader  = compile ps_2_0 Basic_PS(false);
```

```
        ...
    }
}

/**************************************************************/
```

5.7. 效果文件中的非 ASCII 字符集使用问题

基本上来说，除了在注释中，不建议使用非 ASCII 字符集的字符。如果非要用的话，一定要注意字符编码。生成纹理用的图片的完整路径名、控制物体的完整路径名以及效果文件的完整路径名中包含非 ASCII 字符集的字符的情况下，请务必根据所使用的 windows 的语言来修改效果文件的保存编码（如，简体中文必须使用 GB2312，繁体中文必须使用 Big5，日文必须使用 Shift-JIS，等等）。

5.8. 后处理

生成后处理用的效果文件的情况下，一定要在效果文件中声明以下变量（参考 STANDARDSGLOBAL 语义）。

```
float Script : STANDARDSGLOBAL <
    string ScriptOutput = "color";
    string ScriptClass = "scene";
    string ScriptOrder = "postprocess";
> = 0.8;
```

此外，对于典型的后处理效果，technique 的 Script 通常会执行以下步骤：

(1) 变更或清除渲染目标自身的纹理。

Script 样例：

```
"RenderColorTarget0=(RENDERCOLORTARGET 纹理);"
"RenderDepthStencilTarget=(RENDERDEPTHSTENCILTARGET 纹理);"
"ClearSetColor=(清除颜色);"
"ClearSetDepth=(清除深度);"
"Clear=Color;"
"Clear=Depth;"
```

(2) 预处理中对于物体和其他后处理效果的渲染。此时，渲染结果会保存在上个例子中指定的纹理。

Script 样例：

```
"ScriptExternal=Color;"
```

(3) 恢复渲染目标，并且把渲染结果的纹理作为输入来执行其他 pass。

Script 样例：

```
"RenderColorTarget0=;"
"RenderDepthStencilTarget=;"
"Pass=(pass1);"
"Pass=(pass2);"
```

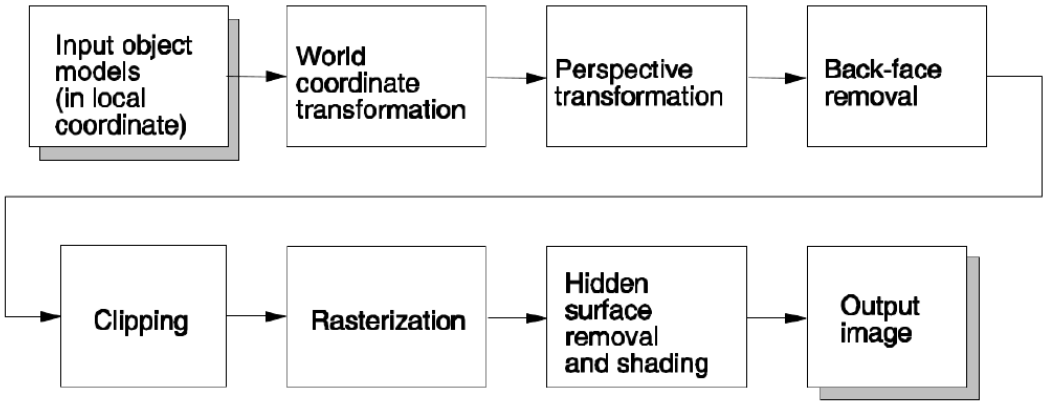
6. 附录

本节内容为译者对原文档进行的补充说明，并非原文档的内容。主要是关于着色器的一些基础知识与大致的渲染流程的说明。已经学习过计算机图形学的读者可以不看。

本节内容并不能代替微软或者 OpenGL 委员会或者其他任何专业指导，只能帮读者建立起一个大概的认识。本节所使用的图片主要来自译者的老师的课件以及 http://learnopengl-cn.github.io。虽然 OpenGL 与 DirectX 是完全不同的两套渲染框架，但是底层的机理是相同的。

本节会对一些复杂的或者不必要的内容进行省略，甚至跳过一些难以理解但是必要的步骤。毕竟，只要建立起大概的认识就足够了。

6.1. 渲染流水线



渲染流水线如上图所示，主要分为 6 个步骤：

(1) 世界坐标系变换：将顶点从局部坐标系变换到世界坐标系。顶点在模型中的坐标是相对于模型原点的（PMX 模型的原点一般在双脚中间），而模型本身在世界中又有一个相对于世界原点的位移。因此，这一步做的就是求出使用模型的局部坐标系的顶点在世界坐标系中的坐标。如果有摄像机的话，还需要根据摄像机的坐标进行一步视图变换，即，求出世界坐标系中的顶点相对于摄像机的位置。

(2) 透视变换：将上一步中的顶点坐标按照近大远小的透视法则进行正规化。透视变换会定义一个锥形的可视空间，然后将落在这个空间内的顶点按照近大远小的透视法则变换到一个长方体空间中。考虑一个大小一定的三角形面，显然，这个面距离视点越近，面三个顶点看起来彼此相距越远；反之，面距离视点越远，面三个顶点看起来相距越近。此乃“近大远小”。

(3) 背向面剔除：将背朝摄像机的面去除。这样做可以节省渲染的开销，因为一般情况下，背朝摄像机的面是不会被看到的。但是，如果存在透明面，那个

背向面就有可能被看到。DirectX 9 不具有次序无关透明度的功能，因此只能通过面排序来渲染透明面。不透明的先渲染，透明的后渲染。面排序并不总是有效的。事实上，有一段时间使用经历的用户都知道，MMD 的透明面渲染经常出问题。

(4) 面裁切：将屏幕外的面剔除，因为它们不会被看到。这一步对渲染速度影响十分大。需要注意的是，在使用全局光照算法、使用镜子或者使用其他具有反射光线能力的材质时，开启面裁切很容易造成错误的结果。

(5) 光栅化：将面映射到屏幕像素。这一步将确定每个面将会被渲染到哪些像素上。需要注意的是，对于某一个像素，结果只有“渲染”和“不渲染”两种，因此锯齿的来源全都在这一步。

(6) 隐藏面移除与着色：确定像素颜色，并且按照深度的顺序正确排列将要被显示的像素。在渲染中，大部分时候，一个像素会被多个面包含。这时候就需要深度缓冲来找出最贴近视点的那一个面，并且将像素的颜色设置为那一个面所渲染出来的颜色。

6.2. 渲染所需的信息

渲染流水线的基本输入是顶点，最终输出是像素。为了完成渲染流水线，有一些必要的信息要提供给渲染框架。一般情况下会需要这些信息：

(1) 顶点坐标：顶点的 XYZ 坐标。

(2) 三角形下标：每一个三角形面片应该由哪些顶点构成。一般这一项会交由渲染框架处理，在着色器中很少会用到它们。

(3) uv 与纹理：uv 是指定顶点的颜色位于纹理上的哪个位置的参数。纹理一般是二维纹理，即图像，所以只需要一横一纵两个坐标即可确定一个位置。如果要使用三维纹理，那么将要提供 uvw 坐标。一般情况下，一个顶点只有一套 uv 坐标，但是可以使用多张纹理。在不同的纹理上，这个顶点的采样位置相同。虽然从原理上讲，顶点可以包含多套 uv，但是实践中应尽量避免，因为有些硬件不支持。

(4) 切向（法向）：根据具体实现的不同，有些时候需要使用切向，有些时候使用法向。它们的作用主要是为了计算光照。提供了切向的场合可以计算出切线空间。切线空间是以顶点为原点所建立的坐标系。在切线空间中可以很方便地计算光线的入射与反射。

(5) 反射率：用于计算镜面反射。

事实上，根据需要，可以向渲染框架提供任何这之外的信息，或者省略其中的部分信息。比如有些输出纯色的 MME 就可以不要 uv 和纹理，某些使用金属度贴图的物体就可以不要反射率。只是一般情况下会需要这些信息。

6.3. 可编程着色器

效果文件所声明的对象就是可编程着色器。顾名思义，它们是用来声明如何对像素进行着色的。MME 可以声明两种着色器：顶点着色器（Vertex Shader）和片段着色器（Pixel Shader）。这里我沿用了 OpenGL 中对于 Fragment Shader 的翻译。如果读者见过翻译成像素着色器的，应该明白指的是同一样东西。

6.3.1. 顶点着色器

渲染流水线中有一步世界坐标系变换，这一步就是由顶点着色器所完成的。顶点着色器可编程就意味着用户可以按照自己的想法随意决定顶点的最终位置。如果用户按照世界坐标系变换矩阵乘视图变换矩阵乘透视变换矩阵的处理方式进行计算，那么得出来的结果就是透视正确的。如果用户按照其他方式，直接输出一个确定值，就可以实现将某一个物体的全部顶点显示在屏幕上的固定位置（当然，这没什么用）。如果用户直接输出顶点坐标，就可以实现按照模型的局部坐标系显示顶点位置，即，物体显示在屏幕上的特定位置（这在做 GUI 的时候很有用）。屏幕的具体范围可以参考获取鼠标位置那一节。

除了决定顶点位置，顶点着色器还有一项重要的功能是向片段着色器输送其他所需信息。顶点着色器在输入每个顶点时只调用一次，但是片段着色器会在渲染每个面的每个覆盖像素时都被调用。因此，顶点着色器输出的值会被插值。否则，片段着色器将无法获得正确的值。

顶点着色器除了必须输出顶点的坐标给渲染框架以外，其余的所有输出都会被插值后送进片段着色器。

6.3.2. 片段着色器

渲染流水线的最后一步，即着色与隐藏面剔除，也是可以编程的。这就是片段着色器的作用。片段着色器从顶点着色器获取信息，最后只输出顶点颜色。一般情况下，片段着色器会被用来进行光照计算，比如 Blinn-Phong 光照模型。但是，根据用户的需求，片段着色器可以输出任何信息作为顶点色。比如，用户可以通过指定片段着色器输出一个常数来实现纯色渲染，或者仅采样纹理颜色来忽略光照，或者通过计算将法线贴图转化为实际法线来检查表面凹凸程度，或者将最终颜色反向来输出底片……总之，片段着色器的输出将会影响像素颜色。对片段着色器的修改可以立竿见影，而且也是调试效果文件的一个重要途径。

深度缓冲不能在片段着色器中被访问，也不能被修改，所以无法通过对片段着色器编程来实现某个物体总是位于最前的效果。但是，深度测试有一些设置项可以使用。借助它们可以实现上述功能。

6.4. 多次渲染与后处理

GPU 之所以高效，是因为它是并行的。从之前的描述中，我们可以得出一个结论：片段着色器被调用的次数远远大于图像中像素的个数（因为会有多个面覆盖同一个像素的情况）。对于一张 1920*1080 分辨率的图像来说，其像素数量可以达到二百万以上。对于 60FPS 的画面来说，每秒钟需要渲染超过一亿两千万个像素，片段着色器被调用的次数更是多得吓人。为了实现这种性能需求，GPU 的架构是并行的，即，每时每刻都有成百上千个单元在分别渲染不同的像素。在这这种架构下，当用户希望访问相邻像素的值时，很有可能这个像素还没有被计算出来。这会产生一些明显的缺陷。

比如，现实生活中，当我们观察灯光时，会发现灯光周围有一圈光晕。在这里我们不讨论它产生的原理，只讨论它的成像特征。这种光晕会出现在明亮的光源周围，并且随着远离光源位置而逐渐衰减亮度，直至消失。如果想要通过片段着色器实现这种效果，显然是不行的，因为每一个像素在被渲染时不能访问它周围的像素，即，无法获知自己与光源的距离。

对于这种情况，一般会使用一种名为后处理的技术来实现。后处理，顾名思义，在第一遍渲染之后进行的处理。将第一遍的渲染结果保存在一张纹理上，并且将其作为输入交由渲染流水线进行第二遍渲染，这就是后处理。由于在第二次渲染中，第一次渲染的结果已经被以纹理的形式固定下来，因此每一个像素可以通过访问纹理获得周围像素的值。

需要注意的是，后处理所用到的算法一般不被认为是渲染技术，而被认为是图像处理技术，即，这些算法通常会由计算机视觉课程详细解释。

6.5. 一些常见效果的实现思路

6.5.1. 阴影

我们都知道，GPU 无法直接计算光路（不然光线追踪早就实装进游戏了），而游戏中却随处可见动态阴影。一般计算动态阴影的方法是，从光源处采样一张深度贴图。读者可以将其理解为：从光源处向周围发射光线，并且记录下每根光

线第一次碰撞的物体与光源的距离。当进行渲染时，通过比较当前点与光源的距离和光源在此方向上的深度贴图的值，即可获知当前点是否是从光源发出的光线第一个碰撞的物体，也就可以得知当前点是否位于光源的阴影中。

显然，这种做法对于透光的遮挡物体是无效的。此外，受制于深度贴图的精度与尺寸，距离光源较远的地方容易形成锯齿阴影。

6.5.2. 泛光

泛光即上文提到的，光源周围的光晕。一般的实现方法是，在后处理过程中对于渲染结果进行亮度钳制，查找出那些较为明亮的像素，然后对其进行高斯模糊以实现光晕范围。最后，将模糊的结果叠加在原始的渲染结果上。

6.5.3. 自发光

自发光材质并不会真的发光，只是通过泛光使其看起来好像在发光而已。

6.5.4. 线稿

常用做法是分别对深度缓冲和渲染结果逐像素求一阶偏导，筛选出那些变化

非常大的像素作为边缘，然后对两者的结果进行加权平均来考虑这个像素究竟是不是边缘。当然，这个算法的效果不是很好。

另一种稍微复杂一点的办法是先计算渲染结果的灰度，然后对灰度图逐像素求梯度和方向，并且对梯度进行筛选。经过这一步处理后的图像还需要进行非极大值抑制，即，根据梯度方向对查询附近最大的梯度值，并且舍弃那些局部次优值以获得精确的边缘。

6.5.5. 波纹

波纹一般可以认为是正弦波。正弦波的表达式是 A*sin(2π/L*t+φ0)，即可以使用振幅、波长和初始相位来控制正弦波。随着 t 的变化，平面上每个点的值也会跟着改变。我们可以简单地将这个值套用在 uv 的变化上来实现一个看起来差不多的波纹。如果想要精益求精，还需要考虑到能量的衰减对振幅的影响以及使用菲涅尔方程来真正计算 uv 的偏移量。

6.5.6. 洋面模拟

简单的洋面可以使用复数的水波的随机叠加制成。然而，想要真实，去学流体力学和傅里叶变换。

6.5.7. 弥散圆

弥散圆指的是物体没有被刚好投射在底片上时所产生的一种圆形的模糊。为了模拟这个效果，可以使用每个像素渲染的点的距离计算出当前像素的弥散圆的尺寸，然后把颜色分布在整个弥散圆所覆盖的所有像素内。当所有像素都被渲染结束后，累加的结果就相当真实了。

当然，如果真的这么实现了，效率会非常低。另一个比较讨巧的办法是，将正常的渲染结果进行高斯模糊，然后利用深度缓冲在清晰和模糊的两张图像之间线性插值来模拟这种效果。

你也可以把这种效果叫做景深。

6.5.8. 镜子

从与镜子对称的位置重新渲染一次场景。