# GfxCommon库 说明文档

张亚川

目录

[GfxCommon库 设计与使用 1](#_Toc334700431)

[GFX库情况简介： 1](#_Toc334700432)

[CECModel类的设计意图： 2](#_Toc334700433)

[A3DGFXEx类的设计意图： 2](#_Toc334700434)

[A3DSkillGfxEvent/A3DSkillGfxComposer类的设计意图: 3](#_Toc334700435)

[物理集成相关类的设计意图： 3](#_Toc334700436)

[GFX库主要模块的详细说明： 3](#_Toc334700437)

[CECModel 3](#_Toc334700438)

[存储与运行，动态实例与静态数据： 3](#_Toc334700439)

[动作播放与动作队列： 4](#_Toc334700440)

[脚本回调与换装： 4](#_Toc334700441)

[Tick与SyncModelPhys 5](#_Toc334700442)

[模型的注册渲染机制与MotionBlur 6](#_Toc334700443)

[特殊需要注意的函数： 7](#_Toc334700444)

[A3DGFXEx 8](#_Toc334700445)

[特效GFX的渲染批次 8](#_Toc334700446)

[特效元素A3DGFXElement： 9](#_Toc334700447)

[A3DGFXElement的注册渲染机制： 15](#_Toc334700448)

[GFX渲染容器GfxRenderContainer： 15](#_Toc334700449)

[采用自定义HLSL与shader常量动画： 15](#_Toc334700450)

[A3DGFXExMan介绍 16](#_Toc334700451)

[A3DSkillGfxComposer/A3DSkillGfxEvent 17](#_Toc334700452)

[与编辑器导出文件相关的接口 18](#_Toc334700453)

[与客户端的接口 19](#_Toc334700454)

[替换Afx系回调函数，A3DGFXInterface 19](#_Toc334700455)

[客户端加载CECModel的回调接口IGFXECMInterface 20](#_Toc334700456)

[通过客户端逻辑直接控制部分元素 20](#_Toc334700457)

# GFX库情况简介：

GfxCommon库目前分为2.0，2.1，2.2，3.1四个分支，其中2.0和3.1的版本较少维护，2.1和2.2共享同一份代码文件，分别用于笑傲(2.1)和圣斗士(2.2)项目。

2.0的GFX库位于mmorpg vss的$/GfxCommon

2.1和2.2的位于$/Common/GfxCommon2

3.1的位于$/GfxCommon3

GFXCommon是一个位于客户端和引擎中间的库。对客户端提供了CECModel类和A3DGFXEx类作为主要的功能模块。辅以ModEditor编辑器和GfxEditor编辑器，提供给策划和美术创作出.ecm文件和.gfx文件用于客户端项目中。

主要模块介绍CECModel，A3DGFXEx，A3DSkillGfxEvent

## CECModel类的设计意图：

引擎中的A3DSkinModel类封装了骨骼模型，包含了子模型、骨骼动画以及蒙皮的功能，其功能较为底层，而客户端中则需要将模型动作，特效，音效，脚本等按策划和美术的需求组合起来使用，直接使用较为不便，因此需要增加一层封装：CECModel。

A3DSkinModel类所含的动作是美术直接从max中导出，经过SkeletonEditor.exe合成过后的版本（A3DSklTrackSet对应.stck文件，A3DSkinModelActionCore对应基本动作）。

CECModel将A3DSkinModel的基本动作加以组合、封装，增加了循环、衔接、附加特效、音效、脚本等功能，从而提供了组合动作(**A3DCombinedAction**)的概念。

此外，CECModel还可以在编辑器中配置挂载的特效GFX，设置骨骼缩放等等。

上述的类间关系可以参考附录中的：**ECModel&GFX原理与应用.pptx文档**。

除了上述基本功能之外，CECModel后续随着需求的变化，还增加了：

1. 子模型管理，ECM可以通过挂点挂载其他ECM

2. 脚本回调函数，在动作起始、结束、加载完毕、换装等时刻可以执行策划和美术编写的逻辑。

3. 模型上的物理数据**A3DModelPhysSyncData**（对应.mphy文件），以及皮肤(**A3DSKin**)上的布料物理数据(对应.sphy文件)文件。

4. MotionBlur渲染等。

## A3DGFXEx类的设计意图：

在游戏中表现爆炸，烟雾，光环，刀光等特殊效果的时候需要通过特效系统来表现。

A3DGFXEx封装了包括粒子在内的各类特效元素，包括粒子，轨迹，面片，环，电弧等基本元素，以及子效果类**A3DGFXContainer**提供层级关系，通过傀儡Dummy的概念可以让粒子系统喷射的每一个粒子都是另一个特效元素**A3DGFXElement**，将子效果和傀儡组合起来共同起作用的话，可以让粒子系统喷射出的每个粒子都是另一个完整的GFX。

## A3DSkillGfxEvent/A3DSkillGfxComposer类的设计意图:

游戏中需要表现技能的飞行和击中效果，通过将一个A3DGFXEx对象从发出者的位置移动到目标位置来实现飞行，当飞行特效击中目标后，在目标身上播放一个击中特效，这一过程通过**A3DSkillGfxEvent**类来维护，而**A3DSkillGfxComposer**类则负责保存和读取编辑好的数据。

## 物理集成相关类的设计意图：

**A3DModelPhysics**用于协助在**CECModel**中使用物理功能，封装了切换RagDoll状态，开启关闭Cloth布料模拟，切换半物理混合状态，物理加力等功能。

**A3DModelPhysSyncData**类用于存放编辑好的模型相关物理数据，包括刚体、约束、Attacher、破碎属性等。

**A3DSkinPhysSyncData**类用于存放将**A3DSkin**中的部分Mesh被编辑为布料后的物理数据。

**A3DModelPhysSync**类和**A3DSkinPhysSync**是上述两个Data类的运行时实例。

# GFX库主要模块的详细说明：

## CECModel

### 存储与运行，静态数据与动态实例：

编辑器编辑保存的数据位于**CECModelStaticData**类中，**CECModelMan**会根据.ecm路径对静态数据做管理，多份**CECModel**实例可以共用同一份静态数据。包括**A3DCombinedAction**，**EVENT\_INFO**等数据，都存储在**CECModelStaticData**类中的，避免同一模型被new多份时附带有冗余的数据。

动态实例与静态数据的概念在组合动作，动作事件上也有用到：

**A3DCombinedAction** 与 **A3DCombActDynData** 也是一堆静态数据与动态实例的关系，其中**A3DCombinedAction**是在编辑器里编辑出来的，到播放动作的时候则会动态创建一个**A3DCombActDynData**对象。

类似的，动作事件的EVENT\_INFO的若干子类与其动态实例也有类似的关系：

**GFX\_INFO**与**GFX\_BINDING**，

**SFX\_INFO**与**SFX\_BINDING**，

**AUDIOEVENT\_INFO**与**AUDIOEVENT\_BINDING**，等等。

上述BINDING对象都是在对应的\*\_INFO::Start()函数中创建的，随着**A3DCombActDynData**的Tick而Tick。

与此相同，物理数据中也有类似的关系，**A3DModelPhysSyncData**与**A3DModelPhysSync**；**A3DSkinPhysSyncData**与**A3DSkinPhysSync**。

### 动作播放与动作队列：

PlayActionByName可以将动作播放到某个特定的Channel上，QueueAction可以在指定的Channel中追加当前一个动作播完之后继续播放的动作。

动作播放完成的判定：由于动作有可能无限循环，而且动作上可能有时间长于动作本身的GFX，因此，在使用中需要指定动作完成的判定方式，例如是否要等待GFX完全播放完才认为是动作结束。通过SetComActFlag函数可以设置**COMACT\_FLAG\_MODE\_ONCE\_IGNOREGFX**或者**COMACT\_FLAG\_MODE\_ONCE\_MULTIIGNOREGFX**。

通过PlayAttackAction播放的动作，默认已经设置了COMACT\_FLAG\_MODE\_ONCE\_IGNOREGFX标志，从而判定合适结束时，不考虑GFX的影响，而是只考虑动作本身。

### 脚本回调与换装：

CECModelStaticData中保存了一个回调脚本表（通过ecm编辑器编辑），在以下时刻会回调对应脚本：

动作播放enumECMScriptStartAction，

动作停止enumECMScriptEndActioin，

文件第一次加载enumECMScriptInit，

模型实例销毁enumECMScriptRelease，

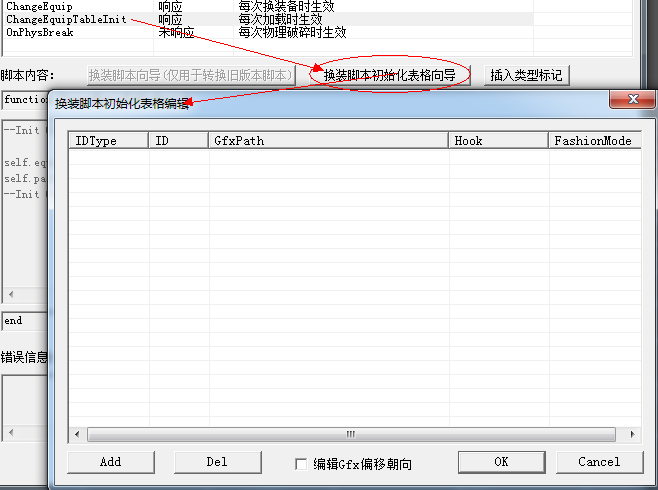
(文件加载）换装脚本表格初始化enumECMScriptChangeEquipTableInit，

模型实例加载enumECMScriptModelLoaded，

换装enumECMScriptChangeEquip，

物理破碎enumECMScriptPhysBreak。

其中换装脚本表格初始化这一回调（OnScriptChangeEquip），是为了把换装脚本的装备id与对应的特效在初始化阶段就做好对应关系，而换装时刻可以通过查表找到需要更换的特效。在ecm编辑器中，有一个专门的编辑界面设置id与特效，而后会根据编辑好的数据自动生成对应表。



### Tick与SyncModelPhys

Tick的速率可以被SetPlaySpeed设置的速度改变。

除此之外，GetSkinModelTRCtrl() 取得的**SkinModelTRControl**结构也维护了一个速度，该速度在PlaySpeed的基础上，再对SkinModel的Update速率产生叠加影响（相乘）。

**SkinModelTRControl**用于处理设置于特效中的模型TickSpeed属性以及是否隐藏模型的逻辑。

增加了物理之后，由于模型可能进入物理驱动的状态，因此在物理引擎Simulate之后，模型的骨骼可能需要根据其绑定的Actor的姿态做调整，因而需要有一个SyncModelPhys调用。

CECModel::Tick

APhysXEngine::Simulate

CECModel::SyncModelPhys

形成上述调用顺序。

由于物理模拟后可能会对半物理状态或纯物理的模型骨骼位置产生影响，因此在物理版本的GFX库中(目前仅在Angelica 2.1的版本中)，我们将模型身上挂载的GFX的位置更新挪动到了SyncModelPhys函数中，放置于TickRelyOnPhysics中。

### 模型的注册渲染机制与MotionBlur

SkinModel的渲染是注册渲染机制，CECModel的Render也只是调用SkinModel的Render，这里只注册，不渲染（除非指定RenderAtOnce参数），而注册渲染机制中并没有将位置信息和渲染的实体打包绑定，只是注册了SkinMesh的指针，因此无法做到对CECModel使用做SetPos，Render，再SetPos，Render的方式去注册多份渲染实体。（未来的设计中应当对这类需求予以考虑）

在CECModel的Render中，会检查当前模型是否触发了残影效果，如果触发了，则会将自身指针注册到A3DGFXExMan类中。在A3DGFXExMan的RenderECMMotionBlur函数中，会依次对其中注册的所有CECModel做处理：首先检查当前帧渲染的过程中，此CECModel中的A3DSkinModel是否已经被渲染过（有可能当前视锥并没有看到此模型，这种情况下我们应当避免渲染残影）。如果渲染过，那么将当前帧的A3DSkinModel中的Transposed Matrix记录下来，如果当前帧没有渲染，则清空以往记录的Transposed Matrix。

接下来，便来当前已经记录的Transposed Matrix列表，依次将该矩阵赋予A3DSkinModel，设置颜色和透明度，设置AlphaBlend模式，并最终调用RenderAtOnce渲染该模型。

残影的实现是通过多次渲染同一个模型实现的。因此**残影数目越多，就相当于要渲染越多个数的模型**。

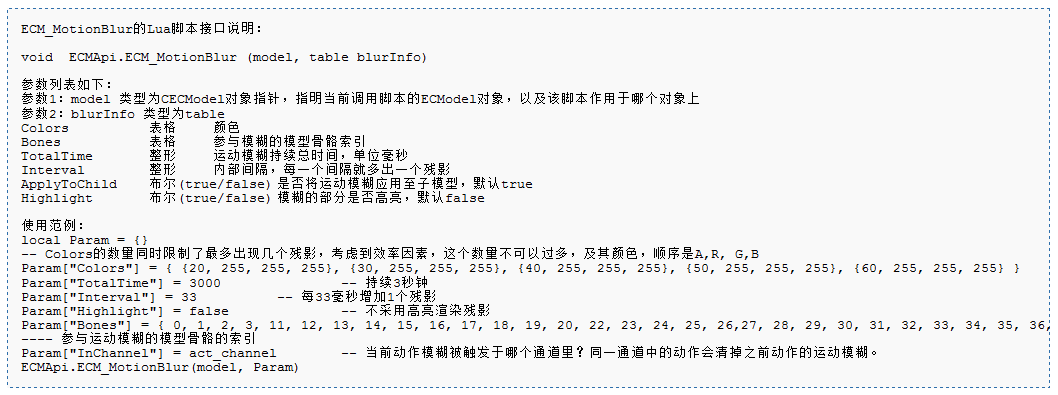
而且由于残影是采用AlphaBlend的模式将自身像素混合到已经渲染完成的帧缓冲上，因此**渲染残影比渲染普通模型耗费的会更多**。

后来因为增加了替换HLSL调用RenderAtOnce的渲染模式，美术可以通过MotionBlur的脚本接口指定选用哪种HLSL效果。

MotionBlur的用法可以参考\\duyuxinati上的Wiki页面：

http://10.64.96.26/wiki/index.php/ECM\_MotionBlur

或者附录中的：**ECM动作条脚本事件说明以及相关API** 文档



MotionBlur中的已知问题：

1. 为什么客户端中渲染残影时，出现不合理的随摄像机位移的错误效果：客户端中有一个地方调用了A3DSkinMan::[SetWorldCenter](http://10.64.96.26/wiki/index.php/SetWorldCenter)函数，这会导致残影随摄像机移动。如果要使用残影，不可以使用该函数，应当将其置为Player所在位置，或者是每隔若干帧更新一次摄像机所在位置，而不要每帧都变动，也不要简单的设置为A3DVECTOR3(0.0f)，因为在某些巨大世界中，这可能导致骨骼变换出现较大的浮点误差。

2. 为什么同一个模型，编辑器中的残影看起来是半透的（正确），而客户端中的残影看起来不是半透的：检查客户端的vs与ps，有可能A3DSkinModel的Transparent没能正确影响输出的像素(输出的像素Alpha必须受到Transparent影响，才能与当前帧缓冲正确混合)。

### 特殊需要注意的函数：

**1. SetGfxUseLOD**，设置是否对粒子特效使用随距离LOD功能。

在某些游戏的商城界面里绘制的模型，由于采用了不同于场景中的摄像机，因此摄像机到GFX的距离很可能超出了LOD的最大范围，以至于根本不产生粒子，因而对于该类模型，我们使用该函数关闭粒子LOD。

**2. SetCreatedByGfx**，GFX库中有一个特效元素会创建CECModel，通过GFX创建出的CECModel对象会设置该标志，该类CECModel会禁止播放Att。

（GFX类上也有一个类似的IsCreatedByGFXECM函数，说明该GFX是由GFX创建的ECM创建出来的，对于该类GFX，禁止再在CECModel类型元素中加载模型）

**3. SetId**，设置CECModel的id，客户端的每个玩家、NPC都有id，CECModel和A3DGFXEx对象都有一些需要用到客户端id的地方，例如GFX内部有需求会用到id回调客户端获取模型，设置是否显示，是否改变Tick速度，以及攻击效果里通过id获得目标位置等等。

**4. ScaleAllRootBonesAndGfx**，由于SkinModel没有提供缩放接口，因此需要设置所有根骨骼的缩放，该函数同时设置根骨骼的缩放与GFX的缩放。

**5. SetColor/ GetColor**，设置、获取模型的颜色，在2.1和2.2中由于模型色彩要牵扯到最终的材质shader是否支持，如果发现设置Color失效的情况，有可能是因为SkinModel材质尚未支持。另由于CECModel的动作条上存在的颜色改变类型事件，内部也会设置颜色，与外部设置的SetColor乘起来共同得到最终的色彩，因此GetColor得到的可能与SetColor并不一致。

#### A3DCombinedAction组合动作中的事件类型

**A3DCombinedAction**类中包含一系列的**EVENT\_INFO**对象，表示组合动作上的事件，会按时间触发。

EVENT\_INFO派生出若干子类，包括：

**1. GFX\_INFO**，特效事件点

**2. SFX\_INFO**，音效事件点 （老的音效，使用AMSoundBuffer）

**3. AUDIOEVENT\_INFO**， 音频事件点，新的音效，使用AudioEngine

**4. SGCAttackPointMark**，攻击事件点

**5. MaterialScaleChange**，模型颜色改变事件

**6. ChildActInfo**，子模型动作事件（子模型动作里有对老版本的刀光的支持，通过m\_bIsTrail开启，通过预计算出轨迹经过的位置并存储于ECM文件中，得到较为平滑的轨迹）

**7. GfxScriptEvent**，脚本事件点，在脚本事件中可以调用的Api，可以参考附录的文档：**ECM动作条脚本事件说明以及相关API**

**8. CameraControlEvent**，摄像机控制事件，编辑好后，可以随着动作播放给出一系列摄像机相对于CECModel的朝向和位置点，客户端根据取得的位置与朝向设置摄像机即可获得与编辑相同的效果，该类事件在实际使用中并不好用，美术编辑起来很不方便，应考虑制作更完善的摄像机控制方案。

**9. BoneScaleChange**，骨骼缩放变换事件，可以在动作播放的过程中修改某些骨骼的缩放，动作播完后会重置。

## A3DGFXEx

### 特效GFX的渲染批次

GFX是注册渲染的，注册渲染分两个层次：

第一个层次是**在游戏World中收集Gfx对象**，这一步通过A3DGFXExMan::RegisterGfx和GfxRenderContainer实现。

第二个层次是**对收集于的GfxRenderContainer中的对象依次调用A3DGFXEx::Render，将其中的A3DGFXElement注册到A3DGFXRenderSlotMan中**，这一步在调用RenderAllGfx，Render2DGfx时都会触发。 （其中RenderAllGfx绘制的是场景中的Gfx，Render2DGfx绘制的是拥有2D属性的Gfx）

在整个场景都渲染完成之后，我们得到了一个场景中的FrameBuffer，对于这个FrameBuffer，会再有下述若干个批次：

1. 空间扭曲 SpaceWarp 批次：这一步实现在客户端中，会调用GetWarpEleList取得当前注册到的空间扭曲元素，然后依次渲染扰动FrameBuffer

（这一步理应封装为一个A3DGFXExMan的函数，供客户端调用即可）

2. DelayedRender批次，这一步主要是为了自定义HLSL中使用了FrameBuffer的RenderTarget的那些元素，主要是某些A3DSkinModel的特殊效果，如冰冻折射的效果

特殊的：特效中的Bloom没有单独的批次，而是在正常的渲染流程中，使用MRT的方法绘制到Bloom RenderTarget上。

参考A3DGFXRenderSlotMan::PreRender中，Bloom RenderTarget被设置成index为1的RenderTarget。 HLSL中如果是要写Bloom的ps，会直接把需要渲到Bloom RenderTarget上去的元素绘制上去。

### 特效元素A3DGFXElement：

**A3DGFXElement**是所有类型特效元素的基类，他主要定义了以下功能接口：

#### 1. 存储相关：

Save, Load

从文件读取GFX元素时，是通过预定义的ID做Runtime的创建的：

A3DGFXElement::CreateEmptyElement 就是创建各类元素的工厂方法。

#### 2. 渲染相关：

Render 函数注册特效元素到渲染管理器的队列中

GetVertsCount函数用于返回该元素需要预留多少顶点的空间，渲染管理器依此决定留存多少buffer给当前元素填写。

FillVertexBuffer函数是渲染管理器填充顶点缓冲的回调，大多数特效元素都需要实现该函数用于填写顶点缓冲。

RenderSkinModel对于SkinModel类型的特效元素的特殊渲染方法，该类型元素也是先被注册到渲染管理器中，而后依次调用该函数绘制。

#### 3. 编辑中用于获取属性或者设置属性的接口：

所有元素都有的Property，通过以下接口编辑：

SetProperty, GetProperty 通过属性id设置属性值

此外，粒子系统中有一系列粒子效果影响器，用于在Tick中Update粒子的位置，颜色，透明度等，通过以下接口编辑：

GetSubPropertyCount, AddSubProperty, RemoveSubProperty, SetSubProperty, GetSubProperty

特效各个元素的ID及其属性ID被定义在A3DGFXElement.h头文件中。

在A3DGFXEditorInterface.cpp文件中针对每类对象都定义了一个元表，表中包括属性ID和属性名称，用于在编辑界面的属性表中自动绑定A3DGFXElement中的各个属性。（遍历对象属性元表，取得其中的ID，根据ID调用对应A3DGFXElement对象的SetProperty和GetProperty函数）

例如：

static const GFX\_PROP\_INFO \_prop\_model\_proxy\_info[] =

{

{ "Alpha模式", ID\_GFXOP\_ALPHA\_MODE },

{ "是否隐藏客户端模型", ID\_GFXOP\_MODELPROXY\_HIDECLIENTMODEL},

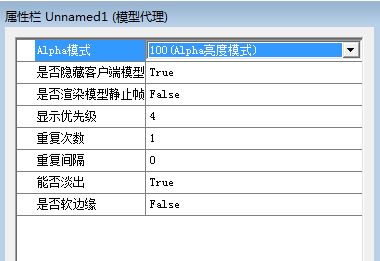
{ "是否渲染模型静止帧", ID\_GFXOP\_MODELPROXY\_USESTATICFRAME },

{ \_prop\_base\_info[13].m\_szName, \_prop\_base\_info[13].m\_OpId }, // 优先级

GFX\_REPEAT\_INFO,

};

对应的属性表：



#### 4. 外部属性设置接口：

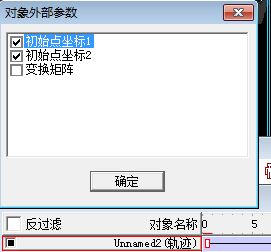
IsParamEnable, EnableParam, BeginUpdateParam, NeedUpdateParam, UpdateParam, EndUpdateParam, GetParam, GetParamType等

特效元素的外部属性，在**CECModel**的GFX\_INFO事件中创建的特效，可以在Update的时候将一些外部参数通过上述UpdateParam接口传递到特效元素当中，例如：轨迹的两个端点，每帧都会根据挂点位置Update其位置和朝向信息。

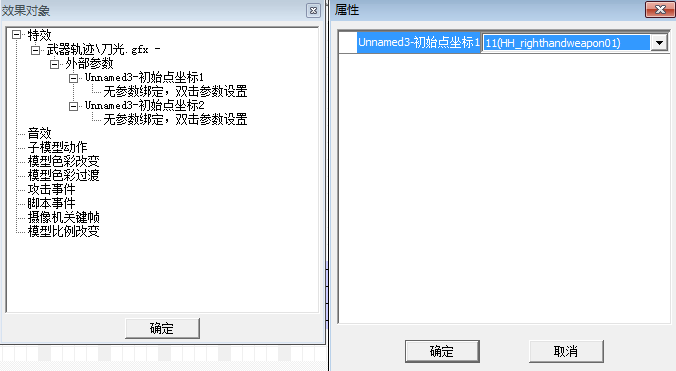
这部分代码参考**GFX\_BINDING::UpdateParam** 函数中对特效元素的外部参数更新。

外部属性可以在编辑器中设置：

GFX编辑器中：



ECM编辑器中（通常，如果使用外部参数，可以把轨迹挂在武器上，用武器的两个挂点决定轨迹的宽度）：



#### 5. HLSL相关编辑接口

GetHLSLInfo, GetAnimationTrack, GetAnimatableProperty等

特效元素可以使用HLSL的ps，并且可以设置ps常量的关键帧内容。

#### 6. Dummy的概念

A3DGFXElement可以被设置为傀儡Dummy元素，设置为傀儡的特效元素可以被粒子发射器当成粒子喷出来，并像粒子一样运动。结合Dummy与A3DGFXContainer元素就可以把一个子效果GFX喷射成粒子。

#### 6. 空间扭曲元素

特效元素可能被渲染到空间扭曲效果中，该类元素在Render中需要被注册到扭曲队列中，渲染的时候是回调的DrawToBuffer函数。[该类元素在渲染空间扭曲的批次里被绘制。](#_特效GFX的渲染批次)

#### 常用的各个元素类型

A3DDecalEx 面片元素，支持2D，3D面片，紧贴地面，使用地面法线等功能。

地面法线和紧贴地面的属性都需要通过回调客户端获取特定信息，地面法线通常只取地形的法线，紧贴地面则是既考虑地形的网格也考虑场景静态模型的网格。

A3DGridDecalEx网格面片元素，支持N\*M格的面片，可以由美术自由编辑网格的形状。

A3DGFXGridAnimation为网格面片元素提供了动画功能。

A3DTrail轨迹，通过两个端点的运动划出一个条带。

A3DTrailEx新轨迹，支持Bloom（Bloom元素需要在Bloom的批次中渲染），支持缩短轨迹长度的消失方式。

A3DParticleSystemEx粒子系统，管理粒子的元素，依据发射器的不同属性，可以形成不同类型的粒子系统（盒型发射器，椭球形发射器，骨骼发射器等）

A3DLightning 电弧，设定起始和末端端点之后，在两点间产生随机条带的效果，同时支持美术拉出一条曲线产生电弧效果。

A3DLightningEx 高级电弧，支持从一端逐渐产生电弧，顶点生命期，以及末尾淡出等效果。

A3DRing 圆环元素，可以产生带有一定张角的圆环。

A3DGFXModel代表一个产生A3DSkinModel模型的元素

A3DGFXECModel代表一个产生CECModel模型的元素，产生的CECModel对象可以完全由GFX来指定ecm文件路径(**NormalLoader**)，也可以是通过gfx的id与用户指令交给客户端决定加载哪个ecm文件(**CustomECMLoader**)，也可以是由GFX加载了指定路径的CECModel对象之后，再由客户端加载该模型的皮肤(**CustomSkinLoader**)。

**注意：使用该类A3DGFXECModel元素的客户端，必须实现IGFXECMInterface接口，并将该接口设置到A3DGFXExMan中，其中的OnLoadCustomECModel函数交给客户端加载模型，而OnLoadCustomSkin则是已经加载了模型，交给客户端取加载皮肤。**

A3DGFXModelProxy 模型代理，代表一个客户端的ECM模型，可以控制原始模型的隐藏、显示，可以使用一个指定的HLSL (ps)多渲染该模型一遍。

A3DGFXContainer 子效果，Play的时候加载一个GFX文件，并使用元素自身的位置朝向去更新加载来的GFX对象。

A3DPhysFluidEmitter物理流体粒子元素，粒子的运动交给物理模拟进行控制，从物理模拟的结果获取粒子的朝向和位置。

A3DPhysPointemitter物理点状粒子，利用了DX9的特性采用POINTSPRITE渲染粒子。

#### 需要特殊注意的元素和特殊用法：

**A3DDecalEx**类可能支持紧贴地面的属性m\_bMatchSurface，这种情况下的渲染，需要回调一次客户端，取得指定位置和半径范围内的顶点与索引，然后立刻渲染出来。这个流程**与其他元素注册渲染的流程不一致，可能导致美术设置的层次关系失效**。（3.1中修正了这个问题，将该类渲染也纳入到RenderSlot中从而保障了层次关系顺利起作用，2.1和2.2尚未）

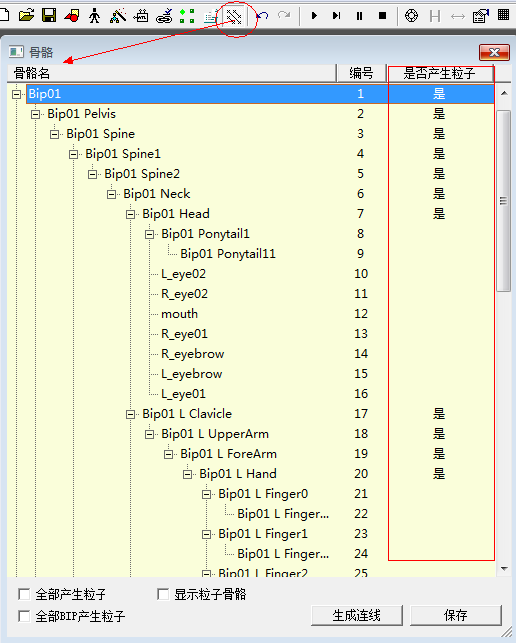
**客户端模型代理元素**，是为了实现笑傲与圣斗士中的特殊需求而加入的元素，该元素代表一个客户端模型，通常情况下代表的模型就是创建该GFX的ECM模型（通过设置id来指定）。通过该元素，美术可以在gfx编辑器中实现对客户端模型的部分控制，包括是否显示，隐藏，再配合下面介绍的**A3DGFXElement::RenderSkinModel**函数实现替换hlsl渲染，以及设置ps constant常量等需求。

**A3DGFXElement::RenderSkinModel** 支持使用一个替换掉的pixelshader去渲染skinmodel。

A3DSkinModel::RenderAtOnce内部会通过传入的**A3DReplaceHLSL**接口指针替换部分贴图，将HLSL　Appear到Device上，再渲染，渲染完毕之后，需要在外部清理掉之前设置到GFX管理的HLSL中的贴图，因为同一个GFX和HLSL可能被反复用于渲染多个SkinModel，某些时刻可能之前的SkinModel及其贴图已经失效了，如果保留在HLSL里，会导致后续的使用变得危险。

**骨骼粒子发生器**，是一类特殊的粒子发射器，传统的粒子发射器都是通过区域或者形状来定义粒子产生点，而该元素则是通过GFX的id获取CECModel，并通过该模型的骨骼状态定义粒子产生点。使用该类型元素的项目，必须实现A3DGFXInterface的GetECModelById接口。

骨骼粒子发射器会使得粒子沿着模型的骨骼的连线均匀发射，在模型编辑器中可以对ECM文件编辑哪些骨骼产生（不产生）粒子：

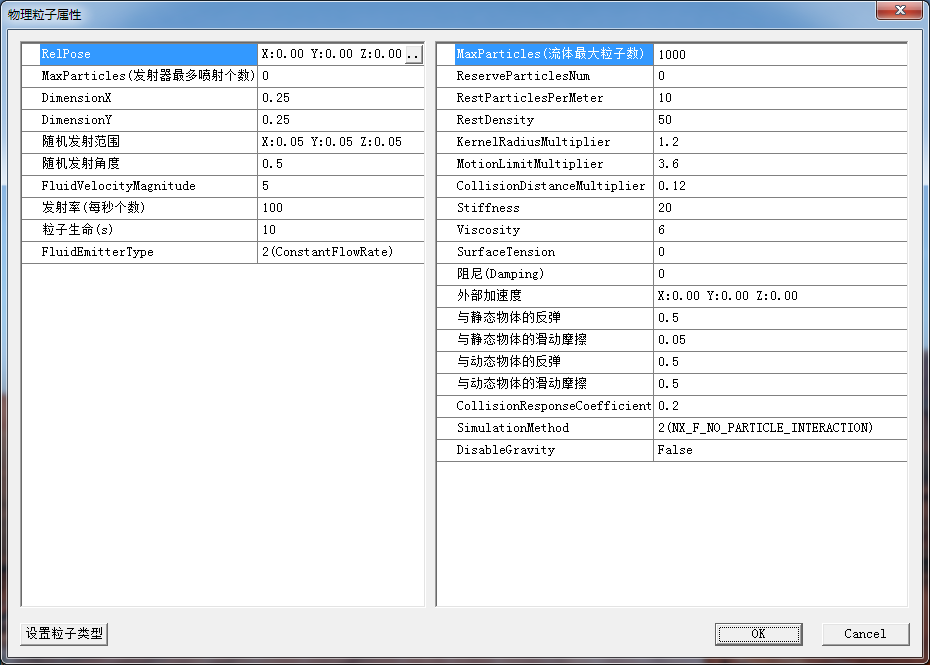


**物理粒子和物理点粒子**

拥有物理数据的元素都从A3DPhysElement类派生。

物理粒子的意图是创建一个物理的流体发射器对象，该流体发射器会不断喷射出粒子，喷出的粒子是按照物理规律运动的，会和场景中的物理实体及力场发生交互，如碰撞、反弹、被风吹动等等。此外，发射器的位置还可以通过GFX关键帧及GFX自身的位置进行更新，从而带动物理粒子发射器运动。

其中物理粒子 A3DPhysFluidEmitter 会创建一个物理的流体对象：APhysXFluidObject，根据流体中的粒子的位置和朝向，更新实际渲染中的面片。因为需要创建物理对象，就需要编辑物理数据，在GFX编辑器中创建物理粒子之后，物理粒子的菜单中有一项“物理粒子发生器属性”，打开以后就可以编辑物理粒子的物理参数了：



**上述属性表中的参数含义可以参考PhysX的文档中对NxFluidDesc，NxFluidEmitterDesc结构的解释**

**（由于APhysX库是在Nx PhysX库的基础上包装了一层，其APhysXFluidParameters, APhysXFluidEmitterParameters的含义就对应了NxFluidDesc，NxFluidEmitterDesc的含义）**

该文档在安装完PhysX开发包之后，应当位于安装目录下，如下：

"D:\Program Files (x86)\NVIDIA Corporation\NVIDIA PhysX SDK

\v2.8.4\_win\SDKs\Docs\PhysXDocumentation.chm"

物理点粒子继承自物理粒子，所不同的是，物理点粒子不支持粒子朝向（通过PointSprite渲染出的都是面向视点的Decal）

GFX文件在支持物理粒子的时候会额外存储一个物理数据文件，后缀名是.gphy。该文件通过A3DGFXPhysDataMan类加载和存储，一个gfx中可能有多个物理元素，每个物理元素的物理对象属性都需要保存，A3DGFXPhysDataMan是通过名字索引，存储了所有的物理数据的。

### A3DGFXElement的注册渲染机制：

特效元素采用注册批量渲染的方式进行绘制。对于采用相同贴图，渲染状态相同的元素，会使用同一个顶点Buffer，并合并到同一个批次里**A3DGFXRenderSlot**绘制。

负责做这件事情的是**A3DGFXRenderSlotMan**类，对不同的贴图和渲染状态组合会建立不同的**A3DGFXRenderSlot**，特效元素会按渲染状态和贴图寻找到对应的Slot被分类收集，最终在A3DGFXRenderSlotMan::RenderOneSlot函数中按照图元类型的不同分别渲染。

**需要注意的是，特效元素一旦使用了自定义的HLSL，就不再对其进行批量渲染了，而是对每一个这样的元素都会新注册一个Slot。**

### GFX渲染容器GfxRenderContainer：

对于某些特殊的需求，比如正常渲染场景过程中，在渲水的时候，有渲染反射，折射的回调函数，需要中断已经渲染了一半的流程，再次遍历一次场景，以收集模型和GFX进行绘制，这个时候如果不加处理，就会导致打乱正常渲染流程中注册进来的GFX对象。

因此设计了**GfxRenderContainer**对象，该对象可以作为一个容器插入到A3DGFXExMan管理器中，用于收集一次渲染流程的过程中所能够获得的GFX对象，通过A3DGFXExMan::RegisterGfx调用注册进来的GFX对象，会被收集到该容器中。

这个容器对象在收集元素进行注册渲染机制的功能上与A3DSkinRender类以及A3DLitModelRender类所起的作用是类似的，只不过它只是一个单纯的容器，而前两者是既有容器功能又包含了渲染代码。

A3DGFXExMan::RenderAllGfx函数中会将**GfxRenderContainer**中收集到的特效元素遍历绘制，这个过程中，每个特效元素都会将自身注册到前述的**A3DGFXRenderSlotMan**中。

使用自定义的GfxRenderContainer时，通过A3DGFXExMan::SetCustomGfxRenderContainer函数插入到A3DGFXExMan管理器中。

### 采用自定义HLSL与shader常量动画：

由于2.1和2.2的A3DHLSL接口有略微的差异，起初是为了屏蔽相关差异，因而设计了**A3DHLPixelShader**类，一方面封装了GFX元素对HLSL的需求，包括设置常量，设置贴图等，另一方面也隔离了2.1引擎和2.2引擎不同的部分。

这部分主要由王蔚实现了一系列的VFX\_系HLSL效果。

在2.1的引擎中位于Shaders\HLSL\GFX2

在2.2的引擎中位于Shaders\2.2\HLSL\GFX2

其中有一类HLSL是需要外部传递**FrameBuffer**到ps里使用的。

使用此类HLSL的特效会被置放到一帧最后的位置，在空间扭曲效果之后再绘制，通过调用**A3DGFXExMan::RenderDelayedEle**函数。（参考[特效GFX的渲染批次](#_特效GFX的渲染批次)）

此类效果通过函数**A3DGFXElement::IsEleShouldBeDelayRendered()**判定。

Shader常量动画依靠A3DAnimatable以及A3DAnimationTrack实现。这部分功能模块是由徐刚实现的。

### A3DGFXExMan介绍

由于历史原因，GFX管理器是一个比较臃肿的类，虽然名为GFX管理器，但是其功能远远超出了其名称所包含的含义。

A3DGFXMan 包含以下几块内容：

1. **基本功能**：初始化Init，释放Release，加载和缓存Gfx（LoadGfx），注册Gfx准备渲染（RegisterGfx），渲染当前注册的Gfx（RenderAllGfx），释放Gfx与淡出Gfx（CacheReleasedGfx,QueueFadeOutGfx)

注册渲染Gfx的时候可以设置不同的GfxRenderContainer作收集器，目前用在水的折射反射中。

注册空间扭曲元素AddWarpEle

注册灯光元素 AddLightParam

注册延迟渲染的元素AddDelayedEle （自定义HLSL一节中说到的需要FrameBuffer的元素）

2. **存放ParticlePool**，Gfx中的粒子喷射器共享一个大粒子池，初始化阶段该池就已经预先分配了20000个粒子，需要新的都可以从里面拿，释放的时候归还到里面。

3. **2D GFX渲染管理**，通过下列函数：

Add2DGfx，Release2DGfx，Play2DGfx，Stop2DGfx，StopAll2DGfx，ReleaseAll2DGfx，Set2DGfxPos，Show2DGfx， Set2DGfxScale，Render2DGfx，Register2DGfx等。

2DGfx主要用于界面上的特效绘制，分为渲染界面前的一次Render2DGfx传递bBackLayer参数，以及渲染界面之后的一次Render2DGfx，采用正交投影摄像机（该摄像机在A3DGFXMan中初始化，与当前屏幕大小保持一致）

4. **各种GFX相关的管理器**

**A3DGFXStreamMan** GFX中使用的顶点Buffer管理器

**A3DGFXSharedTexMan** GFX中使用的贴图管理器，在2.1的版本中该管理器已经将所有管理贴图的功能交给A3DTextureMan去做了，不再自行维护管理，参考A3DGFXSharedTexMan::QueryTexture。

**A3DGFXRenderSlotMan** GFX元素注册渲染管理器，按照渲染状态分类收集GFXElement，然后分批次渲染，这个优化对性能提升影响巨大

**A3DGFXModelMan** GFX中的SkinModel类型元素A3DGFXModel通过该管理器共享模型，对于没有动作的多个模型，可以共享一份A3DSkinModel对象，参考A3DGFXModel::UpdateModel

**A3DGFXSurfaceRender** GFX中的A3DDecalEx类进行紧贴地面形式的渲染时，调用该类。

**ECMMotionBlurPSTable** 用于存放CECModel的MotionBlur中自定义的pixelshader的表

5. **注册、渲染MotionBlur的CECModel**

RegisterECMForMotionBlur，RenderECMForMotionBlur

每帧渲染完毕之后需要再ResetECMForMotionBlur

6. **性能统计**

统计每帧tick， render了多少个GFX，多少粒子等，Tick、Render总共花了多少时间。

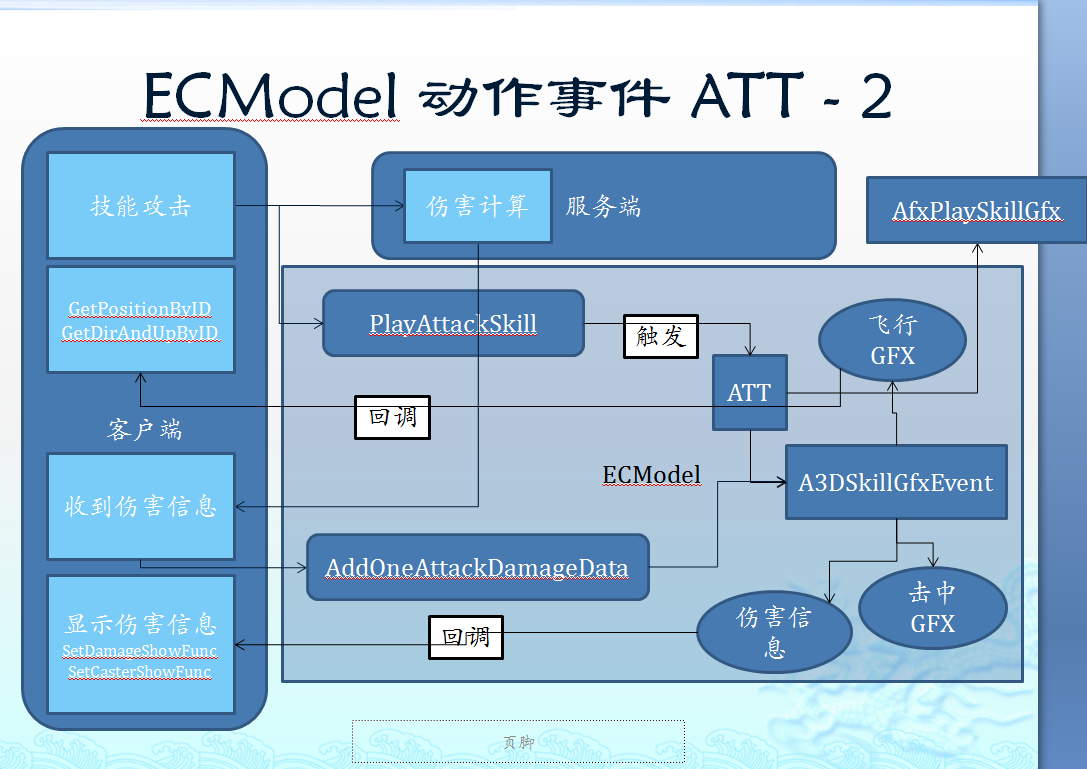
通过GetTotalTickTime，GetTotalRenderTime等

以及对面数进行统计GetElementPerformenceMap()

## A3DSkillGfxComposer/A3DSkillGfxEvent

这两个类主要负责实现客户端中的飞行和击中效果，其中比较复杂的是从飞行到击中的流程，可以参考附录中的PPT：\\Sourceserver\InhouseTools\文档\ECModel&GFX原理与应用.pptx

参考图：



技能攻击的时候客户端调用PlayAttackSkill函数，传入发出者和目标id，同时发送技能攻击的消息给服务器。

PlayAttackSkill触发动作，在动作过程中会触发ATT事件，于是播放飞行效果，飞行过程中会不断通过目标id获取目标位置，直到击中目标，播放击中效果。

在这个过程中，服务器可能会传来伤害信息，客户端收到伤害信息会调用AddOneAttackDamageData函数将伤害数据传入CECModel，然后CECModel干下面的事：

1. 如果ATT尚未触发，伤害数据存入A3DCombActDynData，等到ATT触发时带上。

2. 如果ATT已经触发，正在飞行，则找到飞行中的A3DSkillGfxEvent对象塞入。

如果伤害信息来的时候查到飞行已经结束，或者携带着伤害信息击中了目标，则立即播放伤害信息。

通过id取位置的回调是通过A3DSkillGfxMan实现的，客户端派生自这个类，并实现其定义的纯虚函数：

GetPopsitionByID()

GetDirAndUpByID()

## 与编辑器导出文件相关的接口

编辑器相关接口除了包括**编辑中用于获取属性或者设置属性的接口**以外，还包括编辑器辅助函数：

GfxEditorUtil.h中定义了递归获取GFX、ecm、att、smd文件引用其他文件的函数：

a\_ExpCollectSMDFiles

a\_ExpCollectECMFiles

a\_ExpCollectGFXFiles

a\_ExpCollectATTFiles

这几个函数一直没有在GFX系编辑器中使用过，使用前需要测一下。

# 与客户端的接口

## 替换Afx系回调函数，A3DGFXInterface

早期的Gfx库中有一系列的Afx系回调函数需要客户端实现，后来发现扩展的时候要给回调函数增加功能会非常麻烦，只要增加参数就要求所有的客户端都修改本地的函数实现，并重新编译链接。

**建议逐步将旧的Afx系回调函数(位于A3DGFXExMan.cpp中)都迁移到新接口中，并将复杂参数用struct包裹后传递。**

在3.1版本的GfxCommon中，由于没有老项目的包袱，我们已经统一切换成采用一个纯虚接口**A3DGFXInterface**，并且将复杂的函数参数传递通过struct进行包裹，避免了后续增加参数时的麻烦，类似下面这样：

struct APIPARAM{

param1;

param2;

param3;

// 追加新参数，无需老项目更改函数签名，只需重新编译链接即可，用不到的新参数亦可忽略

}

SomeAPI(APIPARAM\*);

当后续追加新的参数时可以避免强迫老代码更新函数签名。

在2.1和2.2中的新接口我们也开始逐步采用A3DGFXInterface类，其中新增PlaySkillGfx函数用于替代以往的AfxPlaySkillGfx函数，参数改用结构体指针的形式传递，减轻了对客户端部分代码的影响。在默认实现中，该函数PlaySkillGfx直接调用后者，而客户端如果实现了该接口就可以抛弃AfxPlaySkillGfx函数了。

**A3DGFXInterface**接口中有一系列通过id获取模型或者设置模型参数的函数，包括

GetECModelById, SetSkinModelVisibleById, SetSkinModelTickSpeed等，对于使用了**客户端模型代理元素**，**骨骼粒子发射器元素**的客户端项目都需要实现这几个函数。其中，SetSkinModelVisibleById和SetSkinModelTickSpeed函数有标准的实现方案，直接取得CECModel::GetSkinModelTRCtrl()调用**SkinModelTRControl**的对应接口即可。

客户端实现A3DGFXInterface接口后，通过SetGfxInterface函数插入到A3DGFXExMan管理器中。

## 客户端加载CECModel的回调接口IGFXECMInterface

在A3DGFXECModel元素中，需要有客户端配合的地方：

其一是有可能要求客户端帮助加载一个CECModel对象；

其二是可能要求客户端帮助一个已经加载完毕的CECModel对象加载皮肤。

参考 [特效元素中的A3DGFXECModel元素介绍。](#_常用的各个元素类型)

这一需求由IGFXECMInterface接口完成。

客户端实现该接口后，通过SetGfxECMInterface插入到A3DGFXExMan中。

## 通过客户端逻辑直接控制部分元素

在A3DGFXFuncs.h中定义了若干函数，用于在客户端逻辑中控制某些元素属性。

GFX\_UpdaetLightingEdgePos，使用客户端提供的位置更新一个GFX对象的某个电弧元素的两个端点

GFX\_UpdateTrailEdgePos，使用客户端提供的位置更新一个GFX对象中的某个轨迹元素的两个端点

此外，在ECM编辑器中还可以通过更新GFX外部参数的形式控制GFX元素（包括利用挂点位置设置轨迹的两个端点等）。参考A3DCombinedAction.cpp中的GFX\_BINDING::UpdateParam函数。

**部分文件功能说明：**

EC\_Model 文件包含CECModel类和CECModelStaticData类。

EC\_ModelBlur文件包含了CECModel的运动模糊功能，运动模糊通过保存多帧动画的骨骼矩阵，在渲染的时候依次Apply到A3DSkinModel上，再对A3DSkinModel做RenderAtOnce渲染出来。

EC\_ModelHook是ECM文件中的虚拟挂点，这类挂点与A3DSKinModelHook不同，此类挂点只记录相对ECM原点的偏移和旋转，而不考虑A3DSkinModel或者A3DBone的朝向与位置，在ECM编辑器中编辑。（**需要注意，CECModel的原点和A3DSkinModel的根骨骼pos可能不一致**）

EC\_ModelLua是对luawrapper库的简单封装，方便在ECM的脚本事件Api中使用，该文件包含了几个对EventInfo的封装，可以在脚本事件里调用lua接口创建gfx，sfx等。

EC\_ModelLuaApi是将原先位于EC\_Model.cpp中的lua api封装函数提取出来单独列一个文件。

EC\_ModelMan用于管理CECModelStaticData的管理器，通过文件名和引用计数管理CECModelStaticData对象。

EC\_ModelPhys是存放物理应用方面的类的文件，目前有CECModelPhysBlendOverlap，用于表现模型与模型间Overlap时的物理加力和半物理模拟效果。

EC\_ModelShell是在CECModel上套了一个壳，相当于一个Proxy（此文件基本没用上）。

A3DCombinedAction 包含了组合动作类的定义以及动作条上的各类事件点的定义

A3DCombActDynData 定义了组合动作的实例，每播放一个组合动作，都会创建这样一个动态实例对象。

A3DModelPhysics ECM模型使用物理系统的接口

A3DModelPhysSync 创建物理对象，维护模型物理状态(全动画或是半物理或是全物理)，根据动画更新Actor，根据Actor更新动画等。

A3DModelPhysSyncData存储与加载模型物理数据的类

A3DPhysActor 是物理数据中的Actor基类，派生出A3DModelPhysSyncData::ACTOR\_DATA和A3DSkinPhysSyncData::ACTOR\_DATA。

A3DPhysRBCreator 存储了构建物理描述对象，如APhysXSkeletonRBObjectDesc等的信息。

A3DSkinPhysSync 用于创建布料物理对象，更新挂载Actor位姿，依照物理更新ClothMesh等。

A3DAnimatable，A3DAnimatable\_ApplyNone，A3DAnimatable\_PSConst，A3DAnimationTrack等文件用于提供关键帧变量，目前应用于GFX的自定义HLSL 的PS常量关键帧中。

A3DGFXStreamMan 提供了一个定制的Stream，**该Stream使用了延迟Lock的策略帮助提高效率（在Lock时先行返回一个预定的buffer供外部使用，将真正的Lock推迟到Unlock的时刻）**，由于GFX元素填写缓冲时需要将多个元素的顶点缓冲组合到一个Buffer里进行批量渲染，而每个FillVertexBuffer的调用时间都可能比较长，因此上述策略有助于提升性能。

A3DGFXSurfaceRender 用于渲染紧贴地面的面片，使用了正交投影相机从上向下绘制地表面片，中2.0和2.1中通过**D3DTSS\_TCI\_CAMERASPACEPOSITION**的纹理坐标模式获取纹理坐标，2.2中则是直接在HLSL的vs中计算uv。（此处2.1的版本也应可以与2.2的方案统一，减少维护负担）

A3DGFXElement 定义了GFX元素的基类，定义了所有类型特效元素的ID，以及所有类型元素的属性ID。编辑时，可根据属性id名称对应表自动生成相应的属性表。

本文档总结了GFX库的设计，此外还可以从以下文档中查阅GFX库中某些部分的细节。

附属文档：

**针对程序的GFX和ECModel模块总体设计的ppt**

\\Sourceserver\InhouseTools\文档\ECModel&GFX原理与应用.pptx

**ECM动作条脚本事件说明以及相关API**

\\Sourceserver\InhouseTools\文档\[2011.2.24]ECM编辑器动作条脚本事件文档&ECM脚本API文档.doc

该文档的最新版本在 \\elementserver\gameserver\_mmorpg\$/Common/GfxCommon2/ECM编辑器动作条脚本事件文档&ECM脚本API文档.doc

**特效编辑器的使用与制作**

\\Sourceserver\InhouseTools\文档\特效编辑器制作v2.ppt

**ECM编辑器物理编辑功能说明**

\\Sourceserver\InhouseTools\文档\ECM编辑器物理编辑功能说明.doc

**针对GfxCommon 2.0版本的特效自定义shader的使用说明**

\\Sourceserver\InhouseTools\文档\GFX新增的自定义着色器功能说明与教程.docx