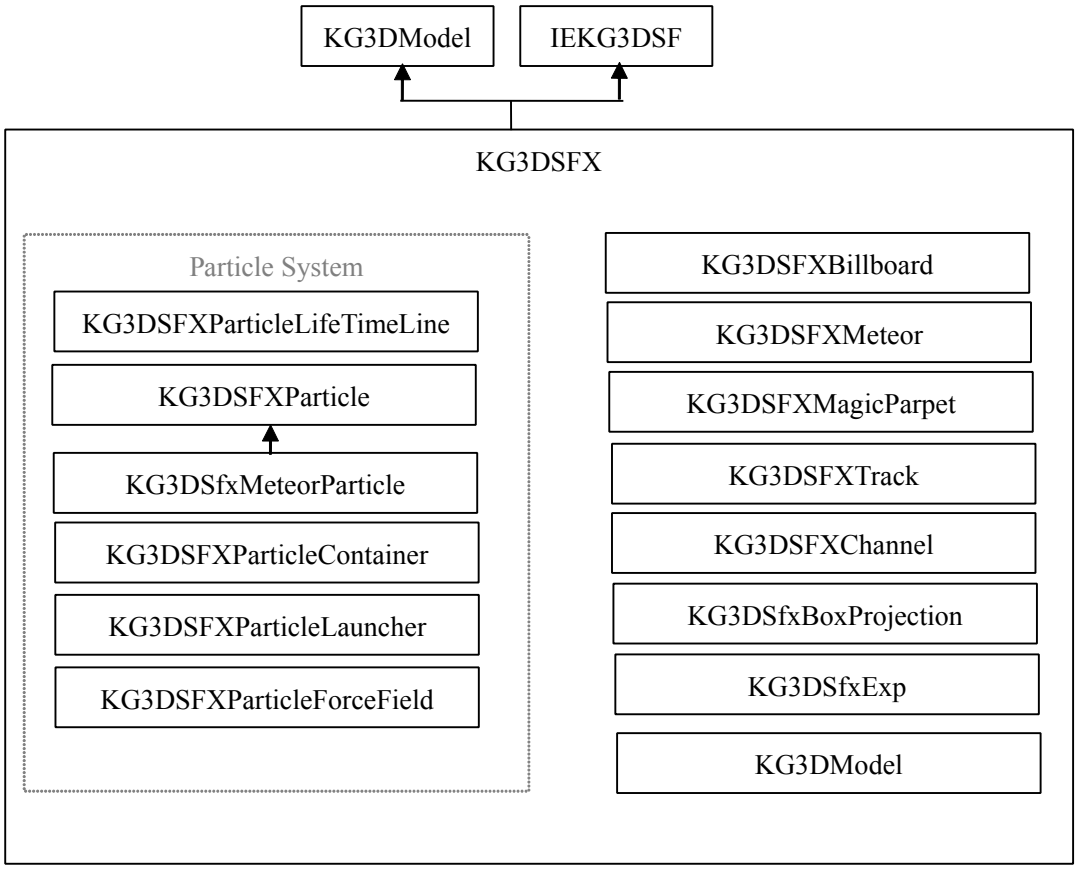
剑网 3-3D 引擎代码文档

SFX(特效)

3D 引擎组
招新宇
2009-09-21

SFX 结构



SFX 结构图(箭头表示继承关系)

为了对外接口一致，KG3DSFX 继承了 KG3DModel，所以对使用者来说可以象 Model 一样，加载，更新，显示等等。为了方便编辑器编辑特效 KG3DSFX 继承了 IEKG3DSFX 接口，主要提供一些修改编辑特效的功能。如上图所示，一个特效可以由各种特效元素组成，包括：粒子系统，公告扳，轨道，力场等等。一个特效不一定包含所有这些特效元素，只是它们的任意组合。

粒子系统

粒子系统由以下 6 部分构成 ：

1）粒子类型属性：KG3DSFXParticleLifeTimeLine
它是一个关键帧序列，主要包括粒子的基本属性：颜色，大小，纹理，旋转量等
粒子更新的时候可以通过当前帧去取得当前这些属性

2）粒子 KG3DSFXParticle
单个粒子，每个粒子是一个矩形面，每帧从 KG3DSFXParticleLifeTimeLine 取得相关性信息，比如大小，颜色来更新自身

3）拖尾粒子 KG3DSfxMeteorParticle
普通粒子 KG3DSFXParticle 的派生物，它的特征是它的形状不是一个矩形面，而是一系列的矩形面连接起来，每一个这种粒子就好比一个拖尾（刀光）

4）粒子发射器 KG3DSFXParticleContainer

在粒子系统更新的时候(FrameMove)，按照发射器形状，和发射属性(数量，生命，速度等)生成粒子，发射器是一些基本形状的表面（矩形，圆形，球面，圆柱面），控制粒子从它们的表面上发射。

PS：以后可以增强一种模型发射器，从模型表面发射粒子（Cool~~）

5）粒子容器 KG3DSFXParticleContainer

管理粒子发射器发射出去的粒子，其实就是一个粒子池(为了效率和渲染状态排序)

6）力场 KG3DSFXParticleForceField

一般粒子只有速度大小和方向的话很难做出复杂的粒子运动，比如旋风，聚积，布而运动。为了让粒子可以进行这些复杂的运动，粒子系统里加入了力场，力场会按照力场类型，使用相应的运动方程，改变粒子的运动位置和方向。

公告扳

KG3DSFXBillboard

每个公告板包含几个属性，比如宽高，颜色等，纹理可以有两层，第二层是 Alpha 层。设计之初没有把公告板的纹理和粒子系统统一处理是考虑到公告板使用比较少，没必要和粒子整体做优化

拖尾（刀光）

KG3DSFXMeteor

初始是在自身坐标系 X 轴上的一条线，特效播放的时候，这条线会拖出一个尾巴，形成一个三角形带。这个东西主要用在刀光，拖尾效果。在客户端帧数比较低的时候，由于两帧更新之间时间差比较大，造成拖尾不平滑（有折角），所以后来增加了骨骼动画关键帧加密插值，修补由于客户端帧数比较低的时候两帧之前丢弃的骨骼动画数据。

Y 投影（贴地面片）

KG3DSFXMagicParpet

最初设计的是紧贴地表的一个面片。一个矩形的网格，每个顶点是通过查询那个位置的高度，再往上抬一点点来决定其 y 值的。这个东西主要用来表示选择，或者施法范围。网络密度和地表网格对齐。

后来由于应用需要，增加了新的两种类型

1）投影：通过屏幕特效（KG3DPostRenderEffectAoe）来处理的贴地效果，优点：可以紧贴地表和物件，同一块地表或者物件可以受多个投影效果影响。缺点：处理 4 个投影需要一次屏幕绘制，所以当数量过多的时候会造成效率上的瓶颈。

2）选择：通过在地形，物件 shader 中加入投影处理来实现紧贴效果，优点，效率高，缺点：每个物件和地表同时只能受一个投影效果影响

轨道

KG3DSFXTrack
为了更精确控制粒子的运动速度和方向，我们把粒子绑定到一个轨道上（沿着轨道运动）轨道是通过轨道导出插件（Track Exporter）来制作的。导出插件导出的是一个 Dummy 在动画时间内的变换矩阵。一般说来，在 max 制作一条轨道，然后把一个 Dummy 绑上去，然后就可以导出了。需要注意的是，Dummy 自身的朝向是会对导出结果有影响的。

通道（两点连线）

KG3DSFXChannel
通过指定两个位置，在两个位置之间产生连接面，通过纹理运动来产生类似电流效果。
这个效果在目前游戏里还没有应用。

立方投影

KG3DSfxBoxProjection
在场景上指定一个立方体，通过搜索场景中所有处于这个立方体内部的面（来自地表和场景物件），通过这些面生成一个新的网格模型，然后通过在对模型进行贴图投影（按照一定的投影规则）产生一些效果。比如把处于立方体内部的所有场景面变成红色 - -b，或者投影一张裂痕贴图。这个效果在目前游戏里还没有应用。也没有相应美术制作，所有还不完善。

特殊控制

KG3DSfxExp
一些特殊的表现效果控制，当前有
1）引擎时间控制：注意，这个是控制整个引擎的时间的，比如你把时间加快了 2 倍，那么整个游戏中的天气效果，模型动画，水流速度，粒子运动速度等都是提速 2 倍 - -b
2）镜头控制：控制当前引擎的主镜头，用来产生诸如抖动之类的效果，当前只有平移抖动，以后可以增加旋转或者更复杂更自然的抖动
3）模型控制：播放的时候可以控制模型的大小（忽肥忽瘦）暂时只能控制大小，以后可以增加更多的控制（比如颜色啊，被绑到轨道上转圈圈啊，哈哈）。

模型动画

KG3DModel
特效里面可以包含模型动画，用来逆补一些用现有特效系统无法很好表现的效果

PS:

1) 现有粒子系统不支持模型发射器和模型粒子，以后可以适当增强。
2) 现有粒子系统使用固定管线绘制，由于粒子属性很多（大小，颜色，纹理 id，速度，力场，自旋，锁轴等等）改成可编程 Instances 方式绘制可能会有一定的困难，不过可以常识，对于 n 卡，或许用 cuda 加速会更有效。

程序接口讲解

```
class KG3DSFX : public KG3DModel, public IEKG3DSFX
{
public :


    // 从.sfx文件加载特效，参数参看KG3DModel
    HRESULT LoadFromFile(const char cszFileName[], unsigned uFileNameHash, unsigned uOption);

    // 保存特效到.sfx文件
    HRESULT SaveToFile(const char cszFileName[], unsigned uOption);

    // 重新加载
    HRESULT Reload();

    // 出始化处理
    HRESULT Init();
    void        ClearUp(BOOL bUnBind);

    // 清理处理（反出始化）
    void        ReSet();

    // 特效是否已经加载完成，包括特效本身的数据和内部的子模型
    BOOL        IsLoaded();

    // 是否是特效排序类型，对于特效来说，永远都是return true
    virtual BOOL IsSortAsSFX();

    // 子模型是否加载完成
    BOOL        IsModelLoad();

    // 子模型的动画数据是否加载完成
    BOOL        IsAniLoad();

    // 数据是否已经准备好，对于特效，等同于IsLoaded()
    virtual BOOL IsResourceReady(BOOL bTestText);

    // 渲染
    HRESULT Render(unsigned uOption, void* pExtInfo);

    // 更新
```

```
        HRESULT FrameMove();

        // 废弃
        HRESULT UpdateParticleSystem();

        // 绘制选择框
        HRESULT RenderSelectMark(DWORD color = 0xFFFFFFFF);

        // KG3DModel接口
        IKG3DAnimationController*       GetAnimationController(enuAnimationControllerPriority
nPriority);
        IKG3DAnimationController*       GetSplitAnimationContorller(DWORD       dwPartIndex,
DWORD dwControllerIndex);

        // 设置/获取当前特效帧
        HRESULT SetCurrentFrame(float fFrame);
        float     GetCurrentFrame() const;

        // 播放特效
        /*
        enum SFX_ANIMATION_PLAY
        {
        SFX_AP_PAUSE,       暂停
        SFX_AP_LOOP,        循环
        SFX_AP_ONE_TIME,    单次

        SFX_AP_MAX
        };
        */
        HRESULT PlayAnimation(SFX_ANIMATION_PLAY sfxAnimationPlay, float fSpeed = 1.f,
int nOffsetTime = 0.f);

        // 参看KG3DModel接口，播放动画（播放特效）
        HRESULT PlayAnimation(
            DWORD    dwPlayType,
            LPCSTR cszAnimationName,
            FLOAT    fSpeed,
            int      nOffsetTime,
            PVOID    pReserved,
            PVOID    pUserdata,
            enuAnimationControllerPriority Priority
        );
        SFX_ANIMATION_PLAY GetPlayAnimation() const;
```

```
    // 设置/获取特效帧间隔
    HRESULT SetFrameInterval(DWORD dwInterval);
    DWORD     GetFrameInterval() const;
    float     GetCurrInterval() const;

    // 设置/获取特效总帧数
    HRESULT SetTotalFrameNum(DWORD dwTotalFrame);
    DWORD     GetTotalFrameNum() const;

    // 当前粒子数量
    DWORD     GetParticleNum() const;
    DWORD     GetMaxParticleNum() const;
    void       SetMaxParticleNum(DWORD dwNum);

    // 得到已经播放的时间长
    int       GetPlayTime() const;

    // 参看模型
    virtual BOOL OnProcessCommands(KG3DModelCommandBase* pCmd);

    // 绑定相关，参看模型
    virtual HRESULT _BindToSocket(KG3DModel *pModel, const char cszSocketName[]);
    virtual HRESULT _BindToBone(KG3DModel* pModel, const char* cszSocketName);
    virtual HRESULT UnBindFromOther();

    HRESULT BindToModel(KG3DModel *pModel);
    HRESULT UnBind();


    // 得到当前世界矩阵
    D3DXMATRIX*    GetCurrentWorldMatrix();

    // 参考模型
    virtual BOOL IsBBoxChanged();
    virtual void SetBBoxChanged(BOOL bBBoxChanged);

    // 计算保围体
    HRESULT             ComputeBBox();

    // 是否和射线相交
    BOOL                IsRayIntersect(D3DXVECTOR3    &Inter,    D3DXVECTOR3    Origin,
D3DXVECTOR3 Direction);

    // 设计alpha
```

```
    HRESULT             SetAlpha(float fAlpha, BOOL bUseSpecial);

    // 是否贴地
    int                 GetIsCloseFloor();

    // 设置/获取是否参与屏幕扭曲效果
    int                 GetIsShockWave();
    HRESULT             SetIsShockWave(int nValue);

    // 设置/获取参与屏幕扭曲方式（只有扭曲还是特效和扭曲同时都用）
    DWORD               GetShockMode() { return m_dwShockMode; }
    void                SetShockMode(DWORD mode) { m_dwShockMode = mode; }

    // 标签相关，参看模型
    void                SetTagUsingInfo();
    DWORD               IsFixedSize() { return m_dwFixedSize; }
    void                EnableEixedSize(DWORD dwEnable) { m_dwFixedSize = dwEnable; }

    // 是否锁定y轴
    DWORD               IsLoacAxisY() { return m_dwLockYAlex; };

    // 开启锁定y轴
    void                EnableLockY(DWORD dwEnable) { m_dwLockYAlex = dwEnable; };

    // 得到特效对于文件名
    LPCTSTR             GetName()     { return m_scName.c_str(); }

    // 设置/获取参与屏幕扭曲的强度（暂时无效）
    void                                                SetShockWavePower(float       power)
{ KG3DModel::SetShockWavePower(power);   }
    float                   GetShockWaePower()                                          {   return
KG3DModel::GetShockWaePower(); }



    // 特效会对粒子系统使用的贴图同一管理
    // 一张贴图可以进行分割，比如分割成x4
    // 4x4的每一个小块叫做一个texture frame

    // 增加贴图
    HRESULT AddTexture(const char szTextureName[]);

    // 删除贴图
    HRESULT RemoveTexture(UINT uTextureIndex);
```

```
    // 当前特效使用贴图数
    DWORD      GetTextureNum() const;

    // 随机得到一个texture frame
    const UINT GetRandTexFrame(UINT uTexIndex) const;

    // 获取贴图相关信息
    HRESULT GetTexture(UINT uTextureIndex, KG3DTexture** ppTexture) const;
    HRESULT GetIKG3DTexture(UINT uTextureIndex, IKG3DTexture** ppTexture) const;
    HRESULT SetTexture(UINT uTextureIndex, const char szTextureName[]);
    HRESULT SetTextureCutNum(UINT    uTextureIndex, UINT    uCuttingNum);
    HRESULT GetTextureCutNum(UINT    uTextureIndex, UINT* pCutNum) const;

    // 获取贴图uv坐标
    HRESULT   GetTextureCoord(FRECT*   pTextureCoord,   UINT      uTextureIndex,   UINT
uFrameIndex);

    // 废弃
    HRESULT GetRenderTargetTextureCount(OUT DWORD* pTexCount);
    HRESULT GetRenderTargetTexture(OUT DWORD* pResourceId, IN DWORD dwTexIndex
= 0);
    HRESULT SetCallbackRender(CallbackRender fpCallbackRender, void* pUserData);

    // 强制刷新贴图（重载）
    HRESULT ReflushTexture();


    // 粒子公用属性相关
    HRESULT AddParticleLifeTimeLine(IEKG3DSFXParticleLifeTimeLine** ppLifeTimeLine);
    HRESULT  GetParticleLifeTimeLine(DWORD  dwIndex,  IEKG3DSFXParticleLifeTimeLine
**ppLifeTimeLine);
    HRESULT RemoveParticleLifeTimeLine(DWORD dwIndex);
    DWORD      GetParticleLifeTimeLineNum() const;


    // 发射器相关，增加，删除，获取，数量
    HRESULT AddParticleLauncher(
        SFX_PARTICLE_TYPE          eParticleType,
        SFX_LAUNCHER_SHAPE           eLauncherShape,
        SFX_PARTICLE_UPDATE_MODE    eParticleUpdateMode,
        KG3DSFXParticleLauncher** ppParticleLauncher
    );
    HRESULT GetParticleLauncher(
```

```
        DWORD                           dwIndex,
        KG3DSFXParticleLauncher** ppParticleLauncher
);
HRESULT GetParticleLauncher(
        DWORD                           dwIndex,
        IEKG3DSFXParticleLauncher** ppParticleLauncher);
HRESULT RemoveParticleLauncher(DWORD dwIndex);
DWORD GetParticleLauncherNum() const;


// 立场相关，增加，删除，获取，数量
HRESULT AddForceField(
        SFX_FORCE_FIELD_TYPE        eForceFieldType,
        IEKG3DSFXParticleForceField** ppForceField
);
HRESULT GetForceField(
        DWORD                           dwIndex,
        IEKG3DSFXParticleForceField** ppForceField
);
HRESULT RemoveForceField(DWORD dwIndex);
DWORD GetForceFieldNum() const;

// 轨道相关，增加，删除，获取，改变，数量
HRESULT AddTrack(const char cszFileName[], KG3DSFXTrack **ppTrack);
HRESULT GetTrack(DWORD dwIndex, KG3DSFXTrack**      ppTrack);
HRESULT GetTrack(DWORD dwIndex, IEKG3DSFXTrack** ppTrack);
HRESULT RemoveTrack(DWORD dwIndex);
HRESULT ChangeTrack(DWORD dwIndex, const char cszNewFileName[]);
DWORD     GetTrackNum() const;

// 拖尾相关
// 增添
HRESULT AddMeteor(KG3DSFXMeteor **ppMeteor);
// 获取
HRESULT GetMeteor(DWORD dwIndex, IEKG3DSFXMeteor **ppMeteor);
// 数量
DWORD     GetMeteorNum() const;
// 删除
HRESULT RemoveMeteor(DWORD dwIndex);
// 插值精度
HRESULT SetMeteorInterpolate(float fInterpolate);
float     GetMeteorInterpolate();

// 获取绑定到的轨道索引
```

```
    DWORD     GetMeteorBindTrackIndex(DWORD dwIndex);

    // 脚印相关
    void      ClearMeteorListNode();//by huangjinshou 2007 8.1
    void      SetMeteorSampling(BOOL val);
    void SetMeteorRunState(BOOL bSampling, KG3DAnimationController *pController);


    // 公告板相关
    // 添加
    HRESULT AddBillboard(KG3DSFXBillboard** ppBillboard);
    // 获取
    HRESULT GetBillboard(DWORD dwIndex, KG3DSFXBillboard **ppBillboard);
    HRESULT GetBillboard(DWORD dwIndex, IEKG3DSFXBillboard** ppBillboard);
    // 数量
    DWORD     GetBillboardNum() const;
    // 删除
    HRESULT RemoveBillboard(DWORD dwIndex);
    // 开启关闭排序
    HRESULT EnableBillboardSort(BOOL bEnable);

    // 贴地面片相关
    HRESULT     AddMagicParpet(IEKG3DSFXMagicParpet     **ppMagicParpet,     DWORD
dwType);
    HRESULT RemoveMagicParpet(int index);
    HRESULT RemoveMagicParpet(IEKG3DSFXMagicParpet* aoe);
    size_t GetMagicParpetNum() { return m_vecMagicParpet.size(); }
    IEKG3DSFXMagicParpet* GetMagicParpet(int index);
    IEKG3DSFXSelectMark* GetSelectMark();

    // 立方投影相关
    HRESULT AddBoxProjection(IKG3DSfxBoxProjection** ppBoxProjection);
    HRESULT RemoveBoxProjection(IKG3DSfxBoxProjection* pBoxProjection);
    HRESULT RemoveBoxProjection(int index);
    IKG3DSfxBoxProjection* GetBoxProjection(int index);
    size_t GetBoxProjectionNum()       { return m_vecBoxProj.size(); }

    // 通道相关
    HRESULT AddChannel(IEKG3DSFXChannel** ppChannel);
    HRESULT RemoveChannel();
    HRESULT GetChannel(IEKG3DSFXChannel ** pChannel);
    KG3DSFXChannel *GetChannel();
    // 更新两个端点的位置
    STDMETHOD(UpdateChannelEffectPos)(D3DXVECTOR3 v1, D3DXVECTOR3 v2);
```

```
// 模型动画相关
KG3DModel* GetModel(size_t index);
HRESULT      SetModel(const char cszFileName[], KG3DModel** ppModel, size_t index);
HRESULT AddModel(const char cszFileName[], KG3DModel** ppModel);
HRESULT RemoveModel(size_t index);
HRESULT GetModel(size_t index, IEKG3DModel **ppModel);

// 模型绑定轨道索引
HRESULT GetModelBindTrackIndex(size_t index, int* pTrackIndex);

// 动画信息
// 速度
HRESULT GetModelAnimationSpeed(float *pValue, size_t index);
// 开始
HRESULT GetModelAnimationStart(float *pValue, size_t index);
// 模式
HRESULT GetModelAnimationMode(DWORD *pValue, size_t index);
// 总帧长
HRESULT GetModelAnimationFrameCount(DWORD* pValue, size_t index);
HRESULT GetModelAnimationFrameTime(float* pValue, size_t index);

void SetModelAnimationFrameTime(float Value, size_t index);
void SetModelAnimationSpeed(float    Value, size_t index);
void SetModelAnimationStart(float Value, size_t index);
void SetModelAnimationMode(DWORD Value, size_t index);
void SetModelAnimationFrameCount(DWORD Value, size_t index);

// 模型接口，对特效无用，反回空串
HRESULT GetMeshFileName(LPSTR pFileName);
HRESULT GetMaterialFileName(LPSTR pFileName);
HRESULT GetAnimationFileName(LPSTR pFileName);

// 模型动画总数
DWORD GetNumModels() const;

// 模型动画开始帧
void SetModelStartFrame(float fStartFrame);
float GetModelStartFrame();

// 得到模型动画总帧长，和GetModelAnimationFrameCount 一样
DWORD GetMdlNumFrame(size_t index);
FLOAT    GetNdlFrameTime(size_t index);
```

```
// 开启关系模型动画循环播放
HRESULT EnagleMdlAniConnect(int nEnagle, size_t index);

// 更新模型动画信息
HRESULT UpdateModelAniInfo(size_t index);
HRESULT UpdateModelAniInfo();

// 跳转模型动画帧
HRESULT SeekModelAniBySFXFrame(float fFrame, float* pSeek, size_t index);
HRESULT SeekModelAniBySFXFrame(float fFrame, float* pfSeek);

// 模型动画是否循环播放
int         IsMdlAniConnect(size_t index);
DWORD     GetModelPlayMode();

// 开启关闭，模型公告板
HRESULT EnagleMdlBillBoard(int nEnagle,    size_t index);
BOOL       IsMdlBillBoard(size_t index);

// 模型颜色控制
HRESULT AddSubsetColorDiffuseLine(DWORD dwIndex, D3DCOLORVALUE initColor);
HRESULT DeleteSubsetColorDiffuseLine(DWORD dwIndex);
HRESULT AddMutiSubsetColorDiffuseLine(DWORD dwIndex, IEKG3DMaterial *piMat);
HRESULT AddTimeLineModelAniBound();
TKG3DTimeLinebase* GetSubsetColorLine(DWORD dwSubset,DWORD dwIndex);
int GetSubsetColorLineSize(DWORD dwSubset);

// 模型动画范围相关
int GetTimeLineModelAniBoundSize();
TKG3DTimeLinebase* GetModelAniBoundTimeLine(DWORD dwIndex);


// 音效相关
// 插入音效帧
virtual HRESULT InsertSoundKey(LPCTSTR szFileName,
      DWORD    dwMode,
      FLOAT    fVolume,
      FLOAT    fSFXKey,
      FLOAT    fMinDis,
      FLOAT    fMaxDis,
      BOOL     bLoop,
      BOOL     bContinue,
      int      nProbability,
      int      nSubIndex
```

```
        );
// 删除
virtual HRESULT RemoveSoundKey(FLOAT fSFXKey, int nSubIndex);

// 修改
virtual HRESULT ModifySoundKey(LPCTSTR szFileName,
        DWORD    dwMode,
        FLOAT    fVolume,
        FLOAT    fSFXKey,
        FLOAT    fMinDis,
        FLOAT    fMaxDis,
        BOOL     bLoop,
        BOOL     bContinue,
        int      nProbability,
        int      nSubIndex
        );
// 查询
virtual HRESULT QuerySoundKey(LPTSTR    szFileName,
        DWORD&    dwMode,
        FLOAT&    fVolume,
        FLOAT     fSFXKey,
        FLOAT&    fMinDis,
        FLOAT&    fMaxDis,
        BOOL&     bLoop,
        BOOL&     bContinue,
        int&      nProbability,
        int       nSubIndex
        );
// 音效帧总数
virtual int GetSoundKeyNum(FLOAT fSFXKey);

// 播放音效
HRESULT PlaySound(_SoundInfo& soundInfo, size_t index);

virtual TKG3DTimeLinebase* GetSoundTimeLine()
{
    return &m_tlSoundKey;
}

// 模型接口，参看kg3dmodel
virtual HRESULT ChangeResource(UINT uFrame, UINT uIndex, LPCSTR strNew);
virtual HRESULT ChangePlayType(UINT uFrame, UINT uIndex, void *pDefault);
virtual HRESULT DeleteResource(UINT uFrame, UINT uIndex);
virtual HRESULT AddResource(UINT uFrame, LPCSTR strAdd, void *pDefault);
```

```
        virtual HRESULT DeleteAllSound();
        virtual DWORD    GetSourceReference(TagSourceInfo *pInfo, DWORD dwSize);

        virtual IEKG3DSfxExp* GetExpEffect() { return m_pExpEffect; }

        // d3d设备丢失恢复
        virtual HRESULT OnLostDevice();
        virtual HRESULT OnResetDevice();
        virtual void ScanlAllTimeLine(float fScanl);

        //  实际加载函数
        HRESULT  _LoadFromFile(const char cszFileName[], unsigned uFileNameHash, unsigned
uOption);

        virtual HRESULT LoadFromFileMultiThread();
        virtual HRESULT LoadPostProcess();
public :
        KG3DSFX();
        virtual ~KG3DSFX();

public :
            typedef HRESULT (KG3DSFX::*ProcessBlockFunction)(BYTE *pBuffer, DWORD
dwOffset);
            static ProcessBlockFunction ms_pfnProcessBlock[SFX_FEID_SIZE];

            DWORD GetUserData()
            {
                return m_dwUserData;
            }
            void SetUserData(DWORD dwData)
            {
                m_dwUserData = dwData;
            }
private :

        // sfx文件读取相关，每一个特效元素（粒子系统，公告板，刀光等）
        //  在文件中都是以block方式存放

        HRESULT ReadGeneralInformationBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadParticleKindBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadLauncherBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadForceFieldBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadTextureBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadBillboardBlock(BYTE *pBuffer, DWORD dwOffset);
```

```
        HRESULT ReadBoxProjBlock(BYTE* pBuffer, DWORD dwOffset);
        HRESULT ReadTrackBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadModelBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadMeteorBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadMagicParpetBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadChannelBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadSoundBlock(BYTE* pBuffer, DWORD dwOffset);
        HRESULT ReadSelectMarkBlock(BYTE *pBuffer, DWORD dwOffset);
        HRESULT ReadExpBlock(BYTE* pBuffer, DWORD dwOffset);

        // sfx存储
        DWORD     WriteGeneralInformationBlock(FILE *pFile);
        DWORD     WriteParticleKindBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteLauncherBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteForceFieldBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteTextureBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteBillboardBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteBoxProjBlock(FILE* pFile, DWORD dwIndex);
        DWORD     WriteTrackBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteModelBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteMeteorBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteMagicParpetBlock(FILE *pFile, DWORD dwIndex);
        DWORD     WriteChannelBlock(FILE *pFile);
        DWORD     WriteSoundBlock(FILE* pFile, DWORD dwIndex);
        DWORD     WriteSelectMarkBlock(FILE* pFile);
        DWORD     WriteExpBlock(FILE* pFile);
        void SortBillboard();
        BOOL      ExpEffectValid();
        BOOL      HaveTimeScalingEffect();

        KG3DSFXAnimationController* m_pSFXAnimationController;

        //  更新包围盒
        HRESULT CalculateAABB();
        HRESULT UpdateMatrix();

        HRESULT PlayAnimationImpl(SFX_ANIMATION_PLAY sfxAnimationPlay, float fSpeed,
int nOffsetTime);

public:
        //Footprint/////////////    add by huangjinshou 2008-03-14
        HRESULT RenderFootprint();
        HRESULT AddSelectMark(IEKG3DSFXSelectMark ** ppSelectMark);
        HRESULT RemoveSelectMark();
```

```
    virtual     HRESULT Playing();
    DWORD m_dwLastTimeUpdateFootprint;
    //////////////////////
public :
    // 粒子属性序列帧
    vector<KG3DSFXParticleLifeTimeLine*>            m_vecParticleLifeTimeLine;
    // 力场
    vector<KG3DSFXParticleForceField*>              m_vecSFXParticleForceField;
    // 发射器
    vector<KG3DSFXParticleLauncher*>                m_vecSFXParticleLauncher;
    // 粒子管理容器
    KG3DSFXParticleContainer                        m_ParticleContainer;

    // 模型动画颜色序列帧
    typedef TKG3DTimeLine<D3DXCOLOR>                COLOR_LINE_T;
    typedef vector<COLOR_LINE_T*>                   MDL_COLOR_LINE_T;
    // faint!
    vector<MDL_COLOR_LINE_T*>                       m_vecSubsetColor;

    typedef TKG3DTimeLine<INT>                      MDL_ANI_BOUND_T;
    vector<MDL_ANI_BOUND_T*>                        m_timelineModelAniBound;

public :
    // 纹理贴图
    vector<KG3DTexture*>                            m_vecTexture;
    // 纹理分割
    vector<UINT>                                    m_vecCuttingNum;

    // 当前帧
    float                   m_fCurrentFrame;
    // 上一帧
    float                   m_fPrevFrame;
    // 总帧
    DWORD                   m_dwTotalFrameNum;
    // 帧间隔
    DWORD                   m_dwFrameInterval;
    // 上一次更新时间
    DWORD                   m_dwLastTime;
    // 当前和上一次更新时间差
    float                   m_fDelayTime;
    // 开始播放时间
    DWORD                   m_dwStartTime;
    // 模型动画开始
    BOOL                    m_bModelAnimationStarted;
```

```
// 特效播放模式
SFX_ANIMATION_PLAY    m_eSFXAnimationPlay;
SFX_ANIMATION_PLAY    m_eSFXAnimationPlaySave;
// 模型动画播放模式
DWORD                 m_dwModelPlayMode;
// 废弃
float                 m_fShockWavePower;
float                 m_fLastParticlFrame;
float                 m_fFixedFrame;

// 播放速度
float                 m_fSpeed;

// 公告板
vector<KG3DSFXBillboard*>     m_vecBillboard;
// 轨道
vector<KG3DSFXTrack*>         m_vecTrack;
// 拖尾
vector<KG3DSFXMeteor*>        m_vecMeteor;
// 贴地面片
vector<KG3DSFXMagicParpet*> m_vecMagicParpet;
// 通道
KG3DSFXChannel*               m_pChannel;
KG3DSFXSelectMark*            m_pSelectMark;    // Add by huangjinshou 2008 1-15
// 特殊效果
KG3DSfxExp*                   m_pExpEffect;
// 立方投影
vector<KG3DSfxBoxProjection*> m_vecBoxProj;

// 公告板是否排序
BOOL            m_bSortBillboard;

// 缩放
float           m_fSizeScale;

// 静态包围盒
AABBOX          m_staticAABB;

// 模型动画开始帧
float       m_fModelAniStartFrame;

// 是否播放完成，对于只播放一次的特效有效
int             m_nPlayFinishFlag;
int             m_nBillBoardMdlNum;
```

```
// 废弃
CallbackRender    m_fpCallBackRender;
void*             m_pUserData;
DWORD             m_dwRenderTarIndex;

// 禁止缩放
DWORD             m_dwFixedSize;          // 0 : not 1 : yes
// 锁定y轴
DWORD             m_dwLockYAlex;          // 0 : not 1 : yes

// 版本
int               m_nVerson;
// 是否在编辑
int               m_nEditor;
DWORD             m_dwLoadOption;

int               m_nTexLoadIndex;
int               m_nMdlLoadIndex;

DWORD             m_dwMaxParticleNum;

struct _SRT {

    D3DXVECTOR3 vScanl;
    D3DXVECTOR3 vTrans;
    D3DXMATRIX    matRot;

    inline D3DXMATRIX* ScanlMat(D3DXMATRIX* mat) {
        return D3DXMatrixScaling(mat, vScanl.x, vScanl.y, vScanl.z);
    }

    inline D3DXMATRIX* TransMat(D3DXMATRIX* mat) {
        return D3DXMatrixTranslation(mat, vTrans.x, vTrans.y, vTrans.z);
    }

};

// 位置信息
_SRT              m_Srt;


// 模型动画信息
struct _SfxModelInfo
```

```
    {
        KG3DModel* pSFXModel;

        FLOAT          fPlayStart;
        FLOAT          fPlaySpeed;
        DWORD          dwPlayMode;
        DWORD          dwNumFrame;
        FLOAT          fFrameTime;
        INT            nLeftFrame;
        INT            nRighFrame;
        INT            nCycleFlag;
        INT            nValidFlag;
        INT            nTrackIndex;
        DWORD      dwIsBillBoard;

        _SfxModelInfo()
        {
            pSFXModel    = NULL;
            fPlaySpeed = 1.0f;
            fPlayStart = 0.0f;
            dwPlayMode = 0;
            dwNumFrame = 0;
            fFrameTime = 0.0f;
            nLeftFrame = 0;
            nRighFrame = 0;
            nCycleFlag = FALSE;
            nValidFlag = FALSE;
            nTrackIndex = -1;
            dwIsBillBoard = FALSE;
        }
    };
    vector<_SfxModelInfo> m_vecModelsInfo;


    // 音效
    TKG3DTimeLine<_SoundInfo> m_tlSoundKey;
    //TKG3DTimeLine<SFX_SOUND_BLOCK> m_vecSoundKey;

    DWORD m_dwUserData;
};
```

```
class KG3DSFXParticle
{
    friend class KG3DSFXParticleContainer;

public :

    // 粒子被发射回调
    virtual HRESULT OnLaunch(
        KG3DSFX*                        pParentSFX,             // 粒子所属的特效
        KG3DSFXParticleLifeTimeLine* pLifeTimeLine,             // 粒子属性
        SFX_PARTICLE_TYPE               eParticleType,          // 类型
        SFX_PARTICLE_UPDATE_MODE        eParticleUpdateMode,    // 更新模式
        SFX_BLEND_MODE                  eBlendMode,             // 混合模式
        DWORD                           dwTrackIndex,           // 轨道id
        float                           fLife,                  // 生命
        D3DXVECTOR3*                    pvPosition,             // 发射位置
        D3DXVECTOR3*                    pvVelocity,             // 速度
        float                           fFrame,                 // 发射争
        DWORD                           dwRandTexDelay,         // 随即贴图因子
        KG3DSFXParticleLauncher*        pLauncher               // 发射它的发射器
    );


    // 更新
    virtual HRESULT Update(float fCurrentFrame);


    // 把粒子的显示数据提交到粒子容易的buffer
    virtual HRESULT PrepareVertexBuffer(KG3DSFXParticleContainer *pContainer );


    // 粒子的属性
    KG3DSFXParticleLifeTimeLine* GetLifeTimeLine() const { return m_pLifeTimeLine; };

    // 得到粒子形状信息，一个矩形和它的uv
    HRESULT GetRectInfo(
        D3DXVECTOR3* pv1,                   /* vector 1 */
        D3DXVECTOR3* pv2,                   /* vector 2 */
        D3DXVECTOR3* pv3,                   /* vector 3 */
        D3DXVECTOR3* pv4,                   /* vector 4 */
        FRECT*          pRectfText,         /* texture rect */
        const D3DXVECTOR3& vWorldPos    /* world position */
        );
```

```
    // 得到粒子的世界举阵
    void GetWorldPos(D3DXVECTOR3* pWorldPos, const D3DXVECTOR3& vLocalPos);

    // 粒子的举行数目，普通粒子只是个矩形面
    virtual size_t GetRectNum()       { return 1; }

   /// 得到粒子类型
    virtual WORD GetPatricleType() { return PARTICLE_TYPE_NORMAL; }


public :
    KG3DSFXParticle(UINT uIndex);
    virtual ~KG3DSFXParticle();

public :
    SFX_PARTICLE_UPDATE_MODE      m_eParticleUpdateMode;
    SFX_PARTICLE_TYPE             m_eParticleType;
    SFX_BLEND_MODE                m_eBlendMode;
    KG3DSFXParticleLauncher*      m_pLauncher;

    D3DXVECTOR3       m_vPositionWorld;
    D3DXVECTOR3       m_vPosition;
    float             m_fWidth;
    float             m_fHeight;
    float             m_fRotation;
    float             m_fRotationStep;
    D3DCOLOR          m_dwColor;
    float             m_fLife;
    float             m_fPreFrame;
    float             m_fStartFrame;
    D3DXVECTOR3       m_vPrevPosition;


    /* If the particle has been bind to track, then m_vVelocity.x means
       particle's speed, and m_vVelocity.y means the distance which the particle
       has moved. In fact, the distance will be passed as current frame to track
       to get a transformation later. */
    D3DXVECTOR3       m_vVelocity;
    float             m_fVelocity;


    KG3DSFXParticleLifeTimeLine* m_pLifeTimeLine;
    KG3DSFX*                     m_pParentSFX;
```

```
    UINT                            m_uPoolIndex;
    DWORD                           m_dwTrackIndex;

    float                       m_fRotationX;
    float                       m_fRotationY;
    float                       m_fRotationZ;
    D3DXVECTOR3                     m_vRotStep;

    DWORD                       m_dwRandomTextureFrame;

    /* random */

    DWORD m_dwRandTexDelay;
    DWORD m_dwLastRandTime;
    DWORD m_dwRandColDelay;
    DWORD m_dwLastRandColTime;
    DWORD    m_dwRandRotDelay;
    DWORD    m_dwLastRandRotTime;
    DWORD    m_dwRandAlpDelay;
    DWORD    m_dwLastRandAlpTime;
    DWORD    m_dwRandFcRotDelay;
    DWORD    m_dwLastRandFcRotTime;
    float    m_fLastRorceFactor;
};


class KG3DSFXParticleLauncher : public IEKG3DSFXParticleLauncher, public
KG3DTransformation
{
public :

    // 发射粒子
    HRESULT LaunchParticle(float fCurrentFrame);
    // 更新
    HRESULT FrameMove();
    // 绘制编辑辅助线
    HRESULT EditorRender();
    HRESULT RenderSelectMark(DWORD dwColor);

    // 绑定到轨道
    HRESULT BindToTrack(DWORD dwIndex);
    // 取消绑定
    HRESULT UnBindToTrack();
```

```
    // 是否绑定
    BOOL       IsBindToTrack() const;
    // 得到轨道的索引
    DWORD      GetBindTrackIndex() const;

    // 克隆
    HRESULT Clone(KG3DSFXParticleLauncher **ppLauncher);

    // 加入影响力场
    virtual HRESULT AddEffForce(IEKG3DSFXParticleForceField* pForce);
    // 删除
    virtual HRESULT DelEffForce(IEKG3DSFXParticleForceField* pForce);
    // 查询
    virtual BOOL       HasEffForce(IEKG3DSFXParticleForceField* pForce);

    // 废弃
    HRESULT                    EnableSpeedInherit(BOOL                    bInherit)
{ m_bSpeedInherit = bInherit; return S_OK; }
    BOOL       IsSpeedInherit()                                    { return
m_bSpeedInherit; }

    // 获取设置绑定到轨道模式
    SFX_LAUNCHER_BIND_TO_TRACK_MODE        GetBindToTrackMode()        const
{ return m_eBindToTrackMode;   }
    void      SetBindToTrackMode(SFX_LAUNCHER_BIND_TO_TRACK_MODE      eMode)
{ m_eBindToTrackMode = eMode; }

    // 获取设置发射粒子类型
    SFX_PARTICLE_TYPE GetParticleType() const                          { return
m_eParticleType; };
    void SetParticleType( SFX_PARTICLE_TYPE eParticleType)             { m_eParticleType
= eParticleType; }

    // 发射器形状
    SFX_LAUNCHER_SHAPE GetLauncherShape() const                        { return
m_eLauncherShape; }
    void          SetParticleShape(          SFX_LAUNCHER_SHAPE          eParticleShape)
{ m_eLauncherShape = eParticleShape; }

    // 发射粒子更新模式
    SFX_PARTICLE_UPDATE_MODE    GetParticleUpdateMode()    const        {    return
m_eParticleUpdateMode; }
    void    SetParticleUpdateMode(SFX_PARTICLE_UPDATE_MODE    eParticleUpdateMode)
```

```
{    m_eParticleUpdateMode = eParticleUpdateMode; }

    // 发射粒子混合模式
    HRESULT                SetBlendMode(SFX_BLEND_MODE                eBlendMode)
{ m_eBlendMode = eBlendMode; return S_OK; }
    SFX_BLEND_MODE GetBlendMode()                                    {    return
m_eBlendMode; }

    // 发射器形状参数
    float GetShapeParam1() const                                     { return
m_fLauncherShapeParam1;     }
    float GetShapeParam2() const                                     { return
m_fLauncherShapeParam2;     }
    void                                SetShapeParam1(float                fValue)
{ m_fLauncherShapeParam1 = fValue; }
    void                                SetShapeParam2(float                fValue)
{ m_fLauncherShapeParam2 = fValue; }

    // 发射粒子属性
    void    SetParticleLifeTimeLine(IEKG3DSFXParticleLifeTimeLine* pParticleLifeTimeLine);
    virtual IEKG3DSFXParticleLifeTimeLine* GetParticleLifeTime();
    KG3DSFXParticleLifeTimeLine*    GetParticleLifeTimeDirect()      {    return
m_pParticleLifeTimeLine; };
    KG3DSFX* GetParentSFX() const;

    // 包围盒
    void GetAABBox(AABBOX *pBox);

    // 随即贴图延迟
    virtual HRESULT SetRandTexDelay(DWORD dwDelay)          { m_dwRandTexDelay
= dwDelay; return S_OK; }
    virtual  DWORD      GetRandTexDelay()                            { return
m_dwRandTexDelay; }

    virtual HRESULT SetRandColDelay(DWORD dwDelay)          { m_dwRandColDelay
= dwDelay; return S_OK; }
    virtual  DWORD      GetRandColDelay()                            { return
m_dwRandColDelay; }

    virtual HRESULT SetRandRotDelay(DWORD dwDelay)          { m_dwRandRotDelay
= dwDelay; return S_OK; }
    virtual  DWORD      GetRandRotDelay()                            { return
m_dwRandRotDelay; }
```

```
    virtual HRESULT SetRandAlpDelay(DWORD dwDelay)              { m_dwRandAlpDelay
= dwDelay; return S_OK; }
    virtual  DWORD      GetRandAlpDelay()                                  { return
m_dwRandAlpDelay; }

    virtual          HRESULT          SetRandForceRotDelay(DWORD          dwDelay)
{ m_dwRandForceRotDelay = dwDelay; return S_OK; }
    virtual  DWORD      GetRandForceRotDelay()                             { return
m_dwRandForceRotDelay; }


    // 位置变换，参看模型

    virtual  void  SetTranslation(D3DXVECTOR3*  pValue)                    { return
KG3DTransformation::SetTranslation(pValue); }
    virtual  void  SetRotation(D3DXQUATERNION*  pValue)                    { return
KG3DTransformation::SetRotation(pValue); }
    virtual  void  SetRotationCenter(D3DXVECTOR3*  pValue)                 { return
KG3DTransformation::SetRotationCenter(pValue); }
    virtual  void  SetScaling(D3DXVECTOR3*  pValue)                        { return
KG3DTransformation::SetScaling(pValue); }
    virtual  void  SetScalingRotation(D3DXQUATERNION*  pValue)             { return
KG3DTransformation::SetScalingRotation(pValue); }
    virtual  void  SetScalingCenter(D3DXVECTOR3*  pValue)                  { return
KG3DTransformation::SetScalingCenter(pValue); }

    virtual  void  GetTranslation(D3DXVECTOR3*  pValue)                    { return
KG3DTransformation::GetTranslation(pValue); }
    virtual  void  GetRotation(D3DXQUATERNION*  pValue)                    { return
KG3DTransformation::GetRotation(pValue); }
    virtual  void  GetRotationCenter(D3DXVECTOR3*  pValue)                 { return
KG3DTransformation::GetRotationCenter(pValue); }
    virtual  void  GetScaling(D3DXVECTOR3*  pValue)                        { return
KG3DTransformation::GetScaling(pValue); }
    virtual  void  GetScalingRotation(D3DXQUATERNION*  pValue)             { return
KG3DTransformation::GetScalingRotation(pValue); }
    virtual  void  GetScalingCenter(D3DXVECTOR3*  pValue)                  { return
KG3DTransformation::GetScalingCenter(pValue); }

    virtual  void  UpdateTransformation()                                  { return
KG3DTransformation::UpdateTransformation(); }
    virtual  void  UpdateByMatrix(const  D3DXMATRIX&  mat)                 { return
KG3DTransformation::UpdateByMatrix(mat); }
```

```
    virtual DWORD GetMotionType()                                   { return
m_dwMotionType;   }
    virtual void   SetMotionType(DWORD dwType)                      { m_dwMotionType
= dwType;}

    virtual DWORD GetForceOption()                                  { return
m_dwForceOption; }
    virtual void   SetForceOption(DWORD optn)                       { m_dwForceOption =
optn; }

    virtual KGRTSTimeLine*          GetRtsTimeLine()                { return
&m_RtsTimeLine;                 }
    virtual TKG3DTimeLinebase*  GetParticleNumLine()                { return
&m_fParticleNumLine;          }
    virtual TKG3DTimeLinebase*  GetParticleSpeedMinLine()           { return
&m_fParticleSpeedMinLine;   }
    virtual TKG3DTimeLinebase*  GetParticleSpeedMaxLine()           { return
&m_fParticleSpeedMaxLine;   }
    virtual TKG3DTimeLinebase*  GetParticleLifeMinLine()            { return
&m_fParticleLifeMinLine;    }
    virtual TKG3DTimeLinebase*  GetParticleLifeMaxLine()            { return
&m_fParticleLifeMaxLine;    }
    virtual TKG3DTimeLinebase*  GetParticleScatterMin()             { return
&m_fParticleScatterMin;     }
    virtual TKG3DTimeLinebase*  GetParticleScatterMax()             { return
&m_fParticleScatterMax;     }

    virtual void InsertRtsKeyFrame(int nFrame, DWORD dwFlag)
{ m_RtsTimeLine.InsertKeyFrameSmark(*this, nFrame, dwFlag);}

    // 缩放所有序列帧
    void ScanlTimeLine(float scanl) {
        m_fParticleNumLine.Scanl(scanl);
        m_fParticleLifeMaxLine.Scanl(scanl);
        m_fParticleLifeMinLine.Scanl(scanl);
        m_fParticleSpeedMaxLine.Scanl(scanl);
        m_fParticleSpeedMinLine.Scanl(scanl);
        m_fParticleScatterMax.Scanl(scanl);
        m_RtsTimeLine.Scanl(scanl);
    }


    // 编辑相关
    virtual BOOL IsHide()                   { return m_nHide;           }
```

```
    virtual void Show(BOOL enable)          { m_nHide = enable;             }
    virtual LPCTSTR GetName()                 { return m_scName.c_str();    }
    virtual void SetName(LPCTSTR szName){ m_scName = szName;            }

public :
    // 发射数量序列帧
    TKG3DTimeLine<float>                    m_fParticleNumLine;
    // 发射粒子生命最大值序列帧
    TKG3DTimeLine<float>                    m_fParticleLifeMaxLine;
    // 发射粒子生命最小值序列帧
    TKG3DTimeLine<float>                    m_fParticleLifeMinLine;
    // 发射粒子速度最大值序列帧
    TKG3DTimeLine<float>                    m_fParticleSpeedMaxLine;
    // 发射粒子速度最小值序列帧
    TKG3DTimeLine<float>                    m_fParticleSpeedMinLine;
    // 发射粒子缩放最小值序列帧
    TKG3DTimeLine<float>                    m_fParticleScatterMin;
    // 发射粒子缩放最大值序列帧
    TKG3DTimeLine<float>                    m_fParticleScatterMax;

    // 发射器位置信息序列帧
    KGRTSTimeLine                           m_RtsTimeLine;

    // 影响发射器的力场
    std::vector<IEKG3DSFXParticleForceField*> m_vecEffForces;
    std::vector<DWORD>                              m_vecEffForcesIDs; // use to load and
save

    D3DXMATRIX                              m_matWorld;

    int                                     m_nSelFlag;
    DWORD                                   m_dwMotionType;
    DWORD                                   m_dwForceOption;

    float                                   m_fParticleNumKeep;

public :
    KG3DSFXParticleLauncher(
        KG3DSFX*                        pParentSFX,
        KG3DSFXParticleContainer* pContainer,
        SFX_PARTICLE_TYPE               eParticleType,
        SFX_LAUNCHER_SHAPE              eLauncherShape,
        SFX_PARTICLE_UPDATE_MODE    eParticleUpdateMode
    );
```

```
        virtual ~KG3DSFXParticleLauncher();

private :
        void GetParticleDirectionAndPos(
                float fScatterMin,
                float fScatterMax,
                D3DXVECTOR3 *pvDirection,
                D3DXVECTOR3 *pvPosition
        );

        void MakeParticle(
                float fLife,
                D3DXVECTOR3 *pvPosition,
                D3DXVECTOR3 *pDirection,
                float fFrame
        );

protected :
        // 发射粒子混合模式
        SFX_BLEND_MODE                  m_eBlendMode;
        BOOL                            m_bSpeedInherit;
        // 发射粒子类型
        SFX_PARTICLE_TYPE               m_eParticleType;
        // 发射器形状
        SFX_LAUNCHER_SHAPE              m_eLauncherShape;
        // 发射粒子更新模式
        SFX_PARTICLE_UPDATE_MODE        m_eParticleUpdateMode;

        // 上一次发射帧
        float                           m_fLastLauncherFrame;

        // 发射器形状参数
        float                           m_fLauncherShapeParam1;
        float                           m_fLauncherShapeParam2;

        // 发射粒子属性
        KG3DSFXParticleLifeTimeLine     *m_pParticleLifeTimeLine;
        // 粒子容器
        KG3DSFXParticleContainer        *m_pParticleContainer;
        // 从属的特效
        KG3DSFX                         *m_pParentSFX;

        // 绑定的轨道index
```

```
    DWORD                                   m_dwBindTrackIndex;
    // 绑定到轨道的模式
    SFX_LAUNCHER_BIND_TO_TRACK_MODE         m_eBindToTrackMode;

};


class KG3DSFXParticleContainer
{
    friend class KG3DSFX;

public :

    // 渲染容器中所有活动粒子
    HRESULT RenderParticle(const KG3DSFX* pSfx, float fCurrentFrame);

    // 活动粒子数量
    UINT GetParticleCount() const;

    // 增加一个活动粒子网格数据
    HRESULT PushRectangleToVB(
        D3DXVECTOR3*            pVector1,
        D3DXVECTOR3*        pVector2,
        D3DXVECTOR3*        pVector3,
        D3DXVECTOR3*        pVector4,
        D3DCOLOR            dwColor,
        FRECT*              pTexCoord
    );

    // 设置贴图总数
    HRESULT SetTextureNum(UINT uTextureNum);

    // 创建一个粒子
    KG3DSFXParticle* NewParticle(UINT uTexIndex, UINT uBlendMode, WORD wType);


public :
    KG3DSFXParticleContainer();
    virtual ~KG3DSFXParticleContainer();


private :

    // 为渲染准备buffer
```

```
    HRESULT PrepareVertexBuffer(const vector<KG3DSFXParticle*>& vecParticles, size_t uSize, size_t* pRectNum);

    // 更新所有粒子
    HRESULT UpdateParticle(float fCurrentFrame);

    // 设置渲染状态
    void SetRenderState(KG3DRenderState& R, UINT eBlendMode);


public :

    /*
     the particles use the same texture, in one texture sort order by blend mode,
     there are 4 blend mode : alpha, add, inv, screen
     */
    struct SortedRectParticle {
        std::vector<KG3DSFXParticle*> m_vecRectParticle[SFX_BM_NUM];
        UINT m_NextFreeIndex[SFX_BM_NUM];
        SortedRectParticle();
        ~SortedRectParticle();
        void Clear();
        KG3DSFXParticle* NewParticle(UINT uBlendMode, WORD wType);
        void UpdateParticle(float fCurrFrame);
    };

    std::vector<SortedRectParticle*> m_vecSortedRectParticle;

    UINT m_uSfxTextNum;

    static LPDIRECT3DVERTEXBUFFER9 m_spVertexBuffer;
    static int m_sVectorNumber;

    VFormat::_Faces_Diffuse_Texture1* m_pRectangleVBData;

    DWORD m_dwActiveNumber;
    UINT    m_uPrimitiveNum;

};
```