# 粒子批次处理框架

1. 新建RenderManager类,管理创建的所有Render.该类不属于任何层级.
2. 把Technique里面的Renderer对象在RenderManager类里面备份一份.根据材质和分组来管理Renderer.
3. 创建特效实体时,添加接口setRenderGroup(size\_t id),设置渲染组ID.根据这个ID来确定渲染批次.
4. 之前mRender在Technique里的某些操作修改.同事RenderManager里存储的Renderer对象也要做添加删除的操作.

难点

1. 根据材质和渲染组ID来存储Renderer.

设两层map.一层用材质名作为key值,Renderer指针为value值.

外面一层用groupID作为key值,第一层的map作为value值.

查询,增加和删除操作都优先遍历groupID操作.

# 接口

新建RendererWrapper类,封装下ParticleRenderer

class RendererWrapper

{

public:

RendererWrapper();

~RendererWrapper();

ParticleRenderer\* getRenderer();

void setRenderer(ParticleRenderer\* renderer);

public:

int mRefCount;

private:

ParticleRenderer\* mRenderer;

};

设计RenderManager类,需要的功能

class \_ParticleUniverseExport ParticleRendererManager :

public Singleton<ParticleRendererManager>

{

typedef map<const String, RendererWrapper\*> RendererMap;

typedef map<uint8, RendererMap> GroupRenderer;

typedef RendererMap::iterator RendererMapIterator;

typedef GroupRenderer::iterator GroupRendererIterator;

public:

ParticleRendererManager();

~ParticleRendererManager();

// 创建Renderer

ParticleRenderer\* createRenderer(ParticleTechnique\* technique, uint8 groupID, const String& rendererType);

// 移除Renderer

void removeRenderer(ParticleRenderer\* renderer);

// 销毁Renderer

void destroyRenderer(ParticleRenderer\* renderer);

// 获取/设置 renderer是否初始化.

bool isRendererInitialised(ParticleRenderer\* renderer);

void setRendererInitialised(ParticleRenderer\* renderer, bool rendererInitialised);

//

void \_prepareRenderer(ParticleRenderer\* renderer);

//

void \_unprepareRenderer(ParticleRenderer\* renderer);

// 获取PU文件名 加上Technique在ParticleSystem里的序号

String getPSNameIndex(ParticleTechnique\* technique);

// 在map中找到相应的renderer.

ParticleRenderer\* findRendererInMap(ParticleRenderer\* renderer);

//在map中找到相应的renderer.(createRenderer时调用.此时还没有renderer)

ParticleRenderer\* findRendererInMap(uint8 groupid, ParticleTechnique\* technique);

// 获取指针

static ParticleRendererManager& getSingleton (void);

// 获取对象

static ParticleRendererManager\* getSingletonPtr (void);

private:

// 存储的Renderer列表

GroupRenderer mGroupRenderer;

RendererMap mRendererMap;

RendererWrapper\* mRendererWrapper;

};