GameMaster Studio

Hoshizora核心组件开发指南

Griffin Bu - darkfall@gamemastercn.com

SoraRenderSystem

```
bool update()
```

更新RenderSystem状态，帧数控制在此进行.

通常的实现为

```
while(g_timer->update()) {

      SoraCore::Instance()->update();
}
```

然后再由SoraCore回调主窗口的update和render.

在如果RenderSystem作为已有的渲染引擎的配接器时(例如HGERenderer)，往往已有的渲染引擎需要使用自己的帧速控制，并且注册回调，这时候可以使用SoraInfiniteRendererCallback.h内的bool_updateFrame，void_updateFrame，it_exit等函数注册为已有的渲染引擎的回调，然后把已有的渲染引擎的帧数设置为不限制.从而让Sora取得帧数控制权.

例如在使用hge时，可以使用如下方式让hge的帧数控制来回调Sora的帧数控制函数，从而让Sora核心取得帧数控制权

```
      hge->System_SetState(HGE_FPS, HGE_INFINITE);

      hge->System_SetState(HGE_FRAMEFUNC, bool_updateFrame);

      hge->System_Start();
```

或者也可以采用直接修改已有渲染引擎的方式(SoraHGERenderer)，但是这样不利于已有渲染引擎的维护，如果已有渲染引擎升级频繁，将导致维护工作难度加大，一般只适用于稳定且修改难度系数低的引擎，例如hge.

```
void beginScene(ulong32 c, ulong32 t)
```

开始新scene/target的渲染

c为屏幕/target清空颜色, t为RenderTarget

在t不为零的情况下, RenderSystem应当使用t作为贴图渲染目标, 在遇到endScene之前所有的渲染都应该渲染到目标而不是屏幕.

对于用户来说, RenderTarget是通过
```
    createTarget(width, height, bzbuffer)
```
函数来创建的, 所以在RenderSystem内部, 一个RenderTarget的实现起码应该包含Target的大小, Target的贴图, Target的ZBuffer缓冲.

```
void endScene()
```

结束scene/target的渲染.

```
void beginFrame()
```

帧渲染开始函数, 由SoraCore在一帧的最初回调. 通知RenderSystem准备开始渲染.

```
void endFrame()
```

帧渲染开始函数, 由SoraCore在一帧的最末回调. 通知RenderSystem一帧已经结束了.

```
void start(SoraTimer* timer)
```

Sora引擎开始运作, 由SoraCore在createWindow成功时回调. timer为使用的Timer, 在使用SoraInfiniteRendererCallback.h里的函数作为回调前, RenderSystem有义务设置g_timer = timer, 否则将导致程序错误.

```
ulong32 createTarget(int width, int height, bool zbuffer=true)
```

创建一个RenderTarget

width和height表示大小, zbuffer表示是否使用zbuffer深度缓冲.

用户可以创建多个Target, RenderSys-
tem内部应该使用链表之类的结果储存所有的Target.

返回值应当是这个RenderTarget唯一的Handle, 一个比较简单的实现是直接把Ren-
derTarget的指针转为ulong32.

```
void freeTarget(ulong32 t)
```

释放一个RenderTarget

参数是createTarget返回的Handle

```
ulong32 getTargetTexture(ulong32 t)
```

获取Target的Texture的Handle, 返回值应当和HSORATEXTURE同含义

参数是createTarget返回的Handle

```
SoraWindowHandle createWindow(SoraWindowInfoBase* windowInfo)
```

创建主窗口函数

RenderSystem在这里应该依据操纵系统创建供Sora使用的主窗口.

SoraWindowInfoBase指针是用户要创建的SoraWindow的指针, 可以从那里获取
Window的大小, Window的位置, Window的Title等信息.

创建成功之后应该返回表示这个Window的唯一的Handle, 通常是直接转换windowIn-
fo指针到ulong32

如果返回0, 则是通知SoraCore创建失败, 将导致引擎关闭.

如果在已经创建主窗口的情况下用户再次创建一个主窗口，则RenderSystem应提供变化窗口大小，位置和标题功能，以在不同的主窗口之间切换.

同时SoraCore内的主窗口会切换到新创建的主窗口.

贴图相关函数:

这些函数都返回SoraTexture结构, 通常SoraCore会转换为ulong32句柄返回给用户, 以避免用户修改贴图的属性.
同时, 如果RenderSystem接受到ulong32或者HSORATEXTURE作为参数的贴图, 转换到SoraTexture*总是是安全的.

```
class SoraTexture
    mTextureID: 储存贴图的Id, 在DirectX下可能是LPDIRECT3D9TEXTURE, 在OpenGL以及OpenGLES下则为Gluint.
    mTextureWidth, mTextureHeight: 贴图在内存中的宽和高, 通常为2的整数倍
    mOriginalWidth, mOriginalHeight: 贴图的实际宽高
在创建贴图的时候, 这些值必须被正确的设置, 否则可能会导致Sprite Transform错误.
```

```
SoraTexture* createTexture(
    const SoraWString& sTexturePath,
    bool bMipmap=false
)
```

从文件创建贴图函数, 由于资源管理系统的存在, 改函数暂时无用.

```
SoraTexture* createTextureWH(int32 w, int32 h)
```

创建一张指定宽, 高的空贴图

```
SoraTexture* createTextureFromMem(
    void* ptr,
    ulong32 size,
    bool bMipmap=false
)
```

从贴图数据中创建一张贴图, ptr为指向数据块的指针, size为数据块大小, 这两个参数通常为getResourceFile的返回值

```
SoraTexture* createTextureFromRawData(
    unsigned int* data,
    int32 w,
    int32 h
)
```

从原始颜色数据中创建一张贴图, 数据必须为R8G8B8A8格式, w和h分别代表宽和高

```
uint32* textureLock(SoraTexture* ht)
```

锁定一张贴图，返回贴图的原始颜色数据指针，最好复制一份贴图数据保存到内部.
SoraTexture内有dataRef域，可以用于储存数据.
被锁定的贴图必须做好被修改的准备.

```
void textureUnlock(SoraTexture* ht)
```

解除对一张贴图的锁定，把修改过的数据放回贴图.

```
void releaseTexture(SoraTexture* pTexture)
```

删除一张贴图，释放内存.
这个函数通常由用户手动调用或者由SoraTextureMap调用(在开启贴图Cache的情况下)

渲染相关函数:

```
void renderQuad(SoraQuad& quad)

void renderTriple(SoraTriple& trip)

void renderWithVertices(
    SoraTexture* tex,
    int32 blendMode,
    SoraVertex* vertices,
    uint32 vsize,
    int32 mode
)

void renderRect(
    float32 x1,
    float32 y1,
    float32 x2,
    float32 y2,
    float32 fWidth,
    uint32 color=0xFFFFFFFF,
    float32 z=0.0f
)


void renderBox(
    float32 x,
    float32 y,
    float32 width,
    float32 height,
    uint32 color=0xFFFFFFFF,
    float32 z=0.0f
)

void setClipping(
    int32 x=0,
    int32 y=0,
    int32 w=0,
    int32 h=0
)
```

```
void setTransform(
    float32 x=0.f,
    float32 y=0.f,
    float32 dx=0.f,
    float32 dy=0.f,
    float32 rot=0.f,
    float32 hscale=0.f,
    float32 vscale=0.f
)

void setTransformWindowSize(float32 w, float32 h)
void setViewPoint(
    float32 x=0.f,
    float32 y=0.f,
    float32 z=0.f
)
```