HGE 引擎教程

1.简介

HGE 是一个硬件加速（Hardware accelerated）的 2D 游戏引擎（Game Engine），HGE 是一个富有特性的中间件，可以用于开发任何类型的 2D 游戏．HGE 封装性良好，以至于你仅仅需要关系游戏逻辑（Game Logic），而不需要在意 DirectX，Windows 消息循环等．

HGE 架构在 DirectX 8.0 之上，能够跑在大多数的 Windows 系统上．

1. 选用 HGE 的理由：

1）专业化 --- 专注于 2D 领域

2）简单化 --- 非常容易使用

3）技术优势 --- 基于 Direct3D API 有较好的性能和特性

4）免费 --- 对于个人或者商业用户都免费，遵循 zlib/libpng license

5）代码高度的一致性 --- 代码是否具有一致性，是衡量代码质量的标准之一（<< Code Reading: The Open Source Perspective >>）

2. 体系结构：

HGE 有 3 个抽象层（layers of abstraction）：

1）核心函数（Core Functions）

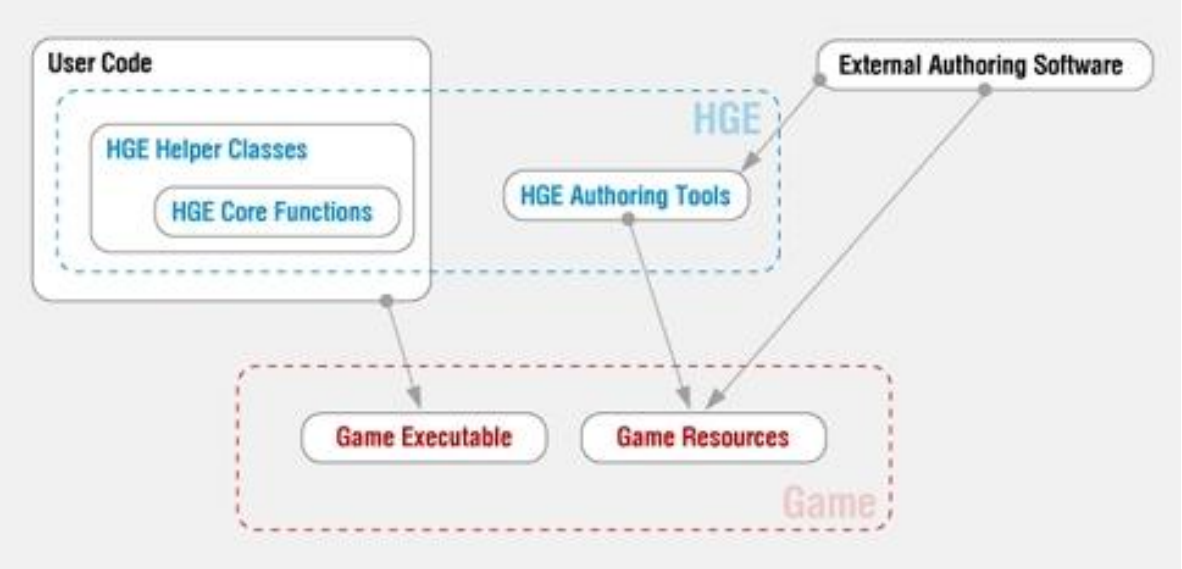
处于核心的函数和例程（routines），被整个系统所依赖．

2）辅助类（Helper Classes）

游戏对象相关的类，架构于 HGE Core Functions 层之上，辅助用户进行游戏开发．

3）创作工具（Authoring Tools）

用于游戏开发的一组工具．



从上图可以看见 ：

1) 用于代码只需要架构在 HGE Helper Classes 之上

2) 通常游戏资源 (Game Resources) 需要使用 HGE 创作工具来产生

3. 体系结构概述 ：

1) Core Functions 层

<1> 图形格式支持 ： 支持 BMP, JPG, PNG, TGA, DDS, DIB 文件格式

<2> 支持窗口模式和全屏模式

<3> 音频支持和音乐回放 (music playback) ： 支持 WAV, MP3, MP2, MP1 and OGG 音频文件格式 (audio file formats) ，支持 MO3, IT, XM, S 3M, MTM, MOD and UMX 音乐文件格式 (music file formats) ，支持压缩流的回放 . 声音大小和声道的控制

<4> 输入设备支持 ： 鼠标和键盘

<5> 资源 ： 读取硬盘上的资源 ，支持 ZIP 打包的文件格式

<6> 日志支持

2) Helper Classes 层

<1> 精灵 (Sprites) 和动画 (Animations)

对于所有硬件设备特性的直接支持：锚点（anchor）支持，伸展．缩放．旋转的支持，不同的回放模式的支持

<2> 字体

读取和渲染（render）位图字体，多种字体排列方式，旋转和缩放字体，字符串宽度计算等

<3> 粒子系统（particle systems）和网格变形（distortion mesh）

高效的粒子系统，可用于创建烟雾．爆炸．魔法效果等，提供粒子系统的管理，支持定界盒（bounding box）计算和冲突检测（collision detection）

<4> 资源管理：通过简单的函数调用，来创建复杂的对象，自动的内存管理．对于资源组（resource groups）采用预先缓存和特殊的清除处理（这是一种通过控制对象分配和释放来提高效率的方法）

<5> GUI：强大而灵活的 GUI 管理，支持动画式的 GUI

<6> 矢量（Vectors），对于 2D 矢量的完全支持

3）Authoring Tools 层

<1> 资源的打包（pack）：HGE 使用 ZIP 格式的资源包，你可以使用任何的打包工具，甚至还可以给资源包加密

<2> 纹理（Texture）工具

<3> 粒子系统编辑器：能够设定粒子的速度，方向，生命周期，轨迹，颜色，透明等

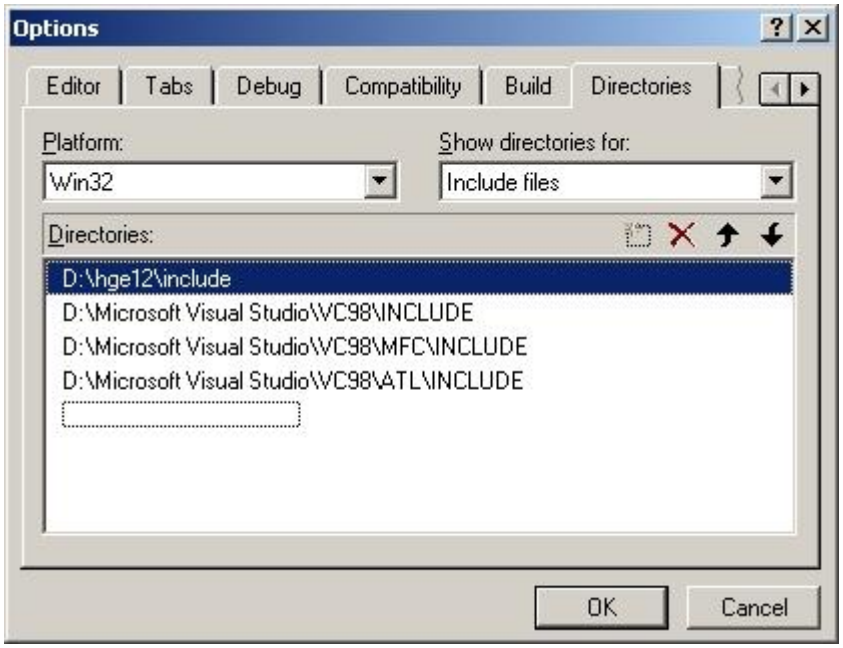
<4> 位图字体编辑器：运行通过系统中已经安装的字体来创建位图字体，你可以使用图形编辑器来为位图字体添加额外的效果

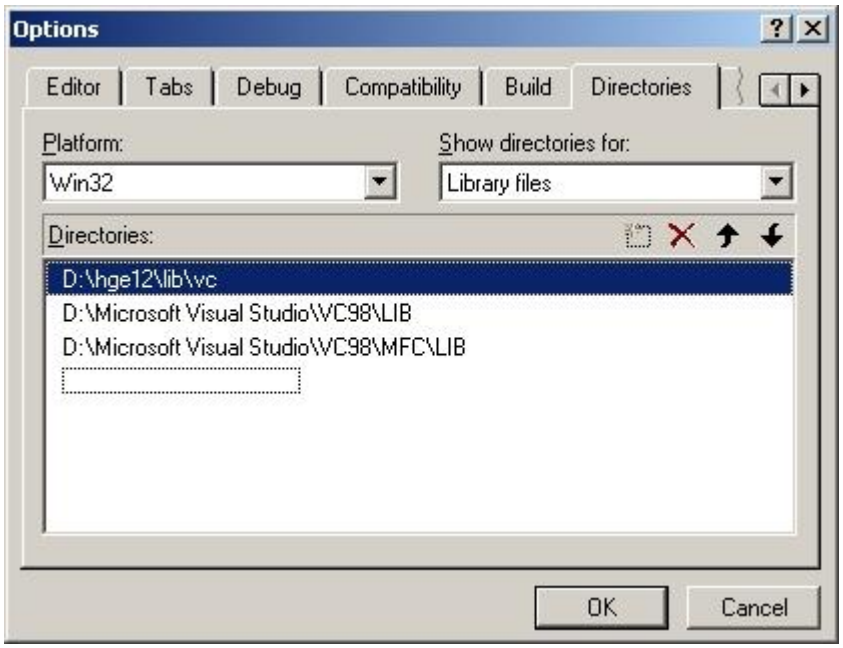
2.安装配置

在 HGE 的文档中有详细谈到如何安装的问题，这里讲一下 VC6 平台的安装问题：

1. 下载完 HGE 之后，需要使用到 lib\vc 文件夹下的库文件以及 include 目录下的头文件

2. 打开 **Tools->Options->Directories**

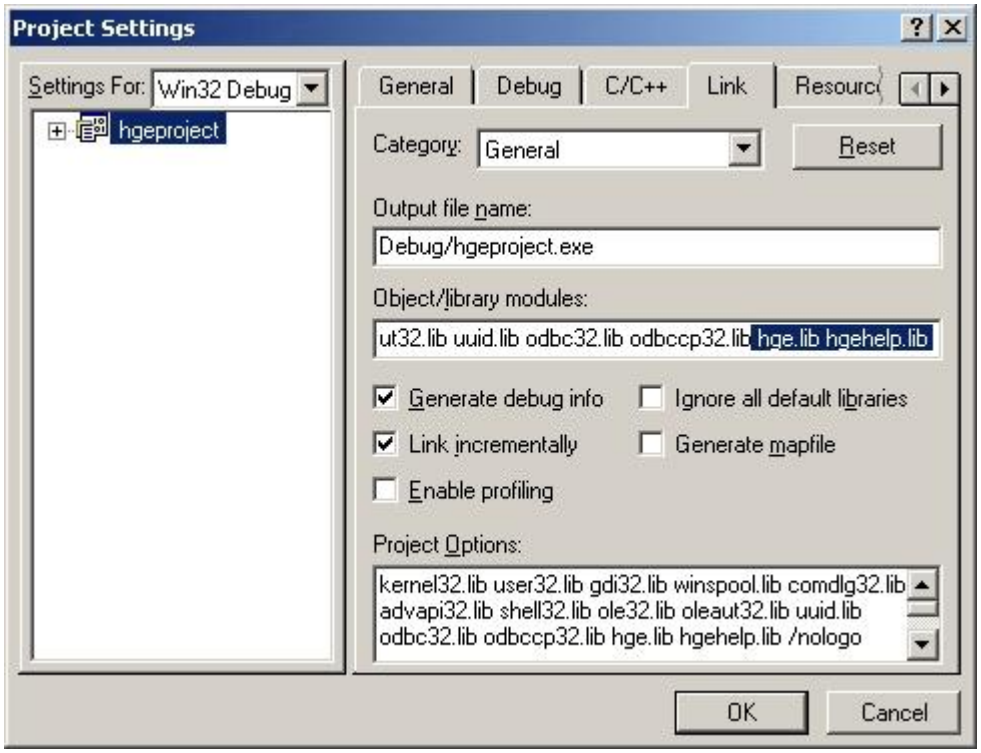




如上两图 ，添加路径

3. 在游戏开发中使用 HGE

首先建立一个空的 Win32 工程 ，然后选择 **Project->Settings...->Link**



按图所示 ，输入 hge.lib 和 hgehelp.lib

当然 ，也可以使用预编译器指令 pragram 来打到同样的目的 .

3. 初试 HGE

当 HGE 安装完成之后 ，就可以使用了 ，关于 HGE 的安装 ，可以参考 << HGE 系列教材 (2) --- 安装 HGE >>

现在使用 HGE 开发一个极小的程序 ：

1. 包含 hge.h 文件 ，并且定义一个 HGE 的指针 ，通过这个指针 ，我们可以访问 HGE Core Functions 层的函数 .

```
#include <hge.h>

HGE *pHge = 0;
```

使用完 HGE 指针之后 ，需要释放这个指针 ，pHGE->Release();

2. 帧函数 (Frame Function) 是一个用户定义的函数 ，每一帧时间 ，它会被 HGE Engine 调用一次 ，函数返回 true ，则调用停止 ：

```
bool FrameFunc()
```

```
    {
      if (hge->Input_GetKeyState(HGEK_ESCAPE))
      {
        return true;
      }
      return false;
    }
```

3. 建立一个 WinMain 函数，WinMain 函数是标准的 Windows 应用程序入口，这里，我们首先初始化 HGE 指针：

```
    int WINAPI WinMain( HINSTANCE hInstance,
              HINSTANCE hPrevInstance,
              LPSTR lpCmdLine,
              int nShowCmd)
{
   pHge = hgeCreate(HGE_VERSION);
   // ...
       pHge->Release();
   return 0;
}
```

通过 HGE 指针，我们才可以访问 HGE Engine 的接口．调用了 hgeCreate 函数之后，不要忘记了使用 Release 函数释放资源．

4. 初始化操作:

有一些初始化操作需要完成，使得程序能够跑起来：

```
    // 设置帧函数
    pHge->System_SetState(HGE_FRAMEFUNC, FrameFunc);
    // 设置窗口模式
    pHge->System_SetState(HGE_WINDOWED, true);
```

```
// 设置不使用声音
    pHge->System_SetState(HGE_USESOUND, false);
// 设置标题为 "Minimal HGE"
    pHge->System_SetState(HGE_TITLE, "Minimal HGE");
```

最后需要调用函数 System_Initiate 来完成初始化操作 ，这个函数返回值是一个 bool 类型的变量 ，如果是 true 那么表示初始化成功 ，如果是 false 表示出错 ，这时候可以通过 System_GetErrorMessage 函数来获取错误消息 ：

```
if (pHge->System_Initiate()){
        pHge->System_Start();
} else {
        MessageBox(NULL,
                pHge->System_GetErrorMessage(),
                "Error",
                MB_OK | MB_ICONERROR | MB_APPLMODAL);
}
```

在程序结束的时候 ，需要释放资源 ：

```
pHge->System_Shutdown();
pHge->Release();
```

5. 整个完整的程序如下 ：

```
#include <hge.h>
HGE* pHge = 0;
bool FrameFunc(){
```

```
    if (pHge->Input_GetKeyState(HGEK_ESCAPE)) return true;
        return false;
    }

int WINAPI WinMain( HINSTANCE hInstance,
            HINSTANCE hPrevInstance,
            LPSTR lpCmdLine,
            int nShowCmd){
  pHge = hgeCreate(HGE_VERSION);
  pHge->System_SetState(HGE_FRAMEFUNC, FrameFunc);
  pHge->System_SetState(HGE_WINDOWED, true);
  pHge->System_SetState(HGE_USESOUND, false);
  pHge->System_SetState(HGE_TITLE, "HGE 小程序");

  if (pHge->System_Initiate())
  {
      pHge->System_Start();
  } else {
      MessageBox(NULL,
          pHge->System_GetErrorMessage(),
          "Error",
          MB_OK | MB_ICONERROR | MB_APPLMODAL);
  }
  pHge->System_Shutdown();
  pHge->Release();
```

```
    return 0;
}
```

注意，程序运行之后，一直调用函数 FrameFunc 直到用户按下 ESC，那么跳到 pHge->System_Shutdown() 处执行 .

4.HGE Core Function

HGE Core Functions 层中的函数需要通过 HGE 指针来访问，就如 << HGE 系列教材（3） --- 初试 HGE >> 所谈到的一样，通过调用 hgeCreate 函数来初始化 HGE 指针，HGE Core Functions 层中的函数，大致分层一下几类：

1. 接口函数（Interface functions）：

hgeCreate --- 初始化 HGE 指针，这是一个全局函数，除了这个函数，HGE Core Funtions 中所有的函数都需要通过 HGE 指针调用 .

Release --- 释放 HGE 接口，调用了 hgeCreate 就应该调用 Release 释放 .

2. 系统函数（System functions）：

这类函数都是以 System_ 开头，后面加上表示函数意义的单词（不出现下划线），之后介绍的函数也将使用这种命名方式，即类型前缀 + 有意义的单词：

System_Initiate	初始化相关软件和硬件
System_Shutdown	恢复声音模式并且释放资源
System_Start	开始运行用户定义的帧函数
System_SetState	设置系统内部状态
System_GetState	返回内部状态的值
System_GetErrorMessage	返回最后出错的 HGE 错误描述符
System_Log	在日志文件中书写格式化消息
System_Launch	运行 URL 或者外部可执行文件或数据文件
System_Snapshot	截屏并保存到一个文件

3. 资源函数 (Resource functions) ：

Resource_Load	从硬盘上读取资源到内存中
Resource_Free	从内存中删除读取的资源
Resource_AttachPack	附加一个资源包
Resource_RemovePack	移除一个资源包
Resource_RemoveAllPacks	移除之前关联的所有资源包
Resource_MakePath	建立一个绝对文件路径
Resource_EnumFiles	通过通配符来枚举文件
Resource_EnumFolders	通过通配符来枚举文件夹

4. 初始化文件函数 (initialization file functions)

Ini_SetInt	在初始化文件中写入一个整数值
Ini_GetInt	从初始化文件中读取一个整数值
Ini_SetFloat	在初始化文件中写入一个浮点值 (float)
Ini_GetFloat	从初始化文件中读取一个浮点值 (float)
Ini_SetString	在初始化文件中写入一个字符串
Ini_GetString	从初始化文件中读取一个字符串

5. 随机数参数函数 (Random number generation functions)

Random_Seed	设置随机数产生器的种子
Random_Int	产生 int 类型的随机数
Random_Float	产生 float 类型的随机数

6. 计时函数 (Timer functions)

Timer_GetTime	从调用 System_Initiate 函数到现在的时间(s)
Timer_GetDelta	返回上一次调用帧函数到现在的时间 (s)
Timer_GetFPS	返回当前 FPS 的值

7. 声效函数 (Sound effect functions)

Effect_Load	载入从硬盘载入声音到内存
Effect_Free	从内存中删除载入的音效和相关的资源
Effect_Play	开始播放音效
Effect_PlayEx	开始播放音效 , 这个函数含有更多的参数

5.输入声音和渲染

1.渲染 :

在 HGE 中 , 四边形是一种图元 , 对应了结构体 hgeQuad , 另外还有三角形图元 , 对应 hgeTriple , 为了渲染 , 我们现在需要使用 hgeQuad 结构体 , 这个结构体如下 :

```
struct hgeQuad {
    hgeVertex       v[4];       // 顶点描述了这个四边形
    HTEXTURE        tex;        // 纹理的句柄或者为 0
    int             blend;      // 混合模式 ( blending mode )
};
```

HGE 中图元对应的结构体总含有这 3 个部分 : 顶点 , 纹理句柄 , 混合模式 ;

```
struct hgeVertex
{
        float x, y;     // 屏幕的 x , y 坐标
        float z;        // Z-order , 范围 [0, 1]
        DWORD col;      // 顶点的颜色
        float tx, ty;   // 纹理的 x , y 坐标 ( 赋值前需要规格化坐标间隔 ,
                        // 使得 tx , ty 取值范围在[0,1] )
};
```

规格化坐标间隔在后面的例子中会谈到 ;

1. 颜色的表示 :

颜色使用 32 位表示 , 从左开始 , 8 位为 Alpha 通道 , 8 位红色 , 8 位绿色 , 8 位蓝色

对于后 24 位 , 如果全部为 0 , 表示黑色 , 如果全部为 1 , 表示白色

2. 定义颜色的运算 :

我们把颜色看成一个四维向量 , 即 alpha 通道 , 红色 , 绿色 , 蓝色这四个分量

<1> 颜色是可以相乘的

颜色的相乘是对应的四个分量分别相乘的结果 , 即 : alpha 通道的值与 alpha 通道的值相乘 , 红色的值与红色的值相乘 , 绿色的值与绿色的值相乘 , 蓝色的值与蓝色的值相乘 .

<2> 颜色是可以相加的

同上 , 对应分量相加 .

颜色的每个分量使用浮点数表示 , 范围是[0-1] , 相加操作可能导致溢出 , 一种处理的方式就是 , 如果溢出 , 则设定值为 1 .

3. 混合模式 ：

1 ） BLEND_COLORADD

表示顶点的颜色与纹理的纹元 （ texel ）颜色相加 ，这使得纹理变亮 ，可见顶点颜色为 0x00000000 将不造成任何影响 .

2 ） BLEND_COLORMUL

表示顶点的颜色与纹理的纹元颜色相乘 ，这使得纹理变暗 ，可见顶点颜色为 0xFFFFFFFF 将不造成任何影响 .

注意 ：必须在1 ） ，2 ）中做一个选择 ，且只能选择1 ） ，2 ）中的一个 . 处理的对象是纹理颜色和顶点颜色 .

这里有一个技巧 ：

如果我们需要在程序中显示一个气球 ，这个气球的颜色不断变化 ，这时候我们并不需要准备多张不同颜色的气球纹理 ，而只需要一张白色的气球纹理 ，设置 blend 为 BLEND_COLORMUL ，白色的 R,G,B 值被表示成 1.0 ，也就是说 ，纹理颜色和顶点颜色相乘的结果是顶点的颜色 ，那么就可以通过修改顶点颜色 ，得到任意颜色的气球了 .

3 ） BLEND_ALPHABLEND

渲染时 ，将对象的像素颜色 （ 而非顶点的颜色 ）与当前屏幕的对应像素颜色进行 alpha 混合 . alpha 混合使用到 alpha 通道 ，对于两个像素颜色进行如下操作 ，得到一个颜色 ：

R(C)=alpha*R(B)+(1-alpha)*R(A)
G(C)=alpha*G(B)+(1-alpha)*G(A)
B(C)=alpha*B(B)+(1-alpha)*B(A)

这里的 BLEND_ALPHABLEND 使用的是对象像素的颜色的 alpha 通道 . 可见如果对象像素颜色 alpha 通道为 0 ，那么结果就是只有当前屏幕的像素颜色 ，也就是常常说的 100% 透明 ，因此 ，我们可以理解 alpha 混合就是一个是图像透明的操作 ，0 表示完全透明 ，255 表示完全不透明 .

4 ） BLEND_ALPHAADD

渲染时 ，将对象的像素颜色与当前屏幕的对应像素颜色相加 ，结果是有了变亮的效果 .

注意 ：这里的 3 ） ，4 ）必选其一 ，且只能选其一 . 处理的对象是对象像素颜色和屏幕像素颜色 .

5 ） BLEND_ZWRITE

渲染时 ，写像素的 Z-order 到 Z-buffer

6) BLEND_NOZWRITE

渲染时 ，不写像素的 Z-order 到 Z-buffer

这里一样是二者选一

设置举例 ：

```
quad.blend=BLEND_ALPHAADD | BLEND_COLORMUL | BLEND_ZWRITE;
        // quad 为 hgeQuad 变量
```

4. HGE 渲染

1) 定义和初始化 hgeQuad 结构体 ：

```
    hgeQuad quad;  // 定义四边形
```

2) 初始化 hgeQuad 变量 ：

```
    // 设置混合模式
quad.blend=BLEND_ALPHAADD | BLEND_COLORMUL | BLEND_ZWRITE;
    // 加载纹理
quad.tex = pHGE->Texture_Load("particles.png");
```

注意 ，读取硬盘上资源的时候 ，可能会失败 ，因此通常都需要检查 ，例如 ：

```
if (!quad.tex)
{
    MessageBox(NULL, "Load particles.png", "Error", 0);
}
// 初始化顶点
for(int i=0;i<4;i++)
{
```

```
    // 设置顶点的 z 坐标

        quad.v[i].z=0.5f;

    // 设置顶点的颜色 , 颜色的格式为 0xAARRGGBB

        quad.v[i].col=0xFFFFA000;

}

    // 这里假定载入的纹理大小为 128*128 ,

    // 现在截取由点 ( 96 , 64 ) , ( 128 , 64 ) , ( 128 , 96 ) , ( 96 ,
    // 96 ) 这四个点围成的图形 .

    quad.v[0].tx=96.0/128.0; quad.v[0].ty=64.0/128.0;  // 规格化坐标间隔
    quad.v[1].tx=128.0/128.0; quad.v[1].ty=64.0/128.0;
    quad.v[2].tx=128.0/128.0; quad.v[2].ty=96.0/128.0;
    quad.v[3].tx=96.0/128.0; quad.v[3].ty=96.0/128.0;
```

注意 , 对于 hgeQuad 结构体 , 顶点 quad.v[0] 表示左上那个点 , quad.v[1] 表示右上的点 , quad.v[2] 表示右下的点 , quad.v[3] 表示左下的点 .

```
// 设置 hgeQuad 在屏幕中的位置

    float x=100.0f, y=100.0f;

    quad.v[0].x=x-16; quad.v[0].y=y-16;
    quad.v[1].x=x+16; quad.v[1].y=y-16;
    quad.v[2].x=x+16; quad.v[2].y=y+16;
    quad.v[3].x=x-16; quad.v[3].y=y+16;
```

3) 设置渲染函数 (render function) :

```
    System_SetState(HGE_RENDERFUNC,RenderFunc);
```

RenderFunc 原型和帧函数一样 :

```
    bool RenderFunc();
```

4) 编写 RenderFunc 函数 :

```
bool RenderFunc() {

    pHGE->Gfx_BeginScene();          // 在如何渲染之前 , 必须调用这个函数
    pHGE->Gfx_Clear(0);              // 清屏 , 使用黑色 , 即颜色为 0
    pHGE->Gfx_RenderQuad(&quad); // 渲染
    pHGE->Gfx_EndScene();            // 结束渲染 , 并且更新窗口
    return false;  // 必须返回 false
}
```

补充 : Load 函数是和 Free 函数成对出现的 , 即在硬盘上加载了资源之后 , 需要 Free 它们 , 例如 :

```
quad.tex = pHGE->Texture_Load("particles");

// ...

pHGE->Texture_Free(quad.tex);
```

2.音效 :

使用音效是很简单的

1. 载入音效 :

```
HEFFECT hEffect = pHGE->Effect_Load("sound.mp3");
```

2. 播放 :

```
pHGE->Effect_PlayEx(hEffect);
```

或者

```
pHGE->Effect_Play(hEffect);
```

1) Effect_Play 函数只接受一个参数就是音效的句柄 HEFFECT xx;

2) Effect_PlayEx 函数较为强大 , 一共有四个参数 :

```
HCHANNEL Effect_PlayEx(

    HEFFECT effect,   // 音效的句柄
    int volume = 100,   // 音量 , 100 为最大 , 范围是[0, 100]
    int pan = 0,        // 范围是[-100, 100] , -100 表示只使用左声道 ,
                        //100 表示只使用右声道
    float pitch = 1.0,    // 播放速度 , 1.0 表示正常速度 , 值越大播放速度越快 ,
                        //必须大于 0( 不等于 0 )
    bool loop = false    // 是否循环播放 , false 表示不循环
);
```

3.输入 :

仅仅需要调用函数 pHGE->Input_GetKeyState(HGEK_xxx); 来判断输入 , 应该在帧函数中调用它 , 例如 :

```
bool FrameFunc()

{

   if (pHGE->Input_GetKeyState(HGEK_LBUTTOM))

     // ...

   if (pHGE->Input_GetKeyState(HGEK_UP))

     // ...

}
```

6.程序流程和细节

HGE 的一些细节，通过源码可以更加清楚的了解，通过读源码，可以更加高效的使用 HGE Engine.

1.必要的第一步：

Help Classes 层建立于 Core Functions 层之上，这并不意味着用户只需要关心 Help Classes 而忽略 Core Functions，因此我们需要获得一个 HGE 指针，来使用 Core Functions 的函数：

<1> 获取 HGE 指针：

```
HGE* pHGE = pgeCreate(HGE_VERSION);
```

<2> 释放 HGE 指针：

使用之后，需要释放 HGE 指针.

```
pHGE->Release();
```

Create 和 Release 过程使用了引用计数，也就是说，一般来看，除了第一次的 Create 调用之外几乎不消耗 CPU 时间和资源，每调用一次 Create 函数，引用计数器就加一，只有在第一次调用的时候才会真正的分配空间，调用 Release 会使得引用计数器减一，当它为 0 的时候，才真正是释放资源.因此以下代码是可用的：

```
while(true)
{
    HGE* pHGE = pgeCreate(HGE_VERSION);
        // 确保不是第一次调用 pgeCreate 函数，因为如果是第一次调用，会分/
        //配内存.
        // ... do something
    pHGE->Release();
}
```

此外，要成对的调用 pgeCreate 和 Release 函数，每次调用 Release 之后，调用它的指针将被赋值为 0，例如：

```
HGE* pHGE = hgeCreate(HGE_VERSION);

pHGE->Release();

pHGE->Release(); // ERROR: pHGE == 0
```

另外 , pHGE->Release 会调用 pHGE->System_Shutdown();

2.必要的第二步 :

初始化 : pHGE->System_Initiate();

初始化语句放在 Windows 入口函数中 , 这个函数将按顺序完成

1) 窗口类的注册

2) 创建窗口

3) 初始化子系统

4) 显示一个 HGE 的 LOGO (这个东西在 HGE 里面被称之为 HGE splash)

一般使用 System_Initiate() 都会是这样的 :

```
 if (pHGE->System_Initiate()) {

    pHGE->System_Start();
 } else {

     MessageBox(NULL,

                pHGE->System_GetErrorMessage(),

                "Error",

                MB_OK | MB_ICONERROR | MB_APPLMODAL);

 }
```

3.必要的第三步 :

调用 : pHGE->System_Start();

调用了 System_Start 的目的是开始消息循环 , 见必要的第二步代码 ;

pHGE->System_Start 和 pHGE->System_Shutdown 是成对出现的，处于某些原因，即使我们知道 pHGE->Release 会调用 System_Shutdown 函数，我们还是应该去显示的调用 System_Shutdown 函数．System_Shutdown 相比 Release 要安全，我们可以这样调用，而不会出错：

```
pHGE->System_Start();

// ... Something

pHGE->System_Shutdown();

pHGE->System_Shutdown(); // OK
```

不论如何，Create 和 Release 成对调用，Start 和 Shutdown 成对调用，那么就不会有问题出现．

4.还有什么是需要的：

System_SetState 函数

常常需要设置窗口大小或者是设置为全屏模式，需要设置是否使用声音等，这一系列操作被称之为设置系统状态，统一通过调用 pHGE->System_SetState 函数来完成，最为关键的是设置帧函数，调用了 pHGE->System_Start 之后，会在绘制每帧图像时调用帧函数．

pHGE->System_SetState(XXX, XXX) 通常可以在如何地方，如何情况下调用，不要认为它们只能在 pHGE->System_Initiate 之前调用

System_SetState 函数的第一个参数表示状态，在内部实现时，它是 FSM 的状态，而第二个参数表示值，通过这个函数，可以绑定状态和相关的值

补充一下，帧函数必须是一个全局函数，而不能是一个类的成员函数，并且帧函数的原型必须是：

```
bool FunName(void);
```

惯用法：

我们通常会在程序初始化之前设置状态，即在 System_Initiate 调用之前，例如：

```
int WINAPI WinMain( HINSTANCE hInstance,
        HINSTANCE hPrevInstance,
        LPSTR lpCmdLine,
```

```
                int nShowCmd)
{

    pHGE->System_SetState(HGE_FRAMEFUNC, FrameFunc);
    pHGE->System_SetState(HGE_WINDOWED, true);
    pHGE->System_SetState(HGE_USESOUND, false);
    pHGE->System_SetState(HGE_TITLE, "HGE");
    pHGE->System_SetState(HGE_SHOWSPLASH, false); // 用于去除 LOGO

    //另外,下面的方法也可以去掉 LOGO ：
    //m_hgeEngine->System_SetState((hgeIntState)14, 0xFACE0FF);


  if (pHGE->System_Initiate())
  {
     pHGE->System_Start();
  }
  else
  {
     MessageBox(NULL,

                pHGE->System_GetErrorMessage(),

                "Error",

                MB_OK | MB_ICONERROR | MB_APPLMODAL);
  }

    pHGE->System_Shutdown();
    pHGE->Release();

    return 0;
}
```

7.Helper classes 类

1) 字体的使用 ：

1. 头文件:

```
#include <hgeFont.h>
```

2. 载入字体:

```
hgeFont* pFont;

pFont = new hgeFont("font1.fnt");  // 不要忘记 delete
```

fnt 文件是一个字体描述文件 (font description file) ，可以通过创作工具产生

3. 打印字体:

```
pFont->printf( 5,  5,
               HGETEXT_LEFT,
               "dt:%.3f\nFPS:%d (constant)",  // 使用中文将出现"??"
               pHGE->Timer_GetDelta(),
               pHGE->Timer_GetFPS()

);
```

在渲染函数中打印文字，HGE 到目前版本 1.81 依然不支持中文，只能使用第三方支持 . 建议使用微妙的平衡(BOGY)提供的解决方案 .

2) 粒子系统的使用 ：

关于 hgeParticleManager ，可以把很多粒子系统加进去让它自动管理 ，使用 SpawnPS 来加入粒子 ，第一个参数可以直接从 hgeParticleSystem->info 得到 . 然后再每一帧里 Update ，然后 Render . hgeParticleManager 会管理这一切 .

1. 建立一个 hgeSprite 对象，hgeSprite 类的构造函数如下 ：

```
hgeSprite(
        HTEXTURE tex,          // 纹理的句柄
        float x,               // sprite 对应的纹理的 x 坐标
        float y,               // sprite 对应的纹理的 y 坐标(区别于顶点
                               // 中的纹理坐标，这里无需规格化坐标间隔 )
        float w,               // sprite 的宽
        float h                // sprite 的高
);
```

注意，sprite 对应的纹理的坐标，是 sprite 的左上的坐标 . 由此可见，一个精灵对应了纹理中的一个四边形区域，实际的源码中，sprite 类含有一个 hgeQuad 成员变量 .

如果 tex 为 0，那么就使用白色作为纹理的数据 (texture data)

```
hgeSprite* pSpt = new hgeSprite(tex, 32, 32, 32, 32);
```

2. 设置混合模式，根据情况设置混合模式，后面详细讨论 :

```
pSpt->SetBlendMode(BLEND_COLORMUL |
                   BLEND_ALPHAADD |
                   BLEND_NOZWRITE);
// 建议使用 BLEND_ALPHAADD，这样看起来效果会好很多 ( 增亮 ) .
```

3. 设置锚点 (似乎和函数名字有点不符)

```
void SetHotSpot(
        float x,     // 锚点的 x 坐标
        float y      // 锚点的 y 坐标
);
```

锚点是这样的一个点 : 进行一些操作的中心点 . 例如进行旋转操作的中心点，即旋转操作依赖于这个点 . 通常设置 sprite 的中心点为锚点 .

4. 关联 hgeParticleSystem

```
pPar = new hgeParticleSystem("trail.psi", m_pSpt);
```

psi 文件被称之为粒子系统描述文件（particle system description file），这个文件是 hgeParticleSystemInfo 结构对象的硬盘镜像，这里不做详细介绍 .

5. 粒子系统中的基本参数介绍：

系统生命周期（System lifetime）：粒子系统的生命周期,在这个周期内会产生新粒子

Emission：每秒产生多少个新的粒子

粒子生命周期（Particle lifetime）：特定的某个粒子的生命周期

6. 设定 Emission：

```
pPar->info.nEmission=10;
```

7. 调用 Fire 函数

pPar->Fire() 函数会重启粒子系统，但它不会影响当前活跃粒子

8. 渲染

```
pPar->Render();
```

9. MoveTo 函数

pPar->MoveTo(x, y); 用于移动粒子系统到（x，y）处

10. Update 函数

```
pPar->Update(m_pHGE->Timer_GetDelta());
```

在帧函数中应该调用 Update 且使用参数为 m_pHGE->Timer_GetDelta()

(3) 使用 hgeSprite 渲染 ：

前面说了为了渲染，使用了 hgeQuad，那样做是复杂的，我们完全可以使用 sprite 来实现，而不需要使用到过多的 Core Functions 层的函数 .

1. 创建 sprite

```
pSpt = new hgeSprite(tex, 96, 64, 32, 32);
```

2. 设置颜色

```
pSpr->SetColor(0xFFFFA000);
```

SetColor 函数将为 sprite 添加颜色，添加的方式由混合模式决定，设置混合模式，通过调用函数 pSpr->SetBlendMode 实现 .

注意，这里设置的颜色是 sprite 中 hgeQuad 对象的顶点的颜色，四个顶点颜色将设为相同，而混合模式设置的是 sprite 中的 hgeQuad 对象的 blend 值 .

```
pSpr->SetBlendMode(BLEND_COLORMUL |
                   BLEND_ALPHAADD |
                   BLEND_NOZWRITE);
```

这里使用的纹理是 alpha 通道渐变，颜色为白色的纹理，因此会使用到 BLEND_COLORMUL，这点在 << HGE 系列教材 (5) --- 输入 . 声音和渲染 >> 做了详细的说明

3. 设置锚点 ：

```
pSpr->SetHotSpot(16, 16);
```

4. 渲染

在渲染函数中，调用 pSpr->Render(x, y); 方可

8 . HGEResourceMessage 类

hgeResourceManager 是一个资源管理类

1. 构造函数

```
hgeResourceManager( const char* scriptname = 0 );
```

scriptname 表示资源脚本文件名（Resource script filename），如果此参数为 0，表示不使用 Resource script file.

现在来介绍一下资源脚本：

资源脚本是一个文本文件，用于定义资源．资源文件由多个（或一个）命令（command）组成，格式如下：

```
Command ResourceName : BaseResourceName
{
    Parameter1=Value1  ; 这里是注释
    Parameter2=Value2
     ...
    ParameterN=ValueN
}
```

我们来看一个例子：

```
Resource level1
{
    filename=levels\level1.dat
    resgroup=1
}
```

这里只有一个命令：Resource，Resource 命令定义了一个原生资源（raw resource），注意，资源文件是大小写敏感的，资源文件中可以有注释，使用";"开头．资源文件的参数（parameter）是没有顺序限制的．

同种类型的资源，不可以使用相同的资源名（Resource Name）．在定义资源名或者文件路径时，出现空格或者特殊字符，需要把整个字符串用双引号引起来．

BaseResourceName 是可选的，如果被指定，那么就表示对 BaseResourceName 对应的参数（Parameters）的拷贝，例如：

```
Sprite wizard
```

```
{
    texture=characters
    rect=0,0,32,32
    hotspot=16,16
    blendmode=COLORMUL,ALPHABLEND,NOZWRITE
    resgroup=1
}

Sprite orc : wizard
{
    rect=0,64,32,32  ; 设定新值
    color=FF808000  ; 设定新值
}
```

这里 orc 除了 rect 和 color 两个参数以外，其他参数值都和 wizard 一样．

注意，hgeResourceManager 是可以容错的，如果脚本出现错误，不会导致程序的终止，错误信息将被写入日志文件．

1）Command（命令）

Command 表明了资源的含义，含有以下几种：

Include，Resource，Texture，Sound，Music，

Stream，Target，Sprite，Animation，Font，

Particle，Distortion，StringTable

<1> Include 命令：Include 命令用于导入其他的资源脚本文件，例如：

```
Include level2.res  ; level2.res 是一个资源脚本文件
```

注意，自引用和循环引用是可行的，它们会被检查出来，并报告在日志文件中．

<2> Resource 命令：定义原生资源（raw resource）

参数： filename，resgroup．例如：

```
Resource level1
{
    filename=levels\level1.dat
    resgroup=1  ；资源组（resource group）标识符，0 表示没有特定的组
}
```

在 filename 中可以使用绝对或者相对路径，相对路径是相对于应用程序所在的文件夹或者是相对于资源包的根目录，特别应该注意的是，如果它是一个相对路径，相对的是应用程序或者资源包的根目录而不是脚本文件

<3> Texture 命令：定义一个纹理

参数：filename，mipmap，resgroup．例如：

```
Texture background
{
    filename=images\bg.jpg
    resgroup=1
}
```

由于没有设定 mipmap 参数的值，因此它取默认值．

<4> Sound 命令：定义一个音效

参数：filename，resgroup．例如：

```
Sound explosion1
{
```

```
filename=sounds\expl1.ogg
resgroup=1
```

}

<5> Music 命令

9.GUI 对象和控件

1. hge 中 GUI 对象和控件

hge 中 GUI 对象被看作是一个控件的容器，hge 提供了创建 GUI 对象的类 hgeGUI 类

2. hgeGUI 类

1）AddCtrl 函数

```
void AddCtrl( hgeGUIObject  *ctrl ); // hgeGUIObject 对象的指针
```

我们通常可以有这样的写法：

```
gui->AddCtrl(new hgeGUIMenuItem(1,fnt,snd,400,200,0.0f,"Play"));
```

这里 hgeGUIMenuItem 是一个控件，继承于 hgeGUIObject 类，注意，我们创建了 hgeGUIObject 对象，但是却没有去销毁它，因为 hgeGUI 类的析构函数会去处理这些问题．

2）SetNavMode

设置 GUI 导航模式（Navigate mode）：

```
void SetNavMode( int navmode );
```

HGEGUI_NONAVKEYS - 无键盘导航
HGEGUI_LEFTRIGHT - 左右按键导航
HGEGUI_UPDOWN - 上下按键导航
HGEGUI_CYCLED - 循环

默认情况下，navmode 被设置为 HGEGUI_NONAVKEYS，对于一个菜单，我们可以这样设置：

```
SetNavMode(HGEGUI_UPDOWN | HGEGUI_CYCLED);
```

这样,使用上下键导航，并且循环．

3）SetCursor

设置光标 sprite：

```
void SetCursor( hgeSprite *sprite );
```

设置光标对应的 sprite，如果为 0，表示不显示光标，默认情况为 0．注意，光标不受 GUI 对象的管理，也就是用户必须自己释放光标资源．

4）SetFocus

```
void SetFocus( int id );
```

每个控件都有一个对应的 ID 号，这个 ID 号被称之为控件的标识符，这里通过控件标识符来设置焦点．键盘事件只会被分发到成为焦点的控件上．

5）Enter

开始 GUI Enter 动画

3. hgeGUIObject

hgeGUIObject 是一个抽象类，它有一个纯虚函数 Render，hgeGUIObject 类的子类的对象并不是 GUI 对象，而是 GUI 控件，这一点应该清楚．

1）hgeGUIObject 的成员变量

hgeGUIObject 的成员变量都为 public：

```
// 以下为必须在构造函数中初始化的变量

int     id;           // 控件标识符
bool    bStatic;      // 如果为 true 控件无法成为焦点也不会接受键盘事件，
                      // 同时它将被 navigate 例程忽略（前面已谈到设置 na
                      // ivgate）
bool    bVisible;     // 控件是否可见，如果为 false，控件将不被渲染
bool    bEnabled;     // false 时，控件对用户的输入不作出任何回应，但是
                      // 控件是可以接受到用户的通知（区别于 bStatic）
hgeRect rect;         // 控件有界框（bounding box）在屏幕上的区域
DWORD   color;        // 控件颜色


// 以下为不需要在构造函数中初始化的变量
hgeGUI       *gui;    // GUI 对象指针
hgeGUIObject *next;   // 连接 GUI 对象中所有控件，子类不需要改变它
hgeGUIObject *prev;   // 连接 GUI 对象中所有控件，子类不需要改变它
static HGE   *hge;    // hge 指针
```

2）void Render(void)

渲染控件到屏幕

3）void Update(float fDt)

fDt 上次调用 Update 函数到现在所用的时间（单位是秒）

4）void Enter(void)

控件出现在屏幕上的时候被调用，用于播放控件出现时的动画

5）void Leave(void)

控件离开屏幕的时候被调用，用于播放控件离开屏幕的动画

6）bool IsDone(void)

判断控件出现动画和控件离开动画是否播放完毕

7) void Focus(bool bFocused)

控件获得焦点 , bFocused 为 true , 反之为 false

8) bool MouseMove(float x, float y)

以控件左上为原点 , 鼠标指针的坐标 . 如果控件状态改变 , 需要通知调用者 , 那么返回 true , 否则返回 false

9) bool MouseLButton(bool bDown)

bDown 如果为 true , 表示按下鼠标左键 , 如果 bDown 为 false , 表示松开鼠标左键

10) bool KeyClick(int key, int chr)

key 表示按键的虚拟代码 (Virtual code of the pressed key) , 见下表 :

HGEK_LBUTTON	Left mouse button
HGEK_RBUTTON	Right mouse button
HGEK_MBUTTON	Middle mouse button (wheel button)
HGEK_ESCAPE	ESCAPE key
HGEK_BACKSPACE	BACKSPACE key
HGEK_TAB	TAB key
HGEK_ENTER	Any of the two ENTER keys
HGEK_SPACE	SPACE key
HGEK_SHIFT	Any of the two SHIFT keys
HGEK_CTRL	Any of the two CTRL keys
HGEK_ALT	Any of the two ALT keys
HGEK_LWIN	Left WINDOWS key
HGEK_RWIN	Right WINDOWS key

HGEK_APPS	APPLICATIONS key
HGEK_PAUSE	PAUSE key
HGEK_CAPSLOCK	CAPS LOCK key
HGEK_NUMLOCK	NUM LOCK key
HGEK_SCROLLLOCK	SCROLL LOCK key
HGEK_PGUP	PAGE UP key
HGEK_PGDN	PAGE DOWN key
HGEK_HOME	HOME key
HGEK_END	END key
HGEK_INSERT	INSERT key
HGEK_DELETE	DELETE key
HGEK_LEFT	LEFT ARROW key
HGEK_UP	UP ARROW key
HGEK_RIGHT	RIGHT ARROW key
HGEK_DOWN	DOWN ARROW key
HGEK_0	Main keyboard '0' key
HGEK_1	Main keyboard '1' key
HGEK_2	Main keyboard '2' key
HGEK_3	Main keyboard '3' key
HGEK_4	Main keyboard '4' key
HGEK_5	Main keyboard '5' key
HGEK_6	Main keyboard '6' key
HGEK_7	Main keyboard '7' key
HGEK_8	Main keyboard '8' key
HGEK_9	Main keyboard '9' key

HGEK_A	'A' key
HGEK_B	'B' key
HGEK_C	'C' key
HGEK_D	'D' key
HGEK_E	'E' key
HGEK_F	'F' key
HGEK_G	'G' key
HGEK_H	'H' key
HGEK_I	'I' key
HGEK_J	'J' key
HGEK_K	'K' key
HGEK_L	'L' key
HGEK_M	'M' key
HGEK_N	'N' key
HGEK_O	'O' key
HGEK_P	'P' key
HGEK_Q	'Q' key
HGEK_R	'R' key
HGEK_S	'S' key
HGEK_T	'T' key
HGEK_U	'U' key
HGEK_V	'V' key
HGEK_W	'W' key
HGEK_X	'X' key
HGEK_Y	'Y' key
HGEK_Z	'Z' key
HGEK_GRAVE	Grave accent (`)

HGEK_MINUS	Main keyboard MINUS key (-)
HGEK_EQUALS	Main keyboard EQUALS key (=)
HGEK_BACKSLASH	BACK SLASH key (\)
HGEK_LBRACKET	Left square bracket ([)
HGEK_RBRACKET	Right square bracket (])
HGEK_SEMICOLON	Semicolon (;)
HGEK_APOSTROPHE	Apostrophe (')
HGEK_COMMA	Comma (,)
HGEK_PERIOD	Main keyboard PERIOD key (.)
HGEK_SLASH	Main keyboard SLASH key (/)
HGEK_NUMPAD0	Numeric keyboard '0' key
HGEK_NUMPAD1	Numeric keyboard '1' key
HGEK_NUMPAD2	Numeric keyboard '2' key
HGEK_NUMPAD3	Numeric keyboard '3' key
HGEK_NUMPAD4	Numeric keyboard '4' key
HGEK_NUMPAD5	Numeric keyboard '5' key
HGEK_NUMPAD6	Numeric keyboard '6' key
HGEK_NUMPAD7	Numeric keyboard '7' key
HGEK_NUMPAD8	Numeric keyboard '8' key
HGEK_NUMPAD9	Numeric keyboard '9' key
HGEK_MULTIPLY	Numeric keyboard MULTIPLY key (*)
HGEK_DIVIDE	Numeric keyboard DIVIDE key (/)
HGEK_ADD	Numeric keyboard ADD key (+)
HGEK_SUBTRACT	Numeric keyboard SUBTRACT key (-)
HGEK_DECIMAL	Numeric keyboard DECIMAL key (.)
HGEK_F1	F1 key

HGEK_F2	F2 key
HGEK_F3	F3 key
HGEK_F4	F4 key
HGEK_F5	F5 key
HGEK_F6	F6 key
HGEK_F7	F7 key
HGEK_F8	F8 key
HGEK_F9	F9 key
HGEK_F10	F10 key
HGEK_F11	F11 key
HGEK_F12	F12 key

如果控件状态修改了，希望通知调用者，那么返回 true，否则为 false

10．其他

（1）创建 HGE 最简单程序的步骤：

1． 定义 HGE 类型的指针，用来保存引擎指针．

2． 调用 hgeCreate 函数得到 HGE 引擎指针----这是个标准的 C 语言函数，从以前阅读这个引擎的部分代码得知，引擎内部模仿了 COM，采用引用计数的方式创建引擎对象．

3． 设置引擎状态值： System_SetState．在这里必须设置帧更新函数，这是必须的，这个引擎把消息循环隐藏了，程序的表面流程变为：main→renderFunc→end．当然程序底层还是一般的消息循环．设置了帧渲染函数后，引擎会不断地调用这个函数．在这里还可以设置其他状态信息，如渲染状态，窗口尺寸，还可以设置是否需要 LOG 文件．

4． 调用 System_Initiate 函数初始化引擎．

5． 开始，System_Start．这个时候底层基本上就进入了消息主循环了

6． 在渲染函数里（渲染函数没有参数，返回值为布尔类型）返回 TRUE 时，底层消息循环就跳出．System_Start 函数也返回了．

7． 调用 System_Shutdown 做一些恢复工作

8． 调用 Release 彻底销毁引擎．

9． 程度退出．

（2）关于基本渲染图元

文档里描述 Quad is the basic **HGE** graphic primitive . HgeQuad 是个结构体,里面保存着一个纹理对象的 ID 值 , 一个渲染模式值 , 和一个 hgeVertex 结构体 , 这个结构体里又包含了四个 float 和一个 DWORD 值 . 如下 :

```
struct hgeVertex
{
        float       x, y;
        float       z;
        DWORD       col;
        float       tx, ty;
};

struct hgeQuad
{
        hgeVertex   v[4];
        HTEXTURE    tex;
        int         blend;
};
```

其中 , x,y 被描述为屏幕坐标 , tx,ty 被描述为纹理坐标 , col 被描述为颜色 . 回忆下 DX8 中的做法 : 创建一组顶点 , 每个顶点包含了位置坐标 , 和纹理坐标 (纹理坐标一般为 0--1) , 还有颜色等信息 , 于是这里的情况也就很好理解了 .

一个点在屏幕上有坐标 , 一个矩形区域需要把一张图片映射进来 , 如果采用纹理方式 , 就需要为每一个点指定一个二维的坐标 .

如果我没推算错 , 那么在 tutorial2 中显示图片的原理 , 就是利用了渲染顶点的方式, 而不是用 D3Dsprite 去渲染 Texture 的 , 同样的 , texture 还是不能自己渲染自己 .

(3) 利用 hgeQuad 显示图片的过程 :

1 . 用 Texture_Load 载入外部文件作为纹理 .

2 . 设置 hgeQuad 的纹理坐标 , 窗口坐标 , 以及渲染模式 .

3 . 每一帧都调用 Gfx_RenderQuad 函数

注意：设置纹理坐标以及窗口坐标时，v[0]表示左上角坐标，v[1]表示右上角坐标，v[2]表示右下角坐标，v[3]表示左下角坐标．我用了 2 个小时才发现这个．--当然这还是我的猜测，但是这样认为可以正确地贴图．

（4）利用 hgeSprite 和 HTEXTURE 实现贴图．

```
1．HTEXTURE        tex = hge->Texute_Load( filename );
2. hgeSprite       *sprite = new hgeSprite( tex, 0, 0, 64, 64 );
3.sprite->Render( x, y )
```

载入一幅背景透明的 PNG 图片，用以上代码就可以实现透明传送．

3D 里面没有颜色键，只有 alpha-channel．通过对 sprite 的 color 的 alpha 值进行设置，可以实现淡入淡出效果，quad 的四个顶点的颜色进行设置也可以---两者底层其实都一样，都是设置顶点颜色．

HGE 用精灵贴图，即使给的坐标为负数，也能被贴出来----支持裁减．（偶尔会出现闪烁的黑线）

（5）显示汉字

HGE 要显示文字，似乎只能载入 fnt 文件---fnt 文件里又指定了 PNG 图片,PNG 图片里指定了各种字体，例如里面没有汉字，程序里就显示不出汉字．有点麻烦．

显示文字步骤：

1． New 一个 hgeFont 对象，构造函数传其 fnt 文件．
2． 设置文字颜色，以及缩放程度（SetScale）
3． 调用 printf 显示，或 Render．

（6）关于 sprite：

通过使用 hgeSprite::SetFlip 可以在 Render 的时候是水平翻转图象还是垂直翻转图象，还是两者都翻转．

可以动态设置 sprite 的尺寸，以及 texture ，从而达到用一个 sprite 去显示多个 texture 的目的，但是一定要注意在显示一幅 texture 时，必须重新设置 textureRect，如下代码所示：

```
g_sprite->SetTexture( g_texture );
g_sprite->SetColor( 0x55ffffff );
g_sprite->SetTextureRect( 0, 0, 64, 64 );
```

```
g_sprite->Render( 0, 0 );
g_font->printf( 0, 64, "alpha-blend" );

g_fishSpr->SetFlip( false, false );
g_fishSpr->SetColor( ARGB( 255, 255, 255, 255 ) );
g_fishSpr->Render( 100, 0 );
g_font->printf( 100, 100, "normal transparent" );

g_fishSpr->SetFlip( true, false );
g_fishSpr->Render( 0, 200 );
g_font->printf( 0, 300, "horizontally blit" );

g_sprite->SetTexture( g_fish );
g_sprite->SetColor( 0xffffffff );
g_sprite->SetTextureRect( 0, 0, 200, 180 );
g_sprite->Render( 200, 200 );
```

可以直接调用 hgeSprite::RenderEx 进行缩放和旋转 ，如下代码为缩放一半 ：

```
g_sprite->SetTexture( g_fish );
g_sprite->SetColor( 0xffffffff );
g_sprite->SetTextureRect( 0, 0, 200, 180 );
g_sprite->RenderEx( 200, 200, 0, 0.5f );
```

所有 sprite 的默认 blend mode 都是全局的 BLEND_DEFAUT ，即 ： BLEND_COLORMUL | BLEND_ALPHABLEND | BLEND_NOZWRITE

(7) 关于帧函数 FrameFunc

FrameFunc 如果放在类里 ，则必须是静态成员函数 ，否则编译器会报"编译器错误"之类的错误 . 但是如果它作为静态函数了 ，则很多类里的成员无法访问 ，好的方法是 ：创建一个桥梁全局函数 ，然后在 Cgame::Init 时传该函数指针过去 ，然后在 Init 里就可以通过这个指针设置 FrameFunc ，这个桥梁函数 ，会显示地调用 Cgame::FrameFunc ，后者才是真正有意义的帧处理函数 . 游戏运行时 ，引擎先调用那个桥梁函数 ，然后桥梁函数再调用那个类成员函数 . (参看 hge3 代码)

(8) 如何使用粒子系统

使用 HGE 引擎的 Particle Editor 可以很轻易地在程序里实现粒子系统 .

大体步骤 ：

1 . 先用 Particle Editor 编辑好粒子系统

2．把对应的 psi 文件以及 particle.png 图片复制到自己的程序目录下.

4． 程序里 new hgeParticleSystem 对象，以及对应的 sprite，和 texture ，然后就可以 fire 启动粒子系统

5．在 FrameFunc 里 hgeParticleSystem::Update，然后在渲染代码块里 Render 就可以了 .

大致原理，我猜测是：psi 文件保存了粒子系统信息，其对应着要使用 particle.png 这个图片 . 创建 sprite 时具体指定使用哪部分，然后 Update 会更新粒子系统状态,Render 时就用对应的 PNG 图片渲染出来 .

当 hgeParticleSystem::GetAge == -2 且 hgeParticleSystem::GetParticlesAlive == 0 时，表示这个粒子系统所有粒子都消失了，可以被删除了 .

（9）两条注意的:

1) Set hotspot of particle sprite in center (if width and height = 64, then hotspot of x and y must be 32).

2) Be sure that when you create hgeParticleManager, fps in parameter same as in particle editor.

（10）使用 z-buffer 绘制 sprite:

使用 z-buffer 可以方便地绘制各种精灵，z-buffer 越大，就越后面，反之，就被绘制得越前面 . 要使用 z-buffer，大致步骤为：

1．System_SetState 时要让 HGE_ZBUFFER 为 TRUE . 否则所有的 z 值都会被忽略 .

2．渲染时，因为纹理是靠精灵绘制出来的，所以这个时候只需要设置精灵的 Z 值: hgeSprite::SetZ 即可 .

3．渲染 .(particle 虽然也可以设置其内部的精灵 Z 值，但是显示出来就有问题)

（11）HE 中时间的问题：

实验发现：HGE 引擎里的 HGE::Timer_GetDelta()并不是我的 FPS 系统里的 SpeedFactor. 引擎里的 Delta 通常在 0.006 左右，而我的 SpeedFactor 一般为 0.200 . --数量级的差别 . 但是二者的作用似乎都是一样的 . ---去平衡速度与动画快慢 .

Timer_GetTime()返回的是秒，小数点后精确到毫秒级---还要高些 .

（12）关于引擎中的鼠标输入：

鼠标坐标，只需要调用 Input_GetMousePos 函数即可，所得到的坐标是相对于窗口左上角的局部坐标．

而按键状态可以直接通过 Input_GetKeyState(HGEK_LBUTTON); 之类的代码得到．在每一帧里，如果调用 Input_GetKeyState(HGEK_LBUTTON)函数来检测按键状态，实验表明，当我只按一下鼠标左键时，记录的数据也表明这个状态被多次检测到．在使用时，最好设置个时间标志变量----在一段时间内，无论按键多少次，也只算做一.

Input_GetMouseWheel()是用来得到鼠标滚轮状态的．

窗口程序下，当鼠标指针移动到窗口外时，GetMousePos 函数得到的坐标始终是移开前那个坐标．

(13) 关于音乐播放：

播放音乐，可以直接使用 Effect_Load 函数调用，这种方法适用 ogg, mp3, wav 等．该引擎不能播放 MID．但是可以把 MID 转换为 XM 文件播放．

Effect_PlayEx 返回的是 HCHANNEL 类型的变量，根据这个变量可以继续控制其播放属性，而控制播放中的声音属性的一组函数是以 Channel 开头的．

Effect_Play 每次返回的 Channel 值是不一样的，即使是播放同一个 HEFFECT．

(14) 关于数据打包

关于数据打包并读取，最简单的方法是：用 winrar 压缩数据为 zip 格式，可以加密码，然后在程序里 m_hge->Resource_AttachPack("data", "kevinlynx");关联数据文件，然后以后加载图片，声音等数据文件时，给的路径为文件在 zip 里的路径即可自动载入.

引擎里的很多加载资源函数都有这么个描述：

If **filename** specifies relative path, texture file is first searched within all resource packs attached with Resource_AttachPack, then in the application disk folder. If **filename** specifies absolute disk path, texture file is loaded directly from disk.

hgeAnimation 也可以指定在资源脚本文件里，不过使用这个类要注意的是：需要调用 Play 来开启动画．

更多关于 Resource Script File 可以具体参看 Resource Manager.doc 文件．

(15) GUI 系统

关于 HGE 里的 GUI 系统 ，如果要使用按钮控件 ，做以下几步 ：

1 ． 创建 hgeGUI 对象 ，该对象会管理所有的控件
2 ． 创建一个按钮纹理 ，其格式包含两幅横向放着的图象 ，一幅用于按钮未按下时的外观 ，一幅用于按下时的外观 ： The texture must hold two images, one for the button in the unpressed state, and directly to the right from it, the image for the button in pressed state.
3 ． 把创建的按钮加入 hgeGUI 对象 ．
4 ． 在每一帧里调用 hgeGUI::Update,在渲染里 hgeGUI::Render ．
5 ． 可以通过 hgeGUIButton::GetState 来得到按钮是否被按下 ．

注意 ： HGE 里的按钮 ，只有当鼠标按下时才会改变外观显示 ．当鼠标指针放在其上时 ，它不改变状态 ．对于按钮如果 SetMode(true)了 ，按钮将变为一个类似 RadioButton 的控件 ．当 hgeGUI 对象被删除时 ，其 AddCtrl 加进来的对象会被自动删除 ．

要使用文本控件 ，同使用按钮控件一样, 不同的是 ，文本控件需要个 hgeFont 对象 ，文本控件被初始化后 ，改变 hgeFont 对象 ，也会对文本控件产生作用 ．

虽然 HGE 引擎的 GUI 很简单 ，但是其扩展性很好 ．因为 hgeGUI::Update 基本上派发了所有控件需要的消息---键盘操作 ，以及鼠标操作；而 hgeGUIObject 基类的很多成员函数都会处理这些消息 ，我们只需要派生 hgeGUIObject ，然后重载我们需要的消息处理即可 ．这样我们很容易地就可以设计出各种 GUI 控件 ．

(16) 关于 ini 文件操作

所有操作 ini 文件的函数都有这三个参数 ： section ，name ，value ．查看一个 DEMO ini 文件便可知道它们三个的含义 ：

```
[snd_shoot]                    //section
volume=30                      //name = value
```

要注意的 ，要使用 ini 文件 ，需要设置一个引擎状态 ： HGE_INIFILE

(17) Gfx 函数族问题 ：

Gfx_SetTransform 函数是相对于整个屏幕而言的 ，它可以让整个屏幕的内容旋转与缩放 ，但是不能操作局部范围！

注意 ： Gfx_BeginScene 不能嵌套使用 .

关于 Gfx_SetClipping 函数 , 当设置裁减矩形为 (100, 100, 300, 200)时 , 绘制图形在 (100, 100)处绘制不出 , 而在(100+300, 100+200)处则能绘制出 . 也就是说 , 可以被显示的区域为 ：(101, 101, 100 + 300, 100 + 200) (包括这里列出的坐标本身) . 当其宽度或高度为 0 时 , 该函数可能会设置失败 , 从而导致重新设置为整个渲染对象 .

关于 Gfx_RenderQuad 函数 , 如果设置的矩形区域为 (100, 200, 100, 200) , 绘制时 , 不会绘制横坐标为 100 这条线 . 也不会绘制纵坐标为 200 这条线 , 但是可以横坐标方向可以绘制到 100+99 处 , 纵坐标也是 .

关于 Gfx_RenderLine 函数 , 该函数忽略最后一个点 .

(18) hgeSprite 问题

在创建 hgeSprite 对象时 , 给其指定纹理坐标时 , 例如 ：

```
Sp = new hgeSprite( tex, 0, 0, 32, 32 );
```

在绘制 sp 时 , 不会绘制 (0, 0, 32, 32) 0, 32, 32, 32 以及 32, 0, 32, 32 这两条线 .

hgeSprite::GetBoundingBox 返回的是 hgeSprite 拥有的纹理的矩形范围 . 该矩形永远都是正立的 , 它不会因为精灵旋转而旋转 .

(19) Texture 问题 ：

关于 Texture_Lock 函数 , 即使 lock 的是纹理上的一部分 , 在获取纹理上的一点颜色信息时 ：

```
lock_ptr[ y * width + x ]
```

其中 width 是该纹理整个的宽度 , 而不是 lock 的宽度 !

使用 target ： 可以在同一帧里先把需要绘制的纹理绘制到 target 上 , 然后用函数 Target_GetTexture 得到 target 的纹理 , 再把该纹理配合 sprite 即可绘制出来---当 offscreen 使用 !