一些必备的知识

TV3DEngine（TV3D 引擎）

TVEngine：主引擎类。所有用 TV3D 创建的应用程序（无论是图形应用程序还是多媒体演示）都必须包含这个类作为开头。

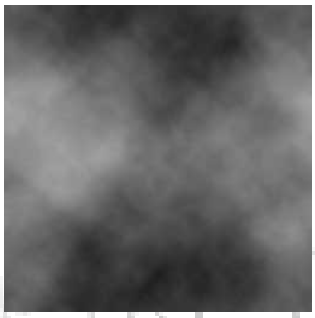
TVScene：所有用 TV3D 创建的应用程序（无论是图形应用程序还是多媒体演示）都必须包含这个类，相当于一场表演的舞台，“演员们”要通过在“舞台”上表演，才能上演一场“好戏”。

TVInputEngine：可以获取键盘、鼠标游戏杆等的输入。主要是面向 Player（玩儿家）的。

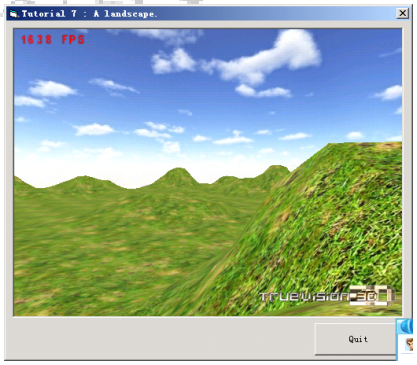
以上各类是一个图形应用程序或是演示程序必须包含的！

其他的东西，它们可以使你的程序变得更酷！

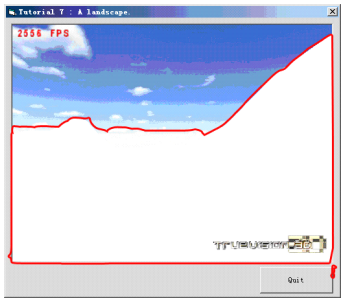
TVLandscape：利用它可以用来生成地形，如山峰，需要加载被称为高程图的图片，就能生成地形。还有像水面，也可以用 Landscape 来做。一个典型的高程图就像这样



黑色部分海拔低，越到白的部分，海拔就越高，就像下面的。



TVTextureFactory：纹理。刚才说到高程图了，能够生成地形，但是如果直接用，就会出现这样的情况。



这是因为没有纹理只有地形，才这样。要想象上面的有绿草，就需要 TVTexture 了。比如我们在荣誉勋章、使命召唤等游戏中看到的美国大兵、大草原、坦克等，就都是用了纹理的，不用纹理，你就根本看不出那是什么东西。

TVMaterialFactory：材质。可以表现物体的材质，就是说用这个你可以看出这玩意是用什么材料做的。如图：

仔细看这个球的表面，是不是显得坑坑洼洼的？这就是 TVMaterialFactory 的功劳。

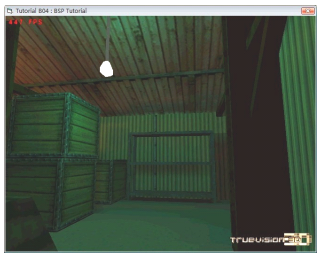
TVAtmosphere：天空盒子。主要是用来表现天空中大气中的一些物质，比如天空、太阳、雾、雨、雪等。还是用图片来说明吧。

天空、雲彩、太陽、光暈

雪片、雲霧

TVLightEngine：光照引擎。顾名思义，它就是用来处理光。有光亮照射的物体就看得见，没有的就呈现暗色。

TVBSPTree：说的专业点，就是二叉分支树，具体就是……，算了算了，我也不知道怎么回事，那时候没认真学，汗……反正就是 CS、Quake 系列用的地图档格式。



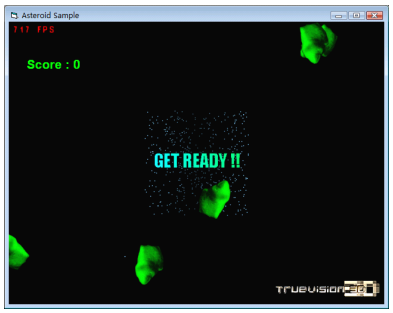
这些类是做一个游戏最基本的，其他的还有 TVAI、TVPath、TVScreen2DImmediate、TVScreenText、TVParticleSystem、TVCollisionResult、TVKeyFrameAnim...

TVAI：AI（Artificial Intelligence）人工智能，不用多说了吧？使命召唤、荣誉勋章的战友、敌人就是人工智能的一个实例。

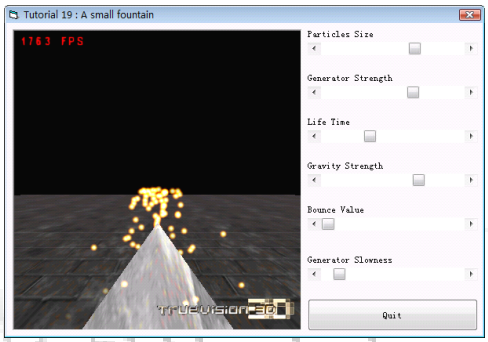
TVPath：这个我不是太清楚，翻译过来是路径处理。不明白。

TVScreen2DImmediate：2D 荧幕场景。用它加上上面提出的那些组件可以做出很酷的 2D 游戏。

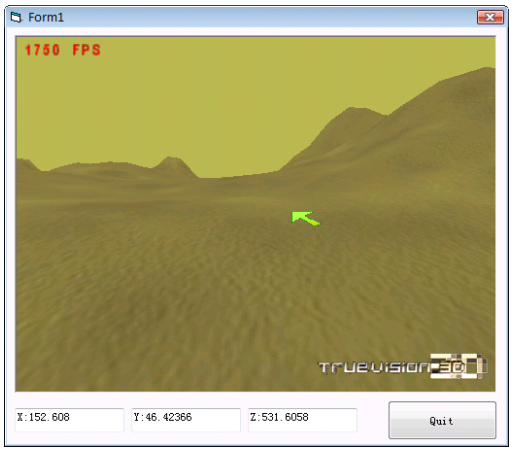
TVScreen2DText：2D 文字，用它可以写一些文字。



TVParticleSystem：particle 的中文意思是粒子的意思。这个类就是可以生成一些小颗粒，粒子。用它可以做出喷泉、火炬或者类似的东西。



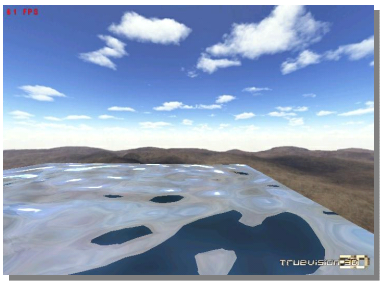
TVCollisionResult：碰撞检测的。用它来检测诸如是否碰到墙壁、射击是否射中等和碰撞有关的是再简单不过了。通过一个函数就行了。



檢測滑鼠當前的 X、Y、Z 位置

TVKeyFrameAnim：用来处理模型动画。它的 PlayAnimation 能够决定模型的动画速度，播放快慢。

TVRenderSurce：渲染表面。像一些特效就可以出来。如动态模糊、环境映射等效果。



TVCamera：游戏中的画面都不是我们直接用眼睛来看见，都是要通过“摄影机”来展现。根据它的不同的角度、位置，看到物体的角度、位置也不一样。

TVGraphicEffect：用于一些特效，比如在游戏开始时的淡入淡出效果。

下面是一些模型对象。

TVMesh、TVActor、TVActor2：3D 模型类，可以用它们建立游戏中人物、车辆、物件等模型。每一个类可以加载的模型档类型是，见下表：

类	TVMesh	TVActor	TVActor2
可加载的档类型	.3DS、.X（微软定义的 3D 模型档）、.TVM（TV3D 定义的 3D 模型档）	.MDL（Half Life 档、.MD2（QUAKE2 的档）、.MD3（QUAKE3 的档）	.MDL、.MD2、.X

这些类函数的具体解释

这些函数在前面的而且用粗体的，都是最常用的，不常用的先不写。

TV3DEngine

Init3DWindowedMode(窗口句柄、是否启用 T&L)

用窗口方式初始化和渲染窗口。

参数

窗口句柄	要渲染的窗口句柄
是否启用T&L（可选的，默认是选定的。）	如果显卡支持T&L，则全速运行T&L（转换和光照）。

返回值为布尔型，如果引擎被正常地初始化了，则返回 True，如果有问题则返回 False。

说明：

- 如果需要从一个对话框中得到结果，你就必须使用 TV.Initialize。
- 在绘制窗口之前，你必须调用 Init3DWindowedMode 函数。
- 桌面显示模式是很重要的，如果它在 24bits 色深下，或者用了一个显卡不支持的模式，引擎将不会启动。所以一定要在 16bits 或 32bits 色深下。

Init3DFullScreen(荧幕宽度，荧幕高度，色深，是否启用 T&L，是否垂直同步，深度缓冲，伽马，窗口句柄)

用指定的参数全屏初始化引擎。

参数

荧幕宽度	要初始化的荧幕宽度(640, 800, 1024,……)
荧幕高度	要初始化的荧幕高度(480, 600, 768,……)
色深	每个像素点的颜色深度
是否启用T&L（可选的，默认是选定的。）	如果显卡支持T&L，则全速运行T&L（转换和光照）。
是否垂直同步（可选的，默认是不）	这个不用说了吧？菜鸟都知道不能选择True
深度缓冲（可选的，默认是TV_DEPTHBUFFER_BESTBUFFER)	在16bits、24bits模板、24bits和32bits中，选择深度缓冲格式，使用得到的最好的性能。
伽马（可选的，默认是0）	伽马因子（默认是1，越大越亮）范围从0到10
窗口句柄（可选的）	

说明：

- 如果需要从一个对话框中得到结果，你就必须使用 TV.Initialize。
- 在绘制窗口之前，你必须调用 Init3DFullScreen 函数。

//初始化引擎 800×600×32，启用垂直同步，全屏

TV.Init3DFullscreen 800, 600, 32, True, TV_DEPTHBUFFER_BETTER, 1, Form1.Hwnd

Initialize（窗口句柄，是否窗口，伽马）
用选定的模式来初始化窗口（在一个对话框中选择）

参数

窗口句柄(可选的)	用于渲染的窗口句柄(窗口模式或者全屏). 如果想用内部窗口就选择0.
是否窗口（可选的，默认是窗口模式）	如果引擎必须初始化为全屏或者窗口，被对话框的选择覆盖
伽马（可选的，默认为0）	伽马因子（默认是1，越大越亮）

返回值为布尔型，如果引擎被正常地初始化了，则返回 True，如果有问题则返回 False。
说明：

- 如果需要从一个对话框中得到结果，你就必须使用 TV.Initialize。
- 在绘制窗口之前，你必须调用 Init3DWindowedMode 函数。
- 桌面显示模式是很重要的，如果它在 24bits 色深下，或者用了一个显卡不支持的模式，引擎将不会启动。所以一定要在 16bits 或 32bits 色深下。
- 当你用了这个对话框，它会使用用户选择的显示模式。你能用 GetVideoMode 得到显示模式。

```
// 显示对话框和根据用户指定的来初始化。
If TV.ShowDriverDialog() = False Then
End   //如果用户点击取消，不要初始化。
End If
TV.Initialize Form1.Hwnd
```


ShowDriverDialog()
显示对话框（用户可以选择显示模式）。

返回值：

返回值	如果用户选择了显示模式，返回 True，如果用户关闭了对话框或者点击了 Cancel，返回 False。

说明：

- 你必须用 TV.Initialize 来初始化引擎。
- 你可以用你自己的选择显示模式窗口替换掉系统的对话框，用 GetDriverSetting 函数。
- 如果用户关闭了对话框或者点击了 Cancel 按钮，那么就不会用对话框的设置初始化引擎。

```
//显示对话框，如果用户没有点击 Cancel 按钮，那么就初始化引擎。
If TV.ShowDriverDialog() = False Then
End   //如果用户点击取消，不要初始化。
End If
TV.Initialize Form1.Hwnd
```

EnableAntialising（是否开启）
开启/关闭抗锯齿效果

参数

是否开启	开启/关闭抗锯齿效果。

说明：
该引擎用两个步骤执行全屏抗锯齿（FSAA）。

- 首先，需要用 TV.MultisampleTp 来设置 Multisample（多级采样）的值，根据你显卡的性能定。比如，Multisample 为 2 或 4，在引擎初始化之前，这个就被调用。
- 然后在引擎初始化之后，调用，TV.EnableAntialising。

如果 3D 显卡不支持抗锯齿，那么引擎将会使用普通的渲染方式。

```
//打开 4 级抗锯齿效果
TV.MultiSampleTp = TV3D_MULTISAMPLE_4SAMPLES
//初始化引擎
TV.Init3DFullScreen 800,600,16
//第二步
TV.EnableAntialising True
```

EnableHardwareTL（开关）
开启/关闭硬件 T&L。

参数

开关	开启/关闭硬件T&L渲染。

说明：
现在大多数显卡都支持了 T&L，所以应该起用这一功能，一些特殊的显卡，可能你必须关闭掉 T&L，因为它会导致伪点精灵（错误）。

EnableShaders（是否启动 Shader）
开启/关闭引擎顶点 shader 的使用。

参数

是否启动Shader(可选的,默认值是True)	开启/关闭顶点shaders。

说明：

- 该函数应该在调用模型之前，在引擎调用之后。
- 如果启用 shader，当它渲染 MD2、MDL 时，顶点 shader 就被启用了，比起其他技术要快（比如 CPU）。权衡一下，虽然不能使用标准光照，如果一定要使用标准光照，你就应该关掉 shader。
- Shader 默认是被打开的。
- Shader 仅影响到 MD2、MDL 的渲染，因为 MD2、MDL 也能用到顶点 shader。

DisplayFPS

在荧幕左上角显示 FPS。

说明：

- 这仅用于调试目的。
- 它可以让你准确地看到你程序的速度。

```
//显示 FPS
TV.DisplayFPS=True
```

GetFPS（）

返回每秒的帧数（FPS）。

```
//显示 FPS，两种方法。
//用内置的方法
TV.DisplayFPS=True
//用文本显示
Scene.DrawText 10,10,"FPS is " & TV.GetFPS
```

ScreenShot（档路径）

对当前的荧幕进行快照。

参数

档路径	要写入快照的路径+文件名。

说明：

- 该快照是 32bits 的 BMP 位图，所以会变得很大。
- 渲染代码中的任何地方都能调用快照功能，但是如果在 TV.RenderToScreen 之前调用，将会抹掉上一帧。
- 该函数非常地慢，所以切记不要滥用！:-)

```
//当用户按了“S”，截取当前图像。
TV.Clear
DrawAllScene
TV.RenderToScreen
If Inp.IsKeyPressed(TV_KEY_S) = True Then
TV.Screenshot "Scrennie.bmp"
End If
```

SetAngleSystem（角度系统系统）

用引擎改变角度系统

参数

角度系统类型	要使用的新的角度系统。

说明：

- 该函数是标准角度系统中的主函数。你可以选择弧度（Radian）或者角度（Degree）。
- 在改变了角度系统以后，所有功能使用的角度将适应使用适当的角度
- 引擎默认的角度系统是弧度。

//把角度系统改成度（Degree）
TV.SetAngleSystem TV_ANGLE_DEGREES

SetSearchDirectory(路径)
设置新的引擎默认的路径

参数

路径	新的引擎默认的查找档路径

说明：
指定的搜索的档是：
- 当前的目录；
- PAK 档；
- TV3D 的 dll 档；
- 图像档、音乐档等多媒体档

SetWatermarkParameters（水印位置，Alpha 浓度）
改变水印的属性

参数

位置	选择显示水印的角度。
Alpha浓度（可选的，默认是0.8）	水印的透明度（你可以在0.6-1之间进行选择）

说明：
- 商业版就没有这个水印了。

GetVideoMode(返回的宽度，返回的高度，返回的色深)
获得当前引擎所使用的显示模式

参数

返回的宽度	渲染窗口的宽度
返回的高度	渲染窗口的高度
返回的色深	代表颜色的深度的值。 (16bits或32bits)

说明：
- 只能是初始化了引擎以后。
- 如果引擎被初始化成窗口以后，则返回值就是渲染的窗口大小。

//获得当前被引擎使用的显示模式，并且把它们保存在变量中。
Dim Width,Height,BPP As Long
TV.GetVideoMode Width,Height,BPP

Pause(毫秒数)

依照制定的毫秒数使计算机“睡眠”

参数

毫秒	暂停的毫秒数

说明：
这种方法可以例如被用来限制帧速率，或在窗口未被启动时，释放CPU资源时。

```
//当窗口未被启动时，使计算机睡眠 500 毫秒
Do
    Do While TV.HasFocus = False
      TV.Pause 500
    Loop
    TV.Clear
    RenderAll
    TV.RenderToScreen
Loop
```

SetREFMode(是否开启)
开启参考光栅器（用于调试模式，软件渲染是非常慢的）。

参数

是否开启	打开/关闭参考光栅器

说明：

- 该功能必须在引擎初始化调用。
- 在 DirectX 的发布版中参考光栅器不会工作，但是可以工作在调试模式中。这就是说通常你的用户在拿到产品后，不会以参考光栅器的方式运行。
- 参考光栅器非常非常地慢，而且不会替换软件渲染。

```
//打开 REF 模式
TV.SetREFMode True
//用它来初始化引擎
TV.Initialize Form1.hWnd
```

SetGeneralSpeed(速度)
改变所有流逝的时间的时间因子

参数

速度	速度因子，小于 1 时速度更快，大于 1 时更慢。

说明：

- 该函数能够减缓渲染或者移动的速度。

Clear（是否开启 Z 缓冲）

清除背景缓冲区，开始新的一帧。

参数

是否开启 Z 缓冲（可选的，默认是 False）	在每次渲染之前，必须调用该函数清除背景缓冲区。

说明：
这个方法是引擎中最根本、重要的，因为它能告诉引擎你要在背景缓冲区开始绘制 3D、2D 元素了。Clear 方法必须和 RenderToScreen 配对！切记！

```
//渲染的基础示例
Do
    DoEvents
    //清除背景缓冲区，开始渲染
    TV.Clear
    //绘制所有的东西
    DrawAllMeshes
    //别忘了，最后把这些东西都渲染在荧幕上。
    TV.RenderToScreen
Loop Until 程序结束标志
```

RenderToScreen()
将所有已经绘制的东西显示在荧幕上。

说明：
- 这个函数是非常重要的，它是渲染的一部分。
- 所有绘制的东西是在 TV.Clear 和 TV.RenderToScreen 之间，然后显示在荧幕上。
- 在执行渲染之前，你必须初始化引擎。

```
//很重要的渲染示例
Do
    //清除背景缓冲区，准备好渲染
    TV.Clear
    //渲染场景
    RenderScene
    //结束渲染阶段，把所有东西显示在荧幕上（窗口或全屏）
    TV.RenderToScreen
Loop
```

IsWindowed()
判断如果引擎运行在窗口模式下，返回 True。

返回值：如果引擎运行在窗口模式，返回 True，如果运行在全屏模式，返回 False。

说明：

你会经常需要处理游戏在窗口或是全屏的情况下的不同,所以如果你不知道用户怎样初始化引擎，这个函数能够帮助你。

ShowWinCursor（是否开启）
显示/隐藏 Windows 游标。

参数

是否开启	关掉/启用 Windows 游标显示。

说明：
该函数可以工作在窗口或者全屏模式中。
//隐藏 Windows 游标
TV.ShowWinCursor False

OpenPAKFile(档路径+文件名)
打开一个 PAK 档（压缩）以便让引擎读取

参数

文件名	PAK 档的文件名

说明：
- 你能同时打开多个 PAK 档。
- 当你打开一个 PAK 档时，引擎会自动将 PAK 解压缩，当你读取一个档时，它将首先检查当前目录，如果该档未找到，将要搜索在 PAK 中的档，从这里解压。
- 别忘了，在读取了所有档后，关闭 PAK 档，这样能释放一些存储器。
- 创建一个 PAK 档是很简单的，用 TV3D SDK 提供的 PAKExplorer 就可以了。
- 目前，引擎支持的 PAK 格式的只有 HalfLife，一个新的压缩档格式正在开发中。

//打开“texture.pak"，再从里面读取“texture1.bmp"。
TV.OpenPAK "texture.pak"
TextureFacactory.LoadTexture "texture1.bmp","texture"
TV.ClosePAK

ClosePAKFile()
关掉所有已经打开的 PAK 档

说明：
当你读取了一个 PAK 档，准备释放一些存储器时，不要忘记用这个方法关掉所有打开的 PAK 档。

SetCamera(摄影机对象)
在当前的视口中，应用一个新的摄影机对象。

参数

摄影机对象	要被应用到当前视口的摄影机对象

说明：

- 当你调用这个函数时，当前的视口就被设置了一个摄影机。
- 改变会立即生效，所以你在渲染中不应该改变摄影机，否则你通过不同的摄影机会看见一些物体它们显得很奇怪。

```
//创建一个新的摄影机并应用到当前视口中
Dim CamFact As New TVCameraFactory
Dim NewCam As TVCamera
Set NewCam = CamFact.CreateCamera()
TV.SetCamera NewCam
// 进行渲染
TV.Clear
RenderAll
TV.RenderToScreen
```

GetCamera()

返回当前的摄影机对象。

返回值：TVCamera 摄影机对象，引擎所使用的摄影机对象（之前已经用 SetCamera 定义了）。

说明：

- 你必须先用 TV.SetCamera 建立了一个摄影机对象。
- 默认的，该函数返回初始化引擎时创建的摄影机。

ResizeDevice()

通过渲染窗口的客户端，调整背景缓冲区的大小（如果有必要就切换回窗口模式）。

返回值：如果引擎正确地重置了，返回True。返回False的情况是很少见的，引擎在不稳定的状态时，你应该完全停止程序（否则可能在下一个渲染中毁掉。）

说明：

- 该函数通常在窗口模式当渲染窗口大小被改变时，被调用（通常跟踪 WM_RESIZE 消息或者改变大小事件）。
- 注意，当改变窗口大小时，你能够让引擎自动改变大小。
- 这个函数只能用在窗口模式。
- 如果目前处在全屏模式下，那么那么该引擎就会切回窗口模式，所以，这个功能可以被用于 Alt+回车，在窗口和全屏转换。

```
//切换到窗口模式
TV.ResizeDevice
```

ResizeFullScreen(荧幕宽度，荧幕高度，色深，Z 缓冲)

切换到全荧幕，改变成指定的显示模式。

参数：

荧幕宽度	新的显示模式的宽度，用像素表示，比如 1024，1280……
荧幕高度	新的显示模式的高度，用像素表示，比如 768，1024……
色深(可选的,默认是-1)	颜色深度（16 或 32 位）
Z 缓冲	引擎将要使用的深度缓冲，如果指定的没有正常工作，引擎就会寻找一个能够工作的。

返回值：如果引擎正确地重置了，返回True。返回False的情况是很少见的，引擎在不稳定的状态时，你应该完全停止程序（否则可能在下一个渲染中毁掉。）

说明：

- 如果目前处在窗口模式下，那么那么该引擎就会切回全屏模式，所以，这个功能可以被用于 Alt+回车，在窗口和全屏转换。

//切换到全屏模式。
TV.ResizeFullScreen 800,600,32

SetShadowMandatory(是否开启)
在引擎初始化之后强制使用阴影

参数

是否开启	渲染中开启/关闭模板阴影

说明：

- 该函数“强制”渲染中允许使用阴影。
- 你必须在引擎初始化之前调用。因为它内置了深度缓冲。

//打开阴影
TV.SetShadowMandatory True
//初始化引擎
TV.Initialize Form1.hWnd

SetShadowParameters(阴影颜色，阴影大小)
改变将要应用阴影的属性

参数

阴影颜色	新的阴影颜色（用 RGBA 宏得到）
阴影大小（可选的，默认是 100）	新的阴影大小。

说明：

- 必须在引擎初始化之后调用。
- 比如如果你想要动态改变阴影，你可以多次调用它。

//设置阴影长度为 2000，设置阴影颜色为(0，0，0，0.5)
TV.SetShadowParameters RGBA(0,0,0,0.5),2000

TickCount()

返回启动计算机开始逝去的毫秒数。

返回值：返回启动计算机开始逝去的毫秒数。

说明：
这个函数模拟了 API 函数 GetTickCount()，能够用于一些类似于时间的东西。

```
//暂停 500 毫秒
Delay = 500
Old = TV.TickCount
Do
DoEvents
Loop Until TV.TickCount - Old > Delay
```

TimeElapsed()
得到在当前帧和最后一帧之间的延时的毫秒

返回值：自从最后一帧逝去的时间

说明：

- 流逝的时间可以被用于平滑移动，因此所有的计算机中，所有的移动速度都是一样的，独立于 FPS。
- 这里有一个小小的优化，你应该把 Timeelapsed 保存在一个变量中，然后用这些变量用于计算，这就能够避免 COM 功能过多地被调用。

```
//使游戏者向前走，每秒 100 个单位
Player.MoveRelative 100 * TV.TimeElapsed / 1000
// /1000 是把流逝的时间转换成秒。
```

AccurateTimeElapsed()
得到在当前帧和最后一帧之间的延时的毫秒

返回值：自从最后一帧逝去的时间

说明：

- 该函数是个增强的 TimeElapsed，你应该用这个来获得最好的精确度。
- 流逝的时间可以被用于平滑移动，因此所有的计算机中，所有的移动速度都是一样的，独立于 FPS。
- 这里有一个小小的优化，你应该把 Timeelapsed 保存在一个变量中，然后用这些变量用于计算，这就能够避免 COM 功能过多地被调用。

```
//使游戏者向前走，每秒 100 个单位
Player.MoveRelative 100 * TV.AccurateTimeElapsed / 1000
// /1000 是把流逝的时间转换成秒。
```

到此，TVEngine 的函数基本上就介绍完毕了。

TVScene

CreateMeshBuilder(名字)
创建一个新的蒙皮网格（模型）

参数

名字（可选的，默认是空）	名字是新的蒙皮网格的名字。可以被 TVMesh.SetMeshName 覆盖。

返回值：要创建一个蒙皮网格，你必须用这个函数。否则因为没有包含在 Scene 中，就不会渲染这个模型。

```
//创建并渲染一个茶壶的模型
Dim Teapot As TVMesh
Set Teaot = New Scene.CreateMeshBuilder()
Teapot.CreateTeapot
Do
    TV.Clear
    Teapot.Render//渲染
    TV.RenderToScreen
Loop
```

DestroyMeshes(Mesh 对象的名字)
从 Scene 中删除指定的 mesh，指定 mesh 的数据将从存储器中释放。

参数

Mesh 对象的名字	要删除的 Mesh 对象

说明：
该方法有时会发现许多不稳定的对象将把它们删除，只有它们可用时，才会适当地使用。
注意：要释放所有涉及的对象，就应该用它。

```
//假装地创建一个 Mesh 然后再删除掉
Set Mesh = Scene.CreateMeshBuilder
Scene.DestroyMesh Mesh
Set Mesh = Nothing
```

DestroyAllMeshes()
从 Scene 中删除所有已存在的 Mesh，所有 Mesh 的数据将从存储器中释放。

说明：该方法有时会发现不稳定的对象将把它们删除，所以要适当地使用它。

```
//在开始新场景之前，删除所有的 meshes。
Scene.DestroyAllMehes
```

CreateBillBoard(纹理，X，Y，Z，宽度，高度，名字，Alpha，源颜色，目标颜色，颜色，是否围绕中心旋转)

创建一个新的广告牌 mesh 对象，并且附加进 Scene 列表。广告牌是个显示在 3D 场景中的 2D 对象，并且总是面向前方。

参数

纹理	要用到的纹理
X	在世界坐标中心的广告牌 X 坐标
Y	在世界坐标中心的广告牌 Y 坐标
Z	在世界坐标中心的广告牌 Z 坐标
宽度	广告牌的宽度（Z 轴坐标）
高度	广告牌的高度（Y 轴坐标）
名字（可选的，默认是空）	广告牌的名字
Alpha（可选的，默认为 0）	如果引擎必须使用 Alpha 混合，就不要设置为 0.
源混合颜色（可选的，默认为 0）	混合的源格式（最终颜色=源混合颜色+目标混合颜色）
目标混合颜色（可选的，默认为 0）	混合的目标格式（最终颜色=源混合颜色+目标混合颜色）
颜色（可选的，默认是白色）	广告牌的主颜色
是否围绕中心旋转（可选的，默认为 False）	如果广告牌必须在中心底部，就可以设为 True。

返回一个新的包含广告牌数据的 TVMesh 对象。

说明：

BillBoard 广告牌能够被用来模拟效果（比如燃烧、爆炸）。如果正确调用了，它能够表现球状的或者柱面状的 mesh，比如树。

```
//创建一棵宽度为 30 的，高度为 50 的，位置在座标原点的树。
Dim Tree As TVMesh
Set Tree=Scene.CreateBillBoard(GetTex("TreeTexture")),0,0,0,30,50,"tree billboard",True)
```

CreateCubicEnvironmentMap(纹理大小)

创建一个新的可用于立方体环境映射的渲染表面。

参数

纹理大小	立方体图的大小，2 的 N 次方大小,从 0 到 12 次方。 1,2,4,8,16,32,64,128,256,512,1024,2048,4096

返回一个表现立方体图的渲染表明对象。TVRenderSurface

说明：

变成立方体的图，只能用一个维度。（宽度=高度）

```
//创建一个立方体图
Dim RS As TVRenderSurface
Set RS=Scene.CreateCubicEnvironmentMap(256)
```

CreateRenderSurface(宽度，高度，是否使用 3D 效果，X 位置，Y 位置)

新建一个使你程序的效果好像镜面、动态模糊等的（附加的渲染纹理）渲染表面。该渲染表面支持 Alpha 通道，可以创建显得透明的效果。

参数

宽度	渲染表面的宽度（应该是 2 的 N 次方）
高度	渲染表面的高度（应该是 2 的 N 次方）
是否使用 3D 效果	如果深度缓冲必须要附加在渲染表面上（实际上仅用于渲染表面的 3D 渲染）
X 位置（可选的，默认是 0）	过时了。
Y 位置（可选的，默认是 0）	过时了。

返回一个准备进行渲染的空的渲染表面。

说明：

渲染表面是一个特殊的纹理，所以纹理规则适用于渲染表面（必须是 2 的 N 次方大小）。

渲染表面可以自动地附加在它们上，让你没有任何问题地渲染在 3D 场景上。

```
//创建一个 256×256 大小的渲染表面，并在上面写一些字。
Dim RS As TVRenderSurface
Set RS = Scene.CreateAlphaRenderSurface(256,256,True,0,0)
RS.StartRender
ScrText.TextureFont_DrawText("yes text on the rendersurface!", 100, 100, -1, "texturefont")
RS.EndRender
```

CreateActorTVM(名字)

创建一个空的 TVActor2 对象。

参数

名字（可选的，默认是空的）	新的实体的名字

返回一个创建的 TVActor2 对象。

```
//创建一个新的Actor，在里面加载模型。
Dim Actor As TVActor2
Set Actor=Scene.CreateActorTVM("barney")
Actor.Load "barney.md1"
```

DrawText(文本，X，Y，颜色)

用内置的字体（Tahoma 字体，8 磅）在荧幕上绘制简单的文本。这个方法现在已经被淘汰了，只用于调试。

参数

文本	要显示的文本，请避免空文本。
X	文本的左上角的 X 坐标
Y	文本的左上角的 Y 坐标
颜色（可选的，默认是白色。）	一段 32bits 的 16 进制颜色值，或者用 RGBA 获得值。

说明：
因为这个函数是很慢的，你应该用 TVScreen2Dtext 类的一个函数来代替。
//在荧幕的 30，50 点处显示出一个 mesh 的位置
Scene.DrawText Mesh.GetPosition.X & “,” & Mesh.GetPosition.Y & “,” & Mesh.GetPosition,30,50,RGBA(1,1,1,1)

GetCamera()
返回当前的摄影机对象。

返回值：TVCamera 摄影机对象，引擎所使用的摄影机对象（之前已经用 SetCamera 定义了）。
说明：
- 你必须先用 TV.SetCamera 建立了一个摄影机对象。
- 默认的，该函数返回初始化引擎时创建的摄影机。

//在 Y 轴，旋转当前的摄影机 80°。
Scene.GetCamera.RotateY 80

SetCamera(X 位置，Y 位置，Z 位置，X 视角，Y 视角，Z 视角)
简单的定义了摄影机的位置和朝向。

参数

X 位置	摄影机的 X 坐标。
Y 位置	摄影机的 Y 坐标。（海拔高度）
Z 位置	摄影机的 Z 坐标。
X 视角	摄影机视角的 X 坐标
Y 视角	摄影机视角的 Y 坐标（海拔高度）
Z 视角	摄影机视角的 Z 坐标

说明：
- 你不能用这个定义一个摄影机的垂直朝向，要这样做，你要用 RotateX 方法，或者用特殊的矩阵。
- 如 Sample 所显示的那样，SetCamera 可以和三角学一起得到一个充分运作的照相机系统。
- 如果用 Position=LookAt，那么该行为不会被系统解释。

//常用的设置摄影机地方法
Scene.SetCamera Pos.x,Pos.y,Pos.z,Pos.x+Cos(AngleY),Pos.y,Pos.z+Sin(AngleY)

GetCollisionImpact(碰撞点，法线，距离)
得到最后一次碰撞的交集（碰撞点）。

参数

碰撞点	返回碰撞交集世界坐标的 3D 矢量（或者是碰撞点）
法线	返回碰撞平面的法线的世界坐标的 3D 矢量（垂直的），能够用于一些物理碰撞。
距离(可选的，默认是 0)	返回起始点和碰撞点之间的距离。

说明：
要得到进一步和更准确的信息，就应该使用 AdvancedCollision 方法，它返回包含所需信息的 TV_COLLISIONRESULT 结构。

```
//返回碰撞结果。
DoCollision
Dim CollisionImpact As D3DVECTOR
Dim CollisionNormal As D3DVECTOR
Dim Distance As Single
Scene.GetCollisionImpact CollisionImpact,CollisionNormal,Distance
```

Collide(起始点，终止点，测试类型，是否为 Landscape)
在大地和 mesh 或 Actor2 等实体做碰撞检测（注意：BSP 和 Actor 的碰撞还不支持！），这是最基本的碰撞检测，返回在起始点到终结点之间最临近的撞击。

参数

起始点	碰撞测试的起点
终止点	碰撞测试的终点
测试类型（可选的，默认的是 TV_TESTTYPE_ACCRATETESTING）	定义碰撞检测的类型。BOX 检测和 SPHERE 是最快的，但是准确度低。
是否为 Landscape（可选的，可选的，默认是 True）	是否检测碰到 Landscape

返回为如果 Scene 的一个元素和另一个发生了碰撞，就是 True。
说明：
这个碰撞检测返回的信息不是很多，要想得到更多，比如碰撞法线、点等，请用 AdvancedCollision。

AdvancedCollision(起始点，终止点，返回的碰撞结果，检测的物体，检测类型)
在 landscape 和 mesh 或 Actor2 等实体做碰撞检测(注意:BSP 和 Actor 的碰撞还不支持!)，这是最基本的碰撞检测，返回在起始点到终结点之间最临近的撞击。

参数

起始点	碰撞测试的起点
终止点	碰撞测试的终点
返回的碰撞结果	如果碰撞发生了，返回的信息。包含了碰撞法线、碰撞点、碰撞的距离，和一些其他信息。
检测的物体	检测物体的类型，你可以把它们组合起来（MESH 和 LANDSCAPE 组合）或者用 0 就表示想要检测所有碰撞。
检测类型	定义碰撞检测的类型。BOX 检测和 SPHERE 是最快的，但是准确度低。

返回为如果 Scene 的一个元素和另一个发生了碰撞，就是 True。
说明：
这个函数所返回的信息可以用于物理运算，要得到最准确的返回信息，就应该用 TV_TESTTYPE_ACCURATETESTING 的检测模式。

```
//对一个 player 进行碰撞检测。
Dim CollisionResult As TV_COLLISIONRESULT
If Scene.AdvancedCollision(OldPlayerPosition, NewPlayerPosition,
CollisionResult, 0, TV_TESTTYPE_ACCURATETESTING, True) = True Then
//碰撞发生了，让 player 往回走。
PlayerPosition = OldPlayerPosition
Else
//没有就继续向前走。
PlayerPosition = NewPlayerPosition
End If
```

CheckGravity(重力摄影机，逝去的时间，下限，上限，是否为 Landscape)*
在碰撞中或者重力中检测摄影机的位置。

参数

重力摄影机	0-1 的一组 CameraGravity 数列，TVCamera(0)是移动之前的位置，TVCamea(1)是新的位置。
逝去的时间	上一次的 CheckGravity 和这次的 CheckGravity 之间的流逝的时间。要创建慢动作或者行为特效，你可以改变这个时间段。
下限	当它之下没有对象时，摄影机速度的下限。建议 0.04
上限	摄影机上升时的最高速度。建议 0.04
是否为 Landscape(可选的，默认是 False)	检测一个 Player 和 Landscape 的碰撞。默认是关掉的，因为太复杂的地形会出现问题。

说明：
新的摄影机位置，准备好，设置成 3D 渲染摄影机。

MousePicking(X 位置，Y 位置，对象类型，碰撞类型)
在完整的 Scene 做一个鼠标碰撞

参数

X 位置	荧幕上鼠标碰撞点的 X 坐标（通常是鼠标在哪儿，碰撞点就在那儿）
Y 位置	荧幕上鼠标碰撞点的 Y 坐标（通常是鼠标在哪儿，碰撞点就在那儿）
对象类型（可选的，默认为 0）	要检测的对象类型，你可以把他们组合起来，如果要检测所有对象，就用 0.
碰撞类型（可选的，默认是 TV_TESTTYPE_BOUNDINGBOX）	定义碰撞检测的类型。BOX 检测和 SPHERE 是最快的，但是准确度低。

返回一个TVCollisionResult对象，你就可以获得所有的选择信息。
说明：
要得到进一步和更准确的信息，就应该使用 AdvancedCollision 方法，它返回包含所需信息的 TV_COLLISIONRESULT 结构。

```
//做一个鼠标碰撞，碰到了一个 Mesh
Dim Pick As TVCollisionResult
Set
Pick=TVScene.MousePicking(CurrentMouseX,CurrentMouseY,0,TV_TESTINGTYPE_BOUNDING
BOX)
```

SetCollisionPrecision(碰撞数值)
改变碰撞精度（针对BSP）。

参数

碰撞数值	碰撞精度值（在0到100之间）

说明：
目前只对BSP碰撞有效。

```
//改变碰撞精度为10
Scene.SetCollisionPrecision（10）
```

SetPickingPrecision(远平面)
设置鼠标碰撞精度（除了BSP有效，其他均已过时）

参数

远平面	在远平面后面，检测不到更多的碰撞。

```
//设置鼠标碰撞的远平面为1000
Scene.SetPickingPrecision 1000
```

GetTriangleNumber()
返回在一个场景中可以看见的三角形数量。(在当前的摄影机中)

返回三角形数量。
说明：要想测试你的显卡每秒生成的三角形数量的性能，你可以就像这样地使用：
NumberOfTrianglePerSec=TV.GetFPS*Scene.GetTriangleNumber
//显示每秒钟三角形数量
*TriPerSec=TV.GetFPS*Scene.GetTriangleNumber*
ScrText.NormalFont_DrawText "每秒钟生成的三角形：" & TriPerSec

LoadCursor(文件名，颜色键，宽度，高度)
读取一个图片档作为引擎要渲染的鼠标游标。

参数

文件名	要读取的图片文件名
颜色键（可选的，默认是0）	透明颜色（用引擎使图片透明的颜色）
宽度（可选的，默认是32）	预期的荧幕上的游标宽度（应该是2的N次方大小）
高度（可选的，默认是32）	预期的荧幕上的游标高度（应该是2的N次方大小）

说明：
- 通常最好使用SetCursorTexture，因为它允许预读一个纹理或者位图，允许设置更多的游标属性。
- 游标在TV.RenderToScreen渲染，所以它是最后渲染的基础图形。

//读取一个位图文件，并使之成为32×32大小。
Scene.LoadCursor "array.Bmp",TV_COLORKEY_BLACK,32,32

LoadShaders(文件名)
读取包含在指定档里的Shader

参数

文件名	包含了Shader脚本的文本档

说明：
Shader是一些有特殊效果的档（结构好像Quake3的Shader档），能够被用于制作纹理动画，或者是特殊的混合纹理。
//读取一段通用的Shader，在一个mesh上应用爆炸效果
Scene.LoadShaders "common.shader"
Mesh.SetShader GetTex("explosion")

LoadTexture(文件名，宽度，高度，纹理名称)
从磁盘中读取一个纹理放入纹理工厂。

参数

文件名	要读取纹理的文件名
宽度（可选的，默认是-1）	纹理的宽度，让引擎自动选择最佳大小，用-1或者留空。
高度（可选的，默认是-1）	纹理的宽度，让引擎自动选择最佳大小，用-1或者留空。
纹理名称（可选的，默认是空）	可选的纹理名称

返回在纹理工厂中的索引。

说明：

- 有了TVTexture.LoadTexture，上面的就是多余的了。现在你应该用TVTexture.LoadTexture代替Scene.LoadTexture。
- 所有纹理大小都应该是2的N次方，相对的，引擎也会把2的N次方变成标准大小。这不是TV3D的局限性，所有的3D显卡都只支持2的N次方大小的纹理。
- 引擎能够读取这些格式：
  - 标准的Windows格式：BMP,JPG,PNG,DIB,TGA
  - DirectX纹理格式：DDS

  PNG,DDS,TGA有Alpha通道，包含了纹理的透明信息。

```
//读取一个房顶纹理，把它应用到mesh
Dim RoofTextureIndex As Long
RoofTextureIndex=Scene.LoadTexture("roof.Bmp",,"roof")
```

RenderAllMeshes(是否Alpha排序)

渲染在Scene中的所有可见的mesh，该方法支持Alpha排序。

参数

是否Alpha排序	如果引擎一定要进行Alpha排序，就设置成True。

说明：

- 如果你的mesh用了Alpha混合或者有带颜色键的纹理，Alpha排序将会出现更准确的结果。
- 注意：如果Scene中包含了大量的mesh，Alpha排序能够稍微减慢处理的速度。

```
//把所有的mesh渲染到荧幕上
TV.Clear
Scene.RenderAllMeshes True
TV.RenderToScreen
```

SetDepthBuffer(Z Buffer或者W Buffer)

改变深度缓冲的方法，你可以从Z Buffer和W Buffer之间进行选择，Z Buffer是系统默认的而且适合于所有显卡，W Buffer是更精确的，但是有些显卡不兼容。

参数

Z Buffer或者W Buffer	选择ZBuffer或者WBuffer

说明：

你能够用WBuffer去解决出现的问题。

//用WBuffer解决水面的假象

Scene.SetDepthBuffer TV_WBUFFER

SetMipMappingBuffer(mipmap贴图的精度)

设置渲染中Mip贴图的精度。

参数

mipmap贴图的精度	0是普通的mip贴图，越大于0就越模糊，越小于0就越锐。

//使东西更模糊

Scene.SetMipMappingBuffer 2

SetRenderMode(渲染方式)

设置渲染多边形的方式。你可以选择WireFrame（线）、Point（点）和实心渲染方式。注意:你可以在运行中改变绘制风格。

参数

渲染方式	新的渲染方式，WireFrame（线）、Point（点）和实心渲染方式

说明：

- 你可以在一帧中，多次调用这个函数。

//绘制一个立方体，先用实心方式，再用线方式。

Scene.SetRenderMode TV_SOLID

Cube.Render

Scene.SetRenderMode TV_LINE

Cube.Render

SetSceneBackGround(红色，绿色，蓝色)
改变场景中的背景颜色（场景的填充颜色）。

参数

红色	红色的数值。范围从0到1
绿色	绿色的数值。范围从0到1
蓝色	蓝色的数值。范围从0到1

说明：
- 默认的背景颜色是黑色。(0.0,0.0,0.0)

//把场景的颜色设置为天蓝色。
Scene.SetSceneBackGround 0.7,1.0,1.0

SetShadeMode(Shade)
改变着色方式。Shade是引擎对三角形的着色方式。
FLAT着色，使每一个三角形都有固定的颜色，没有渐变色。GOURAUD着色，使光线照到每一个顶点，所以看起来比FLAT着色更平滑。

参数

Shade	着色方式

说明：
默认是Gouraud着色。这是一个“快速渲染状态”意味着一个物体在被调用了SetShadeMode后，就立即被渲染了。
//将着色方式改为FLAT
Scene.SetShadeMode TV_SHADEMODE_FLAT

SetSpecularLighting(是否开启)
开启/关闭镜面光。

参数

是否开启	开启/关闭镜面光模式。

说明：
镜面光默认是被关掉的，想在模型上得到高亮的光，就必须把它设置为True。注意：镜面信息需要在材质中的light中获得。镜面光可以提高你的光的质量,但是它非常耗费CPU的资源，所以只有十分必要时才用。
//打开镜面光
Scene.SetSpecularLighting True
//关掉镜面光
Scene.SetSpecularLighting False

SetTextureFilter(过滤器类型)

设置纹理过滤，这是个改变纹理风格的方法。不是所有显卡都支持Cubic过滤和Anisotropic过滤，Trilinear（三线性）过滤和Anisotropic（各向异性）过滤可以使图像呈现出最好的质量，还有mip图。Point（点）过滤弄出的就好像点状的图像。双线性过滤是个好方法，但是没有mip图。

参数

过滤器类型	渲染3D场景的过滤器类型。

说明：

当你启动引擎时，默认的过滤器是三线性过滤。

//改变过滤器为双线性过滤

Scene.SetTextureFilter TV_TEXTUREFILTER_BILINEAR

SetViewFrustum(视野角度，远平面)

打开透视图，设置当前的视图属性。

参数

视野角度	视野角度的角
远平面	视野的最大距离。（假想）远平面的所有东西都不会渲染。

说明：

- 要想控制近平面，请看TVCamera.SetViewFrustm。
- 如果上一个模式是等轴的摄影机，该方法将会切换回透视视图。
- 视野角度定义了你能看到东西的多少，60°和90°是比较好的。如果你缩小视野角度，你就能做出一些放大效果。
- 你应该为你的场景设置远平面最可能小的值，通常在1000到50000之间。如果设置的太大了，你将看到错误的东西，那是因为渲染的太慢，还没有渲染到这里。

//设置视野角度为60°，视野的最大距离为4096.

Scene.SetViewFrustum 60,4096

到此，TVScene 的函数基本上就介绍完毕了。

TVInputEngine

这个类主要有 4 个函数。

IsKeyPressed(键位)

如果用户按下了某个按键，就返回 True。

参数

键位	255 个值中的一个。

返回值：如果用户按下了某个按键，就返回 True。

说明：

这是用键盘控制输入的基础。

```
//检查回车键是否被按下。
If InputEngine.IsKeyPressed(TV_KEY_RETURN)=True then
    //输入代码
End If
```

GetMouseState(鼠标的 X 坐标，鼠标的 Y 坐标，鼠标左键，鼠标右键，鼠标中键，鼠标滚轮)

返回鼠标的相关信息。

参数

X 坐标	TV 荧幕里的 X 坐标
Y 坐标	TV 荧幕里的 X 坐标
鼠标左键	如果用户按下滑鼠左键，则返回一个非空值。
鼠标右键	如果用户按下滑鼠右键，则返回一个非空值。
鼠标中键	如果用户按下滑鼠中键，则返回一个非空值。
鼠标滚轮	返回鼠标滚轮的相关信息。

说明：

该方法经常被用于第一人称游戏，因为它允许用户简单地而且没有限制的旋转视图。

```
//通过鼠标移动旋转摄影机。
Dim MouseX As Long, MouseY As Long
GetMouseState MouseX, MouseY
Camera.RotateY -MouseX * 0.02
Camera.RotateX MouseY * 0.02
```

GetAbsMouseState(鼠标的 X 坐标，鼠标的 Y 坐标，鼠标左键，鼠标右键，鼠标中键)
返回鼠标的完全信息

参数

鼠标的 X 坐标	在渲染窗口中的 X 坐标。
鼠标的 Y 坐标	在渲染窗口中的 Y 坐标。
鼠标左键	如果用户按下滑鼠左键，则返回一个非空值。
鼠标右键	如果用户按下滑鼠右键，则返回一个非空值。
鼠标中键	如果用户按下滑鼠中键，则返回一个非空值。

说明：
TV 游标的位置完全不同于 Windows 游标的位置,但是你要知道 TV 游标比 Windows 游标更好，尤其是在碰撞检测中。

```
//显示游标在荧幕上的位置。
Dim MouseX As Long,MouseY As Long
Button1 As Integer
InputEngine.GetAbsMouseState MouseX,MouseY,Button1
If Button1 Then ButtonString="左键被按下"
DrawText "X: " & MouseX & "  Y: " & MouseY & "  Button : " & ButtonString
```

SetMousePosition(X,Y)
设置当前鼠标位置为一新位置。

参数

X 位置	X 坐标的新位置
Y 位置	Y 坐标的新位置

说明：
用它可以使 Windows 游标坐标和 TV 坐标一致。

```
//强制使 Windows 游标坐标和 TV 坐标一致。
//得到 Windows 的 X,Y 位置
GetCursorPos X,Y
InputEngine.SetMousePosition X,Y
```

到此，TVInputEngine 的函数基本上就介绍完毕了，其实还有一些，不过这些就足够编写游戏了。

TVGlobals

主要就是 Get××和 Set××。

CreateDecal(位置，法线，纹理，大小，消失时间)

创建一个贴花的 mesh。

参数

位置	贴花中心的世界坐标的位置。
法线	贴花法线的方向的世界坐标。
纹理	将要被贴花应用的纹理。
大小	贴花的大小（用 TV 坐标）
消失时间（可选的，默认为 0）	贴花被自动删除的延迟时间，用毫秒表示。

返回一个描绘贴花的 mesh。

说明：

贴花能够被用于在渲染表面添加一个纹理精灵，就像山峰上的小路。通常用 Collision 或者 MousePicking 来计算贴花的法线和位置。

FinalizeShadows()

渲染每一阴影之后，结束它。

说明：

要绘制 Alpha 混合的阴影，你就必须调用它，所以好方法是：渲染所有的 mesh，然后渲染阴影，最后在 2D 工作之前结束阴影。

GetActor(Actor 的名字)

从名字中返回一个 TVActor。

参数

Actor 的名字	要得到的 Actor 的名字。

返回值是个有指定名字的 Actor 对象。

说明：

在某个地方替代 GetActor，试图保存 TVActor 对象。

GetActor2(Actor2 的名字)
从名字中返回一个 TVActor2。

参数

Actor2 的名字	要得到的 Actor2 的名字。

返回值是个有指定名字的 Actor2 对象。
说明：
在某个地方替代 GetActor2，试图保存 TVActor2 对象。

GetMat(材质名称)
获得指定名字材质的索引。

参数

材质名称	材质名称

返回一个定名字材质的索引的整数。
说明：
如果你不想保存材质索引，这个很有用。但是小心，这个方法不要用的太频繁了，因为它很慢。

GetTex(纹理名称)
获得指定名字纹理的索引

参数

纹理名称	纹理名称

说明：
如果你不想保存纹理索引，这个很有用。但是警告，如果每一帧都用，它会花费一些时间。

GetMesh(Mesh 名称)
获得指定名字 mesh 的索引

参数

Mesh 名称	要得到的 Mesh 名称。

返回一个被命名的 mesh 对象。
说明：
如果你不想保存 mesh 对象，这个很有用。但是它是个基于字符串的函数，你要避免过多的调用。

GetMesh2(MeshID 号)
从索引值中返回 mesh 对象。

参数

MeshID 号	要得到 Mesh 的 MeshID 号。

返回一个关联着 ID 号的 Mesh。
说明：
高级碰撞方法返回 Mesh 索引，所以如果你想要从那里（高级碰撞方法）得到 Mesh 对象，该方法是强制的。

GetShader(Shader 名称)
从一 Shader 名称中返回 Shader 索引，只有这一种方法能够得到应用到 Mesh 的 Shader 索引 。

参数

Shader 名称	要得到的 Shader 名称。

返回一个描述 Shader 的长整型数。
说明：

- 只有这一种方法能够得到应用到 Mesh 的 Shader 索引!!
- 在得到 Shader 之前，应该用 TVScene.LoadShaders 读取进一个或者更多的 Shader 档 。

GetTexNameFormId(纹理索引)
返回指定索引纹理的名称。

参数

纹理索引	选定的纹理索引。

返回一个使用纹理名称的字符串。
说明：
如果在纹理工厂中没有对应的索引号，引擎将会变得不稳定。因此请确保索引号的正确。

RGBA(红色，绿色，蓝色，Alpha 透明强度)
创建一个 TV 风格的颜色。

参数

红色	红色强度，在 0 和 1 之间的小数取值。
绿色	绿色强度，在 0 和 1 之间的小数取值。
蓝色	蓝色强度，在 0 和 1 之间的小数取值。
Alpha 透明强度	透明强度，在 0 和 1 之间的小数取值。

返回一个描述颜色的长整形数值。
说明：
在这个引擎中，有两种颜色构建方式，一种是颜色组件（红色、绿色、蓝色、Alpha 透明），一种是长整形数，可以使用就像 0XAARRGGBB 的 16 进制格式（aa 是 Alpha 透明，rr 红色，gg 绿色，bb 蓝色），比如纯红色就是 0XFFFF0000。
警告：对于第一种颜色构建方式，你必须遵守从 0-1 取值的规则，否则就会出现稳定性问题 。

Vector(X,Y,Z)
返回一个 3D 矢量。

参数

X	矢量的 X 坐标。
Y	矢量的 Y 坐标。
Z	矢量的 Z 坐标。

返回一个许多引擎通用的 D3DVECTOR 结构体。
说明：
非常简单地创建一个之前没有定义的矢量。

```
//示例，添加创建一个 AI 的若干节点。
AI.AddNode Vector(100,100,250)
AI.AddNode Vector(150,150,200)
AI.AddNode Vector(160,100,200)
```

Vector3(X,Y,Z)
和 Vector 一样。

到此，TVGlobals 的函数基本上就介绍完毕了。

TVLandscape

AdvancedCollide(起点，终点)

在陆地上做高级碰撞测试

参数

起点	测试的起点，用世界坐标。
终点	测试的终点，用世界坐标。

返回碰撞信息。

返回所有 TVCollisionResult 类的信息。

```
//得到点（0，0，0）和点（50，50，0）之间的交集
Dim Coll As TVCollisionResult
Set Coll=Land.AdvancedCollision(Vector(0,0,0),Vector(50,50,0))
//如果碰撞了，显示信息
If Coll.IsCollision Then
    Scene.DrawText "碰撞了！",20,20
End If
```

ChangPointAltitude(X 坐标，Z 坐标，高度，是否更新四叉树，现在不要更新四叉树，关系)

动态编辑地形的点，改变海拔高度。

参数

X 坐标	在陆地上点的 X 坐标。（用世界坐标）
Z 坐标	在陆地上点的 Z 坐标。（用世界坐标）
高度	如果关系为 False，则为高度，如果关系为 True，则为添加的海拔总数。
是否更新四叉树（可选的，默认是 True）	如果必须更新四叉树信息（如果你做 MousePicking 和 Collision 测试是有用的）
现在不要更新四叉树(可选的,默认是 False)	在内存中保持改变，但是不要更新地形的数据，用 FlushHeightChanges 更新地形数据。
关系（可选的，默认是 False）	如果高度的变化是相对于旧的高度则设为 True。

说明：

能用这个函数动态改变点。如果你通过一个不太精确的位置，作为一个地形顶点，然后它将改变最近的顶点。它同样用于 Landscape 的两个接缝（在边框上）。

```
//编辑，从坐标原点提升 100 个海拔高度。
Land.ChangePointAltitude 0, 0, 100, True, False, False
//第二个实例。在 Landscape 上随机地改变点，用 ChangePointAltitude 方法。
For I = 1 To 1000
Land.ChangePointAltitude(Rnd * 1000, Rnd * 1000, Rnd * 1000, True, True, false)
Next I
Land.FlushHeightChanges()
```

CreateDynExpandTexture(纹理宽度，纹理高度，是否保存解压缩的纹理)
创建一个动态将要覆盖当前地形的纹理。

参数

纹理宽度	动态纹理的宽度
纹理高度	动态纹理的高度
是否保存解压缩的纹理（可选的，默认是 False）	如果存在已经解压缩的纹理（要把它用于动态纹理）或者动态纹理必须以黑色开始。

说明：

- 你必须决定用动态纹理来做什么。解压缩的动态纹理或者简单的动态纹理不能同时初始化，否则会不稳定。
- 解压缩的动态纹理替换了应用 Landscape.ExpandTexture 做的解压缩的纹理。动态纹理占用了很多的显存（不能用系统存储器来存储），所以确保在程序的初始化中，创建了少量的动态纹理。这将要帮助存储器管理器，请在引擎生成地形之后调用这个函数。

```
//读取预置的地形档
Landscape.LoadTerrainData "data.ter"
//创建纹理压缩档，大小为 1024×1024.
Landscape.CreateDynExpandTexture 1024，1024，False
```

CreateDynTextures(纹理宽度，纹理高度，是否保存旧的纹理，是否使用存储器管理)
创建少量的动态纹理，每一个动态纹理将被用于一个块状物体。

参数

纹理宽度（可选的，默认是 256）	动态纹理的宽度
纹理高度（可选的，默认是 256）	动态纹理的高度
是否保存旧的纹理（可选的，默认是 False)	如果必须在新的动态纹理上绘制旧的静态的纹理，则设为 True，如果设为 False，动态纹理一开始就是黑色的。
是否使用存储器管理(可选的,默认是 False)	目前不支持。

说明：
动态纹理占用了很多的显存（不能用系统存储器来存储），所以确保在程序的初始化中，创建了少量的动态纹理。这将要帮助存储器管理器，请在引擎生成地形之后调用这个函数。

```
//读取预置的地形档
Landscape.LoadTerrainData "data.ter"
//创建纹理压缩档，大小为 256×256.
Landscape.CreateDynTexture 256，256，False,False
```

CreateEmptyTerrain(精度，高度，宽度，X 位置，Z 位置)

在 Landscape 上创建新的空的地形。

参数

精度	地形的精度，实际上是每个顶点之间的距离。
高度	横向的片的数量。
宽度	纵向的片的数量。
X 位置	空的 Landscape 左上角的 X 坐标。
Z 位置	空的 Landscape 左上角的 Z 坐标。

说明：

一个 Terrain 是有许多 256×256 的小块组成的，它们是更大的 Terrain 精度的基础。它们将更平滑地组成地形。

//创建一个 10×10 的空地形。

```
Landscape.CreateEmptyTerrain TV_PRECISION_AVERAGE,10,10,0,0
//贴图
Landscape.ExpandTexture GetTexture("expandtexture"),0,0,10,10
```

LoadDynTexture(文件名，是否为动态，是否解压缩纹理)

读取保存的动态纹理进 Landscape。

参数

文件名	动态纹理的文件名
是否为动态（可选的，默认为 False）	如果引擎要创建为动态的，设为 True。如果他要创建为静态的，设为 False。
是否解压缩纹理（可选的，默认为 False）	如果读取的纹理是 CreateDynTexture 模式的，设为 True，如果是 CreateDynExpanded，设为 False。

说明：

- 你可能还会用 TVTextureFactory 读取其他格式的纹理，也通常会简化管理。

LoadTerrainData(文件名，偏移量)

读取地形数据并把它应用到当前地形。

参数

文件名	要读取的地形数据档，可以是 PAK。
偏移量（可选的，默认是 1）	引擎必须开始读取档的所在位置。

说明：

如果读入的档和创建地形的大小不匹配，或者如果生成的没有地形，平的，引擎将会清空存储器，生成一个新的地形。

//读取一个预置的地形，之前不需要生成空地形，他将会自动生成。

```
Landscape.LoadTerrainData "data.ter"
//创建大小为 256×256 的动态纹理。
Landscape.CreateDynTexture 256,256,False,False
```

ExpandTexture(纹理，起始的 X 位置，起始的 Y 位置，宽度，高度，是否解开细节)
使一个纹理覆盖上大片的一个矩形区域，通常用于把纹理扩展到整个地形。

参数

纹理	将要被应用到大片上的纹理。
起始的 X 位置	纹理要覆盖的水平的大片的索引号。
起始的 Y 位置	纹理要覆盖的垂直的大片的索引号。
宽度	纹理覆盖的水平大片数量
高度	纹理覆盖的垂直大片数量
是否解开细节（可选的，默认是 False）	如果纹理必须应用扩展的细节图，设为 True。

说明：
人们经常忘记这个方法能应用一个纹理到 Landscape 的一小部分，不仅是整个的。比如你有一个地形是 16×16 的片，你可以在第一个 8×8 的片扩展一个纹理，然后在其他的地方再应用别的纹理。

```
//创建一个 10×10 的空地形
Landscape.CreateEmptyTerrain TV_PRECISION_AVERAGE.10,10,0,0
//贴图
Landscape.ExpandTexture GetTex("expandtexture"),0,0,10,10
```

DeleteAll()
从一个地形中删除所有东西，并且清空存储器。

说明：
当你 Landscape 不用东西时，应该考虑删除。

```
//清空存储器。
Landscape.DeleteAll()
```

DeleteMaterial(Terrain 的数量)
删除部分 Terrain 的 Material，或者是整个的 Terrain。

参数

Terrain 的数量（可选的，默认是-1）	将要被修改的 Terrain 大片的索引号。

```
//删除 5 个片
Landscape.DeleteMaterial 5
//把整个地形的 Material 删除掉。
Landscape.DeleteMaterial -1
```

DestroyDynTexture(是否删除纹理)
删除动态纹理并且释放存储器，如果你不设置是否删除纹理标志的话，那么，最后创建的动态纹理将保留在内存中。

参数

是否删除纹理（可选的，默认为 True）	如果引擎必须删除静态纹理。

说明：
如果不用一些东西删除掉动态纹理是个好做法，因为纹理系统会占用大量的显存，会使渲染变慢。

```
//读取预置的地形（Terrain）
Landscape.LoadTerrainData "data.ter"
//创建大小为 1024×1024 的动态纹理
Landscape.CreateDynExpandTexture 1024,1024,False,False
//在上面绘制
DrawThingsOnLandscapeTex()
//删除动态纹理
Landscape.DestroyDynTexture
```

DynDrawTexture(纹理，X，Z，X1，Z1，颜色，是否混合色，是否重载，纹理平铺的 X 坐标，纹理平铺的 Y 坐标，改变纹理的 X 坐标，改变纹理的 Y 坐标)
在 Terrain 上绘制动态地形。

参数

纹理	需要绘制成动态纹理的纹理。
X	左上角的 X 坐标。
Z	左上角的 Z 坐标。
X1	右下角的 X 坐标。
Z1	右下角的 Z 坐标。
颜色（可选的，默认是-1）	纹理的颜色,通常可以用 RGBA(1,1,1,1)或者 &HFFFFFFFF 还原到原来的颜色。如果用了混合色，可以用 Alpha 控制透明度。
是否混合色（可选的，默认是 False）	开启/关掉混合色。
是否重载（可选的，默认是 False）	如果引擎没有必要覆盖纹理，设为 True。意思是引擎会平铺纹理。
纹理平铺的 X 坐标（可选的，默认是 1）	如果绘制的纹理被应用成动态纹理，则给出平铺的水平值。范围在 1-128 之间。
纹理平铺的 Y 坐标（可选的，默认是 1）	如果绘制的纹理被应用成动态纹理，则给出平铺的垂直值。范围在 1-128 之间。
改变纹理的 X 坐标（可选的，默认是 0）	在 X 轴上的改变纹理的值。
改变纹理的 Y 坐标（可选的，默认是 0）	在 Y 轴上的改变纹理的值。

说明：

- 你必须用 CreateDynTexture 或者 CreateDynExpandTexture 来创建动态纹理。
- 但是用 CreateDynTexture 能获得更佳的纹理质量，这也会占用过多的显存。

GernerateHugeTerrain(高程图，精度，宽度，高度，X，Z，是否关联地形)

在 TVLandscape 中应用最多的函数。从高程图（灰度图）生成地形。

参数

高程图	高程图文件，可以是任意大小的，但是最好是 2 的 N 次方大小。
精度	地形的精度，在每个顶点之间描述距离。
宽度	地形的宽度
高度	地形的高度
X	新的陆地左上角的 X 坐标
Z	新的陆地左上角的 Z 坐标
是否关联地形（可选的，默认是 False）	如果地形必须是“平滑的",可以设置为 True。

说明：

- 一个高程图是个可以描述海拔高度的灰度图，它就像一个立视图。
- 关联细节需要之前就生成好了地形，因为它对地形的生成起了作用。

```
Set Land=New TVLandscape
//因为我们不需要太高的山脉，所以要调整高度因子。
Land.SetFactorY 0.7
//从灰度图生成地形图。
Land.GenerateHugeTerrain "..\..\..\Media\heightmap.jpg", TV_PRECISION_LOW, 8, 8,
 -700, -1024, True
//然后，读取陆地的纹理。
TextureFactory.LoadTexture "..\..\..\Media\dirtandgrass.jpg", "LandTexture"
//给Land贴上图。
Land.SetTexture GetTex("LandTexture")
```

SetFactorY(因子)

在地形生成之前，改变地形的海拔高度因子。

参数

因子	Y 轴上的缩放因子。1 是默认的，小于 1 更低，大于 1 更高。

说明：

该方法必须在地形生成之前调用。

```
//修改海拔高度为 2
Land.SetFactorY 2
Land.SetAffineDetail TV_AFFINE_HIGH
Land.GenerateHugeTerrain "Terrain.jpg",TV_PRECISION_HIGH,4,4,0,0,True
```

SetAffineDetail(TV_LANDSCAPE_AFFINE 的值)
改变关联生成地形的细节，更高的关联将使地形图更平滑。

参数

TV_LANDSCAPE_AFFINE 的值	生成地形的精度。

说明：
在生成地行之前，关联细节需要设置好了，因此它影响了地形的生成。

```
//把关联地形的高度设置为高，然后生成地形。
Land.SetFactorY 1
Land.SetAffineDetail TV_AFFINE_HIGH
Land.GenerateHugeTerrain "Terrain.jpg",TV_PRECISION_HIGH,4,4,0,0,True
```

FlushHeightChanges(是否更新四叉树，是否更新法线)
应用改变到 Terrain 中。

是否更新四叉树（可选的，默认为 True）	如果四叉树必须被更新的话，设为 True。
是否更新法线（可选的，默认为 True）	如果引擎要通过海拔高度重新计算法线，设为 True。

说明：
如果你要更新许多高度信息，FlushHeightChanges 很有用。当你完成时，这个函数是很好的优化方法。

```
//把（0，0）点提高到 100.
Land.ChangePointAltitude 0，0，100，True，False，False
//另一个例子：建立几个随机的点，应用改变到 Terrain 中。
For i=1 To 1000
    Land.ChangePointAltitude Rnd*1000，Rnd*1000，Rnd*1000，True，True，False
Next i
FlushHeightChanges
```

GetHeight(X，Z)
从指定的点返回海拔高度。

参数

X	Landscape 一个点的 X 坐标。
Z	Landscape 一个点的 Z 坐标。

```
//创建 100 颗树，把它们放在适当高的位置。
Dim Height As Long
Dim X As Single, Y As Single
For i=1 To 100
    Set Tree(i)=Tree(0).DuplicateMesh("tree" & i)
    X=Rnd*1024
    Z=Rnd*1024
    Height=Land.GetHeight(X,Z)
    Tree(i).SetPosition X,Height,Z
Next i
```

InitClouds(云的纹理，标志，云的高度，层，层的总数，平铺 X，平铺 Y，云扩展)

初始化云彩系统，或者定制云彩的层。

参数

云的纹理	将被应用到云上的纹理，通常是半透明的纹理或者是 Alpha 透明的纹理。
标志（可选的，默认是 TV_CLOUD_MOVE）	图层的动作。移动还是不移动。
云的高度（可选的，默认是 500）	云层的海拔高度。为了得到最好的效果，应该在所有物体的上面。
层（可选的，默认是 1）	一定要被更新的层的索引号。范围从 1 到层的总数。
层的总数（可选的，默认是 1）	层的总数。取值范围从 0 到 16，0 是没有云彩。
平铺 X（可选的，默认是 1）	云层的横向平铺。
平铺 Y（可选的，默认是 1）	云层的纵向平铺。
云扩展（可选的，默认是 0）	略。

说明：

在初始化云层之前就应该生成地形。

```
//初始化 3 个云层。
Land.InitClouds GetTexture("Cloud1"),TV_CLOUD_MOVE,400,1,3,1,1
Land.InitClouds GetTexture("Cloud2"),TV_CLOUD_MOVE,450,2,3,1,1
Land.InitClouds GetTexture("Cloud2"),TV_CLOUD_MOVE,500,3,3,1,1
//设置他们的流动速度。
Land.SetCloudVelocity 1,0.03,0.01
Land.SetCloudVelocity 2,0.05,0.02
Land.SetCloudVelocity 3,0.02,0.03
```

SetCloudVelocity(图层索引，X 速度，Z 速度)
改变云层流动的速度。

参数

图层索引	云层的索引数。（取值范围从 1 到云层数量）
X 速度	流动的水平速度。（范围从-2 到 2）
Z 速度	流动的垂直速度。（范围从-2 到 2）

说明：
通常是不同的方向，每个云层有不同的速度会有很好的效果。

```
//初始化 3 个云层。
Land.InitClouds GetTexture("Cloud1"),TV_CLOUD_MOVE,400,1,3,1,1
Land.InitClouds GetTexture("Cloud2"),TV_CLOUD_MOVE,450,2,3,1,1
Land.InitClouds GetTexture("Cloud2"),TV_CLOUD_MOVE,500,3,3,1,1
//设置他们的流动速度。
Land.SetCloudVelocity 1,0.03,0.01
Land.SetCloudVelocity 2,0.05,0.02
Land.SetCloudVelocity 3,0.02,0.03
```

SetCloudTexture(纹理，云层)
应用纹理到指定的云层。

参数

纹理	要应用的纹理云层的索引号。
云层（可选的，默认是-1）	云层的索引数。取值范围从 1 到云层的数量。

说明：
通常要得到较好的效果，就要在 TGA、PNG 或者 DDS 格式的档中应用上 Alpha 通道。

SetCloudMaterial(材质索引，云层)
应用材质到指定的云层。

参数

材质索引	要应用材质的云层的索引号。
云层（可选的，默认是-1）	云层的索引数。取值范围从 1 到云层的数量。

说明：
必须首先用 InitClouds 初始化云彩。

SetFlipMode(是否翻转)
按 X 轴翻转地形修复坏的纹理朝向。

参数

是否翻转	翻转，设为 True。默认是 False，保持原样。

说明：
该方法运用几次翻转重叠也能创建洞穴。

SetHeightMapMode(高程图模式)
改变高程图模式

参数

高程图模式	新的高程图模式。

说明：
- 所有新生成的地形将会使用高程图模式。
- 颜色模式用于避免灰度高程图 256 色的上限，允许使用 3 个颜色的通道红、绿、蓝创建 16777216 色的高程图。

SetPagingSystem(是否开启)
开启页面系统，地形变成了循环的。

参数

是否开启	开启/关闭页面系统。

说明：
理论上应该很严丝合缝的。因为如果边界和海拔不匹配，引擎会分析他们。
//打开页面系统，使地形不断地循环。
Land.SetPagingSystem True

SetBumpMapping(是否开启，凹凸纹理，灯光 ID，凹凸的程度)
在地形上应用凹凸贴图。

参数

是否开启	开启关闭凹凸贴图。
凹凸纹理	将被用于凹凸贴图的纹理。
灯光 ID	凹凸贴图用方向光，必须用 TVLightEngine 创建的光。
凹凸的程度（可选的，默认是 5）	凹凸的幅度，取值范围从 1-10，默认是 5.

说明：
如果显卡不支持凹凸贴图，引擎可以跳过。

SetCustomLightMap(光照贴图)
用一个自定义的光照贴图来代替凹凸贴图。

参数

光照贴图	必须应用在地形光照贴图的，在纹理工厂中的已经读取好的纹理

说明：
- 一个自定义的光照贴图将被扩展到整个地形。

SetDetailTexture（纹理）
用指定的纹理作为细节贴图。

参数

纹理	要作为细节贴图的纹理。

说明：
细节贴图必须是黑白对比的灰度图才能提供最好的效果。
//打开细节纹理贴图
Land.SetDetailTexture GetTex("DetailTexture")

SetDetailTextureMode(模式)
改变细节贴图的模式，纹理混合有 3 种模式。

参数

模式	用于渲染 Landscape 的模式。

说明：
知道什么是适合你的最好方法，试用所有的方法之后，然后选择最适当的一个。

SetTextureScale(U 的缩放，V 的缩放，地形索引)
修改地形纹理的缩放比例，只能在非扩展的纹理下工作。

参数

U 的缩放	在一个大片上的地形纹理的水平平铺
V 的缩放	在一个大片上的地形纹理的垂直平铺
地形索引（可选的，默认为-1）	要被修改的大片的索引。-1 可以更新整个地形。

//在整个 Terrain 上平铺 32×32 的纹理。
Land.SetTextureScale 32,32,-1

SetDetailTextureScale(**U 的缩放,V 的缩放,地形索引**)
在一个区域内设置细节纹理成平铺。

参数

U 的缩放	水平平铺的因子。
V 的缩放	垂直平铺的因子。
地形索引（可选的，默认是-1）	要被修改区域的索引。-1 可以更新整个地形。

说明：
必须用 SetDetailTexture 把细节设置为活动的。

```
//用 SetDetailTexture 设置细节为活动的。
Land.SetDetailTexture GetTex("DetailTexture")
//改变为平铺。
Land.SetDetailTextureScale 32,32,-1
```

SetTerrainScale(X,Y,Z,是否缩放法线)
在地形生成以后，改变地形的缩放大小。

参数

X	X 轴的缩放比例。默认是 1,-1 将要反转地形。
Y	Y 轴的缩放比例。默认是 1,-1 将要反转地形。
Z	Z 轴的缩放比例。默认是 1,-1 将要反转地形。
是否缩放法线（可选的，默认为 True）	通常当你想要在已缩放的地形上得到精确的光时需要缩放法线。

说明：
利用地形缩放可以产生真正的巨大地形。

```
//使地形放大 5 倍的面积。
Land.SetTerrainScale 5，1，5
//提升 5 倍的海拔高度
Land.SetTerrainScale 1，5，1
```

InitRain（雨的密度，雨的纹理，落下速度，风的 X 坐标，风的 Z 坐标，随机数，目标）
初始化雨的粒子系统。

参数

雨的密度	粒子的密度，影响粒子数量。（粒子数量=密度×1000）
雨的纹理	将要应用在粒子上的纹理。
落下速度（可选的，默认为-1）	粒子掉下的速度。（通常是 1）
风的 X 坐标（可选的，默认为 0）	风影响的 X 坐标，添加风，以便于到指定的方向。
风的 Z 坐标（可选的，默认为 0）	风影响的 Z 坐标，添加风，以便于到指定的方向。
随机数（可选的，默认为 0.1）	被添加的随机值以变得更接近现实。
目标（可选的，默认为 20）	围着雨转的摄影机的半径，这是一个优化以便雨水不会远离摄影机而掉到别的地方去。

说明：
你同样可以用 TVAtmophere 的 Rain_Init，有更多的方法和预置。

```
//读取雪片粒子，用这个粒子的纹理设置雪片效果。
//100 的密度会使得粒子系统使用最多 1000 个粒子。
Texture.LoadTexture "snow.bmp","Snow",,,TV_COLORKEY_BLACK
Land.InitRain 100,GetTex("Snow")
```

MousePick(鼠标的 X，鼠标的 Y)
在陆地上做鼠标碰撞测试，如果有交集，就返回一串精度信息。

参数

鼠标的 X	荧幕的 X 坐标，通常用绝对位置。
鼠标的 Y	荧幕的 Y 坐标，通常用绝对位置。

返回包含 TVCollisionResult 碰撞类的所有信息。
说明：

- 该方法是快速的，而且用 Landscape 的四叉树系统优化的，所以做碰撞测试，没有任何问题。
- 你应该用 TVScene.MousePicking 代替 TVLandscape.MousePick 函数，那样会更好。

```
//检测鼠标是否碰到了地面。
If Land.MousePick(MouseX,MouseY).IsCollision=True Then
    Scene.SetCursorTexture GetTex("MouseOver")
Else
    Scene.SetCursorTexture GetTex("MouseNotOver")
End If
```

RenderWater(是否开启水雾)
渲染水面。

参数

是否开启水雾（可选的，默认是 False）	如果引擎必须开启雾，设为 True，如果摄影机在水下，设为 False。

说明：
当你渲染 Alpha 混色有问题时，在这一问题上，能够对你有帮助。

```
//首先渲染地形，然后水下的物体，最后让水面保持透明效果。
Land.Render False,False
DrawAllUnderwaterObjects
Land.RenderWater False
```

Render(是否开启水雾，是否渲染水面)
渲染包括天空、太阳、水的陆地。

参数

是否开启水雾（可选的，默认为 True）	如果引擎必须开启雾，设为 True，如果摄影机在水下，设为 False。
是否渲染水面（可选的，默认为 True）	如果引擎同时渲染水，设为 True，否则可以用 RenderWater。

说明：
这是渲染陆地通常的方法，应该在渲染物体之前调用它。

```
//首先渲染地形，然后水下的物体，最后让水面保持透明效果。
Do
    TV.Clear
    DoEvents
    Land.Render False,False
    DrawAllUnderwaterObjects
    Land.RenderWater False
Loop
```

SetWaterAltitude(海拔高度)
改变水面高度。

参数

海拔高度	水平面的海拔高度。

//打开水面，设置纹理，然后设置水面高度为 30.
Land.SetWaterEnable True
Land.SetWaterTexture GetTex("WaterTex")
Land.SetWaterAltitude 30

SetWaterEnable(打开/关闭)
打开或者关闭 Landscape 的水面。

打开/关闭	打开或者关闭 Landscape 的水面

说明：
水实际上是一个在指定的y高度上的大平面。它覆盖了所有的风景，如果想要，甚至可以延长。
//打开水面，并且应用纹理
Land.SetWaterEnble True
Land.SetWaterTexture GetTex("WaterTex")

SetWaterTexture(纹理)
应用指定的纹理，贴到水面上。

参数

纹理	将要被应用纹理的索引号。

//把“WaterTex”贴到水面上。
Land.SetWaterEnble True
Land.SetWaterTexture GetTex("WaterTex")

SetWaterTextureScale(缩放纹理级别)
把水面改变缩放大小。

参数

缩放纹理级别	平铺的水面纹理因子

说明：
用 SetWaterTextureScale 之前必须用 SetWaterEnable 打开水面。
//改变水面平铺
Land.SetWaterTextureScale 4

SetWaterEffect（是否透明，透明因子，是否动画，源混合，目标混合，是否反光，水面扩展）

在水面上应用特效。

参数

是否透明	如果水面使用 Alpha 混合，设为 True。
透明因子（可选的，默认是 0.4）	水面的透明度。
是否动画（可选的，默认是 False）	打开/关闭水面卷动动画。
源混合(可选的,默认是 D3DBLEND_SRCALPHA)	Direct3D 的源混和状态，针对于高级用户。
目标混合（可选的，默认是 D3DBLEND_INVSRCALPHA）	Direct3D 的目标缓冲区混合状态，针对于高级用户。
是否反光（可选的，默认是 0）	不支持。
水面扩展（可选的，默认是 0）	水可以扩展，使水的体积大于地形本身。

说明：

必须先调用 SetWaterEnable，然后再调用 SetWaterEffect。

```
//打开水面，设置纹理和效果。水面必须半透明和动画。
Land.SetWaterEnable True
Land.SetAltitude 45
Land.SetTexture GetTex(“Water")
Land.SetWaterEffect True，0.5，True
```

SetWaterAnimation(X 方向，Z 方向，速度)

改变水面的动画方向。

参数

X 方向	水面滚动的 X 方向。取值范围从-1 到 1 之间。
Z 方向	水面滚动的 Z 方向。取值范围从-1 到 1 之间。
速度	水面移动的速度。

说明：

水是按照正弦波移动的，模拟水波。

```
//更改水的运动方向
SetWaterAnimation 1，0，5
```

SetWaterBumpmapping（……）

设置水面的凹凸贴图。

SetWaterColor(红色，绿色，蓝色，透明度)
更改水面颜色。

参数

红色	红色组件。取值范围从 0 到 1
绿色	绿色组件。取值范围从 0 到 1
蓝色	蓝色组件。取值范围从 0 到 1
透明度	透明度。0 透明，1 不透明，取值范围从 0-1.

说明：
你可以用透明度设置透明度。
//把水面设置为蓝色透明度。
Land.SetWaterColor 0，0，1，0.5

到此，TVLandscape 的函数基本上就介绍完毕了。

TVTextureFactory

LoadTexture(文件名，名字，宽度，高度，颜色键，是否过滤纹理，是否使用多重 Alpha)
从磁盘读取一个普通的纹理。

参数

文件名	磁盘上的纹理档。
名字（可选的，默认是空）	纹理实体名称。
宽度（可选的，默认是-1）	纹理宽度。
高度（可选的，默认是-1）	纹理高度。
颜色键（可选的，默认是 0）	纹理的颜色键，这一部分不会被渲染。结果是透明的。
是否过滤纹理（可选的，默认是 True）	把过滤器用在这个纹理上。
是否使用多重 Alpha（可选的，默认是）	用于屏蔽。如果是多重 Alpha，你只有 2 个 Alpha(0 和 1),如果被设置为 False,在 32bit 色深下，将有 64 个 Alpha。

返回值：返回纹理的索引号。
说明：
注意，所有的纹理大小都应该是 2 的 N 次方大小，否则，引擎将会拉伸，你将会看到和你想象的不一样。

```
//读取一个用于石头的纹理，没有颜色键，用默认的宽度和高度。
TexFactory.LoadTexture("data\textures\rock.bmp","RockTex",-1,-
1,TV_COLORKEY_NONE,true,true)
//读取一个有颜色键的纹理用于 2D 的平视显示器。
TexFactory.LoadTexture("data\hud\hud_main.bmp","MainHUD",-1,-
1,TV_COLORKEY_MAGNETA,true,true)
```

LoadBumpTexture(文件名，名称，是否生成法线图，振幅)
读取一个将被用于凹凸贴图的 mesh 或 terrain。和普通纹理的读取过程不同，因为引擎需要从纹理中读取法线贴图。

参数

文件名	磁盘上纹理档的路径。
名称（可选的，默认是空）	纹理实体名称。
是否生成法线图（可选的，默认是 True）	如果要生成法线图，设为 True。
振幅（可选的，默认是 1）	设置凹凸纹理的深度

返回纹理的索引号。

```
//读取凹凸映射纹理。
TexFactory.LoadBumpTexture("rockbump.bmp","RockBump",True,1)
```

LoadCubeTexture(文件名，名称，大小)
读取一个特殊的立方体纹理用于立方体环境反射。比如动态水面。

参数

文件名	磁盘上纹理档的路径
名称（可选的，默认是空）	纹理的实体名称
大小（可选的，默认是-1）	立方体纹理的大小。

返回纹理的索引号
说明：
因为是一个立方体纹理，所以要有相同的宽度和高度。

```
//读取一个立方体纹理，设置名字，用默认的宽度和高度
TexFactory.LoadCubeTexture "water_reflection.bmp","waterreflection",-1
```

GetFreeTextureMemory()
得到剩余显存的总数。

```
//检查如果剩余的显存在 32MB 以上，就继续。
FreeMem=TexFactory.GetFreeTextureMemory
If FreeMem<32000000 Then
    MessageBox("不能继续，因为剩下的显存不够！")
    MainQuit ()
End If
```

GetUsedTextureMemory()
返回已用显存的总数。可以用于调试程序用了多少显存。
返回已用显存的总数。

```
//记录已用的显存。
AddToLog("已用的显存是：" & TexFactory.GetUsedTextureMemory)
```

GetTextureInfo(纹理索引)
返回关于已用纹理的信息。信息包含在 TV_TEXTURE，有宽度、高度、格式、颜色键。

参数

纹理索引	纹理索引，可以用 GetTex 得到。

返回包含纹理信息的 TV_TEXTURE。
说明：
这有助于显示已读取的纹理信息。

```
//将纹理信息保存到 TexInfo 的变量中。
Dim TexInfo As TV_TEXTURE
TexInfo=TexFactory.GetTextureInfo(GetTex"WaterTexture"))
DrawText("纹理的宽度是：" & TexInfo.Width)
DrawText("纹理的高度是：" & TexInfo.Height)
DrawText("纹理的文件名是：" & TexInfo.FileName)
```

DeleteAll
删除所有纹理，通常用于从存储器中卸载那些纹理。

说明：
卸载对象在不同的编程语言中看到的是不同的。
//调用 DeleteAll 释放存储器，防止存储器泄露。
TexFactory.DeleteAll
Set TexFactory=Nothing

DeleteTexture(纹理索引)
删除指定索引号的纹理。

参数

纹理索引	纹理索引，用 GetTex 获得。

//删除掉被称为“MyTexture4”的纹理
TexFactory.DeleteTexture GetTex("MyTexture4")

SetTextureMode(色深)
设置纹理的色深。已存在的 16bits 和 32bits 的，256 色。

参数

色深	纹理用的颜色深度。

说明：
- 要在加载纹理档之前调用。
- 最好的过渡效果是 32bits。

//将所有的纹理设置成 32bits 色深模式。
TexFactory.SetTextureMode TV_TEXTUREMODE_32BITS

到此，TVTextureFactory 的函数完全就介绍完毕了。

TVMaterialFactory

CreateLightMaterial(红色，绿色，蓝色，Alpha，环绕光的程度，材质名称)
创建一个有直接光照的材质。

参数

红色	红色值。取值范围从 0-1
绿色	绿色值。取值范围从 0-1
蓝色	蓝色值。取值范围从 0-1
Alpha	Alpha 透明度。取值范围从 0-1
环绕光的程度（可选的，默认是 0.1）	环绕光颜色的百分比。
材质名称（可选的，默认是空的）	材质的名称，之后要用 GetMat 得到其索引。

返回创建材质的索引。
//创建一个发红光的材质，名字是“redmat”
MatFactory.CreateLightMaterial 1.0，0.0，0.0，1.0，0.3，"redmat"

CreateMaterial(名字)
创建一个空的材质。设置的名字用 GetMat 得到索引。

参数

名字（可选的，默认是空的）	实体的名字,设置的名字用GetMat得到索引。

返回值：返回创建材质的索引
//创建一个名为“DefaultMat”的空材质
MatFactory.CreateMaterial("DefaultMat")

CreateMaterialQuick(红色，绿色，蓝色，Alpha，材质名称)
快速创建材质。

参数

红色	红色值。取值范围从 0-1
绿色	绿色值。取值范围从 0-1
蓝色	蓝色值。取值范围从 0-1
Alpha	Alpha 透明度。取值范围从 0-1
材质名称	材质的名称用于 GetMat，以得到索引。

返回值：返回创建材质的索引
说明：
快速创建材质是很有用的。
//创建一个 RGBA(1.0,0.7,0.2,0.7),70%的透明度的材质。
MatFactory.CreateMaterialQuick 1.0,0.7,0.2,0.7,"QuickMat"

DeleteMaterial(纹理索引)
依据指定的索引删除相应的材质。

参数

纹理索引	材质的名称用于 GetMat，以得到索引。

//删除 DefaultMat 材质。
MatFactory.DeleteMaterial GetTex("DefaultMat")

DeleteAllMaterial
删除所有材质，当卸载引擎，避免任何可能的存储器泄漏时，使用它。

说明：
释放存储器，卸载对象在不同的编程语言中看到的是不同的。
//删除所有的材质，释放存储器。
MatFactory.DeleteAllMaterial
Set MatFactory=Nothing

SetAmbient(材质索引，红色，绿色，蓝色，Alpha)
在指定的材质上，应用环境光。通常在创建空的材质以后调用它。

参数

材质索引	材质索引，可以用 GetMat 得到索引。
红色	红色。取值范围从 0-1
绿色	绿色。取值范围从 0-1
蓝色	蓝色。取值范围从 0-1
Alpha	Alpha。取值范围从 0-1

说明：
使用 TV3D 提供的材质工具，MatED，来创建更有趣的，更好玩的和更强大的效果。

SetDiffuse(材质索引，红色，绿色，蓝色，Alpha)
在指定的材质上，应用漫反射光。通常在创建空的材质以后调用它。

参数

材质索引	材质索引，可以用 GetMat 得到索引。
红色	红色。取值范围从 0-1
绿色	绿色。取值范围从 0-1
蓝色	蓝色。取值范围从 0-1
Alpha	Alpha。取值范围从 0-1

说明：
使用 TV3D 提供的材质工具，MatED，来创建更有趣的，更好玩的和更强大的效果。

SetEmissive(材质索引，红色，绿色，蓝色，Alpha)
在指定的材质上，应用自发光。通常在创建空的材质以后调用它。

参数

材质索引	材质索引，可以用 GetMat 得到索引。
红色	红色。取值范围从 0-1
绿色	绿色。取值范围从 0-1
蓝色	蓝色。取值范围从 0-1
Alpha	Alpha。取值范围从 0-1

说明：
使用 TV3D 提供的材质工具，MatED，来创建更有趣的，更好玩的和更强大的效果。

SetPower(材质索引，镜面高光)
在指定的材质上，应用镜面高光。通常在创建空的材质以后调用它。

参数

材质索引	材质索引，可以用 GetMat 得到索引。
镜面高光	镜面反射的强度，在 0-100 之间取值。

说明：
使用 TV3D 提供的材质工具，MatED，来创建更有趣的，更好玩的和更强大的效果。

SetSpecular(材质索引，红色，绿色，蓝色，Alpha)
在指定的材质上，应用镜面反射。通常在创建空的材质以后调用它。

参数

材质索引	材质索引，可以用 GetMat 得到索引。
红色	红色。取值范围从 0-1
绿色	绿色。取值范围从 0-1
蓝色	蓝色。取值范围从 0-1
Alpha	Alpha。取值范围从 0-1

说明：
使用 TV3D 提供的材质工具，MatED，来创建更有趣的，更好玩的和更强大的效果。

```
//创建完空的材质后，命名为 WaterMat，在它上面应用。
MatFactory.CreateMaterial "WaterMat"
//设置不同的材质值
MatFactory.SetAmbient  GetMat("WaterMat"),0.2,0.2,0.2,0
MatFactory.SetDiffuse  GetMat("WaterMat"),1,1,1,1
MatFactory.SetEmissive GetMat("WaterMat"),0.2,0.2,0.2,0
MatFactory.SetPower    GetMat("WaterMat"),3
MatFactory.SetSpecular GetMat("WaterMat"),0.3,0.3,0.3,0
```

GetAmbient(材质索引号)
得到指定的材质的环境光，保存在 D3DCOLORVALUE。

参数

材质索引号	可以用 GetMat 得到索引

返回包含了红绿蓝 Alpha 值的 D3DCOLORVALUE。

```
//得到 WaterMat 材质的环境光。
Dim MyColorValue As D3DCOLORVALUE
MyColorValue = MatFactory.GetAmbient(GetMat("WaterMAT"))
DrawText("红色是：" & MyColorValue.R)
DrawText("绿色是：" & MyColorValue.G)
DrawText("蓝色是：" & MyColorValue.B)
DrawText("Alpha 是：" & MyColorValue.A)
```

GetDiffuse(材质索引号)
得到指定的材质的漫反射光，保存在 D3DCOLORVALUE。

参数

材质索引号	可以用 GetMat 得到索引

返回包含了红绿蓝 Alpha 值的 D3DCOLORVALUE。

```
//得到 WaterMat 材质的漫反射光。
Dim MyColorValue As D3DCOLORVALUE
MyColorValue = MatFactory.GetAmbient(GetMat("WaterMAT"))
DrawText("红色是：" & MyColorValue.R)
DrawText("绿色是：" & MyColorValue.G)
DrawText("蓝色是：" & MyColorValue.B)
DrawText("Alpha 是：" & MyColorValue.A)
```

GetEmissive**(材质索引号)**
得到指定的材质的自发光，保存在 D3DCOLORVALUE。

参数

材质索引号	可以用 GetMat 得到索引

返回包含了红绿蓝 Alpha 值的 D3DCOLORVALUE。

```
//得到 WaterMat 材质的漫反射光。
Dim MyColorValue As D3DCOLORVALUE
MyColorValue = MatFactory.GetAmbient(GetMat("WaterMAT"))
DrawText("红色是：" & MyColorValue.R)
DrawText("绿色是：" & MyColorValue.G)
DrawText("蓝色是：" & MyColorValue.B)
DrawText("Alpha 是：" & MyColorValue.A)
```

GetPower(材质索引号)
得到指定的材质的镜面高光，保存在 D3DCOLORVALUE。

参数

材质索引号	可以用 GetMat 得到索引

返回镜面反射。在 0-100 之间。
//从"CarMat"返回高光反射。
SpecularPower=MatFactory.GetPower("CarMat")

GetSpecular(材质索引号)
得到指定的材质的镜面反射，保存在 D3DCOLORVALUE。

参数

材质索引号	可以用 GetMat 得到索引

返回包含了红绿蓝 Alpha 值的 D3DCOLORVALUE。
//得到 WaterMat 材质的镜面反射。
Dim MyColorValue As D3DCOLORVALUE
MyColorValue = MatFactory.GetAmbient(GetMat("WaterMAT"))
DrawText("红色是：" & MyColorValue.R)
DrawText("绿色是：" & MyColorValue.G)
DrawText("蓝色是：" & MyColorValue.B)
DrawText("Alpha 是：" & MyColorValue.A)

GetMaterialName（材质索引号）
从索引号，返回索引名称

参数

材质索引号	材质索引。

返回索引名称的字符串。
说明：
它有些像 GetMat，但是过时了。
//根据索引号返回名称。
Dim MyMatName As String
MyMatName=MatFactory.GetMaterialName(6)

到此，TVMaterialFactory 的函数完全就介绍完毕了。

TVAtmosphere

Atmosphere_Render

渲染基于 Atmosphere 的天空盒子、天球、雨、雪、太阳和炫目的光。

说明：

- 你能够单独渲染天空盒子和天球，如果你有一些天空层，那么这是个好方法。
- 所以，你渲染的第一个类型的设置，然后第二个与其他材质及颜色等..
- 你能够用 Alpha 混合过渡。
- Atmosphere_Render 通常在 3D 渲染以后渲染，如果你绘制天空遇到了问题，你可以在 TV.Clear 以后用 Sky_Render 渲染天空，在 Atmosphere_Render 之前关掉天空盒子，以避免天空被渲染两次。

```
//在渲染循环中，渲染大气。
Do
    TV.Clear
    //渲染所有的 mesh
    Scene.RenderAllMeshes
    //在绘制 2D 物体之前渲染大气。
    Atmosphere.Atmosphere_Render
    //绘制 2D 物体。
    DrawAll2DStuff
    TV.RenderToScreen
Loop
```


Fog_Enable(是否打开)

打开关闭雾。

参数

是否打开	打开关闭雾。

说明：

- 雾不是一个 Scene 的组件
- 记住，在渲染中你能够反复开启关闭雾。比如渲染一个物体时，就能够关闭雾。
- 当渲染 2D 物体时，就会自动关闭雾。

```
//用标准参数开启雾。
Atmos.Fog_Enable True
Atmos.Fog_SetColor 0.5，0.7，0.9，1
Atmos.Fog_SetParameters 1,1000,0.005
```

Fog_SetColor(红色，绿色，蓝色，Alpha)
更改当前雾的颜色。

参数

红色	红色组件。取值范围从 0-1
绿色	绿色组件。取值范围从 0-1
蓝色	蓝色组件。取值范围从 0-1
Alpha	Alpha 组件。取值范围从 0-1，通常是 1.

//打开雾，颜色为蓝色。
Atmos.Fog_SetColor 0,1,1,1
Atmos.Fog_Enable True

Fog_SetParameters(开始的雾，终结的雾，密度)
给雾设置属性。

参数

开始的雾（可选的，默认是 1）	起始点雾的效果。
终结的雾（可选的，默认是 1000）	终结点雾的效果。
密度（可选的，默认是 0.01）	雾的密度。通常用于 EXP 或者 EXP2.

说明：
雾的类型影响雾的参数。规则是：
- 选择 LINEAR，必须设置开始的雾和结束的雾，这是线性插值，引擎将会在起始点和终结点之间进行差值运算。
- 当你用 EXP 或者 EXP2 时，你只需要设置密度，可以跳过起始点和终结点。

//在 0 到 1000 之间设置线性插值。
Atmos.Fog_SetType TV_FOG_LINEAR
Atmos.Fog_SetParameters 0，1000，0
//设置 EXP，密度是 0.005
Atmos.Fog_SetType TV_FOG_EXP
Atmos.Fog_SetParameters 0,0,0.005//起始点和终结点可以略过

Fog_SetType(雾的算法，雾的类型)
设置将被用于引擎的雾的类型

参数

雾的算法	计算雾的每一个像素的算法。
雾的类型	决定雾是顶点着色还是像素着色。

说明：
雾的类型影响雾的参数。规则是：
- 选择 LINEAR，必须设置开始的雾和结束的雾，这是线性插值，引擎将会在起始点和终结点之间进行差值运算。
- 当你用 EXP 或者 EXP2 时，你只需要设置密度，可以跳过起始点和终结点。

Fog_GetColor(红色，绿色，蓝色，Alpha)
返回雾的颜色。

参数

红色	充满了红色雾气的浮点值。取值范围从 0-1
绿色	充满了绿色雾气的浮点值。取值范围从 0-1
蓝色	充满了蓝色雾气的浮点值。取值范围从 0-1
Alpha	充满了 Alpha 半透明雾气的浮点值。取值范围从 0-1

```
//得到雾的颜色。
Dim Red As Single,Green As Single,Blue As Single,Alpha As Single
Atmos.Fog_GetColor Red,Green,Blue,Alpha
```

Fog_GetParameters(开始的雾，终结的雾，密度)
返回雾的当前的距离/密度。

参数

开始的雾	描述起始点的浮点值。
终结的雾	描述终点的浮点值。
密度	描述密度的浮点值。取值范围从 0-1，通常是 0.005.

```
//获得雾的参数
Dim StartFog,EndFog,Density As Single
Atmos.Fog_GetParameters StartFog,EndFog,Density
```

Fog_GetType(要返回的雾的算法，要返回的雾的类型)
返回雾的类型。

参数

要返回的雾的算法	雾的算法，linear，exp 或者 exp2
要返回的雾的类型	雾的类型，像素雾或者顶点雾。

说明：
雾的算法准确定义了雾的参数。

```
//获得雾的类型。
Dim Algo As CONST_TV_FOG
Dim TypeFog As CONST_TV_FOGTYPE
Atmos.Fog_GetType Algo,TypeFog
```

LensFlare_Enable(是否开启)
开启炫目的光效果。

参数

是否开启	如果开启，设为 True。

说明：

- 炫目的光的效果只能在有太阳的情况下起作用。它的位置会根据太阳的位置而变化。
- 如果你打开了炫目的光，Atmosphere 必须在 3D 渲染以后调用。

```
//当有太阳时，打开炫目的光。
If Hour >8 Then
    Atmosphere.Enable True
    Atmosphere.LensFlare_Enable True
End If
```

LensFlare_SetLensNumber(炫目的光数量)
炫目的光的数目。

参数

炫目的光数量	炫目的光数量

说明：
该函数必须在定义炫目的光的属性之前调用，否则你将会得到失败。

```
//调整炫目的光数量。
Atmos.LensFlare_SetLensNumber 4
Atmos.LensFlare_Enable True
```

LensFlare_SetLensParams(炫目的光的索引，纹理，大小，炫目的光的位置，普通颜色，高光颜色)
设置其中一个炫目的光的参数。

参数

炫目的光的索引	精灵的索引。
纹理	要被应用到炫目的光的纹理，通常是白热光纹理。
大小	大小。
炫目的光的位置	炫目的光的线的位置，被用于炫目的光的虚拟直线。
普通颜色	精灵的全局颜色，用 Alpha 值来做透明。(用 RGBA 宏)
高光颜色（可选的，默认是白色）	高光颜色，通常和普通颜色相同。

说明：你必须在之前设置炫目的光的数量，用 LensFlare_SetLensNumber。

Rain_Enable(是否开启)
开启雨、雪粒子系统。

参数

是否开启	开启雨、雪粒子系统。

说明：

- 下雨特效应用了当前摄影机的位置和朝向来优化粒子数目。如果看到的结果不是足够好，试试在 Rain_Init 增加雨滴的距离和半径。
- 要想开启雨的功能，你必须要在这之前调用 Rain_Init 初始化。

```
//初始化雨
Atmos.Rain_Init 100，GetTex(“Texture"), 3，0，0，0，50
//开启雨
Atmos.Rain_Enable True
```

Rain_Init(密度，纹理，掉落速度，风向 X，风向 Z，随机值，雨的目标，粒子大小，半径，生成速度，移动速度)
初始化雨。

参数

密度	雨滴的密度，描述了粒子数量。 粒子数量=100×密度。
纹理	将要应用在每个雨滴上的纹理。
掉落速度（可选的，默认是-1）	粒子掉落的速度，考虑到了重力。
风向 X（可选的，默认是 0）	X 轴的速度，取值范围从-2 到 2
风向 Z（可选的，默认是 0）	Z 轴的速度，取值范围从-2 到 2
随机值（可选的，默认是 0.1）	将要被应用到速度的随机值，增加雨滴掉落的真实效果。
雨的目标（可选的，默认是 20）	略
粒子大小（可选的，默认是 16）	粒子大小。
半径（可选的，默认是 40）	略
生成速度（可选的，默认是 20）	生成速度。描述在两个粒子之间的内部时钟，用毫秒表示。
移动速度（可选的，默认是 0.08）	雨滴速度，最好就用默认值。

说明：
当初始化雨后，就应该用 Rain_Enable 初始化，最后用 Atmosphere_Render 进行渲染。

```
//初始化雨
Atmos.Rain_Init 100，GetTex(“Texture"), 3，0，0，0，50
//打开雨
Atmos.Enable True
```

Sun_Enable(是否开启)
开启太阳广告牌。

参数

是否开启	开启太阳广告牌

说明：
如果你想要正确地渲染太阳，就应该在绘制 3D 之后调用 Atmosphere_Render。
//开启太阳，应用太阳的纹理。
Atmos.Sun_Enable True
Atmos.Sun_SetPosition Vector3（-500，500，0）
Atmos.Sun_SetTexture GetTex(“sun”)

Sun_SetBillboardSize（太阳的大小）
更改太阳大小。

参数

太阳的大小	太阳的大小

说明：
默认的太阳大小为 1.
//设置太阳的大小为 4.
Atmos.Sun_SetBillboardSize 4

Sun_SetMaterial（材质索引）
设置需要应用到太阳上的材质。

参数

材质索引	将要应用到太阳上的材质索引。

说明：
默认的材质是亮红色的。
你可以控制 Alpha 过程模拟 Alpha 混合。
//创建个材质，并应用到太阳上。
NewMaterial=MatFactory.CreateMaterial（）
MatFactory.SetAmbient NewMaterial, 1, 1, 0, 1
MatFactory.SetDiffuse NewMaterial, 1, 1, 0, 1
Atmos.Sun_SetMaterial GetMat（NewMaterial）

Sun_SetTexture(纹理索引)
设置将要应用到太阳上的纹理。

参数

纹理索引	将要应用到太阳上的纹理的索引。

说明：
之前，必须用Sun_Enable开启太阳。
//设置太阳的纹理。
Atmos.Sun_SetTexture GetTex("sun")

Sun_SetPosition(X位置，Y位置，Z位置)
设置太阳的位置和摄影机的位置相关联。

参数

X位置	和摄影机的位置相关联的X坐标。
Y位置	和摄影机的位置相关联的Y坐标。
Z位置	和摄影机的位置相关联的Z坐标。

说明：
你可以根据时间修改太阳的位置
//开启太阳，更改位置，应用纹理。
Atmos.Sun_Enable True
Atmos.Sun_SetPosition Vector3(-500, 500, 0)
Atmos.Sun_SetTexture GetTex(“sun")

SkyBox_Enable（是否开启，是否自动渲染）
开启天空，并且自动渲染。

参数

是否开启	开启、关闭天空盒子的渲染
是否自动渲染（可选的，默认是True）	

SkyBox_Render
渲染天空盒子。

说明：
单独的天空盒子渲染能够用于多图层的天空，或者是消除Alpha混合错误。

```
//以下是一段渲染天空的例子。
Do
    TV.Clear
    Atmos.SkyBox_Enable True
    Atmos.SkyBox_Render
    Atmos.SkyBox_Enable False
    DrawMeshObjects
    Atmos.Atmosphere_Render
    Draw2D
    TV.RenderToScreen
Loop
```


SkyBox_SetColor(红色，绿色，蓝色，Alpha)
调整天空的颜色。

参数

红色	天空颜色红颜色。取值范围从0-1
绿色	天空颜色绿颜色。取值范围从0-1
蓝色	天空颜色蓝颜色。取值范围从0-1
Alpha	Alpha透明组件，取值范围0-1，0看不见，1不透明。

说明：
你可以在天空盒子之间减低Alpha值。

```
//Alpha降低到0.5，以便于天空盒子有些透明。
Atmos.SkyBox_SetColor 1，1，1，0.5
//或者你可以把天空做成浅红色，好像日出那样。
Atmos.SkyBox_SetColor 1，0.7，0.6，1
```

SkyBox_SetDistance（距离）
在天空和摄影机之间设置距离。

参数

距离	将要被用在天空和摄影机之间的距离，通常不需要麻烦，1000就是一个很好的值。

说明：
就像在例子中看到的，距离值可以很小，为什么？因为当绘制天空盒子时，Z缓冲是关闭的，这意味着在Z缓冲里是没有信息的，所以像一个新的“背景”，但是不会影响其他的渲染。

```
//设置天空盒子距离为2.
Atmos.SkyBox_SetDistance 2
```

SkyBox_SetRotation（X轴半径，Y轴半径，Z轴半径）
设置天空盒子的旋转角度。

参数

X轴半径	绕着X轴旋转的角度，用°或弧度。
Y轴半径	绕着Y轴旋转的角度，用°或弧度。
Z轴半径	绕着Z轴旋转的角度，用°或弧度。

说明：
该方法可以用于通用角度系统，这意味你可以用°或弧度，用TVEngine.SetAngleSystem来选择角度系统。

```
//旋转天空盒子。
Angle=Angle+0.001*TV.Elapsed
TV.SetAngleSystem TV_ANGLE_DEGREES
Atmos.SkyBox_SetRotation 0，Angle，0
```

SkyBox_SetTexture（前面，后面，左边，右边，上面，下面）
设置天空盒子的6个纹理，前面，后面，左边，右边，上面，下面。

参数

前面	前方纹理的索引号。
后面	后方纹理的索引号。
左边	左边纹理的索引号。
右边	右边纹理的索引号。
上面	上面纹理的索引号。
下面	下面纹理的索引号。

说明：
必须严格执行纹理的顺序，否则你会看到不可思议的东西。

```
//给天空盒子应用上不同的纹理。
//必须在纹理工厂中已经读取了这些纹理。
Atmos.SkyBox_SetTexture
GetTex("front"),GetTex("Back"),GetTex("Left"),GetTex("Right"),GetTex("Top"),Get
Tex("Bottom")
```

SkySphere_Enable(是否开启)
打开天空球，自动渲染。

参数

是否开启	开启天空球，自动渲染，设为True。

```
//关掉天空球。
Atmos.SkySphere_Enable False
```

SkySphere_Render
在场景上，单独渲染天空球。

```
//这是一个Alpha混合的演示，所以必须在所有的3D渲染和太阳、炫目的光渲染之前调用。
Do
    TV.Clear
    Atmos.SkySphere_Enable True
    Atmos.SkySphere_Render
    Atmos.SkySphere_Enable False
    DrawMeshObjects
    Atmos.Atmosphere_Render
    Draw2D
    TV.RenderToScreen
Loop
```

SkySphere_SetColor(红色，绿色，蓝色，Alpha)
更改天空球的颜色。

参数

红色	应用于天空球的红色组件，取值范围从0-1
绿色	应用于天空球的绿色组件，取值范围从0-1
蓝色	应用于天空球的蓝色组件，取值范围从0-1
Alpha	应用于天空球的Alpha组件，取值范围从0-1

说明：
在不同的天空图层中，改变能够用于透明的Alpha值。
//修改天空球的颜色为红色。
Atmos.SkySphere_SetColor 1,0,0,1
//修改天空50%的透明度
Atmos.SkySphere_SetColor 1,1,1,0.5

SkySphere_SetPolyCount(切片数量)
改变天空球的画质。

参数

切片数量	天空球的切片数量，最小：6，最大30.

SkySphere_Radius(半径)
改变天空球的半径。

参数

半径	天空球的半径，通常远平面距离是个很好的值。

说明：
因为引擎的Z缓冲是只读的，改变半径不是很有用的，但是如果你的世界非常大，你就应该增加远平面的值，否则你会看到不可思议的闪烁的东西。

SkySphere_SetRotation(X轴的弧度，Y轴的弧度，Z轴的弧度)
设置天空球旋转的角度。

参数

X轴的弧度	绕X轴旋转的旋转角度，（用°或者弧度）
Y轴的弧度	绕Y轴旋转的旋转角度，（用°或者弧度）
Z轴的弧度	绕Z轴旋转的旋转角度，（用°或者弧度）

说明：
- 该函数在构造云上能有很好的表现。
- 该函数用通用角度系统，这意味着你可以用°或者弧度，使用TVEngine.SetAngleSystem选择。

```
//天空球绕Y轴旋转。
Y=Y+TV.TimeElapsed*0.001
Atmos.SkySphere_SetRotation 0,Y,0
Atmos.SkySphere_Render
```

SkySphere_SetTexture(纹理)

更改天空球的纹理。

参数

纹理	要被应用到天空球纹理的索引号。

```
//应用纹理到天空球。
Atmos.SkySphere_SetTexture GetTex("SkySphere")
```

Unload

卸载所有对象清空存储器。

说明：

如果你不需要任何大气服务了，比如从室外到室内。

```
//卸载大气。
Atmos.Unload
Set Atmos=Nothing
```

到此，TVAtmosphere的函数就介绍完毕了。

TVLightEngine

Count

返回创建的灯光的总和。

```
//打开所有灯光。
For i=1 To LightEngine.Count
    LightEngine.EnableLight True
Next i
```

EnableLight(灯光的索引号)

打开/关闭创建的灯光。

参数

灯光的索引号	灯光索引。可以用GetLightByName得到。

说明：

记住3D显卡的光源极限是很重要的，所以管理好你的光源，正确开启它们。

```
//关掉sunlight。
LightEngine.EnableLight(sunlight)=False
```

DeleteLight(灯光索引)

关闭或者删除指定的光源。

参数

灯光索引	灯光索引。可以用GetLightByName得到。

说明：

```
//现在是晚上，关掉太阳光。
If Time>18 And SunNotDeleted=True Then
    LightEngine.DeleteLight SunLight
    SunNotDeleted=False
End If
```

DeleteAllLights

关掉和删除点所有光源。

说明：

如果你要退出一个场景，就要用到这个方法。

```
//删除所有光源，到其他场景。
Sub ChangeScene()
    LightEngine.DeleteAllLights
    LoadNewScene
End Sub
```

GetLightByName(灯光名字)

返回指定光的结构。

参数

灯光名字	灯光名字

返回光结构的D3DLIGHT8。

说明：

该方法能够被用于获得光的信息。

```
//得到“sun”的信息，更新光的位置。
Dim Info As D3DLIGHT8
Info=LightEngine.GetLightByName("sun")
Info.Position=Vector3(100,200,300)
LightEngine.UpdateLight LightEngine.GetLightIndexByName,Info
```

GetLightIndexByName(灯光名字)

从灯光名字获得灯光索引。

参数

灯光名字	要得到灯光索引的灯光名字。

返回一个灯光索引值的整数。

```
//得到“sun”的信息，更新光的位置。
Dim Info As D3DLIGHT8
Info=LightEngine.GetLightByName("sun")
Info.Position=Vector3(100,200,300)
LightEngine.UpdateLight LightEngine.GetLightIndexByName,Info
```

CreateLight(灯光信息，灯光名称，是否用默认属性)

用指定的灯光信息（Direct3D的灯光结构）创建灯光。

参数

灯光信息	灯光信息（Direct3D的灯光结构）
灯光名称（可选的，默认是空的）	可选的名称。
是否用默认属性（可选的，默认是False）	如果是True，引擎就会忽略灯光信息，创建一个默认的方向光，通用于往模型上着色。

返回描述光的光索引。

说明：

记住3D显卡的光源极限是很重要的，所以管理好你的光源，正确开启它们。

```
//创建一个简单的方向光。
Dim LightEngine As New TVLightEngine
Dim Light As D3DLIGHT8
Dim DirectionalLight As Integer
Light.Type=D3DLIGHT_DIRECTIONAL
Light.Diffuse=DXColor(1,1,1,1)
```

```
Light.Ambient=DXColor(1,1,1,1)
DirectionalLight=LightEngine.CreateLight(Light,"light1")
```

CreateQuickDirectionalLight(方向，红色，绿色，蓝色，光线名字，镜面反射因子)
快速建立方向光（也可以说是平行光）。

参数

方向	描述方向的矢量，该矢量不需要被法线化。
红色	红色光组件，取值范围从0-1
绿色	绿色光组件，取值范围从0-1
蓝色	蓝色光组件，取值范围从0-1
光线名字（可选的，默认是空的）	光线名字
镜面反射因子（可选的，默认是0）	模型上的反光的能力。取值范围从0-1

返回描述光的索引。
说明：

- 在光的种类中，方向光是用的最多的。通常它很容易计算能得到最好的结果。它能模拟未知位置的点光，这光线类似于平行发光的点光（比如太阳光、月光）。
- 记住3D显卡的光源极限是很重要的，所以管理好你的光源，正确开启它们。

```
//创建类似于太阳的光。
LightEngine.CreateQuickDirectionalLight(Vector3(-1, -1, -1), 1, 1, 1,“sun”, 0.3)
```

CreateQuickPointLight（位置矢量，红色，绿色，蓝色，范围，光线名字，镜面反射因子）
快速创建点光。也被称为泛光灯，因为光线围绕在每一个方向的光点。

参数

位置矢量	灯光的位置矢量。
红色	灯光颜色的红色组件，取值范围从0-1.你可以用大于1的值增强光感。
绿色	灯光颜色的绿色组件，取值范围从0-1.你可以用大于1的值增强光感。
蓝色	灯光颜色的蓝色组件，取值范围从0-1.你可以用大于1的值增强光感。
范围	灯光没有影响的距离以外。
光线名字（可选的，默认是空）	表述灯光特性的名字。
镜面反射因子（可选的，默认是0）	模型上的反光的能力。取值范围从0-1。0为没有镜面反射效果，1为白色。

返回描述光的索引。
说明：

- 点光是有颜色和位置的，但是没有方向。它们在所有位置都发出相等的光，就像下面所说的。
- 比如一个电灯泡就是个很好的点光源的例子，点光是被光线的衰减度和范围影响，在顶点上照亮一个模型。在灯光中，TV3D用点光的位置和顶点坐标“照”出灯光的方向矢量，游历了光的距离，两者都使用，随着顶点法线，计算光线照亮表面的值。
- 记住3D显卡的光源极限是很重要的，所以管理好你的光源，正确开启它们。

```
//在火把上，创建一个点光源。
Dim TorchLight As Integer
LightEngine.CreateQuickPointLight Torch.GetPosition,1,0.8,0.6
```

SetLightAttenuation(灯光索引，衰减1，衰减2，衰减3)

更改灯光的衰减度。

参数

灯光索引	灯光索引。
衰减1（可选的，默认是0.1）	线性衰减因子。可无限取值。
衰减2（可选的，默认是0）	平方衰减因子，可无限取值。
衰减3（可选的，默认是0）	指数衰减因子，可无限取值。

说明：

- *衰减控制光线减弱的最大距离。D3DLIGHT8结构成员描述了光线减弱，这些成员包含了从0到无限的浮点数。控制光线的衰减。有些应用程序把Attenuation1设为1.0，其他的设为0.0，光的结果减弱为1/D，D是光源到顶点的距离。最大的光线强度是在源物体上，减弱为1/融合度。典型的，一个应用程序设置Attenuation0为0.0，Attenuation1为一个常数，Attenuation2为0.0.*
- *你可以组合衰减度以便得到复杂的衰减效果。或者，你可以把它们设置为超过范围的值，创建一个奇怪的效果。*

```
//放置一个默认的衰减因子
LightEngine.SetLightAttenuation LightID, 1, 0, 0
```

SetLightColor(灯光索引，红色，绿色，蓝色)

改变扩散光、反射光的颜色。

灯光索引	要改变的灯光索引。
红色	红色组件，取值范围从0-1，有时你可以用比1大的值使光线更强烈。
绿色	绿色组件，取值范围从0-1，有时你可以用比1大的值使光线更强烈。
蓝色	蓝色组件，取值范围从0-1，有时你可以用比1大的值使光线更强烈。

```
//改变光的颜色，从红色到蓝色。
Do
    TV.Clear
    RenderScene
    LightMod=(LightMod+1) Mod 2
    If LightMod=0 Then
        LightEngine.SetLightColor LightID,1.0,0,0
    End If
    If LightMod=1 Then
        LightEngine.SetLightColor LightID,0,0,1.0
    End If
    TV.RenderToScreen
Loop
```

SetLightDirection(光源索引，方向矢量)
改变方向光或者聚光灯的方向。

参数

光源索引	灯光索引
方向矢量	描述灯光方向的矢量。该矢量需要被法线化，不应该是0.

说明：
调用这个函数之后就能影响效果。
//就像现实一样，使阳光旋转。
```
TimeOfDay=TimeOfDay+TV.AccurateTimeElapsed*0.02
Direction=Vector3(Cos(TimeOfDay),Sin(TimeOfDay),0)
LightEngine.SetLightDirection SunIndex,Direction
```

SetLightPosition(灯光索引，位置矢量)
更改聚光灯或者点光灯的位置。

参数

灯光索引	灯光索引。
位置矢量	描述新的位置的矢量。

说明：
调用这个函数之后就能影响效果。
//改变光线1的位置为（100，200，300）
```
LightEngine.SetLightPosition 1,Vector3(100,200,300)
```

SetLightRange(光线索引，光线范围)
改变指定的光影响的范围。

参数

光线索引	已存在的光线索引。
光线范围	指定光线新的范围。

说明：
- 调用这个函数之后就能影响效果。
- 光影响的范围决定了光源和不再接收光线物体的距离。该范围成员包含了灯光的最大范围的浮点值，方向光没有范围的概念。

//改变光线半径为1000
```
LightEngine.SetLightRange LightID，1000
```

UpdateLight（光源索引，更新信息）

用新的光源信息更新指定的光。

参数

光源索引	已存在的光源索引。
更新信息	

说明：

- 该方法能够实时地动态更新光。
- 调用这个函数之后就能影响效果。

```
//得到“sun”光的信息，更新它的位置。
Dim Info As D3DLIGHT8
Info=LightEngine.GetLightByName("sun")
Info.Position=Vector3(100,200,300)
LightEngine.UpdateLight LightEngine.GetLightIndexByName,Info
```

到此，TVLightEngine的函数就介绍完毕了。

TVBSPTree

DetectCollision(起始点，终结点，是否可视)

在有BSP的场景上，进行碰撞测试。

参数

起始点	碰撞测试的起点
终结点	碰撞测试的终点
是否可视（可选的，默认是True）	如果引擎必须测试可视平面，或者在BSP的所有面上，设为True。

如果物体相交到碰撞测试的射线上，设为True。

说明：

当你必须做的不匹配碰撞时，碰撞测试应该被使用。

尽管这样，它可以用来测试两点之间的BSP树的能见度。

AdvancedSphereCollision(球的原位置，球的新位置，球的半径)

在BSP上做高级球碰撞，返回有关碰撞的信息。

参数

球的原位置	做碰撞测试的球的原位置，通常都是当前游戏者的位置。
球的新位置	做碰撞测试的球的新位置，通常都是新的游戏者的位置。
球的半径	球的半径。通常都是绑定游戏者的球半径。

返回有关碰撞信息的TVCollisionResult。

说明：

AdvancedSphereCollision是最好的返回碰撞信息的方法。

DetectSphereCollision(位置，球半径)

在BSP上做简单球碰撞，返回有关碰撞的信息。

参数

位置	测试球的位置。
球半径（可选的，默认是1）	测试球的半径。（通常是绑定球的半径）
保留（可选的，默认是0）	

如果碰撞发生了返回True，否则为False。

说明：

你能够用AdvancedSphereCollision得到更多的碰撞信息。

SlidingCollision(起始点，终结点，返回的位置，半径)

滑动碰撞。

OpenPackWAD(纹理文件名，压缩包数量)

打开一个压缩的纹理档（HL类型的档），以便于纹理被引擎读取。

参数

纹理文件名	要读取的WAD档路径和文件名。
压缩包数量（可选的，默认为0）	TrueVision 6.1已经支持了。

说明：

可以用Wally来创建WAD档。(www.telefragged.com/wally)

ClosePackWAD

关闭已关闭的WAD档。

说明：

在BSP读取之后你应该关闭WAD档，否则可能引起存储器泄露！

Load（文件名，是否读取LightMap，是否读取纹理，纹理数量，纹理路径，是否创建动态光源，是否创建Mipmap）

读取BSP（HL的BSP和Quake3的BSP3），其他格式的BSP不会被读取。

参数

文件名	BSP的文件名。
是否读取LightMap（可选的，默认是True）	如果引擎必须要读取光源信息，设为True。
是否读取纹理（可选的，默认是True）	如果引擎必须读取BSP纹理信息，设为True。
纹理数量（可选的，默认是False）	如果你知道纹理名称和数量（比如1.bmp、2.bmp……），可以设为True。
纹理路径（可选的，默认是空）	查找纹理的路径。
是否创建动态光源（可选的，默认是False）	如果引擎在BSP场景中必须创建动态光源，设为True，以便正确的照射到模型上。
是否创建Mipmap（可选的，默认是True）	如果当引擎读取纹理时，必须创建mipmap设为True，通常设为True能得到最好的结果。

说明：

- 如果你想要从一个WAD纹理上，在OpenPackWAD和ClosePackWAD之间使用。
- 注意：能够在其他目录中或者从WAD或者PAK档中读取纹理。

```
//在读取BSP之前读取压缩档。
BSP.OpenPackWAD “TexturePak.Wad”
//用默认的参数读取“scene.bsp”
BSP.Load "Scene.Bsp",true,true,false,"",true,false
//关闭WAD
BSP.ClosePackWAD
```

DeleteAll
从已经读取好的BSP中，删除掉所有东西，并重置。

说明：
该方法在读取新的场景之前清理现有的场景。
//从BSP中删除所有的东西。
BSP.DeleteAll
//读取一个新的关卡
BSP.Load "newlevel.bsp"

GetAltitudeFromPos(X位置，Y位置，Z位置，是否使用摄影机位置)
从一个位置返回高度。比如从一个指定点返回阶梯的高度。

参数

X位置	3D点的X坐标。
Y位置	3D点的Y坐标。
Z位置	3D点的Z坐标。
是否使用摄影机位置	跳过X、Y、Z，直接使用摄影机位置代替。

返回离指定点最近的阶梯高度。

GetPosition
得到在BSP场景的位置。

返回描述BSP位置的3D矢量。
引擎默认的在原点开始读取。

GetStartPlayerPosition
返回游戏者起始的点。

返回描述游戏者起始点的3D矢量。
说明：
如果有好几个游戏参与者的实体，用第一个作为起始点。该位置自动转换为世界坐标。

LoadShaderDirectory(路径)
在指定的目录中读取Shader档。

参数

路径	Shader档的路径。

说明：
当读取BSP时，引擎将要转换该脚本读取档。当然，这意味着在读取地图之前调用这个函数。

```
// 设置默认的Shader目录
BSP.LoadShaderDirectory App.path & "\shaders\"
//设置法线精度读取地图。
BSP.SetCurvePrecision TV_CURVE_NORMAL
BSP.Load App.Path & "\q3map.bsp"
// standard loop
Do
    TV.Clear
    // 渲染BSP地图
    BSP.Render
    // 绘制GUI
    GUI.Render
    TV.RenderToScreen
Loop
```

Render

渲染BSP场景。

说明：

BSP应该在渲染室外环境以后，在2D之前渲染。

```
//设置法线精度读取地图。
BSP.SetCurvePrecision TV_CURVE_NORMAL
BSP.Load App.Path & "q3map.bsp"
//游戏循环
Do
    TV.Clear
    //渲染BSP
    BSP.Render
    //绘制GUI
    GUI.Render
    TV.RenderToScreen
Loop
```

SetPosition（X位置，Y位置，Z位置）

设置在世界坐标中的BSP位置，比如把一个BSP放在Landscape上，或者把两个BSP连接在一起 。

参数

X位置	BSP新位置的X坐标。
Y位置	BSP新位置的Y坐标。
Z位置	BSP新位置的Z坐标。

SetCurvePrecision(曲线精度)

定义Quake3贝赛尔曲线的精度，必须在读取地图以前。

参数

曲线精度	定义Quake3贝赛尔曲线的精度

说明：

- 调用SetCurvePrecision和Quake3的选项面板中的细节选项效果是一样的。
- 必须在BSP.Load之前调用。
- 高精度的Curve需要花费更多的时间渲染，因为有更多的三角形。

```
//设置曲线的法线精度，读取地图。
BSP.SetCurvePrecision TV_CURVE_NORMAL
BSP.Load App.Path & "q3map.bsp"
// 渲染。
Do
TV.Clear
BSP.Render
TV.RenderToScreen
Loop
```

StayOnFloor（位置，最小的，最大的，半径）

做少量的球测试，在指定的位置检测高度。

参数

位置	要测试的游戏者的位置。
最小的（可选的，默认是-5）	测试游戏者的最小距离。
最大的（可选的，默认是100）	测试游戏者的最大距离。
半径（可选的，默认是5）	用于碰撞检测的球半径（通常是绑定游戏者球的半径）。

返回一个新高度的单精度浮点数。

说明：

游戏者停留在地板上，该函数工作的很好。如果你设置了最小距离和最大距离，它也能用于跳跃或者蹲下。

到此，TVBSPTree的函数就介绍完毕了。

TVAI

AddNode(位置)
增加一个AI节点到3D世界。

参数

位置	3D世界的矢量。

说明：
可能你会增加许多节点，因此推荐用Vector3转换成D3DVECTOR，这样会使程序变得整洁。

```
//增加三个节点。
AI.AddNode Vector3(10,5,7)
AI.AddNode Vector3(15,5,9)
AI.AddNode Vector3(20,5,7)
```

GetNode(索引)
返回节点索引的位置。

参数

索引	节点索引。

返回节点位置的D3D矢量。

```
//保存的序列号是4的节点位置。
MyNode=AI.GetNode（4）
```

CreateAIGraph
创建了节点并且确定了位置后，就可以调用CreateAIGraph方法在节点之间寻路。

```
//增加三个节点。
AI.AddNode Vector3(10,5,7)
AI.AddNode Vector3(15,5,9)
AI.AddNode Vector3(20,5,7)
//创建节点网络好了以后，调用CreateAIGraph方法以便你可以在节点之间寻路。
AI.CreateAIGraph
```

ExportGraph(文件名)
将AI图表保存到一个档中去。如果以后再用，直接调用就可以了，而不用重新创建。

参数

文件名	要保存的AI图表的文件名。

说明：
必须调用了CreateAIGraph之后调用ExportGraph，才能正确导出AI图表。

```
//增加三个节点。
AI.AddNode Vector3(10,5,7)
AI.AddNode Vector3(15,5,9)
AI.AddNode Vector3(20,5,7)
//创建节点网络好了以后，调用CreateAIGraph方法以便你可以在节点之间寻路。
AI.CreateAIGraph
//导出AI图表成为一个档。
AI.ExportGraph "desert.Ai"
```

ImportGraph(文件名)

从磁盘上导入AI图表。

参数

文件名	磁盘上AI图表的文件名。

```
//读取"desert.ai"
AI.ImportGraph "desert.ai"
```

FindPath(起始点，终结点，保存的路径)

通过节点网络寻路，指定起始点和终结点，返回路径信息。

参数

起始点	起始点矢量。
终结点	终结点矢量。
保存的路径	返回路径信息。

返回值：	● 如果路径被找到，返回True。
	● 返回路径信息。

```
//通过节点网络寻路。
Dim SavedPath as TVPath
if AI.FindPath(StartMesh.GetPosition,EndMesh.GetPosition,SavedPath) then
    DrawText("Found Path")
End if
```

ImportFromBSP(BSP)

从BSP中导入节点。

参数

BSP	用TVBSPTree的Load读取的BSP地图。

说明：

在用ImportFromBSP之前，你必须预读取BSP地图。

```
//从反恐精英的Italy地图中导入节点。
AI.ImportFromBSP CS_Italy
```

SetFindPathParameters(寻找半径，最大改变高度)
两个不同的节点之间，最大半径和海拔高度。

参数

寻找半径	两个节点之间的最大半径。
最大改变高度	两个节点之间的最大海拔高度。

说明：
必须在CreateAIGraph和ExportGraph之前调用。

```
//放置节点的子程序。
NodesAdding
//设置两点之间的最大半径和海拔高度。
AI.SetFindPathParameters 100,150
AI.CreateAIGraph
//导出AI图表。
AI.ExportGraph "desert.ai"
```

到此，TVAI的函数就介绍完毕了。

TVPath

AddPathNode(节点位置)
增加一个路径节点为3D矢量。

节点位置	D3DVECTOR3类型的节点位置。

返回节点索引。

```
//增加3个不同位置的节点。
TVPath.AddPathNode(Vector3(14,2,42))
TVPath.AddPathNode(Vector3(35,124,64))
TVPath.AddPathNode(Vector3(32,34,12))
```

GetNode（节点索引）
返回节点的位置。

参数

节点索引	节点索引号。

返回D3DVECTOR8类型的索引号。

```
//返回6号节点索引。
NodePosition=TVPath.GetNode(6)
```


GetNodeCount()
返回节点总数。

返回节点总数的整数。

```
Dim NodePosition As D3DVECTOR
//循环得到所有节点的位置。
For i=0 to TVPath.GetNodeCount-1
    NodePosition=TVPath.GetNode(i)
    debug.Print NodePosition.X,NodePosition.Y,NodePosition.Z
Next i
```

CameraSpline(节点时间，改变值)
摄影机围绕着样条曲线。做个演示是很好的。

参数

节点时间	样条曲线用的时间（节点），取值范围从1到GetNodeCount()。
改变值	是不同的路径时间，在摄影机的位置和视野之间。

说明：
摄影机的样条曲线函数使用了两次GetSplinePoint，返回：

- 摄影机的位置
- 摄影机的视野。

```
//增加3个不同位置的节点
TVPath.AddPathNode(Vector3(14,2,42))
TVPath.AddPathNode(Vector3(35,124,64))
TVPath.AddPathNode(Vector3(32,34,12))
//游戏循环。
Do
    Doevents
    TV.Clear
    RenderAll()
    TV.RenderToScreen
    Time = Time + TV.AccurateTimeElapsed * 0.002
    TVpath.CameraSpline Time, 1
Loop
```

GetSplinePoint(步数)
得到路径的样条曲线点。

参数

步数	步数是单精度的。节点1就是1，节点2就是2

返回D3DVECTOR的位置。

```
//在第20步的地方，得到样条曲线点。
PointPosition=TVPath.GetSplinePoint(20)
```

MoveNode(节点索引，新位置矢量)
将节点移动到新位置。

参数

节点索引	节点索引，当你读取一个新的节点时，你就能得到它。
新位置矢量	节点的新位置。

```
//移动索引为6的节点。
TVPath.MoveNode 6,Vector3(12,56,32)
```

ResetPath
重置路径删除所有节点。

```
//重置节点。
TVPath.ResetPath
```

SetPathType(路径类型)
设置路径的类型。

参数

路径类型	路径类型是这样的，CONST_TV_PATHTYPE，Demo最好的样条曲线。

//设置路径类型为样条曲线。
TVPath.SetPathType TV_PATH_SPLINE

到此，TVPath的函数就介绍完毕了。

TVScreen2DImmediate

ACTION_Begin2D
开启一个为2D渲染的区域。

说明：
- 这是一个优化2D渲染的好方法。
- 在ACTION_Begin2D和ACTION_End2D之间渲染2D画面，当调用ACTION_End2D时，就开始渲染2D了。
- 应该在3D渲染时，渲染2D。

```
Scr2D.ACTION_Begin2D
//绘制、渲染2D
Scr2D.ACTION_End2D
```

ACTION_End2D
结束2D渲染区域。

说明：
- 这是一个优化2D渲染的好方法。
- 在ACTION_Begin2D和ACTION_End2D之间渲染2D画面，当调用ACTION_End2D时，就开始渲染2D了。
- 应该在3D渲染时，渲染2D。

```
Scr2D.ACTION_Begin2D
//绘制、渲染2D
Scr2D.ACTION_End2D
```

DRAW_Background
手动绘制背景图。

说明：
- 你不需要在ACTION_Begin2D和ACTION_End2D之间调用这个。

```
//绘制当前背景
Do
    TV.Clear
    DoEvents
    Scr2D.DRAW_Backgound
    Scene.RenderAllMeshes true
    TV.RenderToScreen
Loop
```

DELETE_Background
删除已载入的背景

说明：
这可以清除背景占用的存储器，所以如果你不用背景了，调用它是很有用的。
//将背景从存储器中删除，结束程序。
Scr2D.DELETE_Background
End

DRAW_Box（X1,Y1,X2,Y2，颜色1，颜色2，颜色3，颜色4）
在荧幕上画矩形。

参数

X1	左上角点的X坐标。
Y1	左上角点的Y坐标。
X2	右下角点的X坐标。
Y2	右下角点的Y坐标。
颜色1	矩形的主颜色，或者是左上角的颜色。
颜色2	第二个颜色。
颜色3	第三个颜色。
颜色4	第四个颜色。

说明：
- 你可以省略后三个（颜色2、颜色3、颜色4）颜色，引擎会用主颜色填充整个矩形。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

//在点（10，10)、(50，50）绘制一个用红色填充的矩形。
Scr2D.DRAW_Box
10,10,50,50,RGBA(1,0,0,1),RGBA(1,0,0,1),RGBA(1,0,0,1),RGBA(1,0,0,1)
//或者是……
Scr2D.DRAW_Box 10,10,50,50,RGBA(1,0,0,1)

DRAW_Circle(圆心X，圆心Y，半径，步长，边框颜色)
在荧幕上划圆圈。
参数

圆心X	要画的圆心的X坐标。
圆心Y	要画的圆心的Y坐标。
半径	圆圈的半径。
步长（可选的，默认是12）	圆圈每一段的数量，你用Step越大，圆圈就越精确。
边框颜色（可选的，默认是白色）	圆圈的颜色。(用RGBA宏获取）

说明：
不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。
//在圆心为200、300的位置上画一个半径50蓝色的圆圈。
Scr2D.DRAW_Circle 200,300,50,20,RGBA(0,0,1,1)

DRAW_FilledBox（X1,Y1,X2,Y2，颜色1，颜色2，颜色3，颜色4）
绘制一个填充颜色的矩形。

参数

X1	左上角点的X坐标。
Y1	左上角点的Y坐标。
X2	右下角点的X坐标。
Y2	右下角点的Y坐标。
颜色1	矩形的主颜色，或者是左上角的颜色。
颜色2	第二个颜色。
颜色3	第三个颜色。
颜色4	第四个颜色。

说明：

- 你可以省略后三个（颜色2、颜色3、颜色4）颜色，引擎会用主颜色填充整个矩形。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

//在点（10，10）、（50，50）绘制一个用红色填充的矩形。
Scr2D.DRAW_FilledBox
10,10,50,50,RGBA(1,0,0,1),RGBA(1,0,0,1),RGBA(1,0,0,1),RGBA(1,0,0,1)
//或者是……
Scr2D.DRAW_FilledBox 10,10,50,50,RGBA(1,0,0,1)

DRAW_FilledCircle**(圆心X，圆心Y，半径，步长，边框颜色)**
在荧幕上划由颜色填充的圆圈。

参数

圆心X	要画的圆心的X坐标。
圆心Y	要画的圆心的Y坐标。
半径	圆圈的半径。
步长（可选的，默认是12）	圆圈每一段的数量，你用Step越大，圆圈就越精确。
边框颜色（可选的，默认是白色）	圆圈的颜色。（用RGBA宏获取）

说明：
不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。
//在圆心为200、300的位置上画一个半径50蓝色的圆圈。
Scr2D.DRAW_FilledCircle 200,300,50,20,RGBA(0,0,1,1)

DRAW_Line(X1,Y1,X2,Y2,颜色1,颜色2)
在荧幕上画一条线。

参数

X1	线条左端点的X坐标。
Y1	线条左端点的Y坐标。
X2	线条末端点的X坐标。
Y2	线条末端点的Y坐标。
颜色1	线条的颜色，如果未指定颜色2，就是线条左边的颜色。
颜色2（可选的，默认是-2）	线条右端的颜色。

说明：
- 如果你要在2D场景上绘制几条线段，可以多次调用DRAW_Line。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

//在（0，0）（100，100）之间绘制线段。
Scr2D.DRAW_Line 0，0，100，100，RGBA(1,1,1,1),RGBA(1,1,1,1)

DRAW_Line3D（X1,Y1,Z1,X2,Y2,Z2,颜色1，颜色2）
在3D世界里，绘制3D线段。

参数

X1	3D线段左端的X坐标。
Y1	3D线段左端的Y坐标。
Z1	3D线段左端的Z坐标。
X2	3D线段末端的X坐标。
Y2	3D线段末端的Y坐标。
Z2	3D线段末端的Z坐标。
颜色1	线条的颜色，如果未指定颜色2，就是线条左边的颜色。
颜色2（可选的，默认是-2）	线条右端的颜色。

说明：
因为这不是2D操作，因此，不能在ACTION_Begin2D和ACTION_End2D之间调用。
//在（0，0，0）（100，100，100）之间绘制线段。
Scr2D.DRAWLine 0，0，0，100，100，100，RGBA(1,1,1,1),RGBA(1,1,1,1)

DRAW_Point（X，Y，颜色）
在荧幕上画一个像素点。

参数

X	要绘制在荧幕上点的X坐标。
Y	要绘制在荧幕上点的Y坐标。
颜色	点的颜色。

说明：
- 如果你要在2D场景上绘制许多点，可以多次调用DRAW_Point，引擎已经优化好了。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

```
//在荧幕上的随机位置绘制点。
Scr2D.ACTION_Begin2D
    For I = 0 To 100
        Scr2D.DRAW_Point Rnd * 600, Rnd * 400
    Next I
Scr2D.ACTION_End2D
```

DRAW_RollOver（鼠标滑过的纹理，普通纹理，左边，顶端，右边，底端）
在荧幕上绘制一个纹理，当鼠标滑过该矩形区域时，改变纹理图。

参数

鼠标滑过的纹理	当鼠标滑过矩形区域，改变的纹理。
普通纹理	当鼠标未滑过矩形区域的纹理。
左边	左上角的X坐标。
顶端	左上角的Y坐标。
右边	右下角的X坐标。
底端	右下角的Y坐标。

如果鼠标滑过了指定的矩形区域。
说明：
- 如果你要绘制许多纹理，可以多次调用DRAW_RollOver和DRAW_Texture，引擎已经优化好了。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

DRAW_Texture（纹理，X1，Y1，X2，Y2，颜色1，颜色2，颜色3，颜色4，U1，V1,U2,V2）
画一个纹理或者纹理的一部分。

参数

纹理	要画的纹理索引。
X1	荧幕上的左上角的X坐标。
Y1	荧幕上的左上角的Y坐标。
X2	荧幕上的右下角的X坐标。
Y2	荧幕上的右下角的Y坐标。
颜色1（可选的，默认是白色）	纹理的左上角颜色。
颜色2（可选的，默认是-2）	纹理的右上角颜色。
颜色3（可选的，默认是-2）	纹理的左下角颜色。
颜色4（可选的，默认是-2）	纹理的右下角颜色。
U1（可选的，默认是0）	左上角的水平纹理坐标，0全部纹理。
V1（可选的，默认是0）	左上角的垂直纹理坐标，0全部纹理。
U2（可选的，默认是1）	右下角的水平纹理坐标，1全部纹理。
V2（可选的，默认是1）	右下角的垂直纹理坐标，1全部纹理。

说明：

- 如果你要绘制许多纹理，可以多次调用DRAW_Texture，引擎已经优化好了。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

```
//画“Start”的纹理。
Scr2D.Draw_Texture GetTex("GUI"), 0, 0, 800, 600, -1, -1, -1, -1, 0, 0, 1, 1
```

DRAW_TextureRotated（纹理，X，Y，宽度，高度，角度，U1,V1,U2,V2,颜色1，颜色2，颜色3，颜色4）

绘制旋转纹理或者旋转纹理的一部分。

参数

纹理	要画的纹理索引。
X	荧幕上的左上角的X坐标。
Y	荧幕上的左上角的Y坐标。
宽度	纹理的宽度。
高度	纹理的高度。
角度	纹理的旋转角度。用°或者弧度。
U1（可选的，默认是0）	纹理的左上角颜色。
V1（可选的，默认是0）	纹理的右上角颜色。
U2（可选的，默认是1）	纹理的左下角颜色。
V2（可选的，默认是1）	纹理的右下角颜色。
颜色1（可选的，默认是白色）	左上角的水平纹理坐标，0全部纹理。
颜色2（可选的，默认是-2）	左上角的垂直纹理坐标，0全部纹理。
颜色3（可选的，默认是-2）	右下角的水平纹理坐标，1全部纹理。
颜色4（可选的，默认是-2）	右下角的垂直纹理坐标，1全部纹理。

说明：

- 如果你要绘制许多纹理，可以多次调用DRAW_Texture，引擎已经优化好了。
- 不要忘记在ACTION_Begin2D和ACTION_End2D之间调用。

DRAW_TileMap（贴图映射，纹理索引，偏移量X，偏移量Y，视口，缩放X，缩放Y，Alpha）

用纹理绘制指定的贴图映射。

参数

贴图映射	要渲染的包含所有瓦片的贴图映射。
纹理索引	包含瓦片纹理的纹理索引。
偏移量X	在X轴的贴图映射的偏移量。
偏移量Y	在Y轴的贴图映射的偏移量。
视口	必须被渲染的贴图映射的视口，荧幕的一个矩形区域。
缩放X（可选的，默认是1）	贴图映射缩放。（默认是1）
缩放Y（可选的，默认是1）	贴图映射缩放。（默认是2）
Alpha（可选的，默认是1）	贴图映射的透明度。0：完全透明，1：不透明。

如果操作成功执行，返回True。

Load_Background（文件名，是否平铺）
读取位图作为背景。

参数

文件名	读取背景的文件名。
是否平铺（可选的，默认是False）	如果必须平铺背景，以便覆盖整个荧幕的话，设为True。

说明：
可以读取以下档作为背景：BMP、JPG、TGA、PNG、DDS.

```
//打开自动渲染，读取“Background”作为背景。
Scr2D.EnableAutoBackgroundRendering True
Scr2D.LoadBackground “Background”
```

SETTINGS_EnableAutoBackgroundRendering(打开/关闭)
更改背景渲染模式。

参数

打开/关闭	打开/关闭自动背景渲染功能。

说明：
如果你打开了这个功能，引擎将会在TVEngine.Clear以后自动绘制背景。

```
//启用自动渲染模式读取背景的文件名“Background.bmp”
Scr2D.SETTINGS_EnableAutoBackgroundRendering True
Scr2D.LOAD_Background
```

SETTINGS_SetAlphaTest(是否测试，Alpha测试的深度)
更改Alpha测试。

参数

是否测试	打开/关闭2DAlpha测试。
Alpha测试的程度	Alpha测试的参考值，（在0和255之间），

说明：
去除一些低Alpha的点使用Alpha测试是很有用的。通常用于3D树。

```
//打开Alpha测试。参考值128.
Scr2D.SETTINGS_SetAlphaTest True, 128
```

SETTINGS_SetBlendingMode(目标，源，打开Alpha)
更改2D的混合模式。

参数

目标	混合操作的目标。
源	混合操作的源。
打开Alpha（可选的，默认是True）	如果要打开Alpha混合，设为True。

说明：
通常仅仅用于高级用户。
//设置默认的Alpha混合模式。
Scr2D.SETTINGS_SetBlendingMode D3DBLEND_SRCALPHA,D3DBLEND_INVSRCALPHA,True
//应用附加模式。
Scr2D.SETTINGS_SetBlendingMode D3DBLEND_ONE,D3DBLEND_ONE,True

SETTINGS_SetTextureFilter(纹理过滤模式)
更改纹理过滤。

参数

纹理过滤模式	只能用于2D的纹理过滤。

说明：
通常是为了获得良好的像素到像素的行为你应该选择“点”的过滤器。
//选择“点”过滤器。
Scr2D.SETTINGS_SetTextureFilter TV_FILTER_NEARESTPOINT

到此，TVScreen2DImmediate的函数就介绍完毕了。

TVScreen2DText

ACTION_BeginText
开始渲染文本。

说明：
- 如果你要在荧幕上渲染许多文字，应该使用。
- 你不应该在ACTION_Begin2D和ACTION_End2D之间调用ACTION_BeginText和ACTION_EndText对，这样会出现一些错误。
- 优化方案是用DrawText或者DrawTextFontID来渲染文本。

```
//在荧幕上渲染文本。
ScrText.ACTION_BeginText()
    ScrText.TextureFont_DrawText  "hahaha",  100,  100,  RGBA(1,  1,  1,  1),
"Arial10"
ScrText.ACTION_EndText()
```

ACTION_EndText
在你渲染了所有文字之后，结束文本的渲染。

说明：
必须和ACTION_BeginText配对，文本才能被渲染。

```
//在荧幕上渲染文本。
ScrText.ACTION_BeginText()
    ScrText.TextureFont_DrawText  "hahaha",  100,  100,  RGBA(1,  1,  1,  1),
"Arial10"
ScrText.ACTION_EndText()
```

Fonts_DeleteAllFonts
从存储器中删除已创建的字体。

说明：
如果你不在需要这些字体，删除它们是很有用的。能够清空存储器，加速渲染。

```
//删除所有字体，然后退出。
ScrText.Fonts_DeleteAllFonts
End
```

NormalFont_Create(名称，选择的字体名，文本大小，是否粗体，是否斜体，是否下划线)
创建文本。

参数

名称	称呼该文本的名称。
选择的字体名	Windows包含的字体列表，要用的字体名。
字体大小	文本大小，用像素点表示。
是否粗体	设为True，粗体。
是否斜体	设为True，斜体。
是否下划线	设为True，文字有下划线。

返回创建文本的索引。
说明：
渲染字体，这不是最快的，如果你需要速度，用TextureFont_Create更好。
//创建文字，字体名Arial，10个像素点，粗体。
Dim NewFont As Long
NewFont=Scr2DText.NormalFont_Create("Arial10pts","Arial",10,True,False,False,False)

NormalFont_DrawText(文本，X，Y，颜色，用户字体)
用指定的字体名绘制文本。

参数

文本	要显示在荧幕上的文本。
X	文本的X坐标。
Y	文本的Y坐标。
颜色	文本的颜色，用16进制数表示，或者用RGBA宏获得。
用户字体（可选的，默认是空的）	称呼该文本的名称。

说明：
NormalFont不是渲染文本最快的方法,TextureFont更好。但是前者有TextureFont所没有的，就像渲染一些非英语的字母，就是可能的。
//渲染名为Font1的红色，内容是“hey, how are you?"的文本。
Scr2DText.NormalFont_DrawText "hey, how are you?",10,10,RGBA(1,0,0,1),"font1"

NormalFont_DrawTextFontID(文本，X，Y，颜色，文本ID)
用普通字体的索引绘制文本。

参数

文本	要被渲染的文本内容，也可以有多行文本。
X	文本开始的X坐标。
Y	文本开始的Y坐标。
颜色	文本的颜色。用16进制或者是RGBA获得。
文本ID	创建好文本的索引。

说明：

- 如果要渲染许多文字，可以用TextureFont_DrawTextFontID提高速度，因为它避免了字符查找。
- 在ACTION_BeginText和ACTION_EndText之间渲染文本。

//渲染名为Font1的红色，内容是“hey, how are you?"的文本，索引号是2
ScrText.NormalFont_DrawTextFontID "hey, how are you?", 10, 10, RGBA(1,0,0,1),2

SETTINGS_SetTextureFilter(纹理过滤器类型)
设置文本的纹理过滤器。

参数

纹理过滤器类型	要应用到纹理上的纹理过滤器。

说明：

- 该设置不会覆盖3D的和2D的纹理过滤器，仅能用于文本过滤。
- 通常线性过滤抗锯齿是最好的

//切换为线性过滤
Scr2DText.SETTINGS_SetTextureFilter TV_TEXTUREFILTER_BILINEAR

TextureFont_Create(名称，字体名称，大小，是否粗体，是否斜体，是否下划线，是否国际化)
创建一个纹理字体。

参数

名称	称呼该文本的名称。
字体名称	Windows包含的字体列表，要用的字体名。
大小	字体大小。
是否粗体（可选的，默认是False）	如果要粗体，设为True。
是否斜体（可选的，默认是False）	如果要斜体，设为True
是否下划线（可选的，默认是False）	如果要下划线，设为True
是否国际化（可选的，默认是False）	如果需要显示非英语字符，比如法语、西班牙语等，设为True

返回创建字体的索引。
说明：

- TextureFont是渲染字体最快的，NormalFont就比较慢了。因为它是基于GDI的。
- 警告：引擎是从Windows中可用的字体中创建字体，所以如果你用了指定（不是所有的Windows都有的）的字体，程序会显示不出来。确保把它放在了Windows的字体档夹里。

```
//创建一个字体名是Arial的，大小是10点的。
Dim Arial10 As Long
Arial10=ScrText.TextureFont_Create("Arial10","Arial",10,False,False,False,True)
```

TextureFont_DrawBillboardText(文本内容，X,Y,Z,颜色，字体索引，放大的X坐标，放大的Y坐标)
用指定的字体纹理在3D场景中绘制公告板文本。

参数

文本内容	要渲染在公告板上的文本。
X	公告板的X坐标。
Y	公告板的Y坐标。
Z	公告板的Z坐标。
颜色	将要被应用到公告板上的颜色，用16进制数0xaarrggbb或者RGBA宏获得。
字体索引	要写到公告板上的字体索引。
放大的X坐标(可选的，默认是1)	X轴上的纹理字体缩放因子。
放大的Y坐标(可选的，默认是1)	Y轴上的纹理字体缩放因子。

说明：

- 必须在3D场景中调用。它是一个Alpha混合的3D公告板，所以应该在所有不透明的物体渲染以后渲染。
- 文字将被放到公告板中间。

```
// 在游戏者的位置上方绘制他的名字。
ScrText.TextureFont_DrawBillboardText
Player(1).Name,Player(1).Position.x,Player(1).position.y+50,Player(1).Position.
z+50,FontArialID,2,2
```

TextureFont_DrawText(文本内容，X,Y,颜色，用户字体)
用指定的纹理字体绘制文本。

参数

文本内容	要渲染在荧幕上的文本。用"/n"多行。
X	文本开头的X坐标。
Y	文本开头的Y坐标。
颜色	文本的颜色。通常用RGBA宏或者16进制数0xrrggbbaa获得。
用户字体（可选的，默认是空的）	称呼该文本的名称。用户字体就是你用TextureFont_Create创建的名称。

说明：

- 如果你要在荧幕上渲染许多文字，可以用TextureFont_DrawTextFontID提高速度。因为它避免了纹理字体的字符查找
- 在ACTION_BeginText和ACTION_EndText之间渲染文本。
- 仅用于2D渲染，。

//绘制字体为Arial的文本，之前已经用TextureFont_Create创建。

ScrText.TextureFont_DrawText "some text here",10,10,RGBA(1,1,0,1),6

TextureFont_DrawTextEx(文本内容，文本矩阵，颜色，字体id)

用特殊的矩阵和指定的纹理字体绘制一段文本，用于高级用户。

参数

文本内容	要绘制的文本内容。
文本矩阵	一段D3D用于文本的矩阵。
颜色	文本的颜色。通常用RGBA宏或者16进制数0xrrggbbaa获得。
字体id	纹理字体的索引。

说明：

- 应保留只为高级用户将要使用特殊的3D渲染的字体。
- TVScreen2DText.TextureFont_DrawBillboardText能够用于简单的3D广告牌文字的渲染。
- 就像在2D渲染那样，用在3D渲染里也一样好。

TextureFont_DrawTextFontID(文本，X,Y,颜色，字体ID)

用指定的纹理字体索引绘制文本。

参数

文本	要绘制的文本内容。用“/n”表示多行。
X	文本开始的X坐标。
Y	文本开始的Y坐标。
颜色	文本的颜色。通常用RGBA宏或者16进制数0xrrggbbaa获得。
字体ID	纹理字体的索引。

说明：

- 如果要渲染许多文字，可以用TextureFont_DrawTextFontID提高速度，因为它避免了字符查找。
- 在ACTION_BeginText和ACTION_EndText之间渲染文本。
- 只能用在2D渲染中。

//用6的索引绘制文本。

ScrText.TextureFont_DrawTextFontID "some text here",10,10,RGBA(1,1,0,1),6

TextureFont_GetTextSize(文本内容，字体id，返回的文本宽度，返回的文本高度)
计算文本宽度和高度。

参数

文本内容	要计算的文本内容。
字体id	指定的文本的索引。用TextureFont_Create创建的。
返回的文本宽度	返回的文本宽度。
返回的文本高度	返回的文本宽度。

说明：

- 只能用于纹理字体。
- 可以迅速、精确的知道荧幕上的纹理字体的大小。比如用来适应Windows标准文本字号是很有用的。

```
//返回“hey how are you?”大小。
Dim Width As Single
Dim Height As Single
Dim Text As String
Text="hey how are you?"
ScrText.TextureFont_GetTextSize Text,2,Width,Height
//将信息保存到debug档中，以便查看。
TV.AddToLog "文本的宽度：" & Width & "文本的高度：" & Height
```

TextureFont_GetTexture(字体id)
从字体id返回字体的纹理。

参数

字体id	要返回纹理的字体id，用TextureFont_Create创建的。

返回字体的纹理，然后可以用于任何东西。
说明：
该函数提供了访问字体纹理的功能。

```
//将字体纹理保存到一个档中去。
TexFactory.SaveTexture "TextureFont.Bmp",ScrTex.TextureFont_GetTexture(4)
```

到此，TVScreen2DText的函数就介绍完毕了。

TVParticleSystem

CreateBillboardSystem(粒子总数，开始的粒子数目，粒子大小，位置，半径，纹理)
初始化粒子系统。

参数

粒子总数	粒子最大的数量。直到最大数目前，不会被发射到最大数目。
开始的粒子数目	发射之前的粒子数目。
粒子大小	粒子大小，最大是64.
位置	发射器的位置。是个矢量。
半径	发射器的半径。
纹理	要应用到粒子上的纹理。

说明：
在其他函数能用之前，应该初始化粒子系统，创建粒子系统。

CreateSystemFromPreset(预置，粒子数量，位置矢量，方位矢量，纹理，颜色)
用系统预置创建粒子系统。

参数

预置	选择的预置。
粒子数量	最大的粒子数量
位置矢量	描述发射器的位置。
方位矢量	发射器的方向。
纹理	要应用在粒子上的纹理。
颜色（可选的，默认是-1）	预置的颜色。

说明：
如果预置的不够好，那就用CreateBillboardSystem来代替它。

Destroy
删除当前的粒子系统清空存储器。

说明：
如果你不在需要粒子系统了，能够使用这个清理一些存储器。

GetActiveParticleNumber
得到活动的粒子数量。

返回活动粒子的数量。
说明：
每一次调用时，都会有变化。

GetEmittorPosition
返回当前粒子发射器的位置。

返回粒子发射器的位置矢量。

GetParticleDirection(粒子)
返回指定粒子的朝向。

参数

粒子	要得到方向的粒子索引。

返回描述粒子方向的3D索引。

GetParticleKey(粒子)
返回指定粒子的当前的关键帧。

参数

粒子	要得到关键帧的粒子索引。

返回指定粒子的关键帧。

GetParticlePosition(粒子)
返回指定粒子的当前位置。

参数

粒子	要得到位置的粒子索引。

返回指定粒子的当前位置矢量。

SetAlphaBlendingMode(改变，源，目标)
改变粒子系统的Alpha混合模式。

参数

改变	通过在粒子生存期改变粒子颜色。
源（可选的，默认是D3DBLEND_ONE）	源混合模式。
目标（可选的，默认是D3DBLEND_ONE）	目标混合模式。

说明：
Alpha混合模式描述了在渲染缓冲区内粒子怎样混合。
//更改Alpha混合模式，以便于粒子降低它的Alpha效果。
TVPar.SetAlphaBlendingMode TV_CHANGE_ALPHA, D3DBLEND_ONE, D3DBLEND_ONE

SetAlphaTest(是否开启，返回值)
设置粒子的Alpha测试。

参数

是否开启	开启粒子系统的Alpha测试，设为True。
返回值	粒子的参考值，通常是128.取值范围从0-255

说明：
当写到目标渲染表面时，应用程序能控制Alpha测试。使用Alpha测试，设置你的应用程序以便于它通过Alpha测试测试没一个像素点。如果测试通过，像素将要充满渲染表面，如果没有，D3D将要忽略像素。引擎将只允许像素大于阿尔法值的所有参考的像素。
//用默认的Alpha测试值测试粒子。
TVPar.SetAlphaTest True，128

SetEmittorMoveMode(移动方式)
设置发射器移动的新方式。

参数

移动方式	发射器移动的新方式

说明：
- 当粒子发射器生成了许多粒子而且速度很快时，线性移动通常是最平滑的。
- 你应该试所有的方法之后，找到最适合你程序的方式。

SetGeneralSystemPosition(位置矢量)
更改全局位置。不包括发射器位置。

参数

位置矢量	新的粒子的位置的3D矢量。

说明：
这不会改变发射器位置。

SetGerneratorSpeed(速度)
更改全局速度，独立于计算机速度。

参数

速度	两个粒子之间间隔的速度，用毫秒表示。

说明：
- 全局速度独立于计算机速度。内部使用TV.AccurateTimeElapsed
- 可以设置为0，不让粒子被创建。

//设置全局速度，以便于每隔20毫秒生成一个粒子。
TVPar.SetGerneratorSpeed 20

SetGravity(重力，反弹因子)
设置粒子系统的重力。

参数

重力	重力常数，通常是1.
反弹因子（可选的，默认是0.5）	将要被应用到粒子上的反弹因子。通常是0.5，将要反弹一般的速度。

说明：
重力是将要被增加到粒子速度的加速值。
//用默认的重力值和反弹因子设置粒子系统。
TVPar.SetGravity 1，0.5

SetNewParticleData(X速度，Y速度，Z速度，红色，绿色，蓝色，Alpha，生存期，X位置，Y位置，Z位置)
设置将要生成的粒子信息。

参数

X速度	粒子速度的X坐标。
Y速度	粒子速度的Y坐标。
Z速度	粒子速度的Z坐标。
红色	粒子的红色部分。取值范围从0-1
绿色	粒子的绿色部分。取值范围从0-1
蓝色	粒子的蓝色部分。取值范围从0-1
Alpha	粒子的Alpha，不透明度。取值范围从0-1
生存期	粒子的生存期，用秒表示。这以后粒子将会消失。
X位置	粒子位置的X坐标。
Y位置	粒子位置的Y坐标。
Z位置	粒子位置的Z坐标。

说明：
- 你能够把它用在所有事件中。
- 如果你要控制更多的粒子，要想简单些，NewParticle事件是更好的。

SetParticleColor(红色，蓝色，绿色，Alpha，粒子)
更改指定的例子颜色。

参数

红色	粒子的红色组件。
蓝色	粒子的蓝色组件。
绿色	粒子的绿色组件。
Alpha	粒子的Alpha透明度组件。
粒子	粒子索引。

说明：
当粒子被创建时，你应该赋予颜色。在NewParticle事件中，或者在自动生成模式。

SetParticleDirection(方向矢量，粒子)
改变粒子的方向。

参数

方向矢量	描述方向的3D矢量。
粒子	要更改的粒子索引。

说明：
当粒子被创建时，你应该赋予方向。在NewParticle事件中，或者在自动生成模式。

SetParticleLifeTime(生存期，粒子)
更改指定粒子的生存期。

参数

生存期	粒子的生存期，用秒表示。
粒子	要更改的粒子索引。

说明：
当粒子被创建时，你应该赋予生存期。在NewParticle事件中，或者在自动生成模式。

SetParticlePosition(位置，粒子)
改变粒子的位置。

参数

位置	描述粒子位置的3D矢量。
粒子	要更改的粒子索引。

说明：
当粒子被创建时，你应该赋予它们的位置。在NewParticle事件中，或者在自动生成模式。

SetParticleSpeed(速度)
更改粒子移动的速度。

参数

速度	粒子的移动速度，用毫秒表示。

说明：
在找到合适的速度之前，需要反复尝试。
//设置粒子速度为1.
TVPar.SetParticleSpeed 1

SetParticleSystemPreset(预置模式)
设置粒子系统的预置模式。

参数

预置模式	要用的新预置。

说明：
- 目前只支持爆炸模式。
- 你可以用粒子编辑器来创造更好的效果。
- 当你要渲染粒子系统时，不要忘记使用RenderPreset和RenderParticles!

SetParticleTexture(纹理)
更改粒子的纹理。

参数

纹理	要应用在粒子的纹理。

说明：
- 你只可以在所有粒子上应用一样的纹理。
- 虽然你用不同的纹理创建了一些粒子。

```
//设置粒子纹理为"parttexture"
TVPar.SetParticleTexture GetTex("parttexture")
```


MoveEmittor(新位置矢量)
把发射器移动到别的位置。

Pause
暂停当前的预置效果。

Continue
使已终止的粒子系统继续。

RenderParticles
将粒子渲染到荧幕。

说明：
如果你用了预置的粒子系统，用RenderPreset。

```
//更新和渲染粒子。
TVPar.UpdateParticles TV_UPDATE_BOTH
TVPar.RenderParticles
```

RenderPreset
渲染预置的粒子系统。

ResetAllParticles
渲染整个的粒子系统。

说明：
该方法只能工作在预置的粒子系统上。
//重置粒子系统。
TVPar.ResetAllParticles

到此，TVParticleSystem的函数就介绍完毕了。

TVCollisionResult

GetCollisionResult
返回要测试碰撞项目的类型。Mesh、actor、bsp……

GetCollisionActor
如果碰撞对象类型是Actor，返回Actor。

如果没有碰撞发生，或者碰撞的对象类型不是Actor，返回Nothing。
说明：
如果你不知道Actor是否碰撞，用GetCollisionActor来测试碰撞。

```
//当“捡起”了Actor，设置选定的材质颜色为红色。
Set Pick=Scene.MousePicking(MX,MY,TV_COLLIDE_MESH,BOUNDINGBOX)
If Pick.IsCollision=True Then
    Pick.GetCollisionActor.SetMaterial RedMaterial
End If
```

GetCollisionActor2
如果碰撞对象类型是Actor2，返回Actor2。

如果没有碰撞发生，或者碰撞的对象类型不是Actor2，返回Nothing。
说明：
如果你不知道Actor2是否碰撞，用GetCollisionActor2来测试碰撞。

```
//当“捡起”了Actor2，设置选定的材质颜色为红色。
Set Pick=Scene.MousePicking(MX,MY,TV_COLLIDE_MESH,BOUNDINGBOX)
If Pick.IsCollision=True Then
    Pick.GetCollisionActor2.SetMaterial RedMaterial
End If
```

GetCollisionMesh
如果碰撞对象类型是Mesh，返回Mesh。

如果没有碰撞发生，或者碰撞的对象类型不是Mesh，返回Nothing。
说明：
如果你不知道Mesh是否碰撞，用GetCollisionMesh来测试碰撞。

```
//当“捡起”了Mesh，设置选定的材质颜色为红色。
Set Pick=Scene.MousePicking(MX,MY,TV_COLLIDE_MESH,BOUNDINGBOX)
If Pick.IsCollision=True Then
    Pick.GetCollisionMesh.SetMaterial RedMaterial
End If
```

GetCollisionDistance
返回从两个要碰撞的物体的距离。

返回描述从两个要碰撞的物体的距离的浮点数。
说明：
如果你用“精确测试”模式，你可以得到一些关于碰撞的信息。

```
//检测碰撞，如果距离足够，碰到了Actor2，设置状态为“die”
If Coll.IsCollision=True Then
    If Coll.GetCollisionDistance<100 Then
        Actor.SetAnimationByName "die"
    End If
End If
```

GetCollisionNormal
返回碰撞面的法线。

返回碰撞面的法线的3D矢量。
说明：
如果你用“精确测试”模式，你可以得到一些关于碰撞的信息。

```
//碰撞，写上法线矢量
Dim Pick As TVCollisionResult
Set Pick=Land.MousePick(MX,MY)
If Pick.IsCollision=True Then
    Scene.DrawText "法线的信息:" & Pick.GetCollisionNormal.X & " " &
Pick.GetCollisionNormal.Y & " " & Pick.GetCollisionNormal.Z,20,30
End If
```

GetCollisionTriangleUV(要返回的U,要返回的V)
返回U和V的碰撞重心坐标

参数

要返回的U	要返回的U（第一个边的）坐标。
要返回的V	要返回的V（第一个边的）坐标。

说明：
如果你用“精确测试”模式，你可以得到一些关于碰撞的信息。

GetImpactPoint
返回检测鼠标的碰撞点，模型和地面等交汇就是碰撞了。

返回碰撞交汇点的3D矢量。

```
//检测碰撞，把碰撞点矢量的3个分量打印在荧幕上。
Dim Pick As TVCollisionResult
Set Pick=Land.MousePick(MX,MY)
If Pick.IsCollision Then
    Scene.DrawText "Impact " & Pick.GetImpactPoint.X & " " & Pick.GetImpactPoint.Y & " " & Pick.GetImpactPoint.Z,20,30
End If
```

IsCollision
决定碰撞是否发生。

如果碰撞发生了，返回True。

```
//检测碰撞，把碰撞点矢量的3个分量打印在荧幕上。
Dim Pick As TVCollisionResult
Set Pick=Land.MousePick(MX,MY)
If Pick.IsCollision Then
    Scene.DrawText "Impact " & Pick.GetImpactPoint.X & " " & Pick.GetImpactPoint.Y & " " & Pick.GetImpactPoint.Z,20,30
End If
```

到此，TVCollisionResult的函数就介绍完毕了。

TVKeyFrameAnim

GetBoneMatrix(帧索引，本地位置)
返回指定帧的矩阵。

参数

帧索引	骨骼帧的索引。
本地位置	变换偏移量矢量。

返回：描述骨骼帧的3D矩阵。
//粘上剑mesh到索引为8的骨骼上。
Sword.SetMatrix KeyFrame.GetBoxMatrix(8,vector(0,0,0))

GetBonePosition(帧索引，本地位置)
返回指定帧的位置。

参数

帧索引	骨骼帧的索引。
本地位置	变换偏移量矢量。

返回：描述骨骼帧的3D矢量。

GetKeyFrame
返回关键帧。
//显示当前的关键帧。
Text.TextureFont_DrawText("关键帧： " & KeyFrame.GetKeyFrame , 20, 20)

IsAnimationFinished
如果动画结束了，返回True。如果仍在播放，返回False。

说明：
只能是当循环被关掉，这个方法才工作。

LoadAdditionalAnimations(X档)
读取存储在X档的附加动画。

参数

X档	被保存动画的X档。

说明：
这会简单地在当前动画的末尾添加动画。
//读取档“animation.X"，作为actor的新的动画。
KeyFrame.LoadAdditionalAnimations "Animation.X"

PlayAnimation(帧速)
播放指定的动画。

参数

帧速（可选的，默认是30）	帧速。

//播放动画，每秒24帧。
KeyFrame.PlayAnimation 24

StopAnimation
停止播放当前的动画。

说明：
如果你要再次播放，那将会从头开始。
//停止播放。
KeyFrame.StopAnimation

SetAnimationName(动画名称)
从动画名称中设定要播放的动画。

参数

动画名称	要播放的动画名称。

说明：
如果在列表中未找到指定的动画名称，将要保持当前动画。
//设置当前动画“change”
KeyFrame.SetAnimationName "change"

SetAnimationID(动画索引，是否循环)
通过索引，改变动画。

参数

动画索引	动画索引。
是否循环（可选的，默认是True）	如果动画必须循环，设为True。如果动画只播放一遍，设为False。

说明：
你还能通过调用SetAnimationName改变动画名字，改变动画。
//播放索引为3的动画。
KeyFrame.SetAnimationID 3
KeyFrame.PlayAnimation

SetCustomAnimation(开始的帧，结尾的帧)
自定义mesh的动画范围。

参数

开始的帧	自定义动画的开始的帧
结尾的帧	自定义动画的结尾的帧

说明：
这可以用于，在一个很大的档中保存了很多动画，只需要播放器中的一个。

//在400帧到800帧的地方循环播放。
KeyFrame.SetCustomAnimation 400,800

到此，TVKeyFrameAnim的函数就介绍完毕了。

TVGraphicEffect

ChangeGamma(Gamma值)

当引擎在全屏模式下，更改荧幕的Gamma

参数

Gamma值	Gamma值，取值范围从0.0-10.0，通常1.2是默认的。

说明：

该方法只能工作在全屏模式下。引擎使用1.2作为默认值，以使所有东西更亮。

```
//把Gamma值更改为1.0
GraphicEffect.ChangeGamma 1.0
```

AdvancedGamma(红色Gamma，绿色Gamma，蓝色Gamma)

当引擎处于全屏状态下，允许更改三个通道的Gamma

参数

红色Gamma	红色Gamma的通道，取值范围从0.0-10.0，通常是1.2
绿色Gamma	绿色Gamma的通道，取值范围从0.0-10.0，通常是1.2
蓝色Gamma	蓝色Gamma的通道，取值范围从0.0-10.0，通常是1.2

说明：

更改场景的亮度，或者创建不同颜色的闪光。

```
//创建奇妙的蓝色效果，只需要改变蓝色Gamma。
GraphicEffect.AdvancedGamma 1.0，1.0，1.5
```

FadeFinished

如果结束淡入淡出效果，返回True。

说明：

在RenderToScreen调用之后再调用淡入淡出效果。

```
//直到淡出荧幕，一直显示淡化效果。
GraphicEffect.FadeIn 1000
Do Until GraphicEffect.FadeIn=True
    TV.Clear
    RenderScene
    TV.RenderToScreen
Loop
```

FadeIn(速度)

用指定的速度做一个淡入效果。在3D场景中，从全黑渐变到出现物体。

参数

速度（可选的，默认是2000）。	用毫秒表示。

```
//淡入进入一个新的关卡。
GraphicEffect.FadeIn 1000
Do
    //进行渲染。
    TV.Clear
    RenderScene
    TV.RenderToScreen
Loop
```

FadeOut（速度）

用指定的速度做一个淡出效果。在3D场景中，一点一点到退出程序。

参数

速度（可选的，默认是2000）。	用毫秒表示。

```
//当点击ESC，用淡出效果，退出程序。
Do
    //进行渲染。
    TV.Clear
    RenderScene
    If Inp.IsKeyPressed(TV_KEY_ESCAPE)=True Then
        GrapicEffect.FadeOut 1000
        Fading=1
    End if
    TV.RenderToScreen
Loop Until GraphicEffect.FadeFinished=True And Fading=1
```

StopFade

停止当前的淡入淡出效果，或者，如果当前没有任何淡入淡出效果，则什么也不做。

```
//停止淡入淡出
GraphicEffect.StopFade
```

Flash(红色，绿色，蓝色，速度)

在整个荧幕上生成一个固定长度的闪。

参数

红色	闪电的红颜色，取值范围从0.0-1.0
绿色	闪电的绿颜色，取值范围从0.0-1.0
蓝色	闪电的蓝颜色，取值范围从0.0-1.0
速度（可选的，默认是500）	闪电的时间，用毫秒表示。

说明：

在RenderToScreen调用之后再调用Flash效果。

```
//假设游戏者被枪打中了，显示300毫秒的闪电。
Do
    If GunTouchedPlayer=True Then
        GraphicEffect.Flash 1.0, 0, 0, 500
    End if
Loop
```

FlashFinished

如果Flash效果结束了，返回True。

```
//闪电效果，直到程序结束。
GraphicEffect.Flash 1.0, 0, 0, 300
Do Until GraphicEffect.FlashFinished = True
    TV.Clear
    RenderScene
    TV.RenderToScreen
Loop
```

InstantFlash(红色，绿色，蓝色，闪电因子)

在当前荧幕上生成快速闪电（只有一帧）。

参数

红色	闪电的红色组件。取值范围从0.0-1.0
绿色	闪电的绿色组件。取值范围从0.0-1.0
蓝色	闪电的蓝色组件。取值范围从0.0-1.0
闪电因子	闪电的密度，取值范围从0.0（透明）-1.0（不透明）

说明：

它绘制了一个Alpha混合的闪电，在RenderToScreen调用之后再调用Flash效果。

```
//绘制一个快速的红色闪电。
TV.Clear
RenderScene
GraphicEffect.InstantFlash 1.0, 0.0, 0.0, 1.0
TV.RenderToScreen
```

StopFlash
停止当前的闪电效果，或者，如果当前没有任何闪电效果，则什么也不做。

//停止淡入淡出
GraphicEffect.StopFlash

到此，TVGraphicEffect的函数就介绍完毕了。

TVRenderSurface

Destroy

删除所有的渲染表面，清空存储器。

说明：

因为渲染表面是直接访问记忆体的，当不再需要时，应该释放显存。

```
//删除渲染表面。
RenderSurface.Destroy
```

StartEnvRender(表面)

在立方体渲染表面上开始2D、3D渲染。

参数

表面	引擎要渲染的环境映射的6个面中的一个。

说明：

- 在调用TVEngine.Clear清除背景缓冲区之后，你可以设置摄影机，渲染渲染表面。
- 然后立方体能够用于任何蒙皮网格的的环境纹理。

EndEnvRender

完成渲染立方体环境渲染表面。

说明：

当你要完成渲染立方体表面时，就可以调用这个方法。然后你可以渲染背景缓冲去了。

StartRender(是否清除背景缓冲区)

在渲染表面上开始2D或3D渲染

参数

是否清除背景缓冲区（可选的，默认是True）	如果引擎在渲染之前需要清除渲染表面的背景缓冲区，设为True。

说明：

- 在调用TVEngine.Clear清除背景缓冲区之后，你可以设置摄影机，渲染渲染表面。
- 然后立方体能够用于任何物体的的环境纹理

```
Text1.StartRender
  Text.NormalFont_Create "TV", "Arial", 18, True, True, False
    Scr.ACTION_Begin2D
    Scr.DRAW_FilledBox 0, 0, 128, 32, RGBA(1, 1, 1, 1), RGBA(1, 0, 1, 1),
RGBA(1, 0, 0, 1), RGBA(0, 0, 0, 1)
  Scr.ACTION_End2D
  Text.ACTION_BeginText
    Text.NormalFont_DrawText "TrueVision", 1, 1, RGBA(1, 1, 0, 1), "TV"
  Text.ACTION_EndText
Text1.EndRender
```

EndRender
结束渲染表面的渲染。

```
Text1.StartRender
  Text.NormalFont_Create "TV", "Arial", 18, True, True, False
  Scr.ACTION_Begin2D
      Scr.DRAW_FilledBox 0, 0, 128, 32, RGBA(1, 1, 1, 1), RGBA(1, 0, 1, 1),
RGBA(1, 0, 0, 1), RGBA(0, 0, 0, 1)
  Scr.ACTION_End2D
  Text.ACTION_BeginText
    Text.NormalFont_DrawText "TrueVision", 1, 1, RGBA(1, 1, 0, 1), "TV"
  Text.ACTION_EndText
Text1.EndRender
```

GetCamera
返回关联到渲染表面的摄影机对象。

返回用于渲染表面的摄影机对象。
说明：
当你创建了一渲染表面时，就会自动创建一个摄影机并且关联到渲染表面。

GetD3DTexture
得到内部的将要用于引擎渲染的D3D纹理

说明：
仅为高级用户准备用DirectX8.1库添加自己的函数

GetD3DXRenderSurface
得到内部的将要用于引擎渲染的D3DX渲染表面。

返回用于引擎的D3DXRenderSurface。
说明：
仅为高级用户准备用DirectX8.1库添加自己的函数

GetTexture
返回被绘制的渲染表面的纹理索引。

返回关联到渲染表面的动态纹理的索引。
说明：
你可以看到有动态纹理的渲染表面。要应用动态纹理到物体上，就用GetTexture。

Width
返回渲染表面的宽度。

//在调试信息中，显示渲染表面的宽度。
Debug.Print RenderSurface.Width

Height
返回渲染表面的高度。

//在调试信息中，显示渲染表面的高度。
Debug.Print RenderSurface.Height

SetBackgroundColor(颜色)
设置渲染表面的整体的颜色。如果你用TVRenderSurface.StartRender，将会清除掉渲染表面的背景缓冲。

参数

颜色	用16进制数格式（0xaarrggbb）得到的或者用RGBA宏得到的32bits颜色。

SetCamera(X位置，Y位置，Z位置，视点X，视点Y，视点Z)
设置附加在渲染表面的摄影机。

参数

X位置	摄影机位置的X坐标。
Y位置	摄影机位置的Y坐标。
Z位置	摄影机位置的Z坐标。
视点X	摄影机视点的X坐标。
视点Y	摄影机视点的Y坐标。
视点Z	摄影机视点的Z坐标。

说明：
该函数和TVScene、TVCamara.SetCamera的作用是一样的。
//设置摄影机，在座标原点上。
RenderSurface.SetCamera 0, 0, 0, 50, 0, 0

SetEnvironmentCamera(面，矢量)
设置环境映射渲染的摄影机。

参数

面	当前要渲染的面。
矢量	相关的矢量（摄影机位置）

SetViewFrustum(视野角度，远平面，近平面)
打开透视图，设置当前的视图属性。

参数

视野角度	视野角度的角
远平面	视野的最大距离。（假想）远平面的所有东西都不会渲染。
近平面（可选的，默认是1）	近距离平面，距离前面太远，就不会渲染。

说明：

- 如果上一个模式是等轴的摄影机，该方法将会切换回透视视图。
- 视野角度定义了你能看到东西的多少，60°和90°是比较好的。如果你缩小视野角度，你就能做出一些放大效果。
- 你应该为你的场景设置远平面最可能小的值，通常在1000到50000之间。如果设置的太大了，你将看到错误的东西，那是因为渲染的太慢，还没有渲染到这里。

到此，TVRenderSuface的函数就介绍完毕了。

TVCamera

ChaseCamera(Mesh,结束位置，视点，摄影机速度，指数速度，最快速度，是否避免，最小速度)

使摄影机追踪物体的效果。就像极品飞车系列、GTA那样。

参数

Mesh	要被摄影机追踪的Mesh。
结束位置	摄影机与Mesh的距离。比如：(0,0,0)到(0,0,-20)
视点	摄影机的视点。
摄影机速度	摄影机的跟踪速度。
指数速度（可选的，默认是True）	如果通过和Mesh的距离，改变了速度，设为True。
最快速度（可选的，默认是100）	摄影机的最大速度。
是否避免（可选的，默认是False）	如果摄影机需要避开物体，mesh、地形，设为True。
最小速度（可选的，默认是1）	摄影机的最小速度。

说明：

你要在每一帧调用这个函数以便于摄影机能够正确地更新。

//使摄影机跟踪游戏者。

TVCamera.ChaseCamera Player,Vector3(0,0,-50),Vector(0,30,0),10,True, 40,False,1

ChaseCameraActor2(Actor2,结束位置，视点，摄影机速度，指数速度，最快速度，是否避免，最小速度)

参数

Actor	要被摄影机追踪的Actor2类型的模型。
结束位置	摄影机与Actor2的距离。比如：(0,0,0)到(0,0,-20)
视点	摄影机的视点。
摄影机速度	摄影机的跟踪速度。
指数速度（可选的，默认是True）	如果通过和Actor2的距离，改变了速度，设为True。
最快速度（可选的，默认是100）	摄影机的最大速度。
是否避免（可选的，默认是False）	如果摄影机需要避开物体，mesh、地形，设为True。
最小速度（可选的，默认是1）	摄影机的最小速度。

说明：

你要在每一帧调用这个函数以便于摄影机能够正确地更新。

//使摄影机跟踪游戏者。

TVCamera.ChaseCameraActor2 Player,Vector3(0,0,-50),Vector(0,30,0),10,True,

40,False,1

ChaseCameraMatrix(对象的矩阵,**结束位置，视点，摄影机速度，指数速度，最快速度，是否避免，最小速度**)

参数

对象矩阵	要被跟踪的对象矩阵。
结束位置	摄影机与Mesh的距离。比如：(0,0,0)到(0,0,-20)
视点	摄影机的视点。
摄影机速度	摄影机的跟踪速度。
指数速度（可选的，默认是True）	如果通过和Actor2的距离，改变了速度，设为True。
最快速度（可选的，默认是100）	摄影机的最大速度。
是否避免（可选的，默认是False）	如果摄影机需要避开物体，mesh、地形，设为True。
最小速度（可选的，默认是1）	摄影机的最小速度。

说明：
你要在每一帧调用这个函数以便于摄影机能够正确地更新。
//使摄影机跟踪游戏者。
TVCamera.ChaseCameraMetrix Player,Vector3(0,0,-50),Vector(0,30,0),10,True,40,False,1

GetCameraIndex
返回摄影机的索引。

返回摄影机的索引的整型值。
说明：
全局摄影机默认的索引是0.
//打印出摄影机的索引。
DrawText Camera.GetCameraIndex

GetLookAt
返回摄影机的视点。视点位置通常离摄影机的位置不远，一般它们的距离是1.

返回视点的3D矢量。
说明：
在下面的代码中，你能够看到联合使用GetLookAt和GetPosition来得到摄影机的方向。
//得到摄影机的方向矢量。
Dim Direction As D3DVECTOR
Direction=VSubtract(Camera.GetPosition,Camera.GetLookAt)

GetPosition
返回摄影机的位置。

返回摄影机的位置矢量。
```
//得到摄影机的方向矢量
Dim Direction As D3DVECTOR
Direction=VSubtract(Camera.GetPosition,Camera.GetLookAt)
```

GetMatrix
返回摄影机的矩阵。

```
//把武器附加到摄影机。
Weapon.SetMatrix Camera.GetMatrix
```

SetMatrix(摄影机矩阵)
设置摄影机的变换矩阵。

参数

摄影机矩阵	描述摄影机矩阵的D3D矩阵。

说明：
你能用一个矩阵描述摄影机矩阵，但是不能进行缩放。
```
//从一个物理引擎中得到矩阵并把它作为摄影机使用。
Dim Matrix as D3DMATRIX
PhysicsEngine.GetMatrix Matrix
Camera.SetMatrix Matrix
```

GetRotation(要返回的X角，要返回的Y角，要返回的Z角)
返回摄影机的旋转角度。

参数

要返回的X角	绕X轴旋转的角，用°或者弧度。
要返回的Y角	绕Y轴旋转的角，用°或者弧度。
要返回的Z角	绕X轴旋转的角，用°或者弧度。

说明：
在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，°或者弧度。
注意：如果你直接访问摄影机矩阵，GetRotation也许不会工作。
```
//获得旋转角度。
Dim RotationX As Single, RotationY As Single, RotationZ as Single
TVCamera.GetRotation RotationX, RotationY, RotationZ
```

GetRotationMatrix
返回摄影机的旋转矩阵。

返回摄影机旋转矩阵的D3DMATRIX。

```
//返回摄影机的旋转矩阵。
Dim RotationMatrix As D3DMATRIX
RotationMatrix = TVCamera.GetRotationMatrix
```

LookAtMesh(Mesh)
使摄影机面向一个mesh。注意：这个方法将使摄影机只面向mesh的Y轴。

返回摄影机面向的mesh。

```
//面向游戏者。
Camera.LookAtMesh Player
```

MoveRelative(向前移动，向高处移动，横向移动)
移动摄影机面对的方向。

参数

向前移动	向前移动的总数。
向高处移动	向高处移动量。
横向移动	横向移动量。

```
//如果用户按上，就向前移动。
If Inp.IsKeyPressed(TV_KEY_UP) = True Then
Camera.MoveRelative 5, 0, 0
End If
```

RotateAroundMesh(目标mesh，是否增加旋转，角度，距离，关联mesh的摄影机高度)
在指定的距离和角度围绕一个mesh转。

参数

目标mesh	要围绕旋转的mesh。
是否增加旋转	决定是否附加旋转。
角度	围绕Mesh旋转的角度，° 或者弧度。
距离（可选的，默认是0）	摄影机和mesh间的距离，如果想要保持当前的距离，设为0。
关联mesh的摄影机高度（可选的，默认是0）	关联mesh的摄影机高度

说明：

- 旋转只在Y轴上。
- 在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，° 或者弧度。

```
//绕着游戏者旋转。
TVCamera.RotateAroundMesh Player,True,0,0,40
```

RotateX(X角度)
绕X轴旋转的角度。

参数

X角度	旋转角度。

说明：
在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，° 或者弧度。
//绕着X轴旋转90度。
TV.SetAngleSystem TV_ANGLE_DEGREES
Camera.RotateX 90

RotateY(Y角度)
绕Y轴旋转的角度。

参数

Y角度	旋转角度。

说明：
在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，° 或者弧度。
//绕着Y轴旋转90度。
TV.SetAngleSystem TV_ANGLE_DEGREES
Camera.RotateY 90

RotateZ(Z角度)
绕Z轴旋转的角度。

参数

Z角度	旋转角度。

说明：
在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，° 或者弧度。
//绕着Z轴旋转90度。
TV.SetAngleSystem TV_ANGLE_DEGREES
Camera.RotateZ 90

SetCamera(X位置，Y位置，Z位置，视点X，视点Y，视点Z)
定义摄影机朝向和位置的主要和简单的方法

参数

X位置	摄影机位置的X坐标。
Y位置	摄影机位置的Y坐标。（高度）
Z位置	摄影机位置的Z坐标。
视点X	摄影机视点的X坐标。
视点Y	摄影机视点的Y坐标。（高度）
视点Z	摄影机视点的Z坐标。

说明：

- 用SetCamera的同时，你不能定义垂直方向。要这样做，你必须用RotateX函数，或者用特殊矩阵。
- 在下面的例子中，SetCamera能够被用于工作中的完全三角系统。

//得到摄影机角度和位置常用的方法。
TVCamera.SetCamera PosX, PosY, PosZ, PosX + Cos(AngleY), PosY, PosZ + Sin(AngleY)

SetCustomProjection(投影矩阵)
更改摄影机的投影矩阵。允许使用镜面效果。

参数

投影矩阵	将要被引擎用于投影矩阵的矩阵。

说明：
仅能用于高级用户。

SetLookAt(视点X，视点Y，视点Z)
设置摄影机的视点。

参数

视点X	摄影机视点的X坐标。
视点Y	摄影机视点的Y坐标。
视点Z	摄影机视点的Z坐标。

说明：
用SetCamera的同时，你不能定义垂直方向。要这样做，你必须用RotateX函数，或者用特殊矩阵。
//使摄影机的视点在(100,200,300)
TVCamera.SetLookAt 100,200,300

SetRotation(旋转X，旋转Y，旋转Z)
设置摄影机的旋转角度。（绝对移动）

参数

旋转X	绕X轴旋转的角度。
旋转Y	绕Y轴旋转的角度。
旋转Z	绕Z轴旋转的角度。

说明：
- 在通用角度系统中，你可以用TVEngine.SetAngleSystem选择角度系统，° 或者弧度。
- 该发放使用四元数系统，避免使用万向节锁的出现。
- 该方法完全覆盖了摄影机的旋转，如果你想要相关的旋转，可以使用Rotate（X、Y、Z）函数。

```
//旋绕Y轴作递增式的旋转。
RotationY = RotationY + TVEngine.AccurateTimeElapsed * 0.002
TVCamera.SetRotation 0, RotationY, 0
```

SetRotationMatrix(矩阵)
设置摄影机的旋转矩阵，忽略转换方法。

参数

矩阵	描述摄影机旋转的3D矩阵。

说明：
矩阵的缩放应该总是为1，否则结构将是错误的。

```
//和Mesh相同的摄影机旋转。
Camera.SetRotationMatrix Mesh.GetRotationMatrix
```

SetViewFrustum(视野角度，远平面，近平面)
打开透视图，设置当前的视图属性。

参数

视野角度	视野角度的角
远平面	视野的最大距离。（假想）远平面的所有东西都不会渲染。
近平面（可选的，默认是1）	近距离平面，距离前面太远，就不会渲染。

说明：
- 如果上一个模式是等轴的摄影机，该方法将会切换回透视视图。
- 视野角度定义了你能看到东西的多少，60°和90°是比较好的。如果你缩小视野角度，你就能做出一些放大效果。
- 你应该为你的场景设置远平面最可能小的值，通常在1000到50000之间。如果设置的太大了，你将看到错误的东西，那是因为渲染的太慢，还没有渲染到这里。

```
Camera.SetViewFrustum 60,4096
```

SetViewIsometric(放大缩小数值，远平面)

开启等轴视图，为视图设置属性。

参数

放大缩小数值	等轴视图的缩放大小，取值范围是：1-10000。
远平面	视野的最大距离。在远平面以后不被渲染。

说明：

- 等轴模式是非常聪明的。
- 这是一个特殊的摄影机模式。

//建立标准的等轴模式。大小是50，最大距离是4096。

```
TVCamera.SetViewIsometric 50, 4096
```

SimpleFollowing(Mesh，位置)

使摄影机追踪Mesh。

参数

Mesh	要追踪的Mesh物体。
位置	摄影机相关联的mesh的位置。

说明：

该方法很简单，能够用于第三人称视图，如果你需要更多的跟踪函数，参考ChaseCamera。

//跟踪游戏者，摄影机在他的后面。

```
TVCamera.SimpleFollowing Player, Vector3(0, 20, -50)
```

到此，TVCamera的函数就介绍完毕了。

TVMesh

AddChild(子Mesh)

添加一个作为子Mesh的Mesh。

参数

子Mesh	新的作为子节点的Mesh

```
//读取汽车和轱辘的模型然后把轱辘作为子节点。
Set Car = Scene.CreateMeshBuilder()
Car.LoadXFile "car.x"
Set WheelTemplate = Scene.CreateMeshBuilder()
WheelTemplate.LoadXFile "wheel.x"
//创建4个轱辘。
For i = 0 To 3
Set Wheel(i) = WheelTemplate.Duplicate()
Car.AddChild Wheel(i)
Next i
```

CreateChild(名字)

创建一个mesh的子的新mesh

参数

名字（可选的，默认是空）	mesh的子的新mesh

返回新的mesh对象。

说明：

该函数就象是TVScene.CreateMeshBuilder函数，但是它会在当前的mesh基础上增加新的子节点列表。

```
//创建当前mesh的子节点。
Dim Earth As TVMesh
Dim Sun As TVMesh
Set Sun = Scene.CreateMeshBuilder("sun")
Set Earth = Sun.CreateChild("earth")
Earth.CreateSphere GetTex("EarthTexture"), 6400, 20, 20, RGBA(1.0 , 1.0 , 1.0 , 1.0)
```

DeleteChild
删除mesh的所有子节点。

说明：
引擎不会从存储器中删除该mesh，只会删除分级列表，所以子节点mesh仍然存在。
//汽车出事故了，让我们释放所有的轮子。
Car.DeleteChilds

AddCubicBezierPatch(控制点，细节，纹理)
添加一个由16个点控制的贝赛尔曲线。

参数

控制点	描述贝赛尔曲线16个控制点的3D矢量。
细节	细节，贝赛尔曲线的棋盘形嵌石饰。（最快）1=4x4 2=8x8 3=16x16 4=32x32 5=64x64 6=128x128 7=256x256（最慢）
纹理	将要被应用到贝赛尔曲线上的纹理。

说明：
曲线可以是动态的，所以你可以在每一帧设置为动态效果。

AddDynamicWater(X位置，Y位置，Z位置，细节，是否环境映射，是否涟漪，是否有腐蚀效果)
创建描述水池的动态表面。

参数

X位置	水位置的X坐标。（左上角位置）
Y位置	水位置的Y坐标。（左上角位置）
Z位置	水位置的Z坐标。（左上角位置）
细节	动态水面的细节。
是否环境映射（可选的，默认是False）	如果引擎需要用环境映射以增加反射，设为True。
是否涟漪（可选的，默认是True）	如果需要水面有涟漪效果，设为True，需要消耗更多的CPU资源。
是否有腐蚀效果（可选的，默认是False）	如果需要添加腐蚀效果，设为True，需要消耗更多的CPU和显卡资源。

说明：
你可以动态地添加水波纹，用MakeWave函数，用UpdateWater更新每一帧。

MakeWave(X，Z，广度)
用指定的广度添加动态水面波纹。

参数

X	X位置（顶点位置），取值范围：0-1。
Z	Z位置（顶点位置），取值范围：0-1。
广度	波纹的广度。取值范围从0-100的浮点数。

说明：
当你添加水波纹时，别忘记在每一帧更新水面（用UpdateWater）

```
//在循环中，更新动态水面。
Randomize Timer
Do
    Doevents
    TV.Clear
    Water.Render
    If SomeEvent = True Then
    Water.MakeWave Rnd * 32, Rnd * 32, Rnd * 20
    TV.RenderToScreen
    Water.UpdateWater
Loop
```

UpdateWater
更新动态水。

说明：
只有你用mesh创建动态水面之前，才能用这个方法。

AddVertex(X，Y，Z，颜色，TU，TV，法线X，法线Y，法线Z，是否添加点，TU2，TV2)
添加顶点到用户定义的mesh。

参数

X	新的顶点的X坐标。
Y	新的顶点的Y坐标。
Z	新的顶点的Z坐标。
颜色	淘汰了，用SetColor更改mesh的颜色。
TU	水平纹理坐标。
TV	垂直纹理坐标。
法线X（可选的，默认是0）	新顶点定义的X坐标。
法线Y（可选的，默认是0）	新顶点定义的Y坐标。
法线Z（可选的，默认是1）	新顶点定义的Z坐标。
是否添加点（可选的，默认是False）	如果你添加了顶点，设为True。
TU2（可选的，默认是0）	在第二个纹理坐标的水平纹理坐标。
TV2（可选的，默认是0）	在第二个纹理坐标的垂直纹理坐标。

返回新添加顶点的索引号。
说明：

- 你不能在已经读取的mesh或者一个内置的mesh添加一个顶点。
- 如果你想要在mesh上添加点，你必须用原视点。

```
//创建一个mesh，然后在其上面添加一个三角形。
Dim Mesh As TVMesh
Set Mesh = Scene.CreateMeshBuilder
Mesh.AddFace GetTex("face1")
Mesh.AddVertex 10, 10, 10,  0,  0, 0,  0, 1, 0, False, 0, 0
Mesh.AddVertex 40, 80, 10,  0,  0, 1,  0, 1, 0, False, 0, 0
Mesh.AddVertex 50, 30, 10,  0,  1, 1,  0, 1, 0, False, 0, 0
```

AddTriangle(纹理，X1，Y1，Z1，X2,Y2,Z2,X3,Y3,Z3,瓦片W，瓦片H，新的面，其他的面)
在mesh新增一个新的三角形面。

参数

纹理	要应用到三角形上的纹理。
X1	第一个三角形的X坐标。
Y1	第一个三角形的Y坐标。
Z1	第一个三角形的Z坐标。
X2	第二个三角形的X坐标。
Y2	第二个三角形的Y坐标。
Z2	第二个三角形的Z坐标。
X3	第三个三角形的X坐标。
Y3	第三个三角形的Y坐标。
Z3	第三个三角形的Z坐标。
瓦片W（可选的，默认是1）	水平平铺的纹理坐标数量。
瓦片H（可选的，默认是1）	垂直平铺的纹理坐标数量。
新的面（可选的，默认是True）	被淘汰的，应该总是True。
其他的面（可选的，默认是False）	被淘汰的，应该总是False。

返回新创建的三角形索引。
说明：
AddTriangle会自动调用AddFace。

```
//添加一个简单的三角形。
Set Mesh = Scene.CreateMeshBuilder("triangle")
Mesh.AddTriangle 0, 0, 0, 0, 10, 10, 10, 30, -30, 30, 1, 1, True, False
```

AddFace(纹理，是否光照)
添加一个新的面。

参数

纹理	要应用到面上的纹理。
是否光照	TV3D 6.0就淘汰了。

返回面的索引。
说明：
如果你用AddVertex创建了顶点以后，只能用AddFace创建面。其他的方法，比如AddFloor和AddWall会自动添加面。

AddFaceFromPoint(纹理，X1，Y1，Z1，X2,Y2,Z2,X3,Y3,Z3,X4，Y4，瓦片W，瓦片H，新的面，其他的面)
从4个矢量点上增加一个矩形面。

参数

纹理	要应用到面上的纹理。
X1	第一个面的X坐标。
Y1	第一个面的Y坐标。
Z1	第一个面的Z坐标。
X2	第二个面的X坐标。
Y2	第二个面的Y坐标。
Z2	第二个面的Z坐标。
X3	第三个面的X坐标。
Y3	第三个面的Y坐标。
Z3	第三个面的Z坐标。
瓦片W（可选的，默认是1）	水平平铺的纹理坐标数量。
瓦片H（可选的，默认是1）	垂直平铺的纹理坐标数量。
新的面（可选的，默认是True）	被淘汰的，应该总是True。
其他的面（可选的，默认是False）	被淘汰的，应该总是False。

AddFloor(纹理，X1，Z1，X2，Z2，海拔高度，瓦片W，瓦片H，新的面，其他的面)
在当前的Mesh上添加地板。地板是在Y轴常规的水平平面

参数

纹理	需要应用到地板上的纹理。
X1	地板左上角的X坐标。
Z1	地板左上角的Z坐标。
X2	地板右下角的X坐标。
Z2	地板右下角的Z坐标。
海拔高度（可选的，默认是0）	地板的海拔高度，在Y轴的实际数值。
瓦片W（可选的，默认是1）	纹理的水平平铺数值。
瓦片H（可选的，默认是1）	纹理的垂直平铺数值。
新的面（可选的，默认是True）	被淘汰的，应该总是Ture。
其他的面（可选的，默认是False）	被淘汰的，应该总是False。

返回新创建地板的索引。

AddWall(纹理，X1，Z1，X2，Z2，高度，海拔高度，瓦片W，瓦片H，新的面，其他的面)
创建有纹理的墙面。在一个平面上，X和Z坐标是相同的。

参数：

纹理	将要应用到墙面上的纹理。
X1	描述墙面的左上角的X坐标。
Z1	描述墙面的左上角的Z坐标。
X2	描述墙面的右下角的X坐标。
Z2	描述墙面的右下角的Z坐标。
高度	墙面高度。
海拔高度（可选的，默认为0）	墙面的海拔高度。
瓦片W（可选的，默认为1）	水平平铺因子。
瓦片H（可选的，默认为1）	垂直平铺因子。
新的面（可选的，默认为True）	被淘汰的，应该总是Ture。
其他的面（可选的，默认为False）	被淘汰的，应该总是False。

返回新创建的墙面的索引。
说明：
AddWall、AddFloor应该只能用少数几次，因为如果你调用它们很多次，就会很慢了。

```
//创建四面墙，形成一个房屋。
Dim Mesh As TVMesh
Set Mesh = Scene.CreateMeshBuilder("room")
Mesh.AddWall GetTex("Tex1"), 0, 0, 0, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 0, 40, 40, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 40, 40, 0, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 0, 0, 0, 50, 0, 1, 1
```

AdvancedCollision(起始点，结束点，要返回的碰撞结果，测试类型)
在mesh上做一个高级碰撞，这是基础的碰撞测试，返回起点到终点的碰撞结果。

参数：

起始点	测试线的起始点。
结束点	测试线的终结点。
要返回的碰撞结果	如果碰撞出现了，返回结果，包含了法线，碰撞点，碰撞的距离，还有其他信息。
测试类型	碰撞检测的类型。最快的是碰撞盒子和碰撞球，会返回少量的信息。

如果有碰撞，返回True。
说明：
该函数返回的信息，会被用于物理计算。要想得到全部的碰撞信息，你应该用TV_TESTTYPE_ACCURATETESTING模式。

```
//对游戏者检测碰撞。
Dim CollisionResult As TV_COLLISIONRESULT
IF SceneMesh.AdvancedCollision(OldPlayerPosition, NewPlayerPosition,
CollisionResult, TV_TESTTYPE_ACCURATETESTING) = True Then
    //碰撞移动，向后移动。
    PlayerPosition = OldPlayerPosition
Else
    //如果没有碰撞，游戏者继续移动。
    PlayerPosition = NewPlayerPosition
End If
```

Collision(起点，终点，监测类型，要返回的距离)

在一个mesh上进行简单的碰撞。这是基础的碰撞测试，返回起点到终点的碰撞结果。

参数

起点	测试线的起点。
终点	测试线的终点。
监测类型	将要进行碰撞测试的碰撞类型。最快的是碰撞盒子和碰撞球，会返回少量的信息。
要返回的距离（可选的，默认是0）	返回距离。

如果碰撞发生了，返回True。

```
// 对游戏者进行碰撞测试。
Dim CollisionResult As TV_COLLISIONRESULT
IF SceneMesh.Collision(OldPlayerPosition, NewPlayerPosition,
TV_TESTTYPE_ACCURATETESTING, 0) = True Then
    // 如果碰撞发生了，把游戏者向后移动。
    PlayerPosition = OldPlayerPosition
Else
    // 如果没有碰撞发生，继续移动。
    PlayerPosition = NewPlayerPosition
End If
```

GetCollisionMesh

返回碰撞对象。

如果没有碰撞，或者该碰撞不是mesh，返回空。

说明：

如果你不知道一个mesh是不是碰撞了，用这个来测试碰撞。

```
//当一个mesh发生碰撞了，设置选定的项目的材质是红色。
Set Pick = Scene.MousePicking(mx, my, TV_COLLIDE_MESH, TV_TESTTYPE_BOUNDINGBOX)
If Pick.IsCollision = True Then
    Pick.GetCollisionMesh.SetMaterial RedMaterial
End If
```

ComputeNormals
计算mesh的法线。

说明：
当你读取了一个mesh，就自动创建了法线，所以当你读取了mesh或者你读取了图形，ComputeNormals是没什么用的。
当你用AddVertex、Addface创建的mesh，ComputeNormals能用得着。

```
//创建4面墙，组成一个房屋。最后计算法线。
Dim Mesh As TVMesh
Set Mesh = Scene.CreateMeshBuilder("room")
Mesh.AddWall GetTex("Tex1"), 0, 0, 0, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 0, 40, 40, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 40, 40, 0, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 0, 0, 0, 50, 0, 1, 1
Mesh.ComputeNormals
```

ComputeSphere
计算当前mesh的绑定球。

说明：
该函数优化了剔除。当你读取了一个mesh你应该调用它。

```
//创建一个圆柱，计算它的绑定球。
Dim Cylinder As TVMesh
Set Cylinder = Scene.CreateMeshBuilder
Cylinder.CreateCylinder 300, 100, 30, True
Cylinder.ComputeSphere
```

Create3DText(文本，字体名称，字体大小，挤压，误差值，是否粗体，是否下划线)
用指定的字体名称和参数创建一个3D文本。

参数：

文本	要创建的文本内容。
字体名称	字体名称（必须是TrueType字体）。
字体大小	字体大小。单位是像素。
挤压（可选的，默认是1）	文本的深度。
误差值（可选的，默认是0）	文本的误差值，斜体的程度。
是否粗体（可选的，默认是False）	是否粗体。
是否下划线（可选的，默认是False）	是否有下划线。

```
//创建游戏者的名字为3D显示。
Set Mesh = Scene.CreateMeshBuilder("text")
Mesh.Create3DText "Player1", "Arial", 16, 30, 0, True, False
```

CreateBox(宽度，高度，深度)
创建一个立体盒子。

参数

宽度	盒子的宽度，X轴。
高度	盒子的高度，Y轴。
深度	盒子的深度，Z轴。

说明：
立方体是在世界坐标原点（0，0，0）创建。

```
//创建立方体，大小为50、30、20
Dim CubeMesh As TVMesh
Set CubeMesh = Scene.CreateMeshBuilder
CubeMesh.CreateBox 50, 30, 20
```

CreateCylinder(半径，高度，精度，是否阻力)
创建一个有阻力或者没阻力的圆柱体。

参数：

半径	圆柱体的半径。
高度	圆柱体的高度。
精度（可选的，默认是12）	低面圆的步数。取值范围6-64
是否阻力（可选的，默认是True）	如果圆柱体必须有阻力，设为True。阻力意味着圆柱体的边缘是否是封闭的。

说明：

- 你能应用纹理TVMesh.SetTexture到圆柱上。
- 只有封闭的圆柱体才是相对完美的。

```
//创建一个简单的圆柱体。
Dim Cylinder As TVMesh
Set Cylinder = Scene.CreateMeshBuilder
Cylinder.CreateCylinder 300, 100, 30, True
```

CreateSphere(纹理，半径，片，堆栈程度，颜色)
创建一个有纹理的球。

参数

纹理	应用到球上的纹理。
半径	球的半径。
片	球面的切片数量，数字越大，球就越光滑。取值范围6-64。
堆栈程度	球面的“经络”数量，数字越大，球越真实。取值范围6-64。
颜色（可选的，默认是白色）	球的颜色。

说明：
该球从坐标原点（0，0，0）创建。
通常，片和堆栈程度越大越好。

```
//创建一个球。
Dim Sphere As TVMesh
Set Sphere = Scene.CreateMeshBuilder
Sphere.CreateSphere GetTex("earth"), 6400, 20, 20, RGBA(1.0 ,1.0 ,1.0 ,1.0 )
```

CreateTeapot
创建茶壶模型。

说明：
这个只有原型模式。在游戏中你将要读取更多复杂的模型。
茶壶是测试材质和球面映射最好的材料。

```
//创建茶壶模型。
Dim Teapot As TVMesh
Set Teapot = Scene.CreateMeshBuilder("teapot")
Teapot.CreateTeapot
```

Destroy
删除mesh数据清空存储器。

```
// Destroy the mesh
Mesh.Destroy
Set Mesh = Nothing
```

DuplicateMesh(新的mesh名字)
复制一个mesh返回新的克隆的mesh对象。

参数

新的mesh名字（可选的，默认为空）	新的mesh的名称。

返回克隆的mesh对象。

```
// 创建10棵克隆的树。
Dim TRee(10) As TVMesh
For I = 0 To 9
    Set Tree(i) = TreeTemplate.DuplicateMesh("Tree" & i)
Next I
```

Enable(打开/关闭)
设为False关闭mesh的碰撞功能。

参数：

打开/关闭	设为True打开碰撞功能，False关闭碰撞功能。

说明：
在视图中看不见的你可以关闭他们的碰撞检测。
创建mesh时候，自动打开了碰撞检测。
//关闭mesh1，打开mesh2。
Mesh1.Enable False
Mesh2.Enable True

EnableSphereMapping(打开/关闭，模式)
打开mesh的球面映射。

参数：

打开/关闭	球面映射功能的开关。
模式（可选的，默认是0）	分为0和1。

说明：
能够很好地模拟反射效果。

FlipBillboard(是否左右翻转，是否上下翻转)
如果mesh是一个公告牌，可以上下左右进行翻转。

参数：

是否左右翻转	围着Y轴翻转。
是否上下翻转	围着X轴翻转。

说明：
你可以左右翻转，或者可以上下翻转或者可以在同一时刻，上下左右翻转。
//对TreebillBoard进行左右翻转。
TreebillBoard.FlipBillboard True, False

GetBoundingBox(要返回的最小盒子，要返回的最大盒子)
返回mesh的包围盒。

参数：

要返回的最小盒子	描述最小包围盒的3D矢量。
要返回的最大盒子	描述最大包围盒的3D矢量。

//示例
Dim LocMin As D3DVECTOR
Dim LocMax As D3DVECTOR
Mesh.GetBoundingBox LocMin, LocMax

GetBoundingSphere(要返回的圆心，要返回的半径)
返回mesh的范围球。

参数：

要返回的圆心	描述mesh范围球圆心的3D矢量。
要返回的半径	描述mesh范围球半径的3D矢量。

//示例
Dim LocMin As D3DVECTOR
Dim LocMax As D3DVECTOR
Mesh.GetBoundingSphere LocMin, LocMax

DeleteMaterial()
删除关联到mesh上的材质。

SetMaterial(材质索引)
应用材质到纹理上。

参数：

材质索引	返回材质索引。0是白色索引。

说明：
白色的材质是没有意义的。
//在mesh上应用“wood”
Mesh.SetMaterial GetMat("wood"), -1

GetMatrix
返回当前mesh的旋转矩阵。

返回mesh的旋转矩阵。
说明：
GetMatrix能够被用于旋转、转化和缩放。同样能被用于物力引擎。
//将mesh的矩阵用于camera矩阵。
Camera.SetMatrix Mesh.GetMatrix

SetMatrix(矩阵)
更改mesh的转换矩阵。该矩阵包含了父mesh的信息。用于高级用户。

参数：

矩阵	描述变换矩阵的3D矩阵。（变换、缩放、旋转）

说明：
就像例子那样，SetMatrix能够同时用于粘附在其他东西上，或者用于外部的物理引擎里。

```
//将mesh的矩阵用于camera矩阵。
Camera.SetMatrix Mesh.GetMatrix
```

GetMesh(mesh名)
返回有名称的mesh对象。

参数：

mesh名	要得到的mesh名。

返回有名字的mesh。
说明：

- 如果你不想保存mesh的对象，这个很有用。
- 但是因为它是基于字符串比较的函数，你应该避免在每一帧调用得过多。

GetMeshIndex
返回内部mesh的索引号。

返回mesh索引的整数。

```
//得到mesh的索引。
Index = Mesh.GetMeshIndex
```

GetMeshName
返回mesh名称。

返回包含当前mesh名字的字符串。

```
//在mouse pick以后，比较mesh的名字。
Dim MeshName As String
MeshName = PickedResult.GetCollidedMesh.GetMeshName
If MeshName = "tree1" Then
    //把树的颜色变成红色。
PickedResult.GetCollidedMesh.SetMaterial redMaterial
End If
```

GetMeshScale
返回当前mesh的缩放因子。

返回描述mesh缩放因子的3D矢量。
说明：
如果你用适当的复数，缩放能够用于反射或者翻转。

```
//如果mesh不是1比1的，变为1。
Dim ScaleM as D3DVECTOR
ScaleM = Mesh.GetMeshScale
If ScaleM.X <> 1 Or ScaleM.Y <> 1 Or ScaleM.z <> 1 Then
    Mesh.SetMeshScale 1, 1, 1
End If
```

GetModelMesh
返回关联到Actor上的mesh。

返回和Actor有相同矩阵或方向的mesh。

GetPosition
返回当前mesh的位置矢量。

返回mesh的位置矢量。
说明：
如果当前的mesh是其他mesh的子，该位置不是世界坐标位置。

```
//显示房子的位置。
Dim HousePos As D3DVECTOR
HousePos=House.GetPosition
Scene.DrawText "x:" & HousePos.X & " y:" & HousePos.y & " z: " & HousePos.z, 20,
 20
```

GetPrimitiveType
返回当前mesh的图元类型.

返回图元(CONST_TV_PRIMITIVE)类型.

```
//如果两个mesh不是相同的类型,返回错误信息.
If mesh1.GetPrimitiveType <> mesh2.GetPrimitiveType then
    RaiseError(“错误的元类型”)
End if
```

GetRotation
返回mesh绕着三个轴旋转的旋转角度.

返回mesh绕着三个轴旋转的旋转角度的3D矢量.
说明:
该函数只有在调用了SetRotation之后,才能生效.
记住:在通用角度系统中,你可以选择° 或者弧度，用TVEngine.SetAngleSystem选择。

```
//显示当前的旋转角度。
Scene.DrawText "Rotation around X : " & Mesh.GetRotation.X & VbCrLf & "Rotation
Around Y : " & Mesh.GetRotation.Y & VbCrLf & "Rotation Around Z : " &
Mesh.GetRotation.Z, 20, 20
```

GetRotationMatrix(要返回的 3D 矩阵)
返回当前 mesh 的旋转矩阵(没有变换元素)

参数:

要返回的 3D 矩阵	描述 mesh 的旋转的 4×4 矩阵。

说明:
要返回的矩阵是一个旋转的矩阵，只有左上角的 3×3 矩阵是有趣的。
//设置摄影机为旋转矩阵，以便它们共享同一个方向。
Camera.SetRotationMatrix Mesh.GetRotationMatrix

SetRotation(X 角度,Y 角度,Z 角度)
设置 mesh 的旋转.

参数:

X 角度	绕 X 轴旋转的角度.
Y 角度	绕 Y 轴旋转的角度.
Z 角度	绕 Z 轴旋转的角度.

说明:
可以用 TVEngine.SetAngleSystem 选择°或者弧度.
//绕着 Y 轴旋转 70 度,绕 X 轴旋转 90 度.
Barney.SetRotation 90,70,0

SetRotationMatrix(3D 矩阵)
设置 mesh 的旋转矩阵.(仅用于高级用户)

参数:

3D 矩阵	描述旋转矩阵的 3D 矩阵.仅仅会用到左上角的 3×3 矩阵.(因为这部分描述了整个的旋转).

说明:
该函数主要用于从外部的物理引擎,设置旋转.
仅仅会用到左上角的 3×3 矩阵.(因为这部分描述了整个的旋转).
// 示例 1:创建一个简单的旋转矩阵.
Dim Matrix As D3DMATRIX
TVMathLib.TVMatrixRotationY Matrix, 3.141592 / 2
Mesh.SetRotationMatrix Matrix
// 示例 2:应用摄影机旋转到 mesh 上.
Mesh.SetRotationMatrix Camera.GetRotationMatrix

SetTexture(纹理索引)
对 mesh 应用纹理.

参数:

纹理索引	要应用到 mesh 上的纹理索引.

//在墙面上应用纹理.
Wall.SetTexture GetTex("Wall")

GetTriangleCount
返回组成 mesh 的三角形总数.

返回描述 mesh 的三角形的总数的长整数.
说明:
Mesh 的大小总是用三角形来计算,通常叫做多边形或者面.
//添加 mesh 到 scene 的三角形总数.
SceneTriangleCount = SceneTriangleCount + Mesh.GetTriangleCount

GetTriangleNormal(面的索引)
返回面的顶点法线(垂直于 mesh 面的垂线).

参数:

面的索引	Mesh 面的索引.

返回描述法线的 3D 矢量.
说明:
用这个方法能够得到法线矢量.

GetVertexCount
返回 mesh 的顶点数量.

返回描述顶点数量的长整型数.
//添加 mesh 到 scene 的顶点总数.
SceneVertexCount = SceneVectexCount + Mesh.GetVertexCount

GetVertexInfo(顶点 id,要返回的 X,要返回的 Y,要返回的 Z,要返回的 nx,要返回的 ny,要返回的 nz,要返回的 tu1,要返回的 tv1,要返回的 tu2,要返回的 tv2)
得到指定顶点的信息.

参数:

顶点 id	要读取的顶点索引,
要返回的 X	顶点位置的 X 坐标。
要返回的 Y	顶点位置的 Y 坐标。
要返回的 Z	顶点位置的 Z 坐标。
要返回的 nx	法线矢量的 X 坐标。
要返回的 ny	法线矢量的 Y 坐标。
要返回的 nz	法线矢量的 Z 坐标。
要返回的 tu1	水平平铺的 X 坐标，0 左边，1 右边。第一个纹理坐标。
要返回的 tv1	水平平铺的 Y 坐标，0 左边，1 右边。第一个纹理坐标。
要返回的 tu2	水平平铺的 X 坐标，0 左边，1 右边。第二个纹理坐标。
要返回的 tv2	水平平铺的 Y 坐标，0 左边，1 右边。第二个纹理坐标。

说明:
你可以把这个函数用于所有的 mesh 类型.
不要滥用这些东西,频繁调用它,非常占用存储器.

```
//得到索引为 2 的顶点的位置,并移动顶点.
Dim V as D3DVECTOR
Mesh.GetVertexInfo 2, V.x, V.y, V.z, 0, 0, 0, 0, 0, 0, 0
```

InitLOD(最小顶点)

对当前的 mesh 初始化 LOD 系统.

参数:

最小顶点(可选的,默认是 1)	MeshLOD 最小的顶点数,0 是最好的 LOD 系统.

说明:
必须在引擎读取了 mesh 以后,调用该方法.

```
//打开 mesh 的 LOD 系统.
Mesh.InitLOD 0
```

SetLODIndex(LOD 索引)

设置 mesh 的 LOD 索引.你不能在每一帧都调用它.
参数:

LOD 索引	LOD 索引,1 最精确,5:不精确.

说明:

- 在 TVMesh 中,LOD 工作起来是非常随机的.
- 你能够用 SetLODIndex 或 SetLODVertexes 动态改变 mesh 的细节程度.当然,InitLOD 要在初始化 mesh 后调用.

```
//改变 mesh 的细节度为 3
Mesh.SetLODIndex 3
```

SetLODVertexes(顶点数量)
设置当前 LODmesh 的顶点数.

参数:

顶点数量	顶点数.

说明:

- 在 TVMesh 中,LOD 工作起来是非常随机的.
- 你能够用 SetLODIndex 或 SetLODVertexes 动态改变 mesh 的细节程度. 当然,InitLOD 要在初始化 mesh 后调用.

```
//设置细节度,LOD 的顶点数为 400
Mesh.SetLODvertexes 400
```

IsMeshEnabled
如果 mesh 是有回应的.

如果有回应,返回 True,无回应,返回 False.
如果 mesh 部分回应,仍然会返回 True.

```
//如果 mesh 无回应,则重新启动.
If Mesh.IsMeshEnabled = False Then
    Mesh.Enable True
End If
```


IsVisible
如果 mesh 是可见的,返回 True.

如果 mesh 在当前摄影机的视野中是可见的,返回 True.

```
//如果 mesh 是可见的,显示文本"目标在荧幕上."
If Target.IsVisible = True Then
    Scene.DrawText "Target On Screen", 50, 50, RGBA(1.0, 1.0, 0, 1.0)
End If
```

Load3DSMesh(文件名,是否转换轴心,是否居中模型,是否读取本地矩阵,是否读取材质,是否读取纹理)
读取 3DS 档.

参数:

文件名	要读取的 3DS 档的文件名,用绝对路径.
是否转换轴心(可选的,默认是 True)	因为模型是基于 OpenGL 坐标系统的,用该参数可以转换为 Direct3D 模式.
是否居中模型(可选的,默认是 True)	用这个选项可以把模型放在荧幕的中心.
是否读取本地矩阵(可选的,默认是 False)	如果 3DS 包含了一些对象,用这个选项可以位子对象得到正确的矩阵.
是否读取材质(可选的,默认是 True)	
是否读取纹理(可选的,默认是 True)	

说明:

- 通常当你读取 3DS 档,遇到一些问题时,或者出现不正常的渲染时,试试上面的参数,也许会解决问题.
- 如果纹理没有正常地映射,你应该用 UVW 贴图方式.
- 如果当前的 mesh 不是空的,调用该方法将会失败,出现通用保护性错误.

```
//读取 3ds 的球档,放入新的 mesh.
Dim Ball As TVMesh
Set Ball = Scene.CreateMeshBuilder("ball")
Ball.Load3DsMesh(mediaPath & "ball.3ds", True, True, True)
```

LoadMilkshape3DasciiFile(文件名)

读取用 MilkShape3D 制作的 3DS 档。

文件名	要读取 MS3D ASCII 档的路径。

说明:

如果 mesh 不是空的，调用这个方法将会失败，将会出现“通用保护错误”。

```
//读取用 MS3D 制作的 3D 档。
Dim Ball As TVMesh
Set Ball = Scene.CreateMeshBuilder("ball")
Ball.LoadMilkshape3DAsciiFile(mediaPath & "ball.3ds")
```

LoadSkinMesh(文件名)

读取带有皮肤的 X 动画档。

参数:

文件名	读取带有皮肤的 X 动画档的档路径.

说明:

- 你用 TVActor2 来读取带有皮肤的 X 动画档,会更好,未来版本只支持用 TVActor2 读取.
- TVActor2 的读取和渲染代码和 TVMesh 一样能做.
- 如果 mesh 不是空的，调用这个方法将会失败，将会出现“通用保护错误”。

```
//读取带有皮肤的 X 动画档. Tiny. X
Dim tiny As TVMesh
Set tiny = Scene.CreateMeshBuilder("tiny")
tiny.LoadSkinMesh mediaPath & "tiny.x"
```

LoadTVModel(文件名,是否读取纹理,是否读取材质)

读取 TVM 的静态模型。

参数：

文件名	要读取的 TV3D 定义的模型的文件名。
是否读取纹理（可选的，默认是 True）	如果引擎必须读取关联 mesh 的纹理，设为 true。
是否读取材质（可选的，默认是 True）	如果引擎必须读取关联 mesh 的材质，设为 true。

说明：

- TVM 是有 TrueVision3D 定义的，不是在所有情况下都是最快和最好的。
- 在 SDK 中有 MilkShape3D 的导出器。
- 如果 mesh 不是空的，调用这个方法将会失败，将会出现“通用保护错误”。

```
//读取名为 table.Tvm 的模型。
Dim table As TVMesh
Set table = Scene.CreateMeshBuilder("table")
table.LoadTVModel(mediaPath & "table.tvm")
```

LoadXFile(文件名,是否读取纹理,是否读取材质)

读取 DirectX 格式的 X 静态模型档.

参数：

文件名	要读取的 X 模型档(没有动画)的文件名。
是否读取纹理（可选的，默认是 True）	如果引擎必须读取关联 mesh 的纹理，设为 true。
是否读取材质（可选的，默认是 True）	如果引擎必须读取关联 mesh 的材质，设为 true。(将会影响到光线)

说明:

如果 mesh 不是空的，调用这个方法将会失败，将会出现“通用保护错误”。

```
//读取名为 table.x 的模型.
Dim table As TVMesh
Set table = Scene.CreateMeshBuilder("table")
table.LoadXFile(mediaPath & "table.x")
```

LoadXFileHierarchy(文件名,是否读取纹理,是否读取材质)
读取带有皮肤的 mesh,X 动画档.

参数:

文件名	要读取的 X 模型档的文件名。
是否读取纹理（可选的，默认是 True）	如果引擎必须读取关联 mesh 的纹理，设为 true。
是否读取材质（可选的，默认是 True）	如果引擎必须读取关联 mesh 的材质，设为 true。(将会影响到光线)

说明:
如果 mesh 不是空的，调用这个方法将会失败，将会出现“通用保护错误”。

```
//读取名为 tiny.x 的动画模型档.
Dim tiny As TVMesh
Set tiny = Scene.CreateMeshBuilder("tiny")
tiny.LoadXFileHierarchy(mediaPath & "tiny.x")
```

MoveRelative(向前,向上,横向移动)
移动角色当前的位置和朝向.

参数:

向前	向前移动的总数.
向上	向上移动的总数.
横向移动	向两边移动的总数.

```
//如果用户按了上,向前移动.
If Inp.IsKeyPressed(TV_KEY_UP) then Mesh.MoveRelative 5,0,0
```

Optimize
优化用户的 mesh,使它们更快.

说明:
当你优化了 mesh 时候,你不能重复添加顶点.
该方法不会影响读取 mesh,或者元 mesh,比如球,方盒,圆柱,因为它们已经被优化过了.

```
//添加 4 个墙面组成一间房子.然后计算法线.
Dim Mesh As TVMesh
Set Mesh = Scene.CreateMeshBuilder("room")
Mesh.AddWall GetTex("Tex1"), 0, 0, 0, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 0, 40, 40, 40, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 40, 40, 0, 50, 0, 1, 1
Mesh.AddWall GetTex("Tex1"), 40, 0, 0, 0, 50, 0, 1, 1
Mesh.ComputeNormals
//优化 mesh
Mesh.Optimize
```

Render

渲染 mesh.

说明:

- 渲染 mesh 是很重要的,在渲染了 Landscape,Atmosphere 就要渲染它.
- 如果你想要一次渲染所有的 mesh,并要进行 alpha 排序,你要使用 TVScene.RenderAllMeshes

```
Do
    TV.Clear
    Atmos.Atmosphere_Render
    // 渲染 4 个 mesh.
    Mesh1.Render
    Mesh2.Render
    Mesh3.Render
    Mesh4.Render
    TV.RenderToScreen
Loop
```

RenderShadow

渲染 mesh 的阴影.

说明:

- 阴影必须在所有的 3D 物体渲染以后,在用 TVGlobals.FinalizeShadows 以前渲染.
- 如果你打开了阴影,一个阴影将要通过指定的光从 mesh 被投影.
- 光可以是方向光或者是点光.不允许是聚光.
- 3D 显卡必须支持模板阴影已开启这个功能.
- 要用阴影功能,你必须在 32bits 模式中,且调用 TVEngine.SetShadowMandatory.

```
// 标准渲染
Do
    TV.Clear
    '渲染 3D 物体.
    Mesh1.Render()
    Mesh2.Render()
    '渲染阴影.
    Mesh1.RenderShadow()
    Mesh2.RenderShadow()
    'FinalizeShadows
    FinalizeShadows()
    Tv.RenderToScreen()
Loop
```

ResetMesh
重置 mesh,清空重新使用.
说明:
如果频繁使用它,会非常的不稳定.不要滥用!
```
//重置 mesh
Mesh.ResetMesh
// 在里面读取一个新档.
Mesh.LoadXFile("blabla.x")
```

RotateAroundMesh(目标 mesh，是否添加旋转，角度，距离，摄影机的高度)
使摄影机依指定的距离和角度，绕着 mesh 旋转。

参数:

目标 mesh	要围绕旋转的 mesh
是否添加旋转	如果必须要旋转，设为 True。
角度	要旋转的角度（° 或者弧度）
距离（可选的，默认是 0）	在摄影机和 mesh 之间的距离，如果你想保持当前的距离，设为 0.
摄影机的高度（可选的，默认是 0）	关联到 mesh 的摄影机的高度。

说明:
在 Y 轴旋转。
在通用角度系统中，你可以用 TVEngine.SetAngleSystem 选择用° 或者弧度。
```
// 绕着游戏者旋转
TVCamera.RotateAroundMesh Player, True, 0, 0, 40
```

RotateAxis(轴心,角度,是否旋转)
绕着指定的轴心旋转 mesh.

参数:

轴心	描述要围绕着旋转的自定义的轴心的 3D 矢量.
角度	绕轴心旋转的角度.
是否旋转(可选的,默认是 True)	如果旋转必须添加到当前,设为 True.

说明:
该旋转不必是空矢量,会被法线化.
```
// 绕 X 轴旋转 5°
TV.SetAngleSystem TV_ANGLE_DEGREES
Mesh.RotateAxis(Vector3(1.0, 0, 0), 5, True, False)
// 绕着(0.707, 0.0707, 0)轴心旋转 10°
mesh.RotateAxis(Vector3(0.707, 0.707, 0), 10, True, False)
```

RotateX(角度,是否相关)
绕着 X 轴旋转.

参数:

角度	旋转角度（° 或者弧度）
是否相关（可选的，默认是 True）	如果旋转必须围绕本地 X 轴旋转,设为 True。

说明:
你可以用 TVEngine.SetAngleSystem 来选择° 或者弧度。
//绕 X 轴旋转 5°
TV.SetAngleSystem TV_ANGLE_DEGREES
Mesh.RotateX 5

RotateY(角度,是否相关)
绕着 Y 轴旋转。

参数:

角度	旋转角度（° 或者弧度）
是否相关（可选的，默认是 True）	如果旋转必须围绕本地 Y 轴旋转,设为 True。

说明:
你可以用 TVEngine.SetAngleSystem 来选择° 或者弧度。
//绕 Y 轴旋转 5°
TV.SetAngleSystem TV_ANGLE_DEGREES
Mesh.RotateY 5

RotateZ(角度,是否相关)
绕着 Z 轴旋转。

参数:

角度	旋转角度（° 或者弧度）
是否相关（可选的，默认是 True）	如果旋转必须围绕本地 Z 轴旋转,设为 True。

说明:
你可以用 TVEngine.SetAngleSystem 来选择° 或者弧度。
//绕 Z 轴旋转 5°
TV.SetAngleSystem TV_ANGLE_DEGREES
Mesh.RotateZ 5

ScaleMesh(X,Y,Z)
按照指定的参数缩放 mesh.

参数:

X	缩放的 X 坐标. 正整数放大,负整数缩小.
Y	缩放的 Y 坐标. 正整数放大,负整数缩小.
Z	缩放的 Z 坐标. 正整数放大,负整数缩小.

说明:
1 是原来的大小,-1 是把原来的进行了翻转.
//示例 1,在 Y 轴翻转 mesh
Mesh.ScaleMesh 1, -1, 1
// 示例 2,把 mesh 放大两倍.
Mesh.ScaleMesh 2, 2, 2

SetAdvancedBlending(是否开启,源,目标,Alpha 测试)
设置 mesh 的 Alpha 混合颜色模拟 Direct3D 的 Alpha 混合.

参数:

是否开启	开启/关闭 Alpha 混合.
源(可选的,默认是 D3DBLEND_SRCALPHA)	源操作数.
目标(可选的,默认是 D3DBLEND_INVSCRALPHA)	目标操作数.
Alpha 测试(可选的,默认是 0)	Alpha 测试的参考值.

说明:
如果你需要一些特殊效果, 应该使用高级混合.通常是 SetBlendingMode 是你需要的.

SetBlendingMode(混合模式)
更改混合模式.

参数:

混合模式(可选的,默认是 TV_BLEND_NO)	选择CONST_TV_BLENDINGMODE模拟混合模式.

说明:
允许不同的混合模式.
//关闭混合模式.
Mesh.SetBlendingMode TV_BLEND_NO
//设置缺省的混合模式.
Mesh.SetBlendingMode TV_BLEND_ALPHA, -1

SetAlphaTest(是否开启 Alpha 测试,Alpha 测试参考值,是否写入深度缓冲区)
更改 mesh 的 Alpha 和 Z 缓冲区的行为.

参数:

是否开启 Alpha 测试	开启/关闭 Alpha 测试.
Alpha 测试参考值(可选的,默认是 128)	取值范围是 0-255,如果低于或高于这个范围,则不会渲染.
是否写入深度缓冲区(可选的,默认是 True)	打开/关闭深度缓冲

说明:
如果你打开了 Alpha 测试，引擎将会跳过所有的有 Alpha 的地域参考值的像素
该方法能够消除在物体后面错误的 Alpha 现象，比如树或者草地。
如果你用 Alpha 混合渲染出的 mesh，或者没有实心的 mesh，你应该关闭写入 Z 缓冲
默认的，Alpha 测试是被打开的。参考值设置为 8。

```
//为树木的叶子打开 Alpha 测试，参考值是 128
Tree.SetAlphatest True, 128
//关闭 Alpha 测试，关掉写入 Z 缓冲区。
Mesh.SetAlphaTest False, 0, False
```

SetBillboardType(公告牌类型)
更改公告牌 mesh 的旋转模式.

参数:

公告牌类型	公告牌的旋转类型.

说明:

- 该函数必须在 TVScene.CreateBillboard 创建了公告牌之后,调用.
- 自由旋转意味着公告牌将要围绕着三个轴心旋转,以便于面向摄影机.
- 强迫在 Y 轴旋转,意味着公告牌将只能围绕着 Y 轴旋转,对于树木或者圆柱是个好方法.

```
// 为第一个公告牌作自由旋转,强迫树绕 Y 轴旋转.
FirstBillboard.SetBillboardType TV_BILLBOARD_FREEROTATION
Tree.SetBillboardType TV_BILLBOARD_YAXIS
```

SetBumpMapping(是否开启,凹凸纹理,光索引)
设置 mesh 的凹凸贴图.

参数:

是否开启	开启/关闭 mesh 的凹凸纹理.
凹凸纹理	凹凸纹理的索引.
光索引	方向光的索引.

说明:

- 如果你用了一个点光,一个坏的凹凸纹理,凹凸映射将要失败.
- 如果引擎在显卡上,没有发现有 dot3 凹凸映射功能,凹凸映射将要失败.

//设置 mesh 的凹凸映射.
Mesh.SetBumpmapping True, GetTex("BumpTexture"), LightIndex, -1, 5)

SetCollisionEnable(是否碰撞)
开启/关闭一个 mesh 的碰撞测试.

参数:

是否碰撞	开启/关闭碰撞.

说明:
所有新创建的 mesh 碰撞一开始都是默认打开的.
当不用时,一定要关闭碰撞检测功能,这可以释放一些 CPU 的资源.
//在草地的 mesh 上,关掉碰撞检测.
GrassMesh.SetCollisionEnable False

SetColor(颜色,是否有光)
设置 mesh 的全局颜色.

参数:

颜色	Mesh 的颜色用宏 RGBA 或者 RGBA256 获得.或者用 16 进制数获得.&haarrggbb
是否匹配光(可选的,默认为 False)	如果新的材质的颜色必须要匹配与光的颜色,设为 True.

说明:
该方法覆盖了 mesh 上的颜色.它是应用缺省的颜色材质
你能用淡入淡出效果,用 Alpha 因子更改透明度.
//将球置为红色,50%的透明度.
Sphere.SetColor RGBA(1.0, 0.0, 0.0, 0.5)

SetCubicBezierPatch (控制点矢量)
动态更改贝赛尔曲线的 16 个节点,之前已用 AddCubiccBezierPatch 创建.

参数:

控制点矢量	描述贝赛尔曲线的 16 个矢量的数列.

// 设置贝赛尔曲线的节点.
Dim Node(16) As D3DVECTOR
Node(0) = Vector(0, 0, 0)
Node(1) = Vector(10, 50, 0)
Node(2) = Vector(20, 0, 0)
Node(3) = Vector(30, 10, 0)
Node(4) = Vector(0, -20, 10)
Node(5) = Vector(10, 0, 10)

```
Node(6) = Vector(20, +90, 10)
Node(7) = Vector(30, +10, 10)
Node(8) = Vector(0, 0, 20)
Node(9) = Vector(10, 0, 20)
Node(10) = Vector(20, -40, 20)
Node(11) = Vector(30, -20, 20)
Node(12) = Vector(0, 0, 30)
Node(13) = Vector(10, 0, 30)
Node(14) = Vector(20, -40, 30)
Node(15) = Vector(30, -80, 30)
Mesh.SetCubicBezierPatch Node()
```

SetCullMode(剔除类型)

更改 mesh 的剔除模式.

参数:

剔除类型	新的剔除模式.

说明:

- 前端剔除能够删除后面的三角形.
- 双面不删除任何东西并且渲染所有物体.这是默认的.
- 背面删除能够删除前面的三角形.

```
//切换为前端剔除.
Mesh.SetCullMode TV_FRONT_CULL
```

SetDeathTime(毫秒)

设置 mesh 的消失时间.

参数:

毫秒	Mesh 自动消失的延迟时间,用毫秒表示.

说明:

可以用于临时的 mesh,比如贴花或者子弹.

SetEnvironmentMap(是否打开,纹理)

打开设置 mesh 的立方体环境映射的属性.

参数:

是否打开	打开/关闭 mesh 的立方体环境映射.
纹理	要应用的环境映射的立方体纹理的索引.

```
//打开水面的立方体环境映射.
Water.SetEnvironmentMap True, GetTex("cubictexturereflection")
```

SetMeshCenter(X,Y,Z)
定义 mesh 的中心.

参数:

X	新 mesh 中心的 X 坐标。
Y	新 mesh 中心的 Y 坐标。
Z	新 mesh 中心的 Z 坐标。

//定义（20，30，10）为 mesh 的新中心。
Mesh.SetMeshCenter 20, 30, 10

SetMeshName(mesh 名字)
更改 Mesh 的名字.

参数:

mesh 名字	Mesh 的内部名字.

说明:
当你用鼠标做 mouse picking 测试时,mesh 的名字能被直接地标注出来.
//现用一个名称来标志 mesh,然后改变它.
Set Mesh = Scene.CreateMeshBuilder("stupid")
Mesh.SetMeshName "better"

SetPosition(X,Y,Z)
设置 mesh 的位置.

参数:

X	新位置的 X 坐标.
Y	新位置的 Y 坐标.
Z	新位置的 Z 坐标.

//将 mesh1 放到(0,100,50)的新位置.
Mesh1.SetPosition 0, 100, 50

SetPrimitiveType(图元类型)
更改 mesh 的图元类型.

图元类型	多边形的原始类型.

说明:
图元类型分为 3 种:"triangle list""triangle strip""point list".
Triangle list:用三个顶点绘制三角形.
Triangle strip:重复使用顶点绘制三角形.

```
// 更改图元类型为三角形带.
Mesh.SetPrimitiveType TV_TRIANGLESTRIP
```

SetShader(着色器)

设置 mesh 的 shader.

着色器	之前用 GetShader 得到的 Shader 的索引.

说明:

一个 Shader 是一小段脚本,通过 TVScene.LoadShaders 读取包含纹理渲染的信息,能够用于多种纹理的动画.

```
//读取 Shader 档.
Scene.LoadShaders "common.shader
//把 Shader"explosion"设置到 Billboard 上.
Billboard.SetShader GetShader("explosion")
```

SetShadowCast(是否开启,光源索引,影子大小)

开启/关闭 mesh 的阴影.

参数:

是否开启	开启/关闭阴影.
光源索引(可选的,默认是 0)	影子将要被投下的光源索引.
影子大小(可选的,默认是-1)	Mesh 的影子大小.

说明:

- 如果你开启了这个函数,阴影将要通过指定的光做个 mesh 的投影.
- 光线可以是点光或者方向光,不能使聚光灯.
- 3D 显卡必须支持模板阴影以便支持这一功能.
- 要用这个功能,你必须设置为 32bits 颜色,调用 SetShadowMandatory
- 调用 FinalizeShadow,要在 3D 渲染之后,在 2D 渲染之前.

```
//用 LightID 的光使 mesh 投下一个阴影.
MeshCaster.SetShadowCast True, LightID
//渲染
Do
    TV.Clear
    //渲染 3D 场景.
    SceneRenderAll()
    MeshCaster.Render()
    //渲染并最终化阴影.
    MeshCaster.RenderShadow()
    FinalizeShadows()
    TV.RenderToScreen
Loop
```

到此，TVMesh 的函数就介绍完毕了。