# Max Sdk 开发札记

## 环境配置

### appwizard

#### 安装appwizard

max的sdk开发方式需要在vs2005中安装appwizard，当然，不安装也可以，但是会相当麻烦，自己需要手动创建管理很多文件以及设置，但是使用appwizard就不需要关心了。

步骤也很简单，只要到sdk目录下的howto\3dsmaxPluginWizard文件夹中看readme.txt，然后做相应修改就可以了。

1. Open the 3dsmaxPluginWizard.vsz file (in the 3dsmaxPluginWizard directory root)

in a text editor and edit the ABSOLUTE PATH parameter to reflect the new location of the

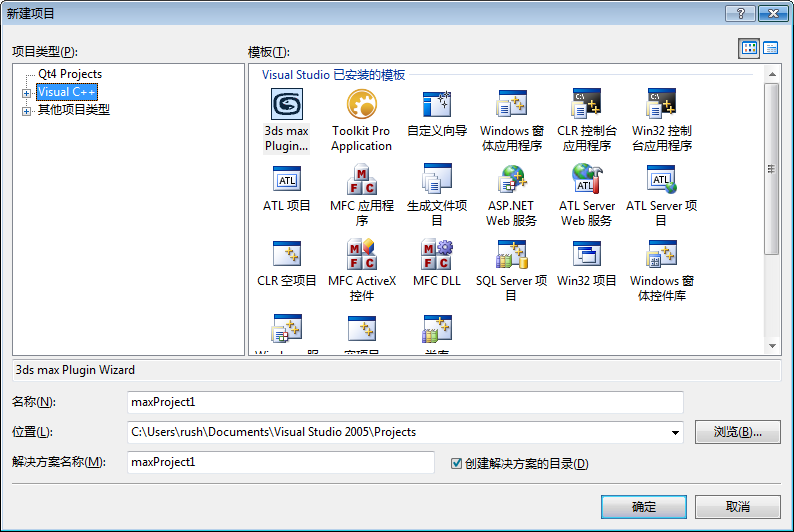
3dsmaxPluginWizard root directory. Do not add a backslash after the directory name.

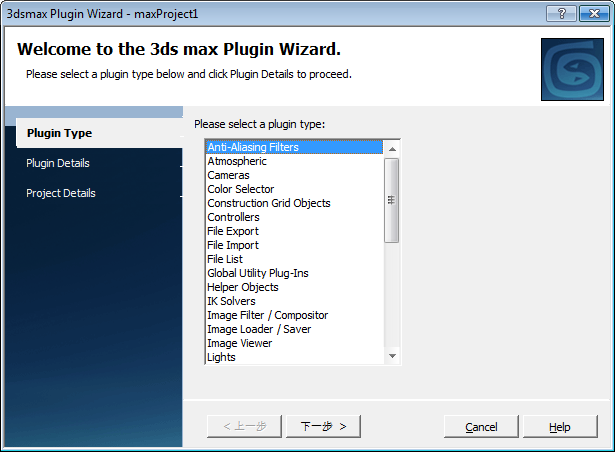
Param="ABSOLUTE\_PATH = [Absolute Path Location of 3dsmaxPluginWizard Root Directory]"

2. Copy the following files from the 3dsmaxPluginWizard root to the 'VC\vcprojects' directory under your Visual Studio install (e.g. C:\Program Files\Microsoft Visual Studio 8\VC\vcprojects):

3dsmaxPluginWizard.ico

3dsmaxPluginWizard.vsdir

3dsmaxPluginWizard.vsz



#### 将appwizard应用于vs2008

以3ds max 9为例，安装时确保同时安装了3ds max sdk。

采用Vs2008为例进行说明，按照max安装目录Autodesk\3ds Max 9 SDK\maxsdk\howto\3dsmaxPluginWizard下的readme.txt中的说明步骤：

1、找到安装目录的3dsmaxPluginWizard文件夹(我的安装目录是C:\Program Files\Autodesk\3ds Max 9 SDK

\maxsdk\howto\3dsmaxPluginWizard

打开此目录下的MaxPluginWizard.vsz 文件，编辑ABSOLUTE PATH参数为：

Param="ABSOLUTE\_PATH=C:\Program Files\Autodesk\3ds Max 9\SDK\maxsdk\howto\3dsmaxPluginWizard"

2、在根据教程复制3dsmaxPluginWizard.ico、3dsmaxPluginWizard.vsdir、3dsmaxPluginWizard.vsz到VS 2008安装目录的VCProjects下面

3、将max安装目录Autodesk\3ds Max 9 SDK\maxsdk\howto\3dsmaxPluginWizard下3dsmaxPluginWizard.vcproj文件及Autodesk \3ds Max 9 SDK\maxsdk\howto\3dsmaxPluginWizard\Templates\1033下root.vcproj文件里的 Version="8.00"改为Version="9.00"；

4、将max安装目录Autodesk\3ds Max 9 SDK\maxsdk\howto\3dsmaxPluginWizard下3dsmaxPluginWizard.vsz文件中 Wizard=VsWizard.VsWizardEngine.8.0改为Wizard=VsWizard.VsWizardEngine.9.0

5、编译的时候可能会出现如下错误：

1>cl : Command line warning D9035 : option 'Wp64' has been deprecated and will be removed in a future release

1>cl : Command line error D8022 : cannot open 'D:\Program'

修改方法：Properties->C/C++->Command Line->Addition Options去掉：/LD @D:\Program Files\Autodesk\3ds Max 2009 SDK\maxsdk\ProjectSettings\AdditionalCompilerOptions.txt

重新编译即可完成。

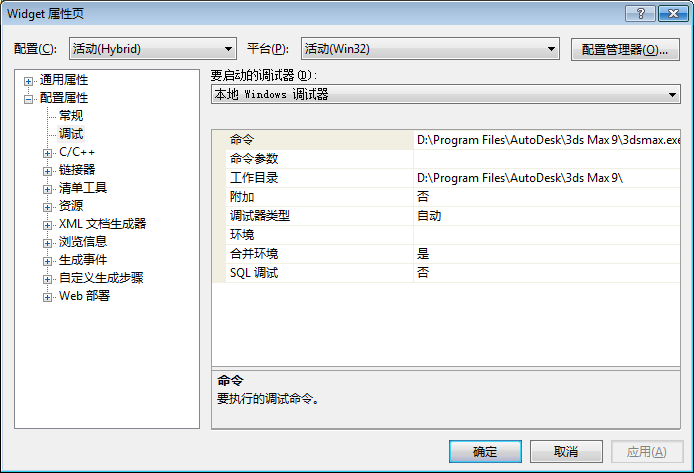
优化方案：

上面的方法是完全可行的，但是直接原地修改的话，就没法保证05和08一起使用了，所以可以拷贝一份3dsmaxPluginWizard，将其改为3dsmaxPluginWizard\_vc9，在3dsmaxPluginWizard.vsz重新指定目录就可以了。

另外上述第四步不需要，只会有兼容性问题（wizard显示有异常）。

### 调试

调试一般使用hybrid编译方案，并且需要设置max.exe为启动程序。



## Class Descriptors (类描述符)

Class descriptors将dll中的插件类的信息提供给系统。Class descriptor的一个方法同样负责这创建plugin class的新实例。开发者通过继承自classDesc2类并且实现一些方法来创建class descriptor. 下面是一个class descriptor的实例，并且它被声明为一个静态实例。

//here

接下来将介绍6个class descriptor的方法：

* IsPublic()
* Create()
* ClassName()
* SuperClassID()
* ClassID()
* Category()

### IsPublic()

如果plugin可以被用户选择并使用，这当然也是大多数情况，那么应该返回TRUE. 但是也有某些plugin被同一个dll中其他plugin 私有使用，这样情况下，则不应该出现在选择列表中。这时应该返回FALSE.

### Create(BOOL loading=FALSE)

当max想要一个该plugin class的一个实例时，它调用该函数。

可选的参数表示是否该类将从一个disk file中被load。如果返回true，那么这个plugin也许不要进行任何object的初始化操作了，loading process会把这一切做完的。

### Void DeleteThis() { delete this; }

当系统需要删除该plugin class的一个实例时，它会调用该方法。开发者通用也必须实现这样一个方法。

### ClassName()

返回类名。该名字将显示在max的ui上。

### SuperClassID()

返回该plugin class的基类的classID，classID是系统定义的描述类的常量。例如，Bend modifier返回OSM\_CLASS\_ID.所有的object space modifiers都使用这个值。有些使用：CAMERA\_CLASS\_ID, LIGHT\_CLASS\_ID, SHAPE\_CLASS\_ID, HELPLER\_CLASS\_ID, SYSTEM\_CLASS\_ID.

### ClassID()

该方法必须返回一个唯一的ID。Sdk提供了一个程序用以生产这个UID。所以如果你想使用一些sample 的源码，你需要重新提供一个UID。如果系统发现重复的UID，它只会加载第一个它找到的plugin（也会打印出一条Class\_ID confilict的错误信息）。

一个Class\_ID由两个32位的数字组成。

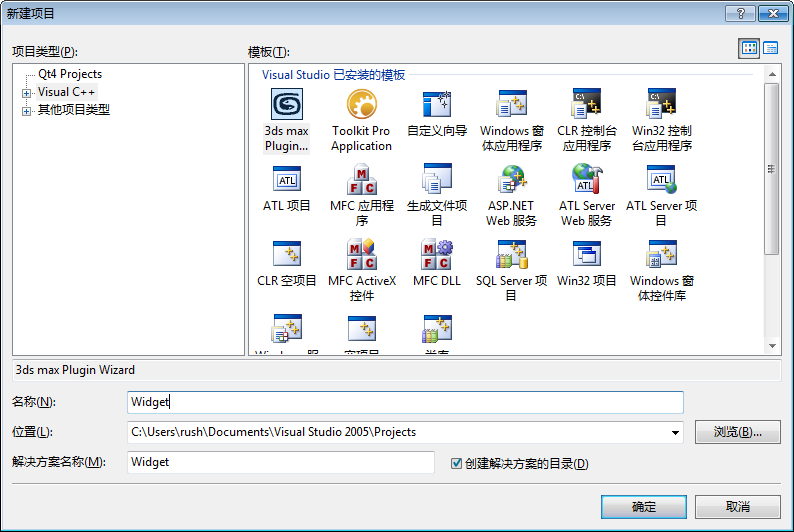
注意sample code plugins中使用0作为第二个class id项。**只有max自身ship的plugin可以将第二个class id项设为0.**开发者开发的插件，两个项都必须不为0.

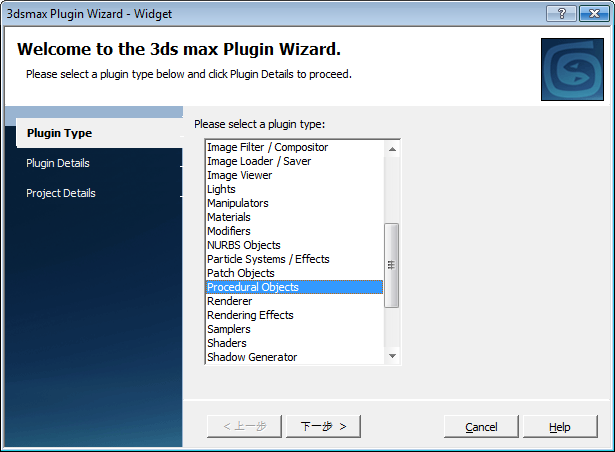
### Category()

该项出现在创建面板的dropdownlist中。如果该项被设置为已经存在的category(比如Standard Primitives)，那么这个plugin将出现在那个category中。开发者不应该把自己的plugin放到max的标准catetory中（由于max只支持在一个category中有12个plugin，所以应尽量将自己创建plugin放到一个新的category中）。如果category不存在，则该项会被创建。如果plugin不需要在此显示其名字，则只需返回一个空字符串。

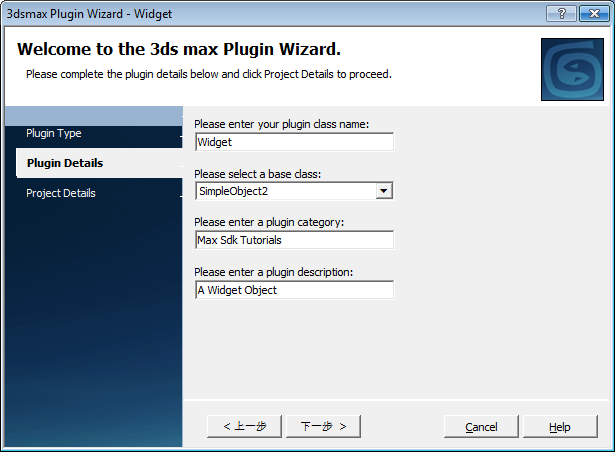
## 一个简单的程序物件插件的制作流程

### 创建project

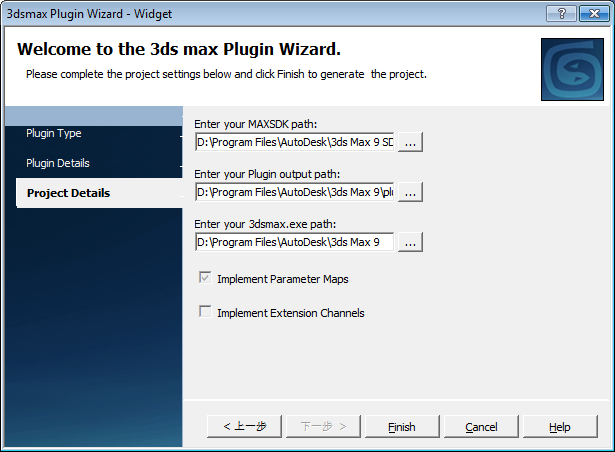




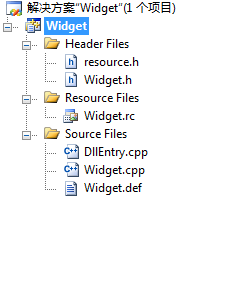
（pluginType选择程序物件）



（Plugin Details： 选择一个基类继承自SimpleObject2，Plugin Category中将会出现在创建命令面板中。）



（设置各种文件路径）



（生成的解决方案）

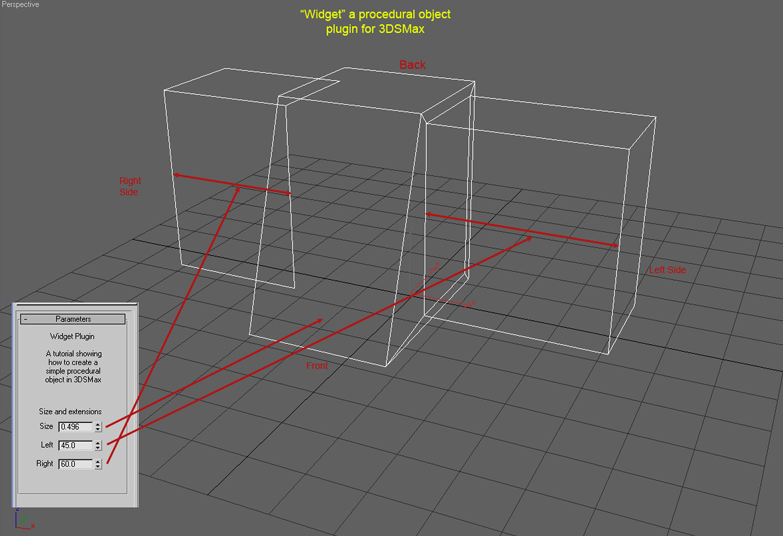
### 设计我们的Widget

1. 该widget应该可以在max的create command panel上创建。
2. 它将有3个参数来调节它的几何体属性。

Size 参数用来表示它总的大小。

Left 参数用来表示他的左翼的长度。

Right 参数用来表示其右翼的长度。



#### 创建Mesh

首先，需要在max中创建网格。这个例子，可以这样做。

创建一个box，将其转化成可编辑mesh….总之，做到和上图差不多的样子即可。

#### 导出网格数据

第二步是将点和面以c++代码的形式导入我们的插件。Simple Objects，以及几乎所有geometric objects，把他们的网格数据存在mesh这个类成员中。

我们的plugin是继承自SimpleObject2的，在SimpleOjbect2中，也有一个mesh成员变量。我们将使用它来建立点和面。我们需要：

1. 告诉mesh对象它将有多少个点以及面。
2. 设置点。
3. 设置面。
4. 我们也可以设置edge visibility flags，以及面属于的smooting groups。

设置顶点数量，将使用setNumVerts(int count)。

设置面数量，将使用setNumFaces(int count)。

设置一个点，将使用setVert(int i, Point3 position)。I表示顶点的索引，注意从0开始索引。

设置面的时候，你需要告诉它使用那些点。所以点应该先比面设置好。mesh对象有一个公共成员变量将faces。可以这样使用：

mesh.faces[3] // return a face object.

所以一部分的代码将看上去像这样：

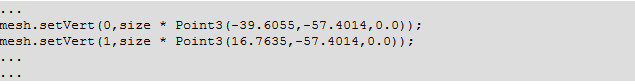
|  |
| --- |
| mesh.setNumVerts(24);  mesh.setNumFaces(44);  mesh.setVert(0,Point3(-39.6055,-57.4014,0.0));  mesh.setVert(1,Point3(16.7635,-57.4014,0.0));  mesh.setVert(2,Point3(-39.6055,22.6108,0.0));  mesh.setVert(3,Point3(16.7635,22.6108,0.0));  ...  mesh.setVert(22,Point3(-49.6055,-24.0253,78.9243));  mesh.setVert(23,Point3(-49.6055,18.0108,78.9243));  mesh.faces[0].setVerts(0, 2, 3);  mesh.faces[0].setEdgeVisFlags(1,1,0);  mesh.faces[0].setSmGroup(2);  mesh.faces[1].setVerts(3, 1, 0);  mesh.faces[1].setEdgeVisFlags(1,1,0);  mesh.faces[1].setSmGroup(2);  ...  mesh.faces[43].setVerts(20, 23, 15);  mesh.faces[43].setEdgeVisFlags(1,1,0);  mesh.faces[43].setSmGroup(0); |

将所有的代码写入c++文件中是一件非常耗时的事情。这个时候maxScript就有用武之地了。我们将使用一个maxscript来把数据都写入c++文件中。

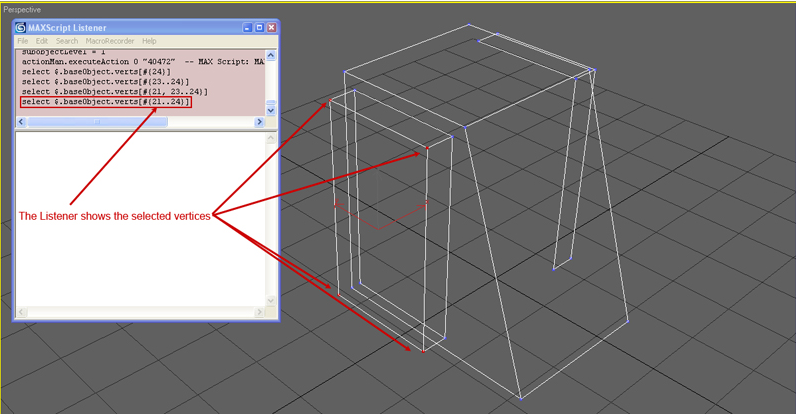
|  |
| --- |
| -----------------------------------------------------------  -- mesh2cpp.ms  -- By Simon Feltman  -----------------------------------------------------------  macroScript Mesh2CPP  category:"SDK Tools"  tooltip:"Mesh to CPP"  (  fn ExportCPP obj stream =  (  format "\tmesh.setNumVerts(%);\n" \  obj.numverts to:stream  format "\tmesh.setNumFaces(%);\n" \  obj.numfaces to:stream  for i = 1 to obj.numverts do  (  pnt = getvert obj i  format "\tmesh.setVert(%,size\*Point3(%,%,%));\n" \  (i-1) pnt.x pnt.y pnt.z to:stream  )  for i = 1 to obj.numfaces do  (  pnt = getface obj i  format "\tmesh.faces[%].setVerts(%, %, %);\n" \  (i-1) \  (pnt.x as integer - 1) \  (pnt.y as integer - 1) \  (pnt.z as integer - 1) \  to:stream  format "\tmesh.faces[%].setEdgeVisFlags(%,%,%);\n"\  (i-1) \  (if (getedgevis obj i 1) then 1 else 0) \  (if (getedgevis obj i 2) then 1 else 0) \  (if (getedgevis obj i 3) then 1 else 0) \  to:stream  format "\tmesh.faces[%].setSmGroup(%);\n" \  (i-1) (getfacesmoothgroup obj i) to:stream  )  )  if selection[1] != undefined and  (classof selection[1]) == Editable\_mesh then  (  local fname = \  getsavefilename types:"Header Files (\*.h)|\*.h|"  local stream = createfile fname  if stream != undefined then  (  ExportCPP (selection[1]) stream  close stream  )  else  messagebox ("Error opening file \"" + fname + "\"")  )  else  messagebox "You must select an editable mesh."  ) |

#### 准备参数

你会发现通过script导出的数据和上面的有一点不同：



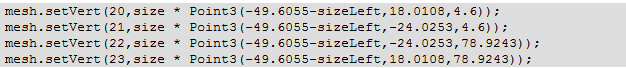
Point3被某个奇怪的size乘。在我们最初的设计中，就是提供了一个size变量，来控制widget的大小，而这个就是那玩意，或者更准确的说，是Size参数的后端。真正的size参数我们后面会实现。

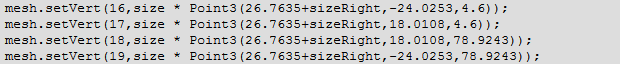


像这样用listener找出mesh两翼的点。

但是要注意max script是1开始计数的，所以要将获得index-1.

这样，我们就能实现我们的另外两个参数，left，right了，只需将这些点在x轴上加减对应的参数，比如：





##### Parameter Blocks

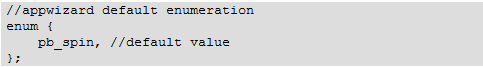
现在打开widget.cpp.我们将为我们的参数创建enumerations以及parameter block descriptor。Parameter block descriptor将自动处理很多max中的用户交互，比如创建移动rollouts，将属性暴露给MAXScript，当max保存时把值存入文件。接着我们将实现网格显示代码，最后我们要处理在视图中的鼠标消息。

###### **Size Parameter**

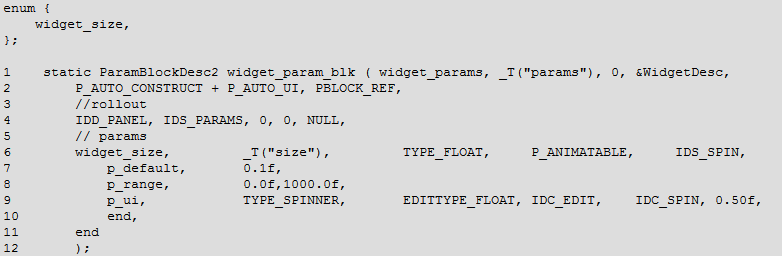
主类定义后，有两个枚举。

第一个我们不管，关注第二个。

第二个枚举appwizard给出了一个值：pb\_spin,

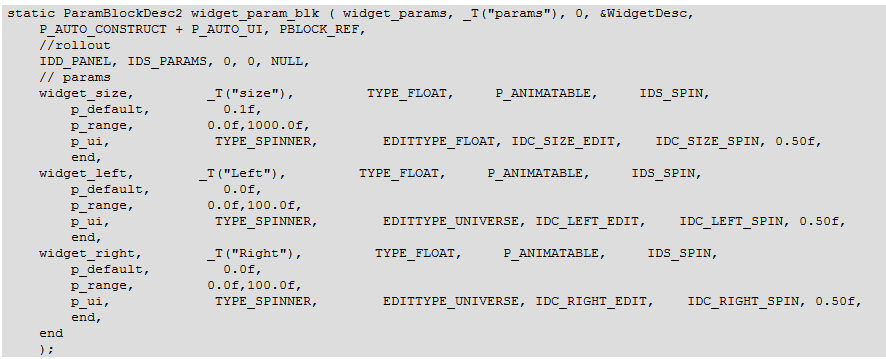


Appwizard同时也在paramBlockDesc2类型的widget\_param\_blk结构的构造函数中定义了pb\_spin。该构造函数使用了很多的参数。Appwizard已经为我们填充了足够的信息来使用参数pb\_spin，而我们可以将其改成我们需要的参数size。同时修改枚举，以及构造函数函数，当修改完后，看起来应该这样：

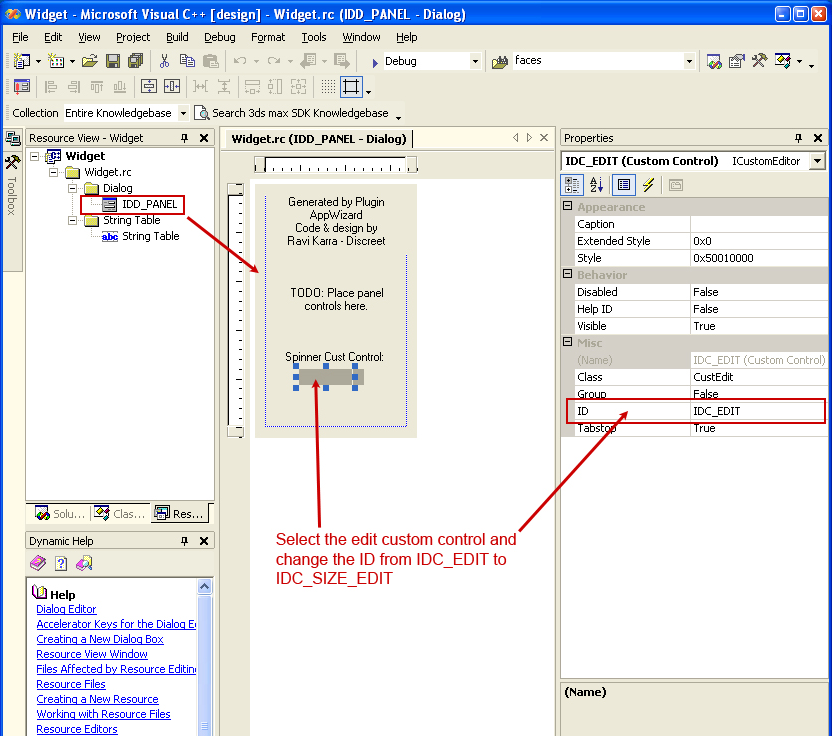


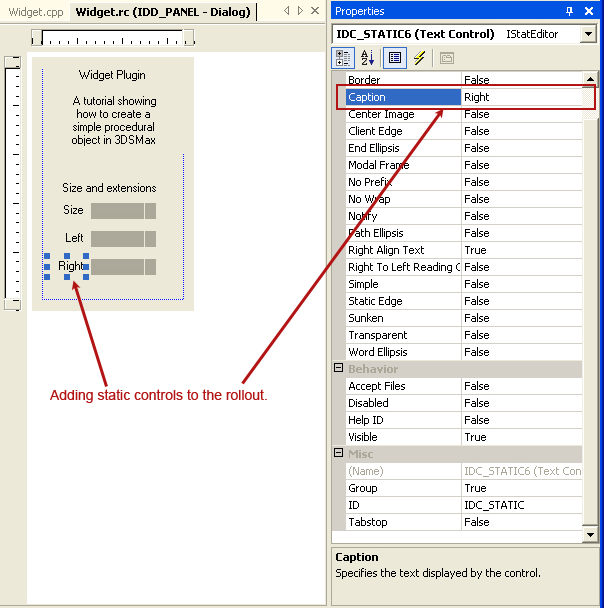
###### **Left and Right Parameters**

添加一个widget\_left和widget\_right枚举。然后扩充prameter block，另外在resource editor中添加两组控件。



#### 设置好用户界面





#### 显示网格

现在，我们将使用我们从max中导出的数据了。

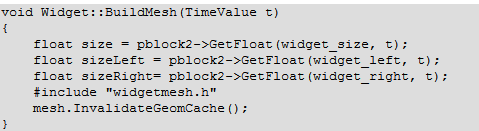
当max需要显示我们的widget，它最终会调用我们的插件中的BuildMesh函数。它会传进一个参数告诉我们的插件什么时候build mesh。插件应该把这个时间参数考虑进去（如果它是有动画属性的）以建立合适的顶点和面。在我们这个例子中，我们同样还需获取3个参数，并将其转换成float。

##### 从parameter block中抽取值

要获取size的值，我们使用GetFloat函数，在IParamBlock类中。



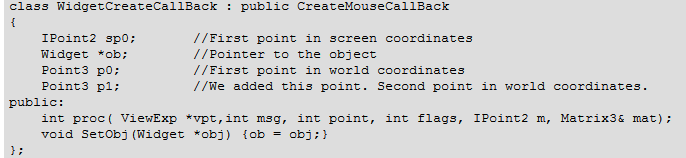
所以，buildMesh函数会看起来像这样：



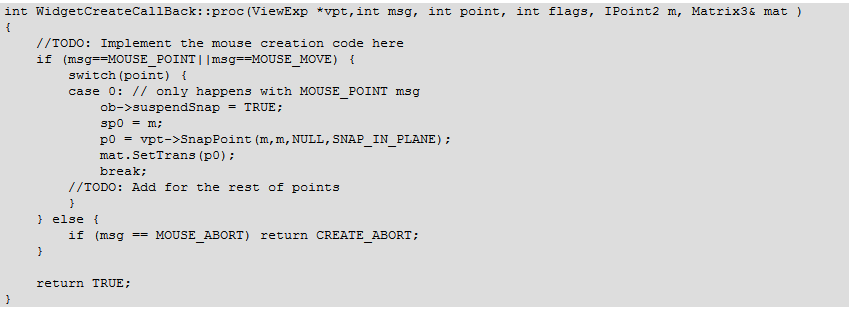
### 处理鼠标事件

最后需要做的事，是处理鼠标的点击和移动。Max需要知道如何建立widget，当用户在viewport中点击和移动时。Max通过调用你提供的回调函数，传进鼠标事件信息，来达成这个目的。而你则通过这些事件信息来更新widget。

在widget.cpp中有个类叫WidgetCreateCallBack，它唯一的用处就是包装一个windows callback function来处理你的鼠标移动和点击事件。我们新加一个point型参数p1紧这个类，这样它看起来像这样：



现在看一下默认的proc：



如果你熟悉windows的消息传递，你会知道普通的window procedures可以捕获WM\_LMOUSEDOWN, WM\_RMOUSEDOWN, WM\_MOUSEMOVE等标准的windows消息。我们的函数与此类似，但是他捕获的是max自定义的消息。比如，WM\_MOUSEMOVE被MOUSE\_MOVE替代了。标准的windows鼠标点击消息被一个特殊的MOUSE\_POINT消息代替了。当你接受到一个MOUSE\_POINT消息，max正是在通知你用户在按上或者按下鼠标（或者移动？）。总共的按上和按下此时通过point参数传入。举例：

* 用户按下鼠标左键，并移动鼠标。首先，proc获得MOUSE\_POINT信息，point值为0.然后，pro获得MOUSE\_MOVE消息，point还是0.
* 用户不再移动，并且释放鼠标左键。Proc获得MOUSE\_POINT消息，point值为1.
* 用户在此按下鼠标左键。Proc获得MOUSE\_POINT消息，point值为2.
* 用户释放鼠标左键。Proc获得MOUSE\_POINT消息，point值为3.

我们将在switch(point)结构中创建我们的物件。首先，当用户第一次click时（case 0：），我们将暂停物件的snapping。



这将组织该widget snap到它自己。接着，我们获取点击的屏幕坐标：



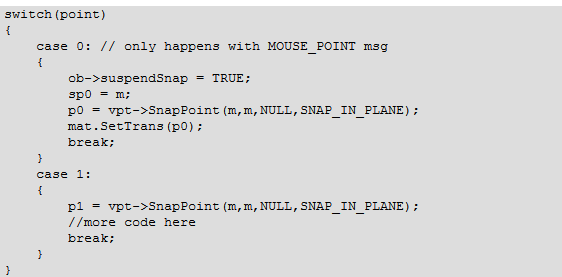
接着我们获取世界坐标。



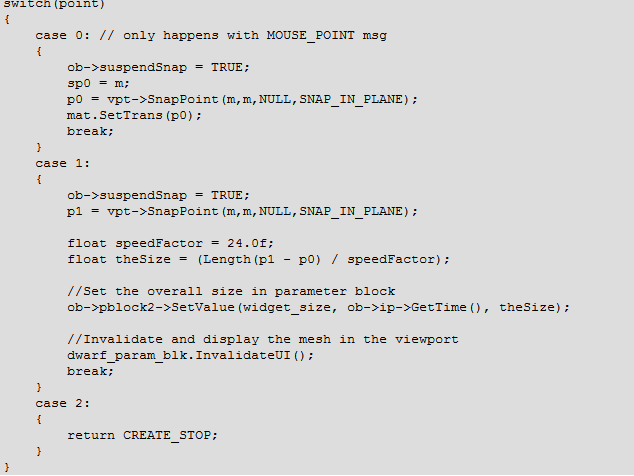
然后，我们储存一些变换信息以留作后用。



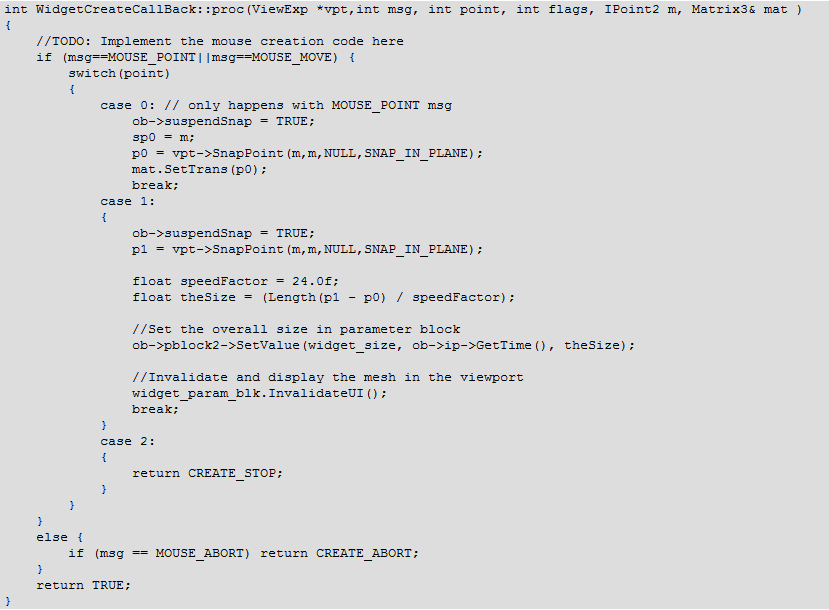
到此，我们的事件处理代码并没有做很多。当我们在viewport中点击并移动鼠标时，我们需要交互式的改变widget的size。因此我们需要获得第一次click出的pos与当前mouse pos在世界坐标轴上的距离。这个将在用户停止移动鼠标，并释放按键时发生。或者当point值为1时。因此，我们应该在case语句中加入另一个处理单元。我们将后来的鼠标位置存放紧p1中，如下：



不错，但是，我们还要更多的信息。我们需要获取两个pos间的距离，并将其直接插入我们的parameter block中。另外，需要一个速度值，以免其快速创建出一个过大的widget。

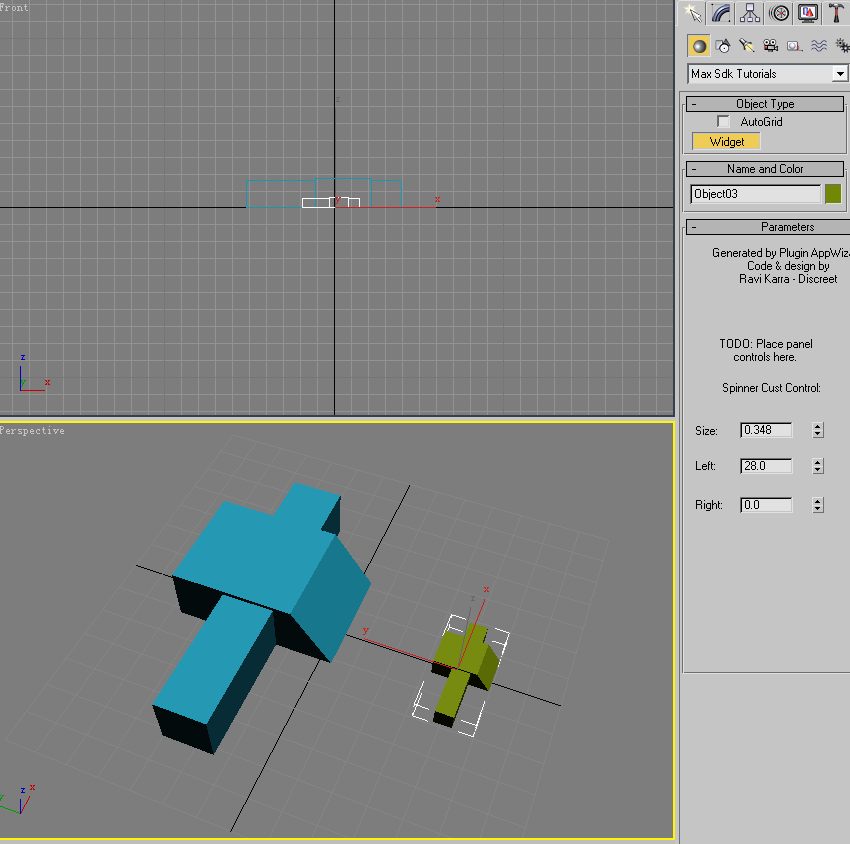


注意我们加了dwarf\_param\_blk.InvalidateUI()来显示mesh。另外，当point值为2时，返回CREATE\_STOP来结束创建的过程。最后，它应该看起来像这样：



### 完成

经过上面的制作，我们的widget插件就正式完成了，它最终的表现如：



你可以像创建一个box一样的方式来创建它，真的是太美妙了啊☺

整个工程

记：其实这个例子在max文档中，并且在howto中有完整的工程及max文件，但是我之前并没有找，所以根据文档自己的理解自己做完了这个流程，生成了上面的widget.zip。

## 一个简单修改器的制作流程

物件修改器都是应用在他们各自的变换空间中的。有些修改器使物件的几何形态发生变形，比如Bend，Taper和Twist。有些修改操纵贴图映射，另外一些则将物件的几何信息完全的用其他东西替代。总的来说，修改器可以对物件和数据有很大范围的操作。这个例子将讨论bend修改器，它可以是物体变形，弯曲现有的顶点。

有些公共的例程（类函数）是所有修改器都应该实现的。分类来说的话：

* Max需要load，classify，manage该插件的Methods。
* 在命令面板上管理修改器的参数的Methods。
* Max可以和插件在viewport中互动的Methods。
* 对物件的真正操作的Methods。
* Max场景中相关元素交互的Methods。

### 简单修改器的定义

修改器的接口被设计的尽量公用，以满足几乎各种类型的修改器的使用。但是有很大一部分的修改器尽管有用，却像但的简单。SimpleMod2类就将这些简单修改器的行为进行了封装。一些简单修改器尽可以继承自SimpleMod2而不是Modifier来获取一些预定义的行为。Bend修改器也是继承自SimpleMod2的。

作为一个简单修改器，有以下的一些假设：

* 修改器仅修改物件的几何学形态。换句话说，修改器可以作用于可变形的物件。
* 修改器使用从Deformer继承类的类的实例来做具体修改。
* 修改器的gizmo用一个3d box表示。

### 需要的插件函数

This first category of methods allow the system to load, classify, and manage the plugins present in a DLL.

The following DLL functions are required. DLL functions are described in detail [here](mk:@MSITStore:D:\Program%20Files\AutoDesk\3ds%20Max%209%20SDK\maxsdk\help\3dsMax9SdkHelp.chm::/prog/main/sdk_start_dll_functions.html).

* LibNumberClasses()
* LibVersion()
* LibClassDesc(i)
* LibDescription()

A class Descriptor is also required. Among the functions required are these important methods shown below. Class descriptors are described in detail [here](mk:@MSITStore:D:\Program%20Files\AutoDesk\3ds%20Max%209%20SDK\maxsdk\help\3dsMax9SdkHelp.chm::/prog/main/sdk_start_class_desc.html).

* Create()
* DeleteThis() (from Animatable).

These functions are required by ALL plugin DLLs except MAXScript plugins.

### 对象修改方法

像了解对象修改器与它的修改的数据之间的关系，你需要了解max中的geometry pipeline system。

#### 一个对象修改器必须告诉系统piplie中的哪个channel它将要去修改。

**相关函数**：ChannelsChanged()

**通用情况**：

要做它，你的修改器需要重载函数simpleMod：：channelsChanged().这个方法指出了修改器真正要去修改的channel。当一个修改器在修改时，channel必须被copy一份。举例来说，考虑TriObject以及一个对他的geometry channel进行修改的修改器。只有TriObject的geometry channel（verteics）需要被拷贝，在修改器被应用到该对象之前（而不是faces， UV coordinates， material assignments， 或其他的channel）。

**Bend情况**：

在Bend这个例子中，基类SimpleMod::ChannelChanged()返回PART\_GEOM. 这意味这只有geometry channel（i.e. the verteics）被真正改变。使用SimpleMod2意味这你只能修改geometry channel。在Bend这个例子中，什么也不需要做，因为父类已经完成了该任务。

#### 修改器必须指示系统在pipleline中那些channel它需要用来执行修改。

**相关函数**：ChannelsUsed()

**通用情况**：

用户必须重载SimpleMod::ChannelsUsed(). 这就指出了这个修改器需要的channel，以用来执行其修改。返回值至少应该包含ChannelsChanged()函数的内容，但是也许会更多。例如一个UV mapping的修改器也许同样需要geometry channel，即使它可能只是修改texture map channel。在simpleMod中的默认实现是：



如果一个修改器没有指定它需要的channel，那么在对象传入修改器的ModifyOjbect()的函数时，也许对象的这个channel是不正确的。

**Bend情况**：  
在Bend这个例子中，父类SimpleMod::ChannelsUsed()返回PART\_GEOM|PART\_TOPO|SELECT\_CHANNEL|SUBSEL\_TYPE\_CHANEL.这意味着它需要geometry channel(the vertics) 以及topology channel（face and polygon structures）， 以及sub-selection channel。

#### 修改器必须提供一个机制来确实修改或变形物件。

**相关函数**：ModifyObject()

**通用情况**：

当一个修改器应用于物件之际，系统会调用修改器的ModifyObject()方法。该方法负责使物件变形并且更新物件的有效期限来反映修改器的有效期限。

修改器可以有很多方式来操作物件。（请先观看working with Modifiers章节）

**Bend情况**：

Bend modifier使用Deformer方案。注意基类SimpleMod实现了ModifyObject方法，而且用到了Deformer方案，所以Bend Modifier同样也许要提供Deformer object(通过实现GetDeformer()).

#### 像Bend这样的继承自SimpleMod的modier需要实现一些接口来和SimpleMod的方法进行交互

**相关函数**: GetDeformer()

**通用情况**：

如果开发者使用Deformer callback approach，那么特需要实现方法来提供callback对象。

即实现GetDeformer()函数，而该函数将返回一个Deformer作为基类的对象实例。

**Bend情况**：

Bend modifier提供了一个继承自Deformer的BendDeformer类。当BendMod::GetDeformer（）被调用时，它会获取prameter block中的需要的参数，构建一个BendDeformer类型的对象，并返回其引用。

#### 开发者必须实现Map 方法，用来实现单个点的变形

**相关函数**：Map()

**通用情况**：

具体去实现单个点的变形的函数是Map()。 Mesh或selection中的每个点都会被Map()遍历。它操作一个点，然后返回修改过的点。

**Bend情况**：

BendDeformer::Map()函数在另外一个坐标轴上操作输入点。所以，它首先将输入点乘以一个变换矩阵来使其变换当local coordinates。当很多操作完成后，点会被变换到其原来的坐标系中。

#### 插件必须实现一个函数用以返回modifier的有效性区间

相关函数：GetValidity(t)

通用情况：

如果一个object在max scene中是静止的，但是有一个animated（活动的）modifier作用于它，系统知道随着时间的改变该modifier需要重新被计算。为了将modifier的有效性区间返回给SimpleMod, modifier会在object上调用GetValidity(t), t为时间。传入这个函数的时间随后会与区间FOREVER相交。所以当一个object在pipeline上移动时，他的有效性区间有可能会越来越小，当animated modifier作用于它上时。

Bend情况：

Bend 使用区间FOREVER来和它的活动的参数区间进行相交。这样计算了modifer总的有效性区间。

#### Modifier需要通过一个函数告知系统他需要deform那种类型的object

相关函数：InputType()

通用情况：

插件通过该函数来告知系统它想要作用的object类型，后者是可以被convert to该类型的object。

Bend情况：

基类SimpleMod已经实现了该函数，它返回defObjectClassID.

### 编辑Modifier参数

Bend modifier通过使用叫做Parameter Map的机制来管理他的用户界面参数。该机制主要由系统管理，开发者倒是不需要做过多的干涉。

#### 插件必须实现函数来处理command panel上用户输入

相关函数：BeginEditParams()和EndEditParams()

通用情况：

插件负责实现两个从Animatable class继承来的函数以处理command panel上用户输入。

* BeginEditParams()

当用户要修改该modifier的参数时（当modifier初始化，或者后面用户修改时），该函数将被系统调用。在这可以往panel上添加rollup page，并且在系统上注册rollup page。一般rollup page会在一个Dialog Proc中创建。

* EndEditParams()

当用户停止编辑modifier参数时它就会被调用。它负责反注册rollup page并删除它。

Bend情况：

通过Parameter Block2和Parameter Map2，这个工作很大程度上被自动化了。你所作的只需要获取Object Parameters（IObjParam）的指针。

#### 插件必须处理用户在command panel上的rollup page上产生的窗口事件

**通用情况**：

当用户在command panel上操作时，消息会被发送到插件的dialog procedure function。这是标准的windows 流程。所以MFC不能被用于plugin ui的这类交互。当rollup page被创建时这个dialog procedure被用来注册成该rollup page的handler。

**Bend情况**：

Bend modifier使用一个parameter map，这样就基本自动处理了大部分的事件。注意这个函数简单的返回FALSE.

#### 插件负责他自己的ui参数的储存于管理

一个modifier的参数可以是活动的，也可以是一个常量。开发者决定那个参数需要是活动的。默认的，在max中所有的参数都是活动的(animatable). 举例来说，Bend modifier的Angle和Direction参数是活动的，但是Axis则不是。

如果一个参数需要是活动的，那它必须有一个controller来控制活动。不同的参数类型需要不同的controller type。比如，一个类似于angle参数的float类型参数，需要一个float的controller。一个node transformation matrix需要一个transform controller。完整的关于参数controller管理的讨论在animation section中。然而，有一个允许开发者大大简化管理参数的机制，这个机制叫做Parameter Block。

Parameter block的一个主要目的在于控制不同的控制器控制不同参数的复杂性。开发者的工作则被大大简化了，要储存或取得活动参数的值，只需调用GetValue()和SetValue()。如果参数还没有被animated，即SetValue()还没有在time=0以外的情况被调用，就存储了一个常量。当SetValue在非time=0的情况时被调用，并且max的“Animate”按钮选中，则一个新的该类型参数的controller实例就被创建并初始化了。插件同样需要在track view中展示他们的活动参数。而使用parameter block，这一步也被自动处理了。

Bend情况：

Bend例子中，Parameter Block用来管理Angle，Direction，Axis，DoRegion，From和To参数。基类SimpleMod以及SimpleMod2管理这个parameter block。Parameter block是simpleMod2类的成员：

//here

在任何时间获取值只需

//here

或者

//here

### 3D Viewport Modifier Methods

#### 插件必须实现一个方法来在3d viewport中显示modifier的gizmo

通用情况：

Modifiers必须实现BaseObject的一个函数Display()。这个函数会在viewport中画出modifier的gizmo。

Bend情况：

在bendmod.cpp中，我们的工作被简化了，因为SimpleMod基类有提供一个默认的实现。默认的实现将gizmo画成一个附着在object上的3d box。注意开发者拥有完全的GraphicsWindow drawing routines的access right，所以他们可以按照他们的想法任意的绘制他们想要的gizmo形态，只要他们不是继承自SimpleMod。

#### 插件必须实现一个方法叫HitTest(). 这个用来检查鼠标是否和modifier的gizmo相交了

#### 插件必须实现一个函数GetWorldBoundBox()来计算modifier gizmo的bounding box

通用情况：

该方法在SimpleMod中通用被实现了。这个返回的是world space bounding box。该方法不必很精确，但应无求快速。

Bend情况：

基类simpleMod已实现。

### 如果modifier支持sub-object selection，插件必须提哦那个方法允许系统操控modifer的sub-object

当一个modifier的参数正在被编辑了，它可以选择像系统提供不同等级的sub-object selection。有两种sub-object selection：

* 1. 选择mdifier的Gizmo或者Center。Modifier可以在scene中有一个可视化的表现用于用户操作。这可以简单到是一个3d box或者复杂到一个FFD lattice。Bend modifier使用一个3d box。当gizmo在max ui的sub-object selection list中被选中时，用户可以变换这个修改的gizmo本身—他们可以调整的location，orientaion，以及scale。当center被选中是，可以修改这个mdifier的gizmo的pivot point。
  2. 第二种sub-object selection的形式是选择object延着pipeline的分解部分。如果object在pipeline中的形式是一个TriObject, 则sub-object selection level也许就是vertex，face，edge等。举例说，这些正是Edit Mesh modifier的sub-object selection level.

当BeginEditParas()函数被调用时，modifier告诉系统他将使用哪种sub-object selection level.

在我们的例子中，bend有Gizmo与Center两种sub-object levels.

当用户在sub-object drop down中切换选择是，modifier会被告知。与此同时，modifier应该提供一个继承自CommandMode的类的实例来支持move，rotate，uniform scale，non-uniform scale，以及squash mods。这些modes替换了他们object层级的响应部分，虽然用户然仍会通过工具栏的标准命令来使用他们。开发者不需要支持所有这些modes，可以只支持move，因为rotate和scale也许对某类插件没有作用。插件不支持的modes将在工具栏上灰掉。除了move/rotate/scale，插件也可以有其他modes。比如Edit Mesh modifier有一个extrude mode，用户可以点击然后拖动以extrude mesh的面。

幸运的是，开发者并不总是需要实现所有这些逻辑。在我们的例子中，SimpleMode就实现了所有逻辑。

### References

这部分的内容关于modifer如何处理与scene中其他相关元素之间的关系，如何交互。

**函数**：

* NotifyRefChanged()
* NotifyDependents()
* NumRef()
* GetReference()
* SetReference()

**通用情况**：

在max的架构中，scene中的元素经常在各自之间产生dependencies。这些dependences是通过References来处理的。References是双路链接，它可以被想象成一个reference maker与一个reference target间的一个记录。Reference maker依赖于reference target，当target在某种情况下变换时，maker会被通知以采取适当的行动。

Max有一个messaging system用来通知在场景中的依赖元素发生改变的情况。Modifier负责该messaging system的某些方面。首先，通过创建对于一个item的reference来建立reference dependency(使用ReplaceRefernce())。再次，向依赖于该modifier的objects发送消息(使用NotifyDependents()).第三，处理来自于依赖项的变动信息(使用NotifyRefChanged()).

插件同时需要实现一个NumRef()函数用于表面它reference的元素数目。函数GetReference(i)和SetReference(i)用来允许系统设置该插件的reference。

**Bend情况**：

因为只有一个parameter block被使用了，bend不需要则额外的操作，所有这些函数都在SimpleMod2中被实现了。如果有多个parameter block，则你需要重载这些函数。

### Undo/Redo System

开发者应该建立好插件，这样max的undo system就能充分的发挥作用了。

## Working with Objects

### Working with Modifiers

这个主题介绍了在max中如何修改objects。讨论了objects与modifiers是如何交互以使objects可以进行修改， 解释了object修改的多种类型，并且展示了各种修改类型的部分关键代码。

#### Working with Modifiers

##### Objects，Modifers，and the Geometry Pipeline

Modifers是沿着geometry pipeline来操控objects的。Max使用geometry pipeline这样的系统来进行objects的修改。Objects与modifers都实现了一些方法来使这个系统正常运转。首先，我们先看一下modifers需要做什么来通知max它们的修改特性。然后我们来探讨下procedural objects需要做什么来使它们自己”可修改化”.

##### Modifers Request a Certain Type of Input

Modifiers也许只能操纵一类object。比如，Edit Spline modifier只能作用于spline shapes。Volume Select modifier只能作用于TriObjects（triangle mesh objects）。Modifer通过实现Modifier::InputType来告知max它将作用于何种object。这个方法将返回该类object的Class\_ID。比如，Edit Spline modifier如此实现该方法：



这样就告诉了max该修改器将作用于何种object。只有本身是spline shapes，或者可以转化自己成spline shapes的objects，才能被该modifier修改。Volume Select modifier也是同样，它需要triangle mesh来作为输入，它的实现是：



当一个object在max中被选中了，只有object匹配的modifier才会被启用。这个是通过查询modifiers的InputType()是否符合选中object的Class\_ID或者可以转化成此来实现的。

##### Modifiers Need To Indicate Which Geometry Pipeline Channels They Require and Alter

Objects沿着geometry pipeline会被分成几个分离channels。Pipleline只能操作objects的部分而不是整体。比如，Objects可以分解成geometry部分（points），topology 部分（polygons or faces），object 的selection level(object, face, vertex, etc...), 以及texture coordinates。这么做的原因是pipeline可以以更有效率的方式运作。

Modifers必须通知max他们将那些channels进行修改。这些channels将被拷贝一份并传入pipeline。Max只会传入最低的channels需求进入pipeline。如果一个channel并没有被任何modifier在任何地方需求，则它既不会被拷贝也不会被传入pipeline。

Modifiers通过Modifier::ChannelsUsed()来通知max它们需要使用那些channels。Channel清单应该包含modifier确实会修改的channel，同样也能包含更多其他channel。这里有个Volume Select modifier的例子：



注意OBJ\_CHANNELS被定义为：



这意味这pipeline需要更新所有上面channel的的数据然后才传递给modifer来进行操作。技术上来说，这么多channel对于这个modifer其实有点过于夸张了。它其实用不到TEXMAP\_CHANNEL,VERTCOLOR\_CHANNEL，它真正使用到的只有GEOM\_CHANNEL, TOPO\_CHANNEL, SELECT\_CHANNEL.

Modifier同样需要通知max那些channel是它真正要修改的。通过Modifer::ChannelChanged()。这里是Volume Select modifier的相应实现：

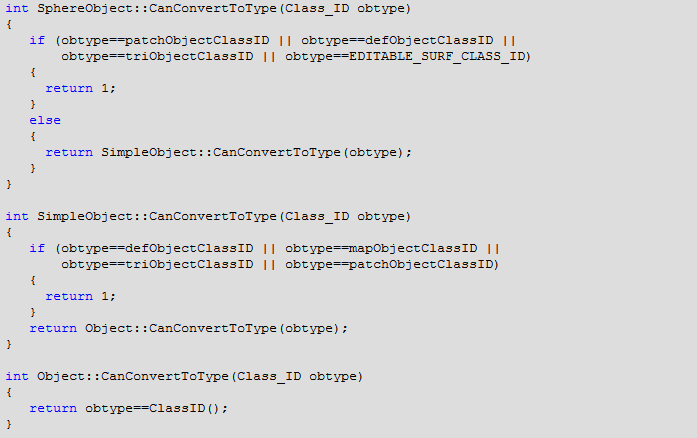


这显示了该修改器修改了selection channel， sub-object channel，geometry channel。注意geometry channel被修改了因为Mesh类的verteSel BitArray使顶点被选中了。

##### Objects May Need to Convert Themselves

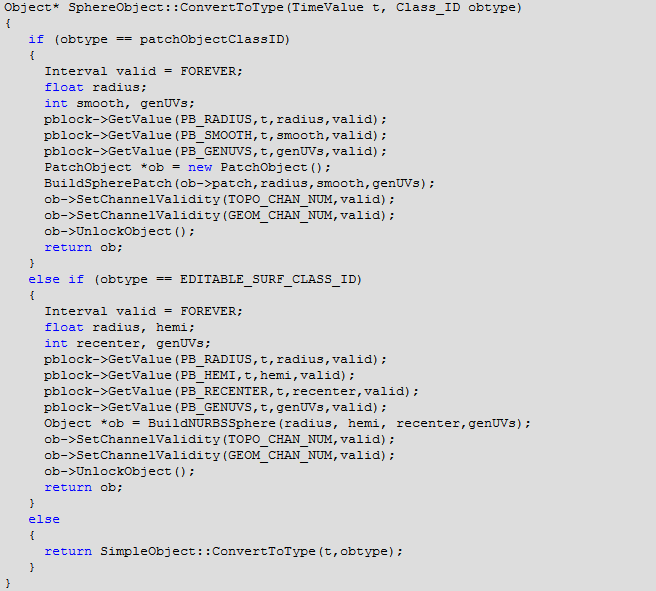
Procedurual objects并不总是以一个合适某种modifer的形式出现。考虑一下一个max的用户想在一个procedural sphere上放置一个volumue select modifier以选择某些点。Procedural sphere并不是通过点和面来定义的，而是通过他的radius， segment count，hemisphere setting等程序参数。但是用户却可以在procedural sphere上使用volume select modifier。原理是max首先会问object是否可以转换成一个triangle mesh object（有vertics和faces），如果object回答是，则max会让要求object转换成triangle mesh object。如此，volume select modifier就可以作用于它了。为此，procedural objects需要实现两个函数，object::CanConvertToType()以及Object::ConvertToType().

Procedural sphere的canConvertToType()的实现如下：



注意sphere在被问及是否可以转换成path object，deformable object(a generic object with points that can be modified), triangle mesh object, editable surf object(NURBS object)时回答yes。其他情况下，它调用父类simpleObject的实现。如果SimpleObject还是无法识别则会调用父类Object的实现。

ConvertToType()函数用来将object转换成指定的Class\_ID的类型。下面是procedural sphere的实现：



##### Summary

最后，modifiers需要更新objects的有效性区间来反映他们自己的有效性。这个通过UpdateValidity()完成。

#### The Modifier::ModifyObject() Method

Max支持很多中类型的modifiers，他们被他们操控的geometry pipeline上的何种channel来进行划分。Modifiers同样也可以操控一个或多个channel。以下列表显示了pipeline被分解成多种channel：

|  |
| --- |
| The pipeline is divided into the following channels:  GEOM\_CHANNEL  The vertices of the object. Most modifiers only alter this channel.  TOPO\_CHANNEL  The topology channel, i.e. the face or polygon structures. Smoothing groups and materials are also part of this channel. Edge visibility is also part of this channels since it is an attribute of the face structure.  TEXMAP\_CHANNEL  The texture vertices and procedural mappings.  MTL\_CHANNEL  This is no longer used. Materials are rolled into the Face data structure and are part of the topology channel.  SELECT\_CHANNEL  The sub-object selection channel. An object's selection flows down the pipeline. What the selection is actually comprised of is up to the specific object type. For example, TriObjects have bits for face, edge and vertex selection. This channel is the actual BitArray used (like selLevel of the Mesh class).  SUBSEL\_TYPE\_CHANNEL  This is the current level of selection. Every object that flows down the pipeline is at a certain level that corresponds to the Sub-Object drop down in the 3ds Max user interface. This channel indicates which level the object is at. This is also specific to the object type. There are 32 bits to represent the level of selection. When all the bits are 0, the object is at object level selection.  DISP\_ATTRIB\_CHANNEL  These are miscellaneous bits controlling the item's display. These bits are specific to the type of object. For the Mesh object these are the surface normal scale, display of surface normals, edge visibility and display flags.  EXTENSION\_CHANNEL  This channel is available in release 4.0 and later only.  This is the channel used by extension channel objects.  VERTCOLOR\_CHANNEL  This is the color per vertex channel. This is also used for the second texture mapping channel.  GFX\_DATA\_CHANNEL  This channel is used internally by 3ds Max for stripping. Plug-In developers don't need to specify this channel as being changed or used in their plug-ins.  TM\_CHANNEL  This is the ObjectState TM that flows down the pipeline. This TM may be modified by modifiers.  GLOBMTL\_CHANNEL  This is no longer used. Materials are rolled into the Face data structure and are part of the topology channel.  The following #defines may be used to specify groups of channels:  #define OBJ\_CHANNELS (TOPO\_CHANNEL | GEOM\_CHANNEL | SELECT\_CHANNEL | TEXMAP\_CHANNEL | MTL\_CHANNEL | SUBSEL\_TYPE\_CHANNEL | DISP\_ATTRIB\_CHANNEL | VERTCOLOR\_CHANNEL | GFX\_DATA\_CHANNEL | DISP\_APPROX\_CHANNEL | EXTENSION\_CHANNEL)  #define ALL\_CHANNELS (OBJ\_CHANNELS|TM\_CHANNEL|GLOBMTL\_CHANNEL)  Note: Some of the sample code specifies these channels as PART\_\* as opposed to \*\_CHANNEL. For example, PART\_GEOM|PART\_TOPO instead of GEOM\_CHANNEL|TOPO\_CHANNEL. The proper usage is the \*\_CHANNEL version. |

Modifier::ModifyObject()，是真正改变这些object的函数，它的定义：



#### Modifiers that change Topology

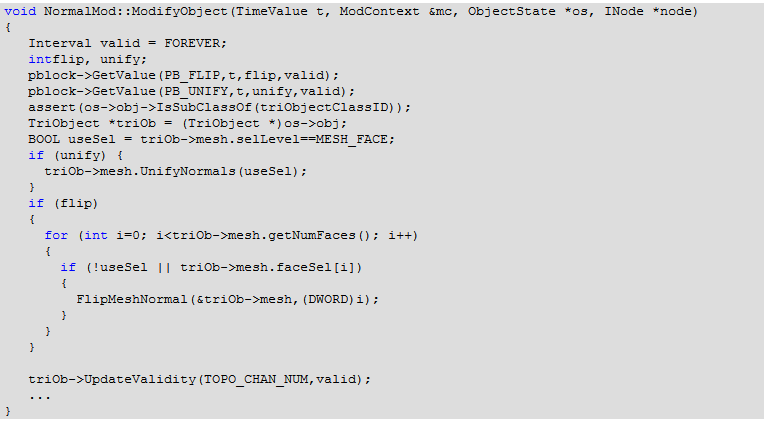
这类修改器修改objects的face或polygon structures。Smooting groups以及materials同样包含于topology channel。Edge visibility也是，因为它是face structure的一个属性。 另外还有Face normals。下面例子源于Normals modifier，它允许用户unify以及flip一个mesh的face normal。全部的源程序位于\MAXSDK\SAMPLES\MODIFIERS\SURFMOD.CPP.

下面是几个函数的实现：



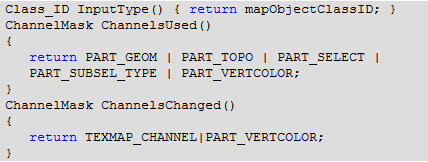
注意到这个modiifier只作用与triangle mesh object或者那些可以把自己转换成triangle mesh object的objects。它需要在使用前更新OBJ\_CHANNELS，并且仅仅修改GEOM\_CHANNEL和TOPO\_CHANNEL.

下面是它的ModifyObject()的实现。注意，当modifier完成修改，它会去给你更新object的相应channel的有效性。

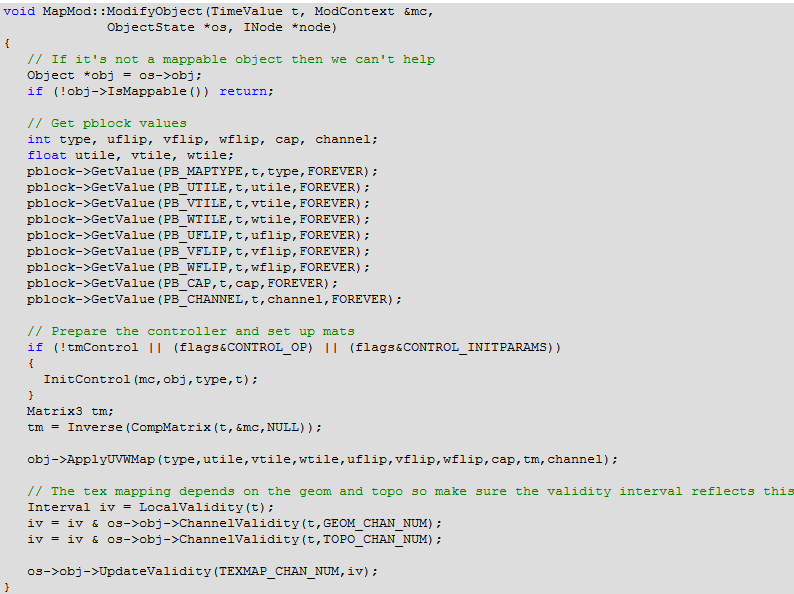


#### Modifiers that change mapping(Texture Coordinates)

该类修改器修改objects的texute coordinates。他们可以为unmapped objects增加新的mapping coordinates，也可以修该objects已有的mapping coordinates。一个增加新的mapping coordinates的修改器就是UVW Mapping Modifier。完整的源程序在\MAXSDK\SAMPLES\MODIFIERS\MAPMOD.CPP, 下面是它的几个实现函数：



修改器需求修改对象是mappable的。而该类型也实现了object::ApplyUVWMap()方法。



#### Modifiers that change Materials

这类修改器修改object的material ID。一个例子是Material modifier(MAXSDK\SAMPLES\MODIFIERS\SURFMOD.CPP).下面是他实现：



