Частное учреждение образования
«Институт современных знаний имени А. М. Широкова»

Кафедра высшей математики и информатики

СОГЛАСОВАНО
Проректор по учебной и научной работе
Козлович М. И.

_____
01.02.2018 г.

ОСНОВЫ ЯЗЫКА MAXSCRIPT

Электронный учебно-методический комплекс
для студентов специальности 1-19 01 01 Дизайн (по направлениям),
направление специальности 1-19 01 01-06 Дизайн (виртуальной среды)

Составитель
Васильева Ю. Д., кандидат технических наук, доцент кафедры высшей математики и информатики частного учреждения образования «Институт современных знаний имени А. М. Широкова»

Рассмотрено и утверждено
на заседании Совета Института
протокол № 7 от 27.02.2018 г.

УДК 004.9(078.5)
ББК 32.973-018.1я73

Р е ц е н з е н т ы:

кафедра «Системы автоматизированного проектирования» Учреждения образования Белорусский национальный технический университет (протокол № 8 от 26.02.2018 г.);

Гривачевский А. Г., заведующий лабораторией моделирования технологических процессов в Объединенном институте проблем информатики НАН Беларуси, доктор технических наук, доцент.

Рассмотрено и рекомендовано к утверждению
кафедрой высшей математики и информатики
(протокол № 7 от 01.02.2018 г.)

О75 **Васильева, Ю. Д.** Основы языка MAXScript : учеб.-метод. комплекс для студентов специальности 1-19 01 01 Дизайн (по направлениям), направление специальности 1-19 01 01-06 Дизайн (виртуальной среды) [Электронный ресурс] / Авт.-сост. Васильева Ю. Д. – Электрон. дан. (0,9 Мб). – Минск : Институт современных знаний имени А. М. Широкова, 2018. – 85 с. – 1 электрон. опт. диск (CD).

Систем. требования (миним.) : Intel Pentium (или аналогичный процессор других производителей) 1 ГГц ; 512 Мб оперативной памяти ; 500 Мб свободного дискового пространства ; привод DVD ; операционная система Microsoft Windows 2000 SP 4 / XP SP 2 / Vista (32 бит) или более поздние версии ; Adobe Reader 7.0 (или аналогичный продукт для чтения файлов формата pdf).

Номер гос. регистрации в НИРУП «Институт прикладных программных систем» 1201815224 от 14.04.2018 г.

Учебно-методический комплекс представляет собой совокупность учебно-методических материалов, способствующих эффективному формированию компетенций в рамках изучения дисциплины «Основы языка MAXScript».

Для студентов вузов.

ISBN 978-985-547-231-6

Пояснительная записка

Учебно-методический комплекс (УМК) по дисциплине «Основы языка MAXScript» разработан на основе учебной программы Института для студентов специальности 1-19 01 01 Дизайн (по направлениям) от 30 июня 2017 г., регистрационный номер № УД-02-311/уч. и предназначен для студентов дневной формы обучения.

УМК представляет собой совокупность учебно-методических материалов, способствующих эффективному формированию компетенций в рамках дисциплины «Основы языка MAXScript», изучается на протяжении одного семестра.

УМК включает в себя курс лекций, планы лабораторный занятий, вопросы для самоподготовки к экзамену, учебную программу дисциплины, список литературы и ресурсов сети Интернет для освоения полного объема знаний, соответствующего стандартам высшей школы.

УМК поможет студентам освоить язык программирования в среде программы Autodesk 3dsMax, который позволяет создавать и отлаживать сценарии для построения пользовательского интерфейса, автоматизировать операции преобразования объектов, текстурирования, освещения, анимации и визуализировать сцены. Знания, умения и навыки, приобретенные в ходе изучения дисциплины, позволят студентам использовать современные программные средства для эффективного решения на высоком уровне специализированных задач, возникающих при дизайн-проектировании виртуальной среды.

В учебно-методическом комплексе рассматриваются основные принципы и методы создания сценариев на языке MAXScript: работа с объектами, назначение им свойств и изменение параметров, изучаются функции для работы с условиями и циклами. Студенты знакомятся с основами объектно-ориентированного программирования в пакете компьютерной графики Autodesk 3ds Max.

1. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1.1. Краткий курс лекций

Тема 1. Основы языка MAXScript [1]

Сценарий

Средства MAXScript позволяют взаимодействовать со сценой 3ds Max и автоматически управлять действиями или операциями из сценария посредством операторов в форме текста. 3ds Max интерпретирует содержимое сценария, а затем выполняет соответствующие действия. Если операторы написаны неверно, сценарий выполняется неправильно или же вообще не выполняется. Поэтому последовательность и организация команд сценария, называемая синтаксисом, должна быть достаточно точной. Отклонения от правильного синтаксиса приводят к ошибкам в сценарии. Как правило, в MAXScript уведомляется о том, что сценарий содержит недоступное для правильной интерпретации содержимое.

Очень важно сохранять ясность и удобочитаемость своих сценариев. Для создания удобочитаемых сценариев пользуйтесь отступами с табуляцией и пробелами, чтобы выделить отдельные элементы или функции. В 3ds Max символы пробелов и табуляции игнорируются во время обработки сценариев. В примерах сценариев, приведенных в этой теме, демонстрируется один из способов организации содержимого сценариев с отступами. Отступами можно пользоваться и по-другому, но при этом всегда нужно быть последовательным.

Пример сценария, где каждая из строк сценария является оператором:

```
/*Построим примитивы Box и Sphere*/
mybox = box()
mysphere = sphere()
```

При написания сценария пользуйтесь следующими правилами:

1. Сохраняйте ясность и удобочитаемость своих сценариев.
2. Следуйте четкому синтаксису написания.

3. Пользуйтесь комментариями.

Команды сценария могут быть выполнены несколькими способами. Самый простой из них – воспользоваться приемником команд MAXScriptListener (рис. 1), который обеспечивает выполнение команд MAXScript в диалоговом режиме. Приемник команд удобен для последовательного выполнения отдельных строк кода или проверки правильности этих строк.

MAXScript Listener можно вызвать через главное меню **MAXScript > MAXScript Listener** или клавишей<F11>.

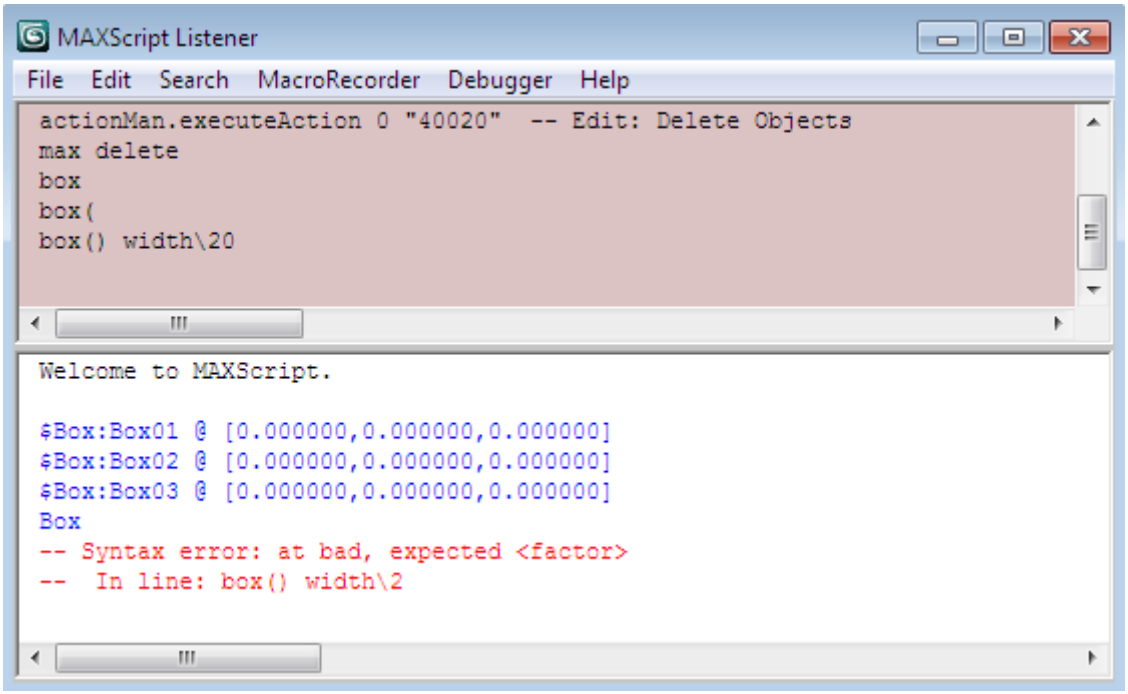

Рис. 1. ОкноMAXScript Listener

Еще один способ записи сценария – окно **MAXScript Editor** (рис. 2). Оно вызывается через главное меню **MAXScript > New Script** при создании нового скрипта, **MAXScript > Open Script**, если необходимо открыть уже созданный скрипт для редактирования.

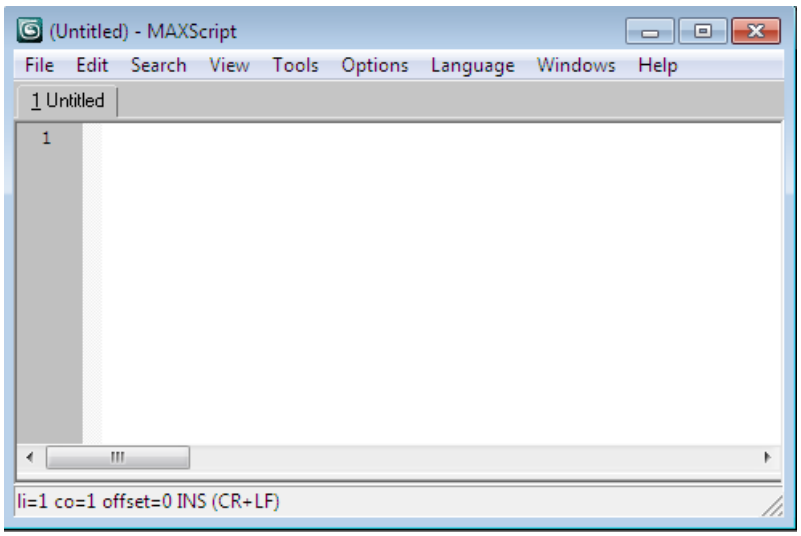

Рис. 2. Окно MAXScript Editor

Данное диалоговое окно предназначено для написания сценариев. Чтобы выполнить сценарий, необходимо выбрать в главном меню Tools>Evaluate All или комбинацией клавиш <Ctrl+E>.

Существует несколько видов сценариев. Их можно приближенно разделить по типу пользовательского интерфейса, который они формируют.

• Сценарная функция – состоит из одной или более функций, но не формирует пользовательский интерфейс. После выполнения сценария такие функции можно вызывать из приемника команд или другого сценария.

• Сценарная утилита – это сценарий, определяющий один или более свитков на панели Utilities в качестве элементов своего пользовательского интерфейса.

• Сценарный подключаемый модуль – это специальный вид сценария, создающий новые инструменты 3ds Max или расширяющий уже существующие. Во всех остальных видах сценариев применяется код для выполнения действий, которые уже доступны из пользовательского интерфейса 3ds Max, например создание параллелепипедов и сфер, назначение контроллеров и конфигурирование видовых окон. Сценарный подключаемый модуль позволяет создавать новые геометрические объекты, карты, модификаторы и прочие элементы оформления сцены. Пользовательский интерфейс нового инструмента появляется в соответствующей части интерфейса 3ds Max. Например, новая карта

появляется в окне Material/Map Browser, тогда как новый модификатор – в списке Modifier List.

• Общие сценарии – любые сценарии, которые могут быть отнесены к категории общих. Такие сценарии позволяют сформировать перемещаемое диалоговое окно в качестве элемента пользовательского интерфейса либо вообще отказаться от интерфейса.

Как правило, сценарий сохраняется в файле с расширением .ms. Для хранения сценариев в организованном порядке макросценарии следует сохранять в файлах с расширением .mcr. Формально оба вида сценариев можно сохранять в файле с любым из двух указанных расширений, и при этом сценарий будет выполняться. Однако правильно выбранное расширение указывает на конкретный вид используемого сценария.

Сценарий можно выполнять автоматически при запуске 3ds Max. Для организации такого режима работы сценария достаточно поместить сценарий в любую из следующих папок.

• 3dsmax/stdplugs/stdscripts;

• 3dsmax/plugins или любую подпапку внутри папки с подключаемыми модулями;

• 3dsmax/ui/macroscripts или в любую другую подпапку

• 3dsmax/scripts/startup.

При запуске 3ds Max программа запуска осуществляет поиск файлов с расширениями .ms, .mcr и .mse в папках, перечисленных в указанном выше порядке. (Расширение .mse используется для шифрованных файлов сценариев.

Комментарии

Комментарии позволяют сделать сценарий более понятным для других. Они представляют собой текстовые блоки в сценарии, написанные простым и понятным языком. Они поясняют или документируют назначение сценария и порядок его выполнения. Старайтесь снабжать комментариями все свои сценарии. Это особенно важно для крупных и сложных сценариев.

Комментарии не относятся к выполняемым операторам сценария. Они выделяются в тексте сценария двойным дефисом (- - который ставится в начале комментария. В таком случае любой текст, начинающийся с двойного дефиса, игнорируется. Двойной дефис можно, если требуется, использовать в нескольких строках, например:

```
b = box()
b.length= 20.0 -- Это комментарий.
-- А это еще один комментарий, имеющий продолжение в
-- этой строке. Следующая строка уже относится к самому сценарию.
b.width= 30.0
```

В следующем примере показано, каким образом комментарий помещается в той же строке, что и оператор сценария:

```
b = box() -- Создать параллелепипед, комментарий игнорируется
```

Комментарии могут занимать несколько строк, если они начинаются двумя символами /* и оканчиваются еще двумя символами */ (или признаком конца файла). Все, что находится между этими символами, считается комментарием. Комментарии, определенные подобным образом, называются *блочными комментариями,* например:

```
s = sphere()
/* Здесь содержится большой комментарий, не требующий ограничителей в каждой строке, поскольку в данном случае используются блочные комментарии.*/ s.radius= 10.0
```

Такие комментарии можно поместить и в одной строке, внутри текста самого сценария:

```
s = sphere/*большой радиус*/ radius:100.0
```

Многострочные операторы

Операторы сценария выполняются, или интерпретируются, построчно. Но иногда оператор MAXScript может оказаться достаточно длинным. Поэтому для размещения длинных операторов в нескольких строках служит символ обратной косой черты (\), указывающий на продолжение оператора. В следующем примере показан длинный оператор, записанный в нескольких строках.

Исходная команда сценария:

torus radius1:10 pos:[0, 0, 20] wirecolor; [225, 230, 100]

Та же команда, разбитая на несколько строк:

```
torus radius1:10 \
pos:[0, 0, 20] \
wirecolor: [225, 230, 100]
```

В каждой новой строке текст сценария указан с отступом. В 3ds Max пробелы игнорируются, а благодаря отступам сценарий легче читать как его автору, так и другим создателям и пользователям сценариев MAXScript.

Переменные и данные

В любом языке программирования отдельные элементы данных представлены переменными. Переменные – это заполнители, или контейнеры, представляющие (и хранящие) данные в программе. У каждой переменной имеется свое имя, которое ей присваивается. Например, переменные могут иметь следующие имена: **a**, **b**, **x** и **countNum**.

Данные, хранящиеся в переменной, называются *значением*, а порядок установки переменной, равной конкретному значению, называется *присвоением* этого значения переменной, например:

x = 5

В данном примере значение **5** присваивается переменной **x**, а данные относятся к числовому типу. Значение может иметь любой тип данных, распознаваемый в MAXScript. Чаще всего используются следующие типы данных: числовые, строковые и логические.

Числовые типы данных

Имеются две разновидности числовых типов данных: целые числа и числа с плавающей точкой.

- Целое число – это положительное или отрицательное десятичное число без дробной части. Например: 0,1,2, -10,345.
- Число с плавающей точкой – это положительное или отрицательное десятичное число с дробной частью. Например: 0.0,33.3,0.75, -5.8.

Если определить переменную и присвоить ей значение числового типа данных, над такой переменной можно будет выполнять математические операции, например:

```
-- Присвоить значение 5.0 переменной x.
x = 5.0
-- Присвоить значение 6.0 переменной y.
y = 6.0
-- Умножить x на y и присвоить результат переменной z.
 z = x * y
-- Теперь z = 30.
```

Конкретный тип данных определяется при присвоении переменной значения, которое может иметь дробную часть, например:

```
-- Присвоить целое значение переменной x.
 x = 5
-- Присвоить переменной x значение с плавающей точкой
x = 5.0
```

Если математические операции выполняются над переменными, содержащими только целые числа, результат всегда будет целым числом. Если же

хотя бы одна переменная содержит число с плавающей точкой, в результате получится число с плавающей точкой.

Строковые типы данных

Данные строкового типа содержат текст. Например, в переменной **msg** можно поместить текст **"File not found."** (Файл не найден), а затем отобразить его для пользователя:

```
msg = "File not found."
messagebox msg
```

Текстовые значения должны всегда указываться в кавычках, чтобы отличить их от данных других типов, например:

```
num = 5.0
message = "5.0"
```

В данном случае математические операции можно выполнять над переменной **num**, но не над переменной **message**. В MAXScript предоставляются самые разные функции для манипулирования строками.

Логические типы данных

В MAXScript часто применяется еще один тип данных – логический (или булев). Логические данные могут принимать следующие значения: **true** (истина), **on** (включено), **false** (ложь) или **off** (выключено).

```
ready = true
ready = off
```

Значение **on** равнозначно значению **true**, тогда как значение **off** равнозначно значению **false**.

Логические данные используются, главным образом, для проверки условий в сценарии с помощью условных операторов.

Тип данных Point3

Для определения позиции, цвета, масштаба объекта используется тип данных Point3. Значение типа **Point3** обозначается тремя числами в квадратных скобках, например **[20, 45, 10].** Для выполнения преобразований эти три числа обозначают координаты **x**, **y** и **z** положения объекта (в данном случае x = **20,** y = **45, z = 10).** Так, для изменения положения эти числовые значения соответствуют положению объекта по осям X, Y и Z.

boxpos:[20,45,10]

Функции

Выражение

box()

называется *функцией.* Функция – это операция, обозначаемая именем функции. В приведенном выше примере имя функции – **box**. В имени функции используются круглые скобки, чтобы уведомить 3ds Max, что в данном случае вызывается функция формирования параллелепипеда. Функция иначе еще называется *вызовом функции.*

В результате ввода выражения

b = box()

вызывается функция **box**. В итоге вся информация о созданном параллелепипеде присваивается переменной **b**. Вы уже знаете о числовых, строковых и логических типах данных, однако переменная b к ним не относится. При создании объекта выбирается тип данных, характерных для конкретного объекта.

Имена функций не зависят от регистра. Иными словами, имя функции **Box()** равнозначно имени **box()**.

При создании объекта MAXScript формирует ссылку на объект. Ссылка представляет собой идентификатор, дескриптор или маркер, предоставляющий

доступ к объекту и всем его свойствам. Ссылка – это не просто имя объекта. Аналогично числовым, строковым и логическим типам данных для переменных существуют типы данных и для объектов. Так, функция **box()** создает сначала параллелепипед в сцене, а затем возвращает ссылку на этот параллелепипед. Если теперь потребуется обратиться к параллелепипеду, достаточно будет воспользоваться ссылкой **b**.

Функции играют очень важную роль в MAXScript, как, впрочем, и в любом языке программирования. Функция представляет собой одну или несколько инструкций, предписываемых компьютеру для выполнения.

Выше уже был показан один из способов применения функции с использованием ее имени и круглых скобок, например **box()**. В функциях можно также передавать отдельные значения, указываемые после имени функции и используемые в ней для выполнения конкретного задания. Так, три или четыре значения, указываемых после имени функции выбора цвета, определяют конкретный цвет, например:

myRedColor = color55 10 255

Подобным образом указываются аргументы функции. Если после имени функции следуют круглые скобки, данная функция вызывается без передаваемых аргументов. А если аргументы передаются функции, круглые скобки опускаются.

Для нормального выполнения некоторых функций необходимо всегда указывать один или два аргумента (т.е. *список аргументов).* Так, для функции выбора цвета требуется указать три или четыре аргумента. Однако для многих функций указывать аргументы необязательно.

Аргументы передаются функции одним из следующих способов.

- В определенном порядке.
- В любом порядке, но после ключевого слова, обозначающего соответствие аргумента и конкретного значения.

В определенном порядке с необязательными ключевыми словами, которые следуют после указываемых по порядку аргументов. Например, у функции могут быть два обязательных аргумента, которые должны быть указаны перед тремя необязательными аргументами. Необязательные ключевые слова могут следовать в любом порядке. Функция такого типа является самой сложной из всех трех рассмотренных типов функций.

Число аргументов наряду с их конкретными типами и порядком следования называется *обозначением функции.*

Например, обозначение функции выбора цвета состоит из имени функции и трех последующих целых чисел: **color целое_число целое_число целое_число**.

Функция создания параллелепипеда имеет ряд дополнительных параметров, которые могут следовать в произвольном порядке, но требуют указания ключевых слов. Покажем на конкретном примере, каким образом указываются такие параметры функции.

1. Введите в скрипте следующее выражение:

b1 = box length:20.5 width:15.0 height:5.6

Сместите параллелепипед в сторону.

2. Далее введите следующее выражение:

b2 = box height:5.6 length:20.5 width:15.0

В итоге будет создан второй параллелепипед с такими же значениями свойств, как и у первого параллелепипеда. Результат выполнения данной функции не меняется, хотя порядок следования аргументов иной. Ключевые слова указывают функции, какие именно значения соответствуют длине, ширине и высоте параллелепипеда.

В MAXScript имеется функция **LoadMaxFile** для загрузки файла сцены 3ds Max. Обозначение данной функции состоит из следующих элементов.

- Обязательное строковое значение с именем файла, которое должно следовать непосредственно после имени функции.
- Два или более необязательных аргументов, сопровождаемых ключевыми словами.

Если функция выполняется успешно, она возвращает логическое значение **true,** а если ее выполнение окажется неудачным (например, при отсутствии указанного файла), она **возвратит** логическое значение **false.**

Ниже приведены четыре разных, но вполне допустимых способа вызова данной функции:

result = LoadMaxFile"c:/scenes/trucks.max"
result = LoadMaxFile "c:/scenes/trucks.max" useFileUnits:true
result = LoadMaxFile "c:/scenes/trucks.max" quiet:trueuseFileUnits:false
result = LoadMaxFile "c:/scenes/trucks.max" useFileUnits:falsequiet:true

Обратите внимание на два последних примера. В обоих случаях функции передаются одинаковые аргументы, изменен лишь порядок следования двух необязательных аргументов.

Свойства объектов

Необходимо не только создавать объекты, но и просматривать и изменять их свойства. Любой объект, создаваемый в 3ds Max, имеет ряд свойств, или атрибутов, определяющих этот объект.

Для обращения к свойствам объектов служит переменная ссылочного типа. Для доступа к конкретному свойству параллелепипеда **b** перед именем этого свойства ставится точка (.), как в следующих примерах:

b.length
b.width
b.height

Свойства объектов можно распознать по названиям соответствующих параметров, появляющихся на панели **Create** или **Modify** при создании объекта непосредственно в видовом окне.

Для обнаружения имеющихся свойств объекта конкретного типа служит функция **showProperties**:

showProperties <узел>

Функцию **showProperties** следует применять к переменной, содержащей сам объект, а не его тип. Для этого необходимо использовать следующий фрагмент кода:

```
b = box()
showProperties b
```

Для видоизменения свойств объекта достаточно установить новые значения этих свойств. Например, в следующей строке кода устанавливается длина параллелепипеда – 40 единиц:

box.length=40

Некоторые свойства являются общими для всех находящихся на сцене объектов, в том числе примитивов и вспомогательных объектов. Такие свойства не перечисляются при выполнении функции **showProperties**, но и они могут быть использованы. Например, у каждого объекта данного типа имеется свойство **.имя**, **.положения**, **.цвет**.

Для доступа к свойству положения можно воспользоваться следующей строкой кода:

```
s = sphere()
s .pos = [2,30,40]
```

А для доступа к отдельным составляющим этого значения следует указать **.x**, **.y** или **.z** после свойства pos. Например, для доступа к координате X положения сферы достаточно воспользоваться следующей строкой кода:

s.pos.x = 40

Аналогичные свойства имеются у преобразований вращением и масштабированием.

С помощью свойства **.wirecolor** выбирается цвет каркаса объекта. Аналогично положению объекта его цвета могут быть представлены типом данных Point 3. В частности, три числовых значения обозначают основные цвета RGB (красный, зеленый и синий соответственно). Каждое такое значение является целым числом в пределах от 0 до 255 и соответствует значениям RGB в селекторе цвета.

- Белый цвет: [255, 255, 255]
- Черный цвет: [0, 0, 0]
- Чистый красный цвет: [255,0,0]
- Ярко-желтый цвет: [255, 255, 0]
- Умеренно-синий цвет: [0,50,1501

Новый цвет каркаса объекта назначается с помощью свойства **.wirecolor**, например:

s.wirecolor = [40,120,200]

Для изменения цветов в любой момент достаточно получить доступ к подсвойствам **.r**, **.g** и **.b**, например:

s.wirecolor.r = 156

Кроме того, цвет можно сохранить в переменной с помощью функции **color**, а затем назначить его для свойства **. wirecolor** объекта:

newColor = color40 120 200

s.wirecolor = newColor

Для указания альфа-составляющей цвета служит следующая конструкция, состоящая из четырех цветов вместо трех:

newColor = color 20 120 200 128

Четвертым параметром в данном выражении является альфа-составляющая цвета (т.е. альфа-канал). Доступ к ней осуществляется с помощью свойства **.a**:

alphaNum= newColor.a

Локальные и глобальные переменные

Терминами «локальная» и «глобальная» обозначается область действия переменной. Область действия определяет место в коде сценария MAXScript, где переменная оказывается доступной. Но как только переменная будет объявлена как глобальная, она окажется доступной в любом сценарии и в любой момент. Локальная переменная может использоваться в том блоке кода, в котором она определена, либо во вложенных в него блоках кода. Блок кода представляет собой любой фрагмент кода, заключенный в круглые или квадратные скобки.

Таким образом, правильнее было бы сказать, что локальная переменная объявляется во вложенном блоке внутри сценария и оказывается недоступной в конце сценария, т.е. за пределами области ее действия.

Созданные до сих переменные действительны вплоть до выхода из 3ds Max. При вводе выражения **x = 5** в окне редактора MAXScript Editor или приемника команд фактически объявляется глобальная переменная **x**. Глобальная переменная сохраняет свое значение вплоть до выхода из 3ds Max, даже если закрыть окно редактора сценариев или приемника команд либо установить 3ds Max в исходное состояние.

Глобальная переменная представляет собой любую переменную, объявленную или определенную вне всех блоков кода.

Глобальную переменную можно также объявить явно. Ниже приведены примеры правильного объявления глобальной переменной:

global a
global a = 2
a = 2 -- если переменная определена внутри сценария, но не блока

В целом, применения глобальных переменных следует избегать. Ведь если объявить глобальную переменную в сценарии, который может быть использован в каком-нибудь другом месте 3ds Max, а затем перезаписать значение такой переменной, то другие сценарии или подключаемые модули могут работать не вполне предсказуемым образом.

Для ограничения области действия переменной ее достаточно объявить как локальную. Локальные переменные действительны в пределах блока кода, в котором они определены, или же во вложенных в него блоках. В этой связи блок кода иногда еще называется *контекстом области действия.* Примеры блоков кода будут представлены при рассмотрении циклов и условных операторов далее в этой главе.

Локальная переменная объявляется несколькими способами. Ниже приведены все действительные способы объявления локальной переменной **b**(напомним, что она должна быть внутри блока):

(local b)
(local b = 2)
(b = 2 – если переменная определена внутри блока)

Если переменная объявлена как локальная вне *всех* блоков кода, компилятор выдаст сообщение об ошибке «No local declarations at top level»(Объявления локальных переменных на верхнем уровне отсутствуют).

Как только локальная переменная будет объявлена, она может быть использована в том же блоке кода, где она была определена, или же во вложенных в него блоках.

Условные операторы

Условный оператор позволяет управлять процессом выполнения программы и может быть реализованы разными способами.

К первому типу условных операторов относится оператор **if/then**

```
if s.pos.x == 10 then
        s.radius = 40
```

В приведенном выше примере оператор изменяет радиус сферы только в том случае, если значение координаты **x** ее положения равно **10**. Двойным знаком равенства (= =) обозначается операция логического сравнения, тогда как одиночным знаком равенства – операция присваивания.

В условном операторе непременно должны присутствовать слова **if** и **then**.

В одном условном операторе может быть указано несколько условий. Для этой цели служат логические операторы НЕ (not), И (and) и ИЛИ (or). Ниже приведен ряд соответствующих примеров.

```
if (not s.radius == 10) then
        messagebox "Radius is not 10."
if (x == 5 and y == 6) then
        z = 0
if (x == 5 or y == 6)
        then z = 0
```

Если оператор not окажется не совсем удобным, условие «не равно» можно проверить и по-другому, используя синтаксис**! =**, например:

```
If s.radius != 10 then
        messagebox "Radius is not 10."
```

При проверке нескольких условий, как правило, требуются круглые скобки для группирования условий в логическом порядке, что особенно важно для

сложных выражений. Выражения в круглых скобках всегда вычисляются в первую очередь. Каждый из операторов в следующем примере дает разные результаты:

```
if (x == 5 or y == 6) and z == 10 then
        w = 0
if x == 5 or (y == 6 and z == 10) then
        w = 0
```

Если в сложном выражении опущены круглые скобки, порядок его вычисления может быть определен с помощью правил предшествования логических операторов. Такие правила определяют старшинство одних логических операторов над другими при вычислении выражения. Наивысшим приоритетом обладает оператор НЕ, который вычисляется в первую очередь. Операторы И и ИЛИ вычисляются слева направо, причем оператор И имеет больший приоритет: **x and y or z**.

В соответствии с правилами предшествования приведенное выше выражение без круглых скобок эквивалентно следующему выражению: **(x and y) or z**.

То же самое имеет место и в следующем выражении: **not x ory and z**. Оно эквивалентно выражению **(not x) or (y and z)**.

Новые операторы сценария вводятся в MAXScriptc новой строки, а для продолжения оператора на следующей строке служит обратная косая черта (\). Но из этого правила имеется исключение для конструкции **if/then**: части **if** и **then** условного оператора могут указываться в отдельных строках. Ведь после выражения **if** непременно должно следовать выражение **then**, поэтому его можно поместить в следующей строке:

```
b = box()
if b.height != 10.0
        then b.length = 40.0
```

Приведенная выше конструкция дает такой же результат, как и следующая:

```
b = box ()
if b.height!= 10.0 then
        b.length = 40.0
```

Оператор **b.length** указан с отступом для обозначения того факта, что он является частью условия **if/then**. Это делается только ради повышения удобочитаемости кода сценария.

Простую конструкцию **if/then** можно расширить, дополнив ее выражением **else**, причем каждая из частей **if**, **then** и **else** условного оператора может быть указана в отдельной строке, например:

```
if b.pos== 10
        thenb.height = 40
elseb.height = 80
```

Выражение **else** дает возможность указать альтернативное действие, если условие внутри выражения **if** не выполняется.

Помимо операторов == и !=, могут быть использованы и другие операторы. В таблице 1 приведен полный их перечень.

Таблица 1. Перечень операторов

Оператор	**Определение**
= =	Равно
1 -	Не равно
>	Больше
> =	Больше или равно
<	Меньше
< =	Меньше или равно

Циклические конструкции

Цикл представляет собой повторяющуюся, или итеративную, операцию. Он обозначает повторяющееся выполнение группы операторов, которое прекращается в определенный момент. Так, если требуется изменить свойства 20 находящихся на сцене сфер, это можно сделать внутри циклической конструк-

ции, повторяющейся 20 раз – по одному для каждой сферы в отдельности. Рассмотрим две циклические конструкции: циклы **for** и **whi1e**.

В цикле **for** используется конструкция, аналогичная следующей:

for i = 1 to 5 do [действие]

В этом цикле значение 1 присваивается индексной переменной **i**. *Индексной* называется такая переменная, которая изменяется (увеличивается или уменьшается) при каждом повторении цикла. Индексные переменные называются также индексными переменными цикла.

Индексная переменная должна быть объявлена в той же строке, в которой объявлен цикл **for**. В цикле **for** сначала выполняется действие, а затем увеличивается значение переменной **i** на 1, после чего действие повторяется. И так до тех пор, пока значение переменной **i** не достигнет предельного значения, указанного после **to**, тогда действие выполняется в последний раз и цикл завершается.

Для построения цикла **for** может быть использована любая переменная, а не только переменная **i**; значение ее приращения может начинаться или оканчиваться любым целым числом. Кроме того, переменную можно использовать и внутри самого действия.

Для выполнения цикла **for** с многими операторами в теле цикла эти операторы следует указывать в круглых скобках, а слово **do** должно предшествовать круглым скобкам. Как и в примерах условных операторов **if/then/else**, цикл не выполняется в скрипте до тех пор, пока не будут введены всего его операторы. Это означает, что до тех пор, пока в цикле не будет введена завершающая круглая скобка, ничего не произойдет. Операторы внутри круглых скобок вводятся с отступом для повышения организованности и удобочитаемости кода сценария. В качестве полезного практического совета каждую новую группу операторов в круглых скобках рекомендуется вводить с дополнительным отступом на одну позицию табуляции.

Если тело цикла содержит только один оператор, в таком случае круглые скобки не нужны. Так, оба приведенных ниже примера равнозначны:

```
For i= 1 to 5 do (
        s = sphere()
)
For i = 1 to 5 do
        s = sphere()
```

Циклы **while** подобны циклам **for** в том отношении, что в них неоднократно повторяется группа операторов. Отличие состоит лишь в том, что в цикле **while** число повторений неизвестно или не указано заранее. Для завершения такого цикла устанавливается определенное условие.

Имеются два типа циклической конструкции **while**: **do/while** и **while/do**.

Для завершения такого цикла указывается условие:

do-while

```
i=1
b = box length:200 name:"redBox"\ wirecolor:[255,0,0]
do(
  copy b
b.pos = [i*25,0,i*25]
  i=i+1
)while b.pos.x<100
```

while-do

```
i=1
b = box length:200 name:"redBox"\ wirecolor:[255,0,0]
while b.pos.x<100 do(
  copy b
b.pos = [i*25,0,i*25]
  i=i+1
)
```

Массивы

Массив представляет собой последовательный ряд элементов. Доступ к каждому элементу массива, или значению, осуществляется с помощью числового *индекса.*

Элементы массива могут иметь любой тип данных, включая числа, логические значения **true/false**, трехмерные объекты, строки и т.д. Объекты массивов создаются с помощью специального конструктора массива. В простейшем случае массив создается с помощью следующей команды:

myArray= #()

Знак # и последующие круглые скобки обозначают массив. С помощью приведенного выше оператора была создана переменная **myArray**, в которой сохранен пустой массив.

Пустой массив пока еще не содержит никаких элементов. Такие элементы можно ввести в массив, например, путем присваивания ему конкретных значений во время его создания:

myArray= #(1,2,4,8,16)

Данный массив содержит пять целочисленных элементов. У каждого из них имеется соответствующий индекс. В частности, у первого элемента массива (перечисленного выше целого значения 1) имеется индекс 1, у второго элемента массива – индекс 2 и т.д. Доступ к отдельным элементам массива осуществляется по его индексу, указываемому в квадратных скобках.

В следующей строке кода возвращается значение третьего элемента массива (целое число 4):

myArray[3]

А в приведенной ниже строке кода значение третьего элемента массива заменяется целым числом 28. Подобным способом можно заполнить массив нужными значениями:

myArray[3] =28

Отдельные элементы присоединяются к массиву с помощью команды **append:**

append myArray 56

В приведенной выше строке кода к массиву присоединяется шестой его элемент (целое значение 56). В массиве можно хранить любой ряд элементов, включая трехмерные объекты.

Массивы весьма полезны для выполнения однотипных операций над несколькими объектами или значениями. Для последовательного обращения к элементам массива совсем не обязательно знать число его элементов. С помощью цикла можно перебрать все значения массива:

```
a = #("one", "word", "at", "a", "time")
for i = 1 to a.count do (
        messagebox a[i]
)
```

Свойство **.count** массива объектов всегда содержит общее число элементов массива.

Для работы с массивами имеется целый ряд дополнительных функций. К их числу относятся **deleteltem** (удаление элементов), **join** (объединение массивов), **sort** (сортировка массива) и **findltem** (находит значения).

Операции со строками

Строковые переменные содержат буквенно-цифровые символы или текст, например «Hello», «MAX4U»или «Введите ваше имя». Строки представляют собой литеральные значения, которые могут использоваться во всплывающих сообщениях или при указании пути к файлу. Строки присваиваются строковым переменным с помощью кавычек. Если не снабдить строку кавычками, это приведет к ошибке.

В MAXScript предоставляется целый ряд операций, которые можно выполнять над строками.

1. Сцепление. Означает объединение двух строк с помощью знака «плюс» (+). Это аналогично сложению двух строк для получения третьей:

textl= "MAXScriptis"
text2 = " fun!"
text3 = textl + text2

В приведенном выше примере переменная **text3** будет иметь значение «MAXScript is fun!».

Для просмотра строки служит функция **messageBox**:

messagebox text3

2. Поиск. Функция **findstring**осуществляет поиск первого экземпляра подстроки в более крупной строке, например:

test = "String example"
location = findString test "ex"

В скрипте возвращается значение 8, указывающее на то, что подстрока **"ex"** начинается с восьмого символа строки **test.** Если искомая подстрока отсутствует, в возвращается сообщение **undefined**.

3. Замена. Эта операция осуществляет замену одной строки другой. Она реализуется с помощью функции **replace** следующим образом:

test = "This is a string"
test2 = "yet another"
test3 = replace test 9 1 test2
messagebox test3

Первым аргументом функции **replace** является строка, содержащая исходный текст. Второй ее аргумент указывает на место, с которого следует заменять строку. Третий аргумент данной функции определяет, сколько символов следует удалить. Четвертый обозначает вставляемую подстроку. В итоге строка test3 будет иметь следующий вид: «This is yet another string»(Это еще одна строка).

Взаимное преобразование чисел и строк

При выполнении файлового ввода-вывода числовые данные в сценарии нередко приходится преобразовывать в текст для последующей записи в текстовый файл. С другой стороны, текст приходится читать из файла и преобрази в числовые данные для последующего использования в сценарии. В подобных случаях требуется взаимное преобразование числовых и строковых. данных В MAXScript такое преобразование выполняется достаточно просто.

Если попытаться сложить число со строкой, поставив между ними плюс. MAXScript выдаст ошибку. Прежде чем присоединить число к строке, необходимо привести числовое значение к строковому типу данных. Это делается с помощью оператора **as string**.

Допустим, что требуется сформировать следующую строку:

string1 = "Your customer number is 345"

Если абонентский номер (customer number) представлен целым числом, данную строку можно сформировать следующим образом:

custNum = 345 – custNum - это целое число

```
custString = custNum as string -- сформировать строку из custNum
stringl = "Your customer code is " + custString
```

С помощью операторов **as integer** и **as float** строку можно преобразовать в целое число или же в число с плавающей точкой соответственно.

Тема 2. Построение пользовательских интерфейсов на языке MAXScript [1]

В MAXScript имеются средства для создания специализированных пользовательских интерфейсов, которые формируются при выполнении сценария. Пользовательский ввод можно получить из свитков и диалоговых окон, а затем выполнить на его основании соответствующие команды. В этой главе описыва-

ется порядок создания элементов пользовательского интерфейса, связанных со сценарием.

В сценарии могут быть сформированы два типа интерфейсов.

1. Свитки на панели Utilities (рис. 3).
2. Перемещаемое диалоговое окно (рис. 4).

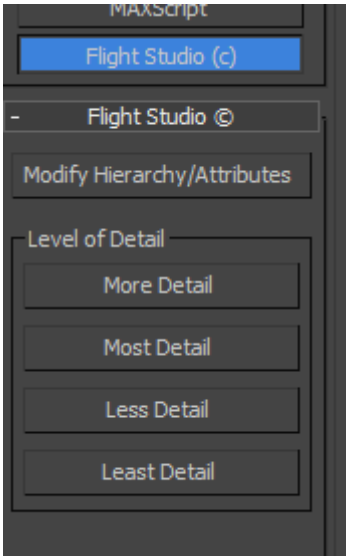

Рис. 3. Пример свитка на панели Utilities

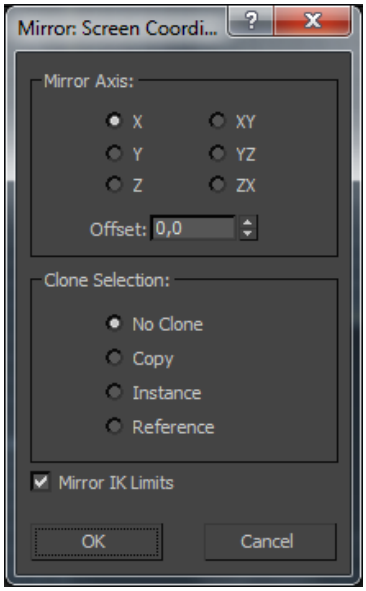

Рис. 4. Пример диалогового окна

Исключение составляет сценарный подключаемый модуль, представляющий собой специальный сценарий, способный формировать свитки в других частях пользовательского интерфейса 3ds Max.

Свитки

Свитки составляют саму основу любого пользовательского интерфейса, формируемого средствами MAXScript. Прежде чем создавать какие-либо другие элементы пользовательского интерфейса, в том числе флажки и счетчики, необходимо сформировать хотя бы один свиток, в котором они будут находиться. Под термином «свиток» в MAXScript подразумевается любой свиток, известный вам из пользовательского интерфейса 3ds Max, например панелей команд или диалоговых окон. Но, если требуется, свиток может быть отображен средствами MAXScript, как отдельное диалоговое окно.

Правда, создать специальный свиток средствами MAXScript и заполнить его элементами пользовательского интерфейса не составляет большого труда.

Для создания свитка используется следующая конструкция:

rollout <переменная> "Имя свитка"(
< Элементы пользовательского интерфейса, в том числе флажки, кнопки и счетчики >
)

В переменной хранится внутреннее имя свитка, используемое в сценарии для обращения к свитку. А строка «Имя свитка» определяет название, появляющееся в заголовке свитка.

Код, находящийся в круглых скобках, называется выражением свитка. Внутри выражения свитка определяются все элементы пользовательского интерфейса и их функции.

Для создания диалогового окна в соответствии с рис. 5 необходимо прописать следующий код:

```
rollout a "Пример диалогового окна"(
        spinner sp "Введите значения: "
        button but "Щелкнуть"
)
createdialog a 230 70
```

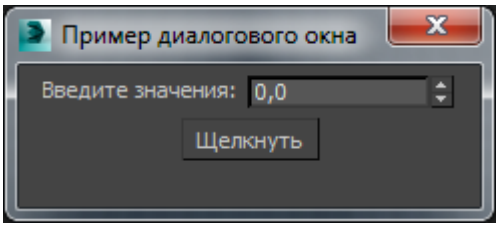

Рис. 5. Простое диалоговое окно

В этом коде определяется свиток, обозначенный как «Пример диалогового окна». Переменная **a** содержит внутреннюю ссылку на свиток, которую можно использовать в остальных частях сценария для обращения к данному свитку.

Команды **spinner** и **button** создают соответственно счетчик и кнопку с метками, указанными в кавычках. А переменные **sp** и **but** содержат внутренние ссылки на счетчик и кнопку.

При выполнении скрипта свиток появится в виде диалогового окна «Пример диалогового окна» шириной 230 и высотой 70 единиц. Он содержит два элемента пользовательского интерфейса: счетчик и кнопку.

По команде **createdialog** создается новое перемещаемое диалоговое окно с использованием команд создания свитка. В связи с тем, что **a**– это внутреннее имя свитка, команда **createdialog a** обращается к определению свитка, сделанному ранее в данном сценарии.

Если изменить значение в счетчике и щелкнуть на кнопке, то ничего не произойдет. Для того чтобы что-то произошло при взаимодействии пользователя с элементами интерфейса в данном диалоговом окне, в сценарий необходимо ввести обработчики событий.

Для ввода обработчика событий в сценарий дописывается следующий код, он должен быть расположен внутри свитка и после описания элементов интерфейса (счетчик и кнопку):

```
on but pressed do (
        d = sp.value
        sphere pos:[d,0,0]
)
```

Если изменить значения в счетчики и щелкнуть по кнопке, то после каждого щелчка будет создаваться сфера, расположенная по оси X так, как указано в счетчике. Счетчик хранится в переменной **sp**. А значение, введенное в его поле, может быть получено с помощью свойства **.value** данной переменной. Следовательно, **sp.value** – это значение, введенное в поле счетчика **sp**.

После щелчка на кнопке **but** значение из счетчика присваивается в сценарии переменной d, а затем создается сфера, располагаемая по оси X так, как указано в счетчике.

Существует еще немало средств для ввода меток, флажков, кнопок-переключателей и других элементов интерфейса. При этом можно контролировать расположение элементов интерфейса справа, слева либо по центру диалогового окна или в единственной строке свитка. Ниже перечислены наиболее часто используемые элементы пользовательского интерфейса в том виде, в каком они называются и применяются в сценариях MAXScript.

- **Label** (Метка) – статический элемент управления, предназначенный для отображения текста (рис. 6). Пользователь не может изменить текст метки, однако это можно сделать в сценарии.

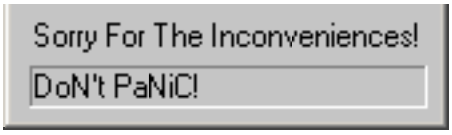

Рис. 6. Элемент интерфейса Label

- **Checkbox** (Флажок) – элемент, который может быть включен или выключен пользователем (рис. 7).

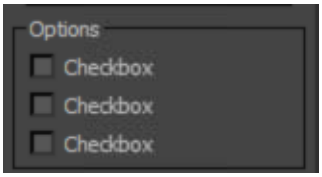

Рис. 7. Пример использования элемента интерфейса Checkbox

- **Button** (Кнопка) – элемент, который утапливается при нажатии и приподнимается обратно при отпускании (рис. 8).

Рис. 8. Пример элемента интерфейса Button

- **Checkbutton** (Фиксирующая кнопка) – элемент, который утапливается при первоначальном нажатии и приподнимается при повторном нажатии (рис. 9).

Рис. 9. Пример элемента интерфейса Checkbutton

- **Pickbutton** (Кнопка выбора) – элемент выбора объектов на сцене.

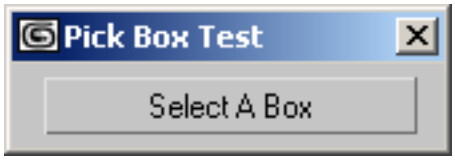

Рис. 10. Пример элемента интерфейса Pickbutton

- **Radiobuttons**(Кнопки-переключатели) –ряд переключателей в свитке. Одновременно можно выбрать лишь один из них (рис. 11).

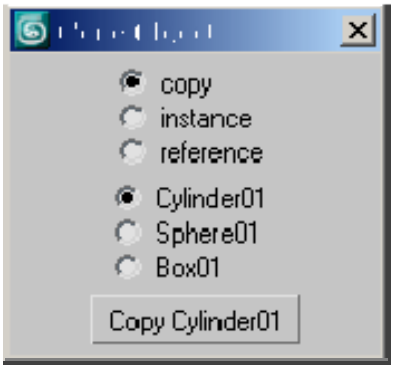

Рис. 11. Пример элемента интерфейса Radiobutton

- **Spinner** (Счетчик) – элемент, предназначенный для ввода числового значения в свитке. Он состоит из поля ввода и стрелок (рис. 12). У счетчика имеются два параметра: диапазон значений и устанавливаемое по умолчанию значение.

Рис. 12. Пример элемента интерфейса Spinner

- **Slider** (Ползунковый регулятор) – альтернативный счетчику элемент. Пользователь перемещает ползунок в ту или иную сторону (рис. 13). У ползун-

кового регулятора те же самые параметры: диапазон значений и устанавливаемое по умолчанию значение.

Рис. 13. Пример элемента интерфейса Slider

- **Edittext** (Поле редактирования текста) – текстовое поле, в котором пользователь вводит и редактирует текст (рис. 14).

Рис. 14. Пример элемента интерфейса Edittext

- **Listbox** (Списковое окно) – список элементов, предоставляемых пользователю на выбор (рис. 15). Пользователь может прокручивать список для выбора нужного элемента из списка.

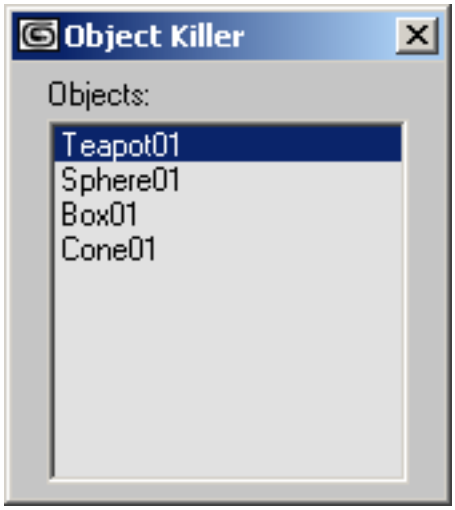

Рис. 15. Пример элемента интерфейса Listbox

- **Combobox** (Комбинированное окно) комбинация списка выбора и поля редактирования текста. Список всегда отображается в свитке полностью, хотя длинные списки могут быть снабжены полосами прокрутки. В расположен-

ном вверху поле редактирования отображается выделенный в данный момент элемент списка.

- **Dropdownlist** (Раскрывающийся список) – аналогичен комбинированному окну, но в данном случае список раскрывается, когда пользователь щелкает на кнопке со стрелкой вниз. Следует иметь в виду, что текст в раскрывающемся списке не вводится.
- **Colorpicker** (Селектор цвета) – элемент, предназначенный для отображения диалогового окна Color Selector(Селектор цвета). В свитке он представлен образцом цвета (рис. 16). Пользователь может щелкнуть на образце цвета, чтобы открыть диалоговое окно Color Selector.

Рис. 16. Пример элемента интерфейса Colorpicker

- **Progressbar** (Индикатор выполнения) – элемент, предназначенный для отображения хода действия или выполнения процесса.
- **Mapbutton** (Кнопка карты) и **Materialbutton** (Кнопка материала) – элементы, предназначенные для отображения карт и материалов в диалоговом окне Material/Map Browser (рис. 17).

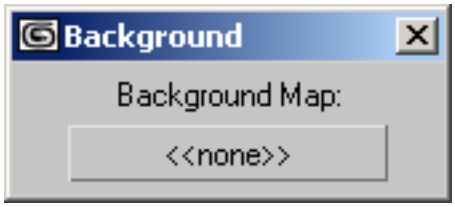

Рис. 17. Пример элемента интерфейса Mapbutton

- **Bitmap** (Растр) – элемент, предназначенный для отображения растрового изображения в свитке (рис. 18).

Рис. 18. Пример элемента интерфейса Bitmap

Для каждого элемента интерфейса, указываемого в выражении свитка, создается соответствующий объект со своими свойствами. Доступ к этим свойствам осуществляется по имени данного элемента интерфейса. Имеются свойства, которые являются общими для всех элементов интерфейса (например, свойство положения), а также свойства, характерные только для элемента конкретного типа.

При создании в 3ds Max элементы интерфейса размещаются на панели со стандартными параметрами расположения. Если требуется другая схема их расположения, конструктору каждого такого элемента можно передать ряд следующих параметров.

- **align** – указывается следующим образом: #left, #right или #center. Знак «решетки» должен быть указан обязательно. Данный параметр определяет выравнивание элемента интерфейса по левому краю, по правому краю или же по центру.
- **pos** – определяет положение элемента интерфейса в конкретной точке с координатами x и y, указываемыми в пикселях. Значение этого параметра указывается с помощью типа данных Point2 ([x, y]). Такое положение элемента интерфейса определяется относительно левого верхнего угла свитка.
- **width** – задает ширину элемента интерфейса в пикселях.
- **height** – задает высоту элемента интерфейса в пикселях. Для спискового и комбинированного окон высота задается в зависимости от числа элемен-

тов в списке. Так, для отображения N элементов в списковом окне его высоту следует задать равной N. А для отображения N элементов в комбинированном окне его высоту следует задать равной N+2.

- **offset** – задает смещение относительно устанавливаемого по умолчанию положения элемента интерфейса. В качестве единиц изменения используются пиксели, а в качестве типа данных – Point2.
- **across** – задает положение элементов интерфейса по горизонтали, а не по вертикали. Применяется к элементу и следующим за ним (N - 1) элементам, где N – число, указываемое после слова across. Этот параметр более подробно рассматривается в следующем разделе.

Как правило, лучше сначала предоставить 3ds Max возможность расположить элементы интерфейса с помощью устанавливаемых по умолчанию параметров, а затем внести необходимые коррективы в расположение элементов. Текстовые метки должны быть краткими, чтобы они не выходили за край свитка, а элементы интерфейса не должны перекрываться.

Обработчик событий

Общая форма обработчика событий имеет следующий вид:

```
on <имя элемента интерфейса> <имя события > do(
    [выполняемые команды]
)
```

Ниже приведен перечень типичных событий, используемых в сценариях.

- **pressed** – вызывается после щелчка на кнопке.
- **changed** – вызывается при изменении состояния управляющего элемента, например при установке флажка или правке значения в счетчике.
- **picked** – вызывается для кнопки выбора, когда пользователь выбирает элемент в сцене.
- **entered** – вызывается при вводе числа в поле редактирования счетчика и последующем нажатии клавиши <Enter>.

- **selected** – вызывается при выборе элемента из списка в списковом или комбинированном окне.

Тип обработчика событий зависит от элемента интерфейса. Если использовать, например, элемент **radiobutton**, то в качестве обработчика событий могут выступать два события: **changed** (срабатывает при переключении между радиокнопками) и **rightClick** (нажатие на правую кнопку мыши). Полная информация о каждом обработчике событий расположена в файле помощи программы 3ds Max и доступна через главное меню Help>MAXScript Help.

По мере укрупнения и усложнения утилит возникает потребность сгруппировать элементы в свитке на панели **Utilities**. Элементы можно сгруппировать в логической последовательности, используя выражение группы. Независимо от того, как и для чего группируются элементы, группирование должно разумно разграничивать функции панели. Группа очерчивается на панели контуром с меткой в левом верхнем углу. А синтаксис группы следующий:

```
Group "описание группы"(
        <элементы пользовательского интерфейса>
)
```

Тема 3. Работа с интерфейсом 3dsMax с помощью MAXScript [1]

В 3ds Max имеется немало элементов пользовательского интерфейса, включая панели команд, свитки, диалоговые и обычные окна. Кроме того, в пользовательском интерфейсе 3ds Max можно выполнять немало тех действий, которые обычно выполняются в видовых окнах, а также команды доступа к внешним файлам. Столь глубокая структура пользовательского интерфейса 3ds Max становится причиной значительных потерь времени при решении сложных или часто повторяющихся задач. Подобные задачи, как правило, автоматизируются полностью или частично с помощью сценариев.

В MAXScript имеется достаточно функций и команд для работы практически со всеми элементами пользовательского интерфейса. Поэтому в интере-

сах индивидуальных пользователей или студий – уделить серьезное внимание вопросам эффективного использования интерфейса 3ds Max, чтобы найти пути повышения производительности своего повседневного труда.

Клонирование объектов

Одной из самых основных операций, выполняемых в видовом окне, является дублирование, или клонирование, объектов. И большую часть этих операций можно выполнять средствами MAXScript.

Следует, однако, иметь в виду, что копирование и получение экземпляра распространяется не только на объекты, находящиеся на сцене, в том виде, в каком они представлены в MAXScript, но и на многие типы данных. Например, копировать можно строки, цвета, растры, значения угла поворота вокруг осей и т.д. Рассмотрим копирование в сцене примитивных объектов.

Для копирования объекта используется следующая команда:

copy<узел>

Данный тип копирования показывает, что объект будет отделен от исходного. Любые изменения в копии не оказывают влияния на исходный объект. У скопированного объекта имеются собственные свойства, модификаторы и прочие атрибуты, которые могут быть изменены, не оказывая влияния на оригинал или другие копии.

Пример использования команды **copy**:

```
b = box heightsegs: 8
c = copy b
move c [25,0,0]
d = copy c pos:      [50,0,0]
```

Для создания экземпляра используют команду:

instance <узел>

Изменения, вносимые в копию, будут сказываться и на исходном объекте. С другой стороны, изменения, вносимые в исходный объект, приводят к обновлению копии. Это классический пример поведения на сцене полученных экземпляров объектов. Экземпляры имеют общие с исходным объектом свойства, модификаторы, материалы и карты, а также контроллеры управления анимацией, но не преобразования, пространственные исказители или путевые имена.

Пример использования команды:

```
b = box heightsegs: 8
i = instance b pos:[0,25,0]
addmodifier i (bend())
```

Для создании ссылки используется команда:

Reference <узел>

Любые изменения, вносимые в родительский объект, будут оказывать влияние на порожденный объект. В то же время изменения в порожденном объекте не будут оказывать влияния на родительский объект.

В MAXScript имеется еще один, более сложный способ клонирования узлов. Приведенные выше способы пригодны для простых объектов, но дают непредсказуемые результаты при клонировании таких объектов с иерархиями, как источники света с собственными целями. Для клонирования с сохранением взаимосвязей между родительскими и порожденными объектами на сцене служит функция:

maxOps.CloneNodes

Пример использования этой функции:

```
group objects name:"Stul"
maxOps.CloneNodes $Stul offset:[0,0,140]
```

В данном примере создается группа объектов с именем «Stul», создается ее копия и перемещается вверх на 140 единиц.

Некоторые параметры функции **maxOps. CloneNodes:**

- **clonetype: <enum>** – этот параметр может принимать одно из трех следующих значений: **#copy** (копия), **#instance** (экземпляр) или **#reference** (ссылка), а по умолчанию – **#copy**.
- **newNodes: <&массив узлов >** – этот массив узлов будет заполнен вновь клонированными узлами.
- **actualNodeList: <&массив узлов>** – в качестве этого необязательного параметра передается пустой массив, который будет заполнен конкретными клонированными узлами. Это объясняется тем, что между узлами возможны взаимосвязи, которые служат причиной появления в списке других узлов, например взаимосвязи между целями источников света и камер, узловыми частями систем, членами групп или расширенных иерархий.
- **offset: <point3 >** – этот параметр перемещает новый узел по указанному вектору типа point3. Он может быть использован для установления различий между вновь клонированными и старыми узлами.

Копирование массивов

Массивы являются весьма эффективным и полезным средством MAXScript. Они могут содержать практически все, что поддается записи. Например, можно организовать массив чисел, строк, функций, структур, свитков и даже находящихся на сцене объектов.

Время от времени массивы приходится копировать. В таком случае целесообразно, казалось бы, воспользоваться следующим методом:

```
h = #(1,2,3)
m = copy h
```

Однако он непригоден. Ведь значение переменной **m** является не массивом, а просто постоянным значением «ОК». Это совершенно не то, что ожида-

лось получить. Для получения более предсказуемых результатов копирования массива необходимо ввести в вызов функции дополнительный аргумент **#noMap:**

m = copy h #noMap

В ответ на введенную выше строку будет выдан следующий результат:

#(1,2,3)

Благодаря дополнительному аргументу **#noMap** результаты копирования массива оказываются вполне предсказуемыми.

Команды и панели команд 3ds Max

В MAXScript имеется ряд самых общих команд управления крупными частями пользовательского интерфейса. Многие из них формируются автоматически в верхней части окна приемника команд (называемого подокном макрорегистрации MAXScript Listener) в ответ на разные события, происходящие на сцене. В частности, имеется команда сохранения файла сцены **max file save**, команда фиксации выделенного **max freeze selection** и команда развертывания видового окна на весь экран **max tool maximize**.

Панели команд являются наиболее важной частью интерфейса 3ds Max. Переходя от одной панели к другой, пользователь получает доступ к разным командам. Например, команды правки каркаса доступны только на панели **Modify**. Переходить от одной панели команд к другой можно и средствами MAXScript, используя следующий синтаксис:

SetCommandPanelTaskMode [mode:] <имя панели>

Для доступа к панели команд имеется также соответствующая функция: **GetCommandPanelTaskMode()**.Эта функция возвращает следующие имена панелей.

- #create
- #modify
- #hierarchy
- #motion
- #display
- #utility

Еще один способ перехода от одной панели команд к другой состоит в применении команд Max. Эти команды следующие.

- max create mode
- max modify mode
- max hierarchy mode
- max motion mode
- max display mode
- max utility mode

Большинство самых важных команд доступно на основной панели инструментов. В MAXScript имеются средства для доступа ко многим из этих команд. В частности, программным путем можно выбрать кнопку отмены операции, открыть диалоговое окно выбора объектов или активизировать инструмент перемещения объектов.

Выбор объекта в сцене

Одной из наиболее часто используемых функций расширенного набора средств MAXScript является функция **PickObject**, позволяющая выбрать объект на сцене, используя стандартные инструменты выделения. Синтаксис данной функции следующий:

PickObject [message: <строка>] [count: n | #multiple] [filter: <функция>]

При вызове функции **PickObject** курсор принимает вид пиктограммы выделения объекта и происходит переход в режим ожидания от пользователя конкретных действий для выделения одного или более объектов в сцене 3ds Max. Ниже приведены некоторые необязательные параметры функции **PickObject**.

- **[message: <строка>]** – это сообщение, появляющееся в строке состояния, расположенной в нижней части пользовательского интерфейса 3ds Max.

- **[count:n | #multiple]** – этот параметр указывает количество выбираемых объектов. Функции можно передать целое число (n) для выбора конкретного количества объектов или же значение #**multiple** для того, чтобы пользователь мог выбрать столько объектов, сколько потребуется, прежде чем нажать клавишу <Esc> или щелкнуть правой кнопкой мыши.
- **[filter–<функция>]** – это функция, которая позволяет отфильтровать разные виды объектов на сцене 3ds Max. Такая функция может упростить выбор объектов, если, например, требуется выделить только объекты формы насцене, заполненной параллелепипедами. При этом фильтр вызывается и передается тому объекту на сцене, над которым находится курсор, например:

Function mySphereFilter obj = (ClassOf obj == Sphere)

В данном случае из тела функции возвращается логическое значение **true,** если объект является сферой; в противном случае возвращается логическое значение **false.** Эта функция фильтрации передается далее функции **PickObject.**

Функция mouseTrack

Функция **mouseTrack** позволяет отслеживать перемещения курсора мыши в видовых окнах. В частности, она способна контролировать щелчки кнопками мыши и перемещения курсора мыши, а также отслеживать пересечение курсора мыши с геометрическими объектами на сцене. Для организации реакции на событие, связанное с мышью, можно написать функцию, которая вызывается, когда подобное событие наступает. Это так называемая функция обратного вызова. Она имеет специальное обозначение и воспринимает определенные аргументы, предоставляемые средствами MAXScript.

Синтаксис функции **mouseTrack** следующий:

mouseTrack [on:<узел>] [prompt:"msg"] [snap:#2D|#3D] [trackCallback:fn|#(fn,arg)]

Ниже приведены некоторые параметры функции **mouseTrack.**

- **[on: <узел>]** – этот необязательный аргумент определяет находящийся на сцене объект, реакция на который будет происходить в функции обратного вызова. Если этот аргумент не предоставляется, функция **mouseTrack** будет отслеживать перемещения курсора мыши на активной координатной сетке.
- **[prompt: "msg"]** – это текстовое сообщение, появляющееся в строке состояния, расположенной в нижней части окна 3ds Max.
- **[snap**:**#20\#30]** – этот аргумент активизирует привязки, но действует лишь в том случае, если перемещение курсора мыши не отслеживается на поверхности объекта.
- **[trackCallback: fn]** – это функция, которая вызывается в ответ на события, связанные с мышью, а ее имя обозначает функцию обратного вызова. Такая функция должна возвращать значение **#continue** для того, чтобы продолжалась обработка событий, фиксируемых функцией **mouseTrack.** Если же эта функция возвращает какое-то другое значение, она просто больше не вызывается.

Реализуемая отдельно функция обратного вызова должна воспринимать ряд аргументов следующим образом:

function CallBack message intRay obj faceNumber shift Ctrl alt = (...)

- **message** – это код, указывающий на действия мыши в виде одного из следующих четырех сообщений:

#mousepoint – нажата левая кнопка мыши;

#mousemove – курсор мыши перетаскивается при нажатой левой кнопке мыши;

#mouseabort – нажата правая кнопка мыши.

- **intRay** – пересечение с лучом, исходящим из курсора мыши и направленным в сторону активной координатной сетки или отслеживаемого объекта. У такого луча имеется свойство положения (.pos) и вектор направления (.dir).

- **obj** – это отслеживаемый объект. Он присваивается при вызове функции mouseTrack с дополнительным аргументом [оп:<узел>]. Если данный аргумент не предоставляется, параметр **obj** не определен.
- **faceNumber** – порядковый номер грани, на которой находится курсор мыши. Этот параметр действует только в том случае, если объект является редактируемым каркасом. В противном случае он не определен.
- **Shift/Ctrl/Alt** – эти параметры указывают на нажатие соответствующих модифицирующих клавиш. Они могут принимать лишь два логических значения: **true** и **false.**

Тема 4. Преобразования и анимация на языке MaxScirpt [1]

Преобразования положением и масштабом

Положение объекта может быть установлено двумя способами:

- с помощью свойства .pos или .position;

```
s = sphere()
s.pos = [150, 10, 0]
s.pos.x = 200
```

- с помощью метода move.

```
s = sphere()
move s [10.0, 10.0, 10.0]
```

Перечень аргументов метода **move** состоит из ссылки на перемещаемый объект и величины этого перемещения в виде данных типа **Point3.** Следует, однако, иметь в виду, что метод **move** перемещает объект не в точку с координатами [x, y, z], а на величину [x, y, z]. Следовательно, если применить данный метод несколько раз, объект будет непрерывно перемещаться на указанную величину.

Значение свойства положения задается явно и не увеличивается всякий раз.

Свойство масштаба (**scale**) определяет сжатие или растяжение объекта вдоль указанной оси. По умолчанию значение составляющих X, Y и Z свойства масштаба равно 1,0, что соответствует масштабу 100% объекта по каждой из осей координат. Если установить значение одной из составляющих свойства масштаба равным 2,0, исходный масштаб объекта увеличится вдвое, а если установить ее значение равным 0, 5, то исходный масштаб объекта сократится наполовину.

Масштаб объекта можно изменить по всем трем координатам следующим образом:

```
s = sphere()
s.scale= [2.0, 1.0, 0.5]
```

Аналогично положению, масштаб указывается в виде данных типа **Point3.** В приведенном выше примере объект растягивается в два раза вдоль оси X, не изменяет свой размер вдоль оси Y и сокращается вдоль оси Z. Если же объект требуется растянуть только вдоль оси X, для этого достаточно ввести следующую строку кода:

```
s = sphere()
s.scale.x = 2.0
```

Для пропорционального масштабирования объекта достаточно умножить свойство масштаба на требуемую величину:

```
s.scale= 3.0 * s.scale
-- или s.scale *= 3.0
```

Аналогично свойству **pos** и методу **move** для свойства масштаба имеется соответствующий метод **scale.** Данный метод реализуется следующим образом:

```
b = box()
scale b [1.0, 2.0, 3.0]
```

В этом фрагменте кода параллелепипед масштабируется на разную величину в направлении каждой из осей координат. Свойство . **scale** и метод **scale** действуют таким же образом, как и свойство **.position** и метод **move**, а именно: значение свойства задается явно в виде данных типа **Point3**, а метод изменяет это значение с указанным приращением.

Преобразование вращением

Вращение является более сложным видом преобразования, чем изменение положения и масштаба, поскольку требует более сложных математических расчетов.

Как и для изменения положения и масштаба, для вращения объектов в MAXScript предусмотрены два способа.

- установка свойства . rotationo бъекта явным образом;
- применение метода rotate.

При изменении положения и масштаба значения по каждой из осей X, Y и Z всегда выражаются одним числом, например 1, 0 или 34,5. А вращение на определенное количество градусов вокруг конкретной оси выражается несколькими способами, причем с разной интерпретацией величины вращения. Итак, для вращения объекта имеются следующие методы.

- EulerAngles;
- Кватернионы;
- AngleAxis.

Метод **EulerAngles** является, вероятно, самым понятным и лучше всего соответствующим тем операциям вращения, которые можно выполнять и наблюдать в интерфейсе 3dsMax.

Конструктор класса **EulerAngles** воспринимает три аргумента, выражаемые в градусах (по одному для каждой из осей). Эти значения используются в данном методе для вращения объекта на указанное число градусов вокруг заданной оси в системе мировых координат:

```
b = box()
b.rotation = eulerAngles 0 45 0
```

В приведенной выше строке кода объект поворачивается на 45˚ вокруг оси Y в системе мировых координат. Вращение по методу **eulerAngles** проще всего понять, поскольку оно соответствует вращению объектов, наблюдаемому в видовых окнах.

Результат вращения по методу **eulerAngles** можно присвоить переменной, чтобы затем использовать в другом объекте:

```
ang = eulerAngles 0 45 0
b.rotation = ang
```

Переменная **ang** в действительности представляет собой объект MAXScript, имеющий тип данных вращения.

Углы вращения вокруг нескольких осей указываются следующим образом:

```
ang = eulerAngles 30 20 44
```

При установке ключевых кадров анимации с помощью метода **eulerAngle s**изменение вращения между двумя последовательными ключевыми кадрами ограничивается углом 180°, поскольку в методе **eulerAngles** используется кратчайший путь перехода от одного ключевого кадра к другому.

Составляющие свойства вращения объекта могут использоваться для задания вращения вдоль конкретной оси. Например, выражения

```
euAngle = eulerAngles 30 45 60
rotate b euAng
```

можно заменить выражениями, в которых свойство вращения устанавливается непосредственно:

b.rotation.x_rotation = 30
b.rotation.y_rotation = 45
b.rotation.z_rotation = 60

При таком способе установки свойства вращения угол поворота ограничивается величиной 360°.

Во время работы с объектами сцены непосредственно в пользовательском интерфейсе 3ds Max ограничения, присущие эйлерову представлению вращения вокруг осей X,Y и Z, зачастую никак не мешают поворачивать объекты и получать вполне предсказуемые результаты. Для этого просто выбирается система координат, наиболее подходящая для конкретной ситуации, и далее вращение выполняется визуально для достижения желаемого результата.

При написании сценариев возможность манипулировать моделью визуально отсутствует. Вращение объекта приходится обозначать точно с помощью конкретных чисел, чтобы добиться искомого результата.

И в этом случае на помощь приходит метод представления процесса вращения с помощью **кватернионов**. Кватернионы явно обозначают всякое возможное вращение, что позволяет точно и однозначно задать вращение вокруг осей X, Y и Z.

Значение кватерниона выражается в виде угла и значения типа **Point 3,** описывающего некий вектор:

quat <угол> <вектор>

Векторная часть данного выражения относится к выражению типа **Point3.** Отдельные значения в выражении типа **Point3** всегда находятся в пределах от -1 до 1. Ниже приведены некоторые примеры обозначения кватернионов:

quat 30 [1, 0, 0]
quat 30 [0.266917, 0.534798, 0.801715]

Для того чтобы лучше понять кватернионы, представьте себе сферу с вектором, направленным от центра сферы к ее поверхности. Векторная часть ква-

терниона обозначает направление вектора, в угол – количество градусов, на которое объект должен повернуться вокруг данного вектора.

Если векторное значение состоит из единицы и двух нулей, то вектор нетрудно себе представить направленным вдоль одной из осей X, Y или Z. При этом 1 определяет ось, вдоль которой вектор направлен. Так, векторное значение [1, 0, 0] направляет вектор вдоль оси X.

Как правило, используются векторы, направленные вдоль оси X, Y или Z. А для того чтобы использовать векторы, имеющие другое направление, нужно знать, как их строить.

Векторные значения кватернионов следует рассчитывать очень аккуратно. Квадратный корень из суммы квадратов всех составляющих должен быть равен 1. Так, из приведенного выше примера кватерниона **quat 30 [0.266917, 0.534798, 801715]** можно установить, что квадратный корень из суммы $(0.266917)^2 + (0.534798)^2 + (0.801715)^2 = 1$

Это ограничение объясняется тем, что длина вектора должна быть всегда равна 1. Следовательно, длина вектора всегда равна квадратному корню из суммы $x^2 + y^2 + z^2$.

Правильное определение векторных значений кватерниона – занятие долгое и утомительное. Поэтому для упрощения расчетов применяется метод **normalize**.

```
rot_vect = normalize [10, 14, 5]
q = quat 90 rot_vect
```

Если объект нужно повернуть на угол более 360°, в таком случае можно воспользоваться объектами класса **AngleAxis.** Такие объекты реализуются подобно кватернионам следующим образом:

```
ang = angleaxis 30 [1, 0, 0]
ang = angleaxis 30 x_axis
```

Оба приведенных выше выражения равнозначны, поскольку в обоих случаях создается объект **ang,** который обеспечивает вращение на 30° вокруг оси X.

У всякого геометрического объекта имеется своя точка опоры, которая представляет собой точку, вокруг которой происходит перемещение, вращение и масштабирование объекта. До сих пор в этой главе преобразования рассматривались лишь относительно устанавливаемой по умолчанию точки опоры объекта.

Для изменения точки опоры объекта в MAXScript используется свойство **.pivot**, чтобы установить или переместить точку опоры, например:

```
object.pivot = [0, 0, 10.0] --или
object.pivot.z = 10.0
```

Если выполняется вращение, в 3ds Max предполагается, что это делается в системе мировых координат относительно точки опоры.

Анимация

Анимация в 3ds Max осуществляется с помощью контроллеров управления анимацией. Контроллеры рассчитывают значения оживляемых свойств объекта с помощью самых разных методов интерполяции в зависимости от типа выбранного контроллера. Чаще всего применяются контроллеры управления положением, вращением и масштабированием, называемые также контроллерами управления преобразованием.

Анимация сцены организуется по трекам. Трек содержит информацию об изменениях, происходящих в объекте, модификаторе, параметре или другом элементе в ходе анимации. С каждым оживляемым элементом сцены связан отдельный трек анимации, для которого может быть назначен контроллер.

Треки анимации сцены и их свойства доступны для правки в окне **TrackView**. Кроме того, контроллеры управления преобразованием можно назначать и править на панели **Motion**. Для того чтобы извлечь максимальную

пользу из материала этого раздела, вы должны уметь пользоваться окном **TrackView**, чтобы просматривать и править треки анимации, а также назначать контроллеры на панели **Motion** и в том же окне **TrackView**.

Для каждого создаваемого объекта автоматически назначаются контроллеры управления преобразованием. А для остальных оживляемых параметров объекта контроллер назначается, как только для анимации параметра устанавливается ключевой кадр.

Конкретные типы контроллеров указываются в подсвойстве **.controller** всякого оживляемого свойства объекта:

```
s = sphere()
s.pos.controller
```

При выполнении этого фрагмента кода возвращается значение **Controller: Position_XYZ,** указывающее на то, что для трека анимации положения сферы назначен контроллер **Postion_XYZ.**

Свойство **.controller** имеет подсвойство **.value**. Как только для параметра будет назначен контроллер, значение этого контроллера может быть установлено следующим образом:

```
s.pos.controller.value = [20,0,50]
```

Некоторые контроллеры, например **Position_XYZ**, имеют три составляющие, каждая из которых имеет свой контроллер. Доступ к этим контроллерам осуществляется по индексу 1, 2 и 3:

```
s.pos.controller[1].controller
```

При выполнении этой строки кода возвращается контроллер, назначенный для составляющей X стандартного контроллера **Position_XYZ,** в качестве которогослужит **Bezier_Float,** т.е. контроллер управления по кривой Безье с плавающей точкой. Подсвойство **.value** может быть использовано и для этого подконтроллера:

s.pos.controller[1].controller.value

При выполнении этой строки кода возвращается значение составляющей X контроллера **Position_XYZ.**

У каждого контроллера имеются свои подсвойства. Так, если для объекта назначен контроллер **Position_XYZ** в качестве контроллера анимации его положения, следующая строка окажется равнозначной предыдущей строке кода:

s.pos.controller.x_Position.controller.value

Доступ к контроллеру, состоящему из подконтроллеров, а также к его значениям может быть всегда осуществлен несколькими способами. Многие программисты предпочитают конструкцию **controller[1].controller** вместо более явно выраженной конструкции **controller.X_Position**, поскольку это дает им возможность работать с любым назначенным для объекта контроллером при условии, что он имеет три составляющие. Если в сценарии используется первая конструкция, используемый в объекте контроллер можно изменить без ошибок.

Для того чтобы назначить новый контроллер для оживляемого параметра объекта, достаточно создать сначала экземпляр этого контроллера, а затем присвоить его:

```
c = linear_float()
s.radius.controller = c
```

В дальнейшем экземпляр контроллера можно использовать для изменения свойств данного контроллера, **c.value = 40**.

Если контроллер из переменной с назначается для других параметров в сцене, при изменении значения переменной с изменяются также все экземпляры **c.value** подобно тому, как это происходит с экземплярами других элементов сцены.

Во многих случаях вместо свойства .**controller** может быть использовано подсвойство **.track**. Тем не менее рекомендуется пользоваться свойством . controller, чтобы ваши сценарии работали при любых обстоятельствах.

В MAXScript имеются средства для создания ключевых кадров на треках анимации и установки их значений.

1. Ключевые кадры анимации можно создавать с помощью контекста **animate**. Этот контекст равнозначен включению режима **Auto Key** и выполнению последующих команд (при этом кнопка Auto Key в пользовательском интерфейсе 3ds Max не активизируется):

```
animateon(
        [команды]
)
```

В данном контексте можно также использовать контекст **at time** для указания места для установки ключевых кадров:

```
s = sphere()
animate on (
        at time 50 s.radius = 10
        at time 100 s.radius = 60
)
```

В этом фрагменте кода создаются ключевые кадры анимации радиуса сферы в кадрах 50 и 100, где устанавливаются значения радиуса 10 и 60 соответственно.

Если контекст **animate** используется для установки ключевых кадров, то в нулевом кадре автоматически создается ключевой кадр, даже если это не указано явно. При этом повторяются те же действия, что и при активизации кнопки **Auto Key**: если ключевой кадр устанавливается не в нулевом кадре, а в последнем он отсутствует, то ключевой кадр автоматически создается в нулевом кадре с исходным значением трека.

Когда создается сфера, контроллер для трека анимации ее радиуса не назначается. Если ключевые кадры устанавливаются с помощью контекста

animate, то по умолчанию для трека анимации радиуса сферы назначается контроллер **Bezier Float.** После расстановки ключевых кадров на треке можно отобразить тип назначенного контроллера, введя в приемнике команд следующую строку кода:

s.radius.controller

2. Кроме того, ключевые кадры анимации могут быть созданы с помощью метода **addNewKey.** Если контроллер уже назначен для трека, то, используя метод **addNewKey,** можно добавить ключевой кадр непосредственно к контроллеру:

```
c = s.radius.controller
addNewKey c 20
```

В данном примере ключевой кадр добавляется к контроллеру, управляющему анимацией радиуса сферы, в кадре 20. Значение контроллера в данном кадре не меняется, а просто устанавливается ключевой кадр.

Если ключевые кадры вводятся с помощью метода **addNewKey**, ключевой кадр не создается автоматически в нулевом кадре.

Для удаления всех ключевых кадров конкретного контроллера можно воспользоваться следующей строкой кода:

deleteKeyss.radius.controller

Для изменения значений ключевых кадров в конкретном кадре имеются следующие возможности.

- Использовать контекст **animate**, чтобы установить значения ключевых кадров непосредственно для контроллера или же самого параметра. Обе приведенные ниже строки кода совершенно равнозначны:

```
animate on ( at time 30 s.radius = 50 )
animate on ( at time 30 s.radius.controller.value = 50 )
```

При написании сценариев предпочтительнее использовать более короткую запись. Но при установке ключевых кадров анимации вращения может возникнуть потребность присвоить значение угла поворота свыше 360° непосредственно контроллеру, не прибегая к методу **angleAxis.**

Рассмотрим для примера две следующие строки кода:

at time 50 s.rotation.z_rotation = 720
attime 50 s.rotation.controller[3].controller.value = 720

Допустим, что в качестве контроллера управления вращением назначен контроллер **EulerXYZ.** При выполнении первой строки кода вращения не происходит, поскольку угол поворота определяется в виде остатка от деления на 360. А во второй строке кода устанавливается ключевой кадр, вынуждающий объект совершить два полных оборота вокруг своей оси Z.

- Использовать свойство .**keys**, содержащее массив ключевых кадров, установленных для конкретного контроллера:

s.radius.keys[2].value = 40

В этой строке кода изменяется значение второго ключевого кадра, установленного для контроллера, управляющего анимацией радиуса сферы.

Если значения ключевых кадров изменяются с помощью свойства **.keys**, нужно точно знать контроллер, которому эти значения присваиваются. Присваиваемое значение определяется для ключевых кадров контроллера. Если создать контроллер в виде экземпляра и назначить его для нескольких оживляемых параметров в сцене, то изменение отдельного значения с помощью свойства **.keys** коснется всех параметров, для которых назначен данный контроллер.

Тема 5. Работа с объектами на языке MAXScript

Применение модификаторов

Все, что создается в MAXScript, считается объектом или элементом определенного типа, даже при отсутствии физического проявления. Модификаторы также считаются объектами, хотя они и не являются трехмерными объектами, а скорее относятся к элементам, обладающим определенными свойствами, которые можно устанавливать и изменять аналогично трехмерным объектам сцены.

Сначала создается объект-модификатор, а затем устанавливаются его свойства. Рассмотрим пример. Для создания модификатора можно воспользоваться конструктором **bend** и указать угол сгибания, введя следующий код:

b = bend angle:90

Переменная **b** содержит ссылку на модификатор **Bend**, тогда как угол сгибания (**angle**) относится к свойствам данного модификатора. Теперь модификатор можно применить к любому трехмерному объекту.

Например, создадим цилиндр:

c = cylinder height:50 heightsegs:20

Для применения модификатора к цилиндру необходимо вызвать функцию **addmodifier**:

addmodifier c b

В примере модификатор применяется к цилиндру, после имени функции указываются ссылка на цилиндр и ссылка на данный модификатор. А поскольку конкретная ось действия модификатора не указывается, то по умолчанию в 3dsMax используется ось Z.

Свойства модификатора можно изменить тремя способами.

1. Присвоить явную ссылку на модификатор. В приведенном выше упражнении переменной **b**была присвоена ссылка на модификатор. После этого

свойства переменной **b** могут быть установлены с помощью таких выражений, как следующее:

b.angle = 80

При установке свойства объект автоматически обновляется в видовом окне. Так, в приведенном выше упражнении это оказало влияние не на новый, а на исходный цилиндр, поскольку к нему был применен модификатор **Bend**, ссылка на который делается в переменной **b**.

2. Изменить свойства модификатора посредством объекта, к которому он применяется. Если используется метод **addmodifier**, модификатор выполняет роль свойства геометрического объекта. Изменения могут быть внесены в модификатор следующим образом:

c.bend.angle = 75

Доступ в этой строке кода осуществляется сначала к свойству **bend** цилиндра **c**, а затем к свойству **angle** модификатора **Bend.** Таким способом удобно пользоваться в отсутствие явной ссылки на модификатор.

3. Получить доступ к массиву свойств модификаторов, применяемых к объекту. Каждый объект имеет свойство **.modifiers**, возвращающее массив модификаторов, применяемых к данному объекту. Первым элементом данного массива является модификатор, находящийся на вершине стека в том виде, в каком он представлен на панели команд **Modify**. Вторым элементом массива является следующий вниз по стеку модификатор и так далее до последнего элемента массива, который соответствует модификатору, находящемуся на дне стека. Если воспользоваться упоминавшимся ранее примером цилиндра, то можно написать следующее:

c.modifiers[1].angle

В этой строке кода осуществляется доступ к свойству **angle** модификатора **Bend**. Такой способ доступа к свойствам модификатора удобен для выполнения операций над всеми модификаторами, применяемыми к объекту.

Создание источников света и камер

В MAXScript можно создать любой тип источника света из тех, которые доступны в пользовательском интерфейсе 3ds Max. У каждого типа источника света имеется свой конструктор. Например, конструктор свободно направленного источника света выглядит следующим образом:

directionalLight()

Многие свойства источников света являются общими для них. К ним относятся следующие свойства:

источник_света.rgb – цвет RGB;

источник_света.excludeList – массив узлов, исключаемых из световых эффектов;

источник_света.projectorMap – карта, хранящаяся в объекте textureMap.

В свойствах источников света можно также указывать растровые текстуры **BitmapTexture**.

В следующем примере создается всенаправленный источник света с устанавливаемыми по умолчанию свойствами и положением.

myLight = omnilightpos:[0, -100, 100] castshadows:on

Управление камерами осуществляется в MAXScript достаточно просто. Ниже приведены некоторые конструкторы для свободных и нацеленных камер:

targetCamera()
freeCamera()

Для создания камеры в сцене сначала создается целевой объект для нацеленной камеры. У данного объекта, выполняющего роль цели камеры, отсутствуют собственные свойства.

tobj = targetObject pos: [7,15,31]

Далее создается сама камера и для нее указывается цель:

tc= targetCamera pos:[0, 0, 40.0] target:tobj

Когда камера перемещается, она остается постоянно направленной на цель. Например, если ввести следующий код:

move tc [-121,-166,60]

Камера постоянно будет направлена на цель.

Для того чтобы назначить текущее видовое окно для камеры используют следующий код:

viewport.setcamera tc

Работа с материалами

Средствами MAXScript можно получить доступ к редактору материалов, а также к отдельным материалам, применяемым к объектам. В частности, для доступа к материалу, назначенному для объекта, служит свойство **.material** или **.mat**:

узел.material
узел.mat

А для доступа к материалу в редакторе материалов служит виртуальный массив **meditMaterials**. Этот массив организуется в 3dsMax автоматически и индексируется номерами соответствующих позиций образцов в редакторе материалов, например:

meditmaterials [3]

При выполнении этой строки кода возвращается материал, находящийся на третьей позиции образца в редакторе материалов.

Массив **meditMaterials** состоит только из 24 элементов. Так, обращение **meditMaterials[25]** к данному массиву приведет к ошибке.

Материалы можно создавать, как отдельные объекты, и назначать их для объектов сцены. Каждый тип материала имеет свой конструктор. Например, конструктор стандартного материала имеет следующий вид:

standard()

Как только стандартный материал создан, его карты становятся доступными в виде свойств данного материала:

sm = standard() sm.diffusemap

Разные виды карт можно также создавать, как отдельные объекты, и назначать их для отдельных каналов проецирования:

ch = checker()
sm.diffusemap = ch

Для отображения карты в видовом окне служит метол **showTextureMap:**

showTextureMap sm ch on

В этой строке кода активизируется режим отображения карты клетчатого рисунка ch в материале sm.

Управление средством визуализации

В MAXScript имеется возможность устанавливать ряд параметров для автоматического управления средством визуализации. Визуализацию проще всего организовать с помощью метода render. Его синтаксис следующий:

render [camera: <camera_node>] [frame: <number> | #current]

Метод **render** имеет около 45 необязательных параметров.

Ниже перечислены некоторые устанавливаемые по умолчанию параметры и режимы визуализации.

- Визуализация в активном видовом окне.
- Использование текущих настроек визуализации.
- Визуализация в виртуальный буфер кадров.

Устанавливаемые по умолчанию параметры могут быть переопределены. Ниже приведены некоторые необязательные параметры визуализации.

- **camera: <camera>** – после создания камеры визуализацию можно выполнять с точки зрения камеры, используя объект камеры.
- **frame: <number>** или **#current** – для визуализации конкретного кадра, например кадра 10, достаточно ввести следующую строку кода:

render frame:10

Для указания средству визуализации текущего номера кадра служит обозначение **#current**:

render frame:#current

- **frameRange**: **<interval>** или **#active** – этот параметр задает диапазон кадров анимации в следующем виде:

frameRange:(interval 0 100)

С помощью обозначения **#active** средству визуализации указывается активный в данный момент временной интервал:

render frameRange:#active

- **fromframe: <number>** и **toframe: <number>** – если используется только свойство **fromframe**, визуализация начинается с указанного кадра и продолжается до последнего кадра. Если же используется только свойство **toframe**, визуализация начинается с текущего кадра и завершается в кадре с номером **toframe**.
- **nthframe: <number>** – устанавливает режим визуализации через каждые n кадров.
- **outputfile: <string>** – указывает путь к файлу вывода.
- **outputwidth: <number>** – задает выходную ширину.
- **outputheight: <number>** – задает выходную высоту.
- **pixelaspect: <number> –** задает выходные пропорции элемента изображения.
- **renderElements**: **<bool>** – если передается логическое значение **true** этого параметра либо его значение вообще не передается, визуализации подлежат любые визуализируемые элементы сцены.
- **renderMultiPassEffects: <bool>** – если передается логическое значение **true** этого параметра либо его значение вообще не передается, а для текущей камеры активизирован эффект многопроходной визуализации, то визуализируется именно этот эффект.
- **renderElementBitmaps: <&var>** – если визуализация выполняется по элементам, растровые изображения отдельно выводимых элементов визуализации помещаются в массив, который возвращается в переменной, указываемой по ссылке.

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

2.1. План практических работ

Практическая работа 1. Работа с примитивами. Элементы трансформации в MAXScript

Цель работы: научиться создавать крипты на языке MAXScript. Работать с примитивами и элементами трансформации.

Вопросы, изучаемые на лабораторной работе

Создание скрипта.

Создание примитивов.

Перемещение, масштабирование и поворот объектов.

Работа с объектами 3dsMax.

Контрольные вопросы

Где создаются скрипты MAXScript? Как выполнить скрипт MAXScript? Где отображаются ошибки, допущенные в скрипте? Как создать сферу и изменить ее радиус в MAXScript?

Практическая работа 2. Условные операторы и циклы в языке MAXScript

Цель работы: научиться работать с условными операторами. Уметь работать с циклами и задавать его пределы.

Вопросы, изучаемые на лабораторной работе

Условные операторы с условием.

Циклы с заданным количеством повторов.

Случайное задание значений.

Контрольные вопросы

Какие условные обозначения используются в операторе условия? Как задать случайную окраску объекта? Каким образом можно создать 30 сфер? Как изменить число повторений в цикле?

Практическая работа 3. Создание пользовательского интерфейса в MAXScript

Цель работы: научиться создавать пользовательский интерфейс на языке MAXScript и работать с его элементами.

Вопросы, изучаемые на лабораторной работе

Создание свитков.

Работа со счетчиками, кнопками.

Создание обработчиков событий.

Контрольные вопросы

Как создать из свитка диалоговое окно? Как создать реакцию пользователя на нажатие кнопки? Как задать размеры для диалогового окна?

Практическая работа 4. Манипулирование объектами средствами MAXScript

Цель работы: познакомиться с дополнительными элементами интерфейса, научиться работать со слайдерами.

Вопросы, изучаемые на лабораторной работе

Создание групп в пользовательском интерфейсе.

Создание слайдеров.

Задание ограничений для счетчиков.

Взаимодействие элементов интерфейса с объектами в сцене.

Контрольные вопросы

Как создать группу из элементов интерфейса? Как задать значения для слайдеров? Как связать элементы интерфейса с объектами в сцене?

Практическая работа 5. Создание сценария для формирования Солнечной системы

Цель работы: познакомиться с расширенными настройками диалогового окна, научиться создавать анимацию вдоль пути средствами языка MAXScript.

Вопросы, изучаемые на лабораторной работе

Визуальные настройки интерфейса.

Анимация вдоль пути.

Динамическое управление параметрами объекта в сцене.

Контрольные вопросы

Как задать ширину для элементов интерфейса? Как задать анимацию вдоль пути? Каким образом можно изменить параметры объекта сцены, используя элементы интерфейса?

Практическая работа 6. Работа с модификаторами в 3dsMax на языке MAXScript

Цель работы: научиться применять и изменять настройки модификаторов 3dsMax на языке MAXScript, работать с файлами помощи.

Вопросы, изучаемые на лабораторной работе

Создание конструктора модификатора.

Изменение параметров модификаторов.

Применение нескольких модификаторов к одному объекту.

Контрольные вопросы

Как применить несколько модификаторов к одному объекту? Как изменить настройки модификатора после его создания. Какая функция используется для применения модификаторов?

Практическая работа 7. Применение материалов в 3dsMax на языке MAXScript

Цель работы: научиться применять и изменять настройки материалов 3dsMax на языке MAXScript, работать с файлами помощи.

Вопросы, изучаемые на лабораторной работе

Поиск нужного параметра в файле помощи.

Создание стандартных материалов.

Изменение параметров материалов.

Контрольные вопросы

Как создать материал Raytrace и применить его к объекту? Как изменить цвет материала Standard? Как применить созданный материал к объекту? Как добавить материал в слот материалов программы 3dsMax?

Практическая работа 8. Анимация объектов в 3dsMax на языке MAXScript

Цель работы: научиться создавать анимацию объектов сцены программы 3dsMax на языке MAXScript.

Вопросы, изучаемые на лабораторной работе

Создание ключевых кадров.

Анимация параметров модификатора.

Изменение скорости анимации.

Контрольные вопросы

Как создать анимацию падающего мяча в MAXScript?Как изменить скорость анимации? Как создать ключевой кадр?

Практическая работа 9. Файловый ввод и вывод в MAXScript

Цель работы: научиться работать с текстовыми файлами и читать из него информацию для создания объектов сцены, применять строковые функции.

Вопросы, изучаемые на лабораторной работе

Открытие текстового файла.

Чтение строк из текстового файла.

Работа со строками, основные функции.

Контрольные вопросы

Как открыть текстовый файл в MAXScript? Как разбить строку на элементы массива по определенному символу? Как работать с массивами строк?

3. РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

3.1. Вопросы для подготовки к зачетам и экзаменам

3.1.1. Вопросы для подготовки к экзамену

Сценарий. Синтаксис и организация сценария.

Операторы. Многострочные операторы. Комментарии.

Переменные. Типы данных.

Типы данных Float и Integer.

Тип данных Point3.

Логические данные, примеры использования.

Локальные и глобальные переменные.

Объекты. Свойства объектов.

Два способа задания свойств объектам.

Понятие функции.

Аргументы функции.

Параметры и обозначения функций.

Способы вызова функций.

Способы обращения к объектам в сцене.

Свойство положения объектов. Функция перемещения.

Свойство масштабирования. Функция изменения масштаба.

Свойство поворота. Функция вращения.

Функции копирования объектов.

Условные операторы.

Циклы for.

Циклы while do и do while.

Массивы. Доступ к элементам массива.

Массивы. Операции с массивами.

Строки. Объявление строк.

Строки. Операции над строками.

Функция messageBox().

Функция LoadMaxFile().

Выбор объектов в сцене с помощью интерфейса.

Создание пользовательской кнопки.

Построение пользовательского интерфейса.

Свитки. Объявление и вызов.

Создание диалогового окна.

Элементы интерфейса. Основные параметры.

Элемент интерфейса spinner, особенности использования.

Элемент интерфейса button, особенности использования.

Элемент radiobutton, особенности использования.

Элемент интерфейса edittext, особенности использования.

Использование элемента интерфейса PickButton.

Создание события для элементов интерфейса.

Применение модификаторов.

Создание и применение модификатора Bevel.

Создание текста средствами MAXScript.

Создание источников света и камер.

Применение материалов.

Создание и применение текстурных карт.

Управление средствами визуализации.

Анимация. Создание ключевых кадров.

Редактор Visual MAXScript.

4. ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

4.1. Учебная программа

ЧАСТНОЕ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
"ИНСТИТУТ СОВРЕМЕННЫХ ЗНАНИЙ ИМЕНИ А.М. ШИРОКОВА"

УТВЕРЖДАЮ
Ректор Института современных знаний
имени А.М.Широкова
_____ А.Л. Капилов

_____
(дата утверждения)
Регистрационный № УД-_____/уч.

ОСНОВЫ ЯЗЫКА MAXSCRIPT
Учебная программа учреждения высшего образования
по учебной дисциплине для специальности:
1-19 01 01 Дизайн (по направлениям), направление специальности
1-19 01 01- 06 Дизайн (виртуальной среды)

2017 г.

Учебная программа составлена на основе образовательного стандарта ОСВО 1-19 01 01-2013 и учебного плана Института современных знаний имени А. М. Широкова по направлению специальности 1-19 01 01-06 Дизайн (виртуальной среды).

СОСТАВИТЕЛЬ:
Ю.Д. Васильева, доцент кафедры высшей математики и информатики Института современных знаний имени А.М. Широкова, кандидат технических наук

РЕЦЕНЗЕНТЫ:
Ю.В. Виланский, ведущий инженер-программист отдела научно-технических разработок и системного сопровождения проектов научно-производственного частного унитарного предприятия «ТЕТРАЭДР», к.т.н., доцент
В.В.Захаров, доцент кафедры интеллектуальных информационных технологий Белорусского государственного университета информатики и радиоэлектроники, кандидат технических наук

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:
Кафедрой высшей математики и информатики Института современных знаний имени А.М.Широкова (протокол № 12 от 28 июня 2017 года);
Научно-методическим советом Института современных знаний имени А.М.Широкова (протокол № 5 от 29 июня 2017 года)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

В курсе «Основы языка MAXScript» предполагается освоение принципов и методов создания сценариев MAXScript. Студенты знакомятся с основами объектно-ориентированного программирования в пакете компьютерной графики Autodesk 3ds Max.

Цель дисциплины – освоить язык программирования, который позволяет создавать и отлаживать сценарии для построения пользовательского интерфейса, автоматизацию операций преобразования объектов, текстурирования, освещения, анимации и визуализации сцен в среде трехмерного моделирования приложения Autodesk 3ds Max.

Для достижения этой цели необходимо решение следующих учебных задач:

изучение синтаксис и организацию сценария;

освоение принципы пользовательского интерфейса;

овладение набором команд для взаимодействия с элементами трехмерной среды;

изучение основных команд для трансформации трехмерных объектов средствами языка программирования;

освоение принципов работы с объектами и их возможных модификаций.

Курс излагается с опорой на развитие самостоятельного мышления студентов, самообразования. Для выполнения студентами выбираются задания, допускающие при желании значительную степень углубления.

Дисциплина «Web-дизайн» относится к циклу специальных дисциплин и опирается на знания и умения студентов, полученные при изучении таких дисциплин как «Компьютерная анимация» и «Информационные технологии в дизайне».

Освоение образовательной программы по учебной дисциплине «Основы языка MaxScript» должно обеспечить формирование следующих академических компетенций:

АК-1. Владеть базовыми научно-теоретическими знаниями в области художественных, научно-технических, общественных и практических задач профессиональной деятельности.

АК-2. Владеть методикой системного и сравнительного анализа, междисциплинарным подходом к решению проблем, находить решения на стыке разных дисциплин, связанных с теорией и практикой дизайна.

АК-3. Владеть исследовательскими навыками.

АК-4. Уметь работать самостоятельно.

АК-5. Быть способным к творческой, креативной работе.

АК-6. Владеть междисциплинарным подходом при решении проблем.

АК-7. Иметь навыки использования современ6ных технических средств обработки информации.

АК-9. Уметь учиться, быть расположенным к постоянному повышению профессиональной квалификации.

Также студент должен приобрести следующие социально-личностные компетенции:

СЛК-2. Совершенствовать и развивать свой интеллектуальный и общекультурный уровень, повышать проектно-художественное мастерство.

СЛК-6. Быть способным к критике и самокритике.

После изучения учебной дисциплины студент должен владеть следующими профессиональными компетенциями и быть способным:

ПК-2. Осуществлять дизайн-проектирование с учетом соотношения смыслообразующих и формообразующих факторов (художественно-формальных, эргономических, инженерно-психологических, технологических, конструктивных, экологических, социально-культурных, экономических) в условиях как аналогового, так и безаналогового проектирования.

ПК-3. Формировать выразительное образное решение объекта проектирования на основе конкретного содержания.

ПК-4. Осуществлять прогностическое дизайн-проектирование с использованием инновационных технологий.

ПК-5. Осуществлять экспертную оценку уровня дизайнерского решения по основным смыслообразующим и формообразующим факторам.

ПК-6. Адаптироваться к изменению объекта профессиональной деятельности, как в пределах специализации, так и направление специальности.

ПК-7. Осуществлять развитие научно-теоретической и практической базы обеспечения дизайн-деятельности.

ПК-9. Собирать, анализировать и систематизировать профессиональный опыт в области дизайн-деятельности.

ПК-10. Выявлять общие закономерности функционирования и развития дизайн-деятельности на основе собранного факто логического материала.

ПК-11. Анализировать композиционные, конструктивные, технологические, эргономические и колористические решения продуктов дизайн-деятельности.

ПК-12. Анализировать результаты собственных дизайн-решений.

ПК-18. Уметь проектировать, организовывать, анализировать процесс педагогического взаимодействия при освоении профессиональных компетенций по направлению специальности.

В результате изучения данного курса студенты должны

знать:

основы алгоритмизации и программирования;

основные методы моделирования компьютерной графики с использованием языков программирования;

уметь:

интерактивно взаимодействовать с пользователем средствами программирования;

создавать интерактивные приложения в трехмерной среде пакета 3ds Max;

иметь представление:

о дополнительных возможностях среды программирования в сфере разработки трехмерных сцен;

о функциях и методах для создания мультимедийных приложений;

о взаимодействии элементов сцены друг с другом средствами языка программирования.

В соответствии с учебными планами специальности «Дизайн» (по направлениям), направление специальности «Дизайн (виртуальной среды)» дисциплина изучается на протяжении одного семестра. Общее количество часов – 140, в том числе аудиторных – 52 часа.

Форма получения высшего образования – дневная.

Дисциплина изучается в седьмом семестре: 12 часов лекций и 40 часов практических занятий. На самостоятельную работу отводится 88 часов.

Текущая аттестация по дисциплине проводится в форме экзамена.

СОДЕРЖАНИЕ УЧЕБНОГО МАТЕРИАЛА

Тема 1. Основы языка MAXScript

Синтаксис и организация сценария. Комментарии. Переменные и данные. Приемник команд MAX-Script Listener. Свойства объектов. Функции. Локальные и глобальные переменные. Условные операторы. Строки. Циклические конструкции. Создание и выполнение сценариев. Грамматика MAXScript. Макрорегистратор.

Тема 2. Построение пользовательских интерфейсов

Типы пользовательских интерфейсов. Виды сценариев. Макросценарии. Файлы сценариев. Свитки. Создание пользовательского интерфейса. Ввод элементов. Обработчик событий. Точная настройка пользовательского интерфейса. Создание рабочего сценария. Ввод событий для элементов. Включение и отключение элементов. Редактор Visual MAXScript.

Тема 3. Работа с интерфейсом 3dsMax

Функция mouseTrack. Клонирование объектов. Копирование массивов. Команды Max. Панели команд. Основная панель инструментов. Выбор объектов на сцене.

Тема 4. Преобразования и анимация

Изменение положений объектов. Масштабирование объектов. Методы вращения объектов. Метод EulerAngle. Локальные вращения. Метод AngleAxis. Работа с анимацией. Контроллеры. Ключевые кадры анимации. Значения ключевых кадров. Типы контроллеров. Ось вращения. Удаление ключевых кадров.

Тема 5. Работа с объектами

Применение модификаторов. Источники света и камеры. Работа с материалами. Управление средствами визуализации.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					Самостоятельная работа студентов (СРС)	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	2	3	4	5	6	7	8	9
1	Основы MaxScript	2	4				8	Результаты выполнения заданий по теме практического занятия
2	Построение пользовательских интерфейсов	2	8				20	
3	Работа с интерфейсом 3dsMax	2	8				20	
4	Преобразования и анимация	2	8				20	
5	Работа с объектами	4	12				20	
	Итого	**12**	**40**				**88**	

ИНФОРМАЦИОННО-МЕТОДИЧЕСКАЯ ЧАСТЬ

СПИСОК ЛИТЕРАТУРЫ

Основная

1. Основы 3ds Max 8 MAXScript: официальный учебный курс от Autodesk. – М. : Издательский дом «Вильямс», 2006. – 256 с.

2. *Бондаренко, С.В.* 3ds Max 8. Библиотека пользователя (+CD) / С. В. Бондаренко, М. Ю. Бондаренко. – СПб. : Питер, 2006. – 608 с.

3. *Мэрдок, К. Л.* 3ds Max 2008. Библия пользователя / К. Л. Мэрдок. – М. : Диалектика, 2008.

Дополнительная

1. *Тозик, В.* 3ds Max 8. Трехмерное моделирование и анимация / В. Тозик, А. Меженин. – СПб. :2006.

2. *Мердок, К. Л.* 3ds Max 2013. Библия пользователя / К. Л. Мердок. – М. : Диалектика, 2013.

3. http://graphic.csportal.ru/3d-graphic/Max-Script-Lessons/

ЭЛЕКТРОННЫЕ РЕСУРСЫ

1. Статьи по MaxScript. [Электронный ресурс]. – Режим доступа: http://www.3dmax-tutorials.ru/publ/max_script/6 – Дата доступа: 16.05.2017.

2. MAXScript. Теория, уроки, видеокурс [Электронный ресурс]. – Режим доступа: http://www.scriptattack.com/theory/theory.html – Дата доступа: 22.04.2017.

3. ScriptSpot | Your community resource for 3ds Max tools [Электронный ресурс]. – Режим доступа: http://www.scriptspot.com – Дата доступа: 02.05.2017.

ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

1. Работа с примитивами. Элементы трансформации в MAXScript.
2. Условные операторы и циклы в языке MAXScript.
3. Создание пользовательского интерфейса в MAXScript.
4. Манипулирование объектами средствами MAXScript.
5. Создание сценария для формирования Солнечной системы.
6. Работа с модификаторами в 3dsMax на языке MAXScript.
7. Применение материалов в 3dsMax на языке MAXScript.
8. Анимация объектов в 3dsMax на языке MAXScript.
9. Файловый ввод и вывод в MAXScript.

ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

№п/п	Название раздела, темы	Кол-во часов на СРС	Задание	Форма выполнения	Цель илизадача СРС
1	2	3	4	5	6
1	Основы MaxScript	8	Изучить литературные источники и электронные ресурсы по теме.	Чтение литературы, посещение интернет-ресурсов	Ознакомиться с основными возможностями программирования в среде приложения 3dsMax.
2	Построение пользовательских интерфейсов	20	С помощью языка MaxScript построить интерактивные элементы в программе трехмерной графики 3dsMax	Предоставить скрипт в формате ms	Закрепление навыков использования знаний построения объектов
3	Работа с интерфейсом 3dsMax	30	С помощью языка MaxScript создать пользовательский интерфейс для применения модификаторов и материалов	Предоставить скрипт в формате ms	Закрепление навыков создания пользовательских интерфейсов
4	Преобразования и анимация	10	С помощью языка MaxScript построить анимацию скачущего мяча	Предоставить скрипт в формате ms	Закрепление навыков работы с объектами и применения пользовательских функций
5	Работа с объектами	20	С помощью языка MaxScript из внешнего файла построить объекты в сцене 3dsMax	Предоставить скрипт в формате ms	Закрепление навыков работы с языком MaxScript

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ УВО

Название учебной дисциплины, с которой требуется согласование	Название кафедры	Предложения об изменениях в содержании учебной программы Института по учебной дисциплине	Решение, принятое кафедрой, разработавшей учебную программу (с указанием даты и номера протокола)

ДОПОЛНЕНИЯ И ИЗМЕНЕНИЯ К УЧЕБНОЙ ПРОГРАММЕ УВО

на _____ /_____ учебный год

№ пп	Дополнения и изменения	Основание

Учебная программа пересмотрена и одобрена на заседании кафедры высшей математики и информатики (протокол № ____от _____201__г.)

Заведующий кафедрой

_____ _____ _____

(ученая степень, ученое звание) (подпись) (И.О.Фамилия)

УТВЕРЖДАЮ
Декан факультета

_____ _____ _____

(ученая степень, ученое звание) (подпись) (И.О.Фамилия)

(ученая степень, ученое звание) (подпись) (И.О.Фамилия)

4.2. Основная литература

1. Основы 3ds Max 8 MAXScript: официальный учебный курс от Autodesk. – М. : Издательский дом «Вильямс», 2006. – 256 с.

2. Бондаренко, С. В. 3ds Max 8. Библиотека пользователя (+CD) / С. В. Бондаренко, М. Ю. Бондаренко. – СПб. : Питер, 2006. – 608 с.

3. Мэрдок, К. Л. 3ds Max 2008. Библия пользователя / К. Л. Мэрдок. – М. : Диалектика, 2008.

4.3. Дополнительная литература

4. Тозик, В. 3ds Max 8. Трехмерное моделирование и анимация / В. Тозик, А. Меженин. – СПб., 2006.

5. Мердок, К. Л. 3ds Max 2013. Библия пользователя / К. Л. Мердок. – М. : Диалектика, 2013.

4.4. Ресурсы Интернет

6. Статьи по MAXScript. [Электронный ресурс]. – Режим доступа: http://www.3dmax-tutorials.ru/publ/max_script/6 – Дата доступа: 16.05.2017.

7. MAXScript. Теория, уроки, видеокурс [Электронный ресурс]. – Режим доступа: http://www.scriptattack.com/theory/theory.html – Дата доступа: 22.04.2017.

8. ScriptSpot | Your community resource for 3ds Maxtools [Электронный ресурс]. – Режим доступа: http://www.scriptspot.com – Дата доступа: 02.05.2017.

4.5. Техническое и программное обеспечение дисциплины

Для проведения лекционных занятий используется аудитория, оснащенная персональным компьютером с подключенной к нему видеопроекционной установкой и экран, на который выводятся слайды презентаций, видеоролики и окна интерфейса программ по тематике лекций.

Для проведения лабораторных работ используется компьютерный класс с персональными компьютерами, работающими под управлением операционной

системы Windows. Для выполнения лабораторных работ используется программа Autodesk 3dsMax.

СОДЕРЖАНИЕ

Учебное электронное издание

Автор-составитель
Васильева Юлия Дмитриевна

ОСНОВЫ ЯЗЫКА MAXSCRIPT

*Электронный учебно-методический комплекс
для студентов специальности 1-19 01 01 Дизайн (по направлениям),
направление специальности 1-19 01 01-06 Дизайн (виртуальной среды)*

[Электронный ресурс]

Редактор *И. П. Сергачева*
Технический редактор *Ю. В. Хадьков*

Подписано в печать 30.07.2018.
Гарнитура Times Roman. Объем 0,9 Мб

Частное учреждение образования
«Институт современных знаний имени А. М. Широкова»
Свидетельство о регистрации издателя №1/29 от 19.08.2013
220114, г. Минск, ул. Филимонова, 69.

ISBN 978-985-547-231-6