

Учебник по Blitz3d

доработанное издание v1.1. Автор и редактор: Солопий Артём.

Содержание

Введение.

Глава1. Введение в Blitz3d.

Пункт1.1. Интерфейс программы.

Пункт1.2. Синтаксис программы.

Глава2. Основы языка программирования.

Пункт2.1. Типы данных.

Пункт2.2. Метод задания числовых величин.

Пункт2.3. Константы.

Пункт2.4. Переменные.

Пункт2.5. Выражения.

Пункт2.6. Метки.

Пункт2.7. Операторы ввода с клавиатуры и вывода на экран.

Пункт2.8. Управляющие конструкции.

Пункт2.8.1. Проверка условий.

Пункт2.8.2. Циклы.

Пункт2.9. Функции.

Пункт2.10. Процедуры.

Пункт2.11. Строка констант.

Пункт2.12. Массивы.

Пункт2.13. Банки.

Пункт2.14. Типы.

Пункт2.15. Операции со строками.

Пункт2.16. Файлы.

Глава3. Расширяем возможности.

Пункт3.1. Подключение файлов проекта.

Пункт3.2. Преобразование типов.

Пункт3.3. Математические функции.

Пункт3.4. Работа с устройствами ввода.

Пункт3.5. Системные функции.

Пункт3.6. Таймер.

Глава4. Работа со звуковыми файлами.

Пункт4.1. Работа со звуками.

Глава5. 2D графика.

Пункт5.1. Графический режим.

[Пункт5.1.1. Подключение 2D графического режима.](#)
[Пункт5.1.2. Графические буферы.](#)
[Пункт5.2. Рисование фигур.](#)
[Пункт5.3. Цвет, экран.](#)
[Пункт5.4. Работа с изображениями.](#)
[Пункт5.4.1. Изображения.](#)
[Пункт5.4.2. Буфер изображения.](#)
[Пункт5.4.3. Анимационные изображения.](#)
[Пункт5.4.4. Проверка на столкновения изображений.](#)
[Пункт5.5. Текст в графическом режиме.](#)
[Пункт5.6. Работа с пикселями.](#)
[Глава6. Работа с видео файлами.](#)
[Глава7. 3D графика.](#)
[Пункт7.1. 3D Графический режим.](#)
[Пункт7.2. Камера.](#)
[Пункт7.3. Примитивы.](#)
[Пункт7.4. Работа с поверхностью объектов.](#)
[Пункт7.4.1. Цвет объекта, прозрачность и др.](#)
[Пункт7.4.2. Текстура объекта.](#)
[Пункт7.4.3. Кисти.](#)
[Пункт7.5. Движение объектов.](#)
[Пункт7.6. Управление объектами. Родитель. Точка вращения.](#)
[Пункт7.7. Статус объектов в мире.](#)
[Пункт7.8. Создание своих объектов.](#)
[Пункт7.9. Загрузка объектов.](#)
[Пункт7.10. Анимация объектов.](#)
[Пункт7.11. Столкновение объектов.](#)
[Пункт7.12. Свет.](#)
[Пункт7.13. Спрайты.](#)
[Пункт7.14. Озвучивание 3D объектов.](#)
[Заключение.](#)

Введение.

Данный учебник разбит на несколько глав, в каждую главу входят несколько пунктов по определённой теме. Если вас интересует не весь учебник, а конкретная тема, то вы можете перейти к ней по ссылке в оглавлении.

Условные сокращения используются в описании команд. Например, перем - сокращенное от "переменная", переменная файла - "перем_файла". Если какие-нибудь значения команд ограничены квадратными скобками [и], значит при написании программы эти значения можно опустить (не

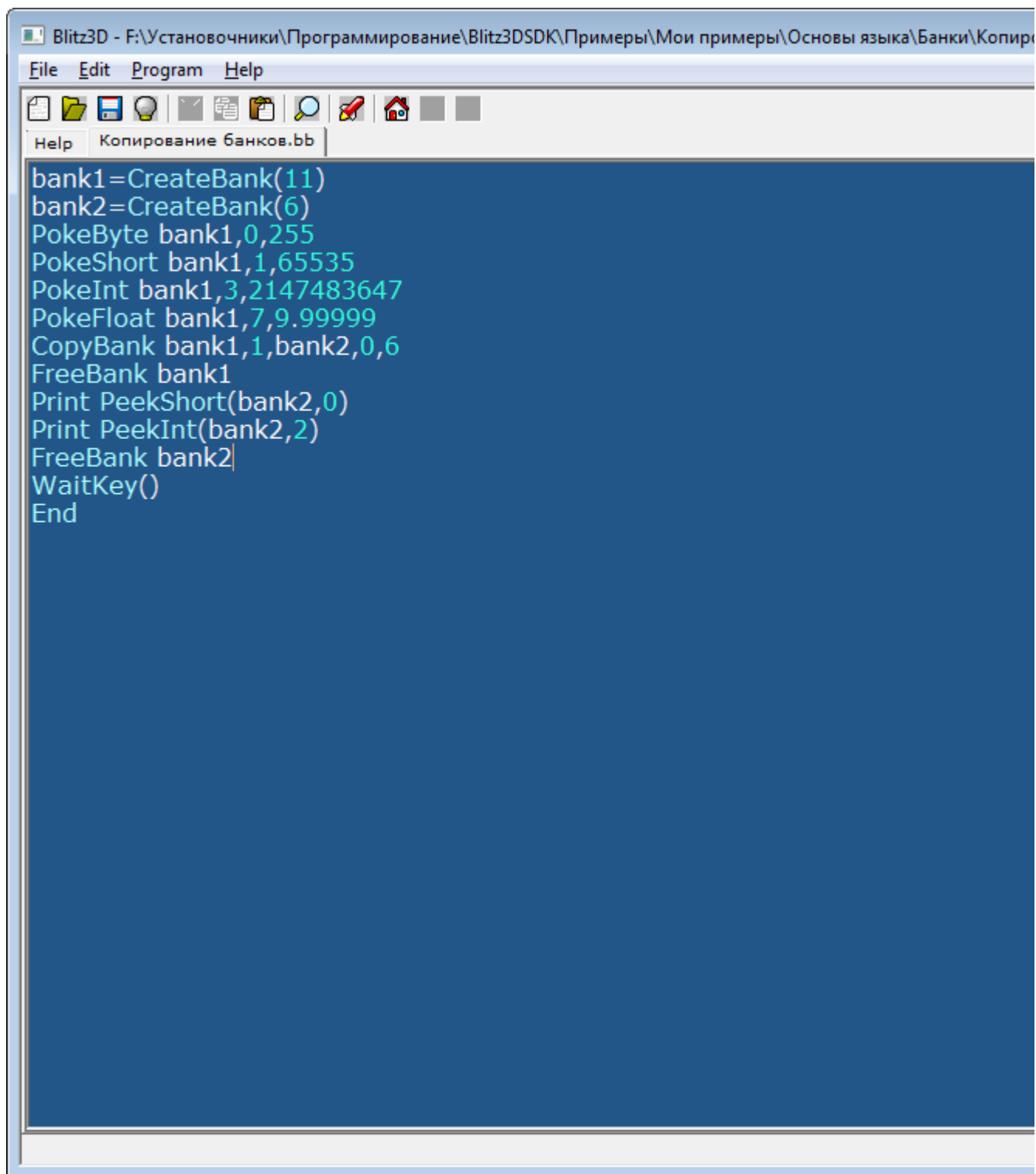
использовать), эти скобки использую для необязательных параметров.

Глава1. Введение в Blitz3D.

Blitz3D - это среда программирования, работающая с двухмерной и трёхмерной графикой, для этого используется Microsoft DirectX 7.0. За основу взят язык Basic, поэтому его знание облегчит изучение Blitz3D. Сперва будет объяснен интерфейс программы, потом сам язык программирования.


Пункт1.1. Интерфейс программы.












Так выглядит окно Blitz3d:



Синее текстовое поле является основным – в нём пишут программный код.

Кнопки панели инструментов:

 - (New) – создание нового проекта

-  - (Open) - открытие существующего проекта
-  - (Save) - сохранение проекта
-  - (Close) - закрытие текущего проекта
-  - (Cut) - вырезать в буфер обмена
-  - (Copy) - копировать в буфер обмена
-  - (Paste) - вставить
-  - (Find) - найти текст
-  - (Run) - запуск проекта
-  - (Home) - перейти на страницу помощи
-  - (Back) - вернуться на предыдущую страницу в окне помощи
-  - (Forward) - вернуться на следующую страницу в окне помощи

Для закрытия\открытия панели инструментов: меню Edit -> ShowToolbars или Shift + Esc

Запуск и отладка программы:

Для запуска программы нажмите F5 или 

Для запуска предыдущей программы нажмите F6

Для проверки ошибок нажмите F7

Для создания .exe файла нажмите: меню Program -> Create Executable

Для включения\отключения отладочного окна: меню Program -> Debug Enabled

В правой части окна находится панель с тремя вкладками:

Funcs - отображает все функции, которые используются в проекте.

Types - отображает все типы, которые используются в проекте.

Label - отображает все метки.

Во время написания программы для краткого описания команды, возле которой находится текстовый курсор, нажмите F1.


Так выглядит окно отладочного окна:





В нем отображается код программы и выделенное место выполнения


текущей команды.


Кнопки на панели инструментов окна отладки:

 - остановка программы (пауза)

 - возобновление программы

 - без входа в функцию

 - вход в функцию

 - выход из функции

 - остановить выполнение

В правой части окна отладочного модуля находится панель с тремя вкладками:

Locals – отображает все локальные переменные и их значения

Globals – отображает все глобальные переменные и их значения

Consts - отображает константы их значения

Пункт 1.2. Синтаксис программы.

В Blitz3D имеются зарезервированные команды:

After, And, Before, Case, Const, Data, Default, Delete, Dim, Each, Else, ElseIf, End, EndIf, Exit, False, Field, First, Float, For, Forever, Function, Global, Gosub, Goto, If, Insert, Int, Last, Local, Mod, New, Next, Not, Null, Or, Pi, Read, Repeat, Restore, Return, Sar, Select, Shl, Shr, Step, Str, Then, To, True, Type, Until, Wend, While, Xor, Include.

Данные названия нельзя использовать в названии идентификаторов (переменных, констант, массивов, функций, процедур и типах). Зарезервированные слова автоматически окрашиваются в голубой цвет.

В программный код можно добавлять комментарии. Для их добавления используется символ ";" (точка с запятой). После этого символа прописывают нужный текст пояснений. Цвет текста комментариев - желтого цвета. Можно записать несколько команд в одну строку, которые должны отделяться знаком ":".

Пример:

```
x=x+1 : Print x
```

```
End; это комментарий
```

Глава2. Основы языка программирования.

В данной главе вы узнаете основы языка Blitz3D. Эта глава разбита на несколько пунктов по определенной теме. Итак, приступим.

Пункт2.1. Типы данных.

Blitz3d имеет следующие типы данных: числовые и строковые. Существует 2 типа числовых: целые и вещественные. Целый тип называется Integer, вещественный - Float, а строковый - String. Целый тип имеет значения в диапазоне(отрезке) от -2147483648 до 2147483647 (4 байта на число), а вещественный 4 байта на число. Вещественный тип - это дробно-рациональное число, записанное в десятичном виде, где запятой является точка. Оно может иметь шесть цифр после точки или записано в экспоненциальном виде. Например, 0.05 или .05 (ноль отбрасывается), или 5.e-001. Строковый тип - это строка различных символов. Значения этих типов данных хранятся в оперативной памяти компьютера и занимают столько места, сколько необходимо в зависимости от типа. Типом обладают переменные, константы, массивы, структуры (типы). О них говорится в следующих пунктах главы.

Пункт2.2. Метод задания числовых величин.

В языке используется 2 вида величин: переменные и константы. Их названия определяются с помощью букв латинского алфавита и цифр. Для их задания пользуются следующим правилом:

- Каждая величина начинается с буквы латинского алфавита.
- Цифры ставят только после букв латинского алфавита.
- Нельзя использовать буквы русского языка.
- Нельзя использовать специальные символы кроме "_", "%", "#", "\$". Их ставят только после букв, причем, после них не должны присутствовать никакие символы. Например, символы "%", "#", "\$" играют особую роль.

Если нарушить эти правила, то при запуске программы компилятор выдаст ошибку.

Пример правильных имен величин: a, b1, c\$, in%, dat#, fr_g.

Теперь поговорим о специальных символах:

% - величина будет целого типа.

Пример целых величин (Integer): **num%**, **life%**, **a_ser%**

- величина будет вещественного типа.

Пример вещественных величин (Float): **flt#**, **sred#**, **b_a#**, **g#**

\$ - величина будет строкового типа (String).

Пример вещественных величин (Float): **stroka\$**, **abc\$**, **ag_g\$**

Знак % для типа Integer можно опустить, так что переменные без % на конце тоже будут целого типа.

Пункт2.3. Константы.

Константой называется величина, которая не меняется в ходе выполнения программы. Она задаётся с помощью слова CONST. Константы бывают числового и строкового типа. Числовые константы могут быть записаны со знаком "+" (можно опустить) или "-". Пример задания константы:

```
Const a%=1
```

Можно задать несколько констант в один ряд через запятую:

```
Const g#=9.8, i=10, str$="Привет"
```

Знак = - это знак присвоения. Есть зарезервированные константы. Число пи: **Pi**.

Пункт2.4. Переменные.

Переменной называется величина, значение которой может меняться во время выполнения программы. Переменные бывают числового и строкового типа. Переменные типа Integer, которым не присвоили ничего, имеют значение 0. Для типа Float - 0.0, а переменные типа String имеют пустую строку.

Пример задания переменных:

```
a=90
```

```
i#=.5
```

```
$="Как дела?"
```

По доступности переменные бывают глобальные и локальные. Глобальные переменные - это переменные, которые используются в любой части

программы (в главной программе и в процедурах и функциях), они объявляются с помощью `Global`. Локальные переменные - это переменные, которые используются внутри функций и процедур, объявляются с помощью `Local`. Имена локальных переменных на разных уровнях программы могут совпадать, но их значения будут разными. Глобальные переменные объявляют только в главной программе до их использования.

Пример глобальной переменной:

```
Global i=5, s$="Hi"
```

Пример локальной переменной:

```
Local f, a$="How are you?"
```

Переменные являются локальными на том уровне программы, где им присвоили в первый раз значение. Переменным и константам можно присвоить значения не только в десятичной системе счисления, но и в двоичной и шестнадцатеричной. Если нужно присвоить число в шестнадцатеричной системе счисления, то перед ним ставят префикс `$`. Для двоичного ставят `%`. Эти префиксы подсказывают компилятору, что числа в шестнадцатеричной или двоичной системе счисления, который преобразует их значения в десятичную форму и заносит в переменную.

Например:

```
a=%0100  
b=$ff  
Print a+" "+b
```

На экран выйдет 4 и 255.

Пункт2.5. Выражения.

Выражением называется последовательность операндов, операций и скобок, задающая некоторые вычисления. Операции подразделяются на четыре класса: арифметические, логические, операции отношения, операции сцепления строк (конкатенация) и операции сдвига битов.

1. Имеется шесть арифметических операций: возведение в степень - "`^`", умножение - "`*`", деление - "`/`", остаток от деления - "`mod`", сложение - "`+`", вычитание - "`-`". Например:

```
c=a+b ; сложить a и b, затем занести в c  
d=e-f ; из e вычесть f, затем занести в d  
g#=r/m ; r разделить на m, затем занести в g
```

$kl = l \bmod n$;остаток от деления l на n занести в kl

$g = c * d$;умножить c на d , занести в g

$f = g^3$;возвести в куб g и занести в f

2. Логические операции: Отрицание - **NOT**, конъюнкция (и) - **AND**, дизъюнкция (или) - **OR**, исключающее или - **XOR**.

Таблицы логических выражений:

Отрицание - NOT

Значение a	Выражение	Результат b
a=1	b=NOT a	b=0
a=0	b=NOT a	b=1

Дизъюнкция - OR

Значение a	Значение b	Выражение	Результат c
a=0	b=0	c=a OR b	c=0
a=1	b=0	c=a OR b	c=1
a=0	b=1	c=a OR b	c=1
a=1	b=1	c=a OR b	c=1

Конъюнкция - AND

Значение a	Значение b	Выражение	Результат c
a=0	b=0	c=a OR b	c=0
a=1	b=0	c=a OR b	c=0
a=0	b=1	c=a OR b	c=0
a=1	b=1	c=a OR b	c=1

Исключающее или - XOR

Значение a	Значение b	Выражение	Результат c
a=0	b=0	c=a OR b	c=0
a=1	b=0	c=a OR b	c=1
a=0	b=1	c=a OR b	c=1
a=1	b=1	c=a OR b	c=0

Логические операции можно выполнять не только с 0 и 1, но и с другими числами. Например, **3 Or 5** даст результат **7**. Так как эти значения в компьютере представлены в двоичной форме, то **11 Or 101** есть **111**, а это

7.

3. Операции отношения: равно - "=", не равно - "<>", меньше - "<", больше - ">", меньше или равно - "<=", больше или равно - ">=". Результатом таких операций является 0 (Ложь) или 1 (Истина). К примеру, **Print 1>6** даст результат 0, **Print 8>5** даст результат 1.

4. Чтобы сцепить строки, используют знак "+". Например:

```
a$="Blitz"
```

```
b$="3D"
```

```
c$=a$+b$ ;значение переменной c$ будет Blitz3D
```

5. Операции сдвига битов в целом значении: **Shr** и **Shl**.

Чтобы удалить в числе определённое количество битов слева, используют:
число Shr число_битов

Чтобы добавить в число определённое количество нулевых битов слева, используют:
число Shl число_битов

Например:

```
a=15 Shr 2 ;Т. к. 15 в двоичной 1111, то 15 Shr 2 удалит 2 бита слева. Результат будет 11, а присвоится 3.
```

```
b=35 Shl 3 ;35 в двоичной 100011, 35 Shl 3 добавит три нуля слева. Результат будет 100011000 b присоится 280
```

```
Print a + " "+b
```

Пункт2.6. Метки.

Очень часто требуется выполнять программу в разных местах, поэтому для этого используют метки. В Blitz3D существует два вида меток: простая метка и метка с функцией возврата. Простая метка не позволяет в любой момент возвратиться в то место, где она вызывалась. Выполнение возвращается в исходное положение, когда дойдет до конца программы. Для описания метки служит "." и имя. Например: .label1. Вызов простой метки осуществляется с помощью GOTO имя. Вызов метки с функцией возврата осуществляется с помощью GOSUB имя. Для возврата в место вызова используется оператор RETURN.

Пример программы с простой меткой:

```
Print "Начало"
```

```
Goto .part1
```

```
Print "Конец"
```

```
.part1
Print "Тело метки"
While Not KeyHit(1)
Wend
End
```

Пример программы с меткой с функцией возврата:

```
Print "Начало"
Gosub .part1
Print "Конец"
While Not KeyHit(1)
Wend
End
.part1
Print "Тело метки"
Return
```

Пункт2.7. Операторы ввода с клавиатуры и вывода на экран.

Часто требуется ввод значений с клавиатуры. Для этого используется команда INPUT("текст"). В скобках указывается текст в кавычках, который выводится на экран перед вводом значения.

Пример:

```
a%=Input("Введите целое число ")
b#=Input#("Введите вещественное число ")
c$=Input$("Введите строку ")
```

Для вывода значений или текста на экран используют операторы WRITE и PRINT. Отличие у них в том, что оператор Print после вывода переходит на новую строку, а оператор Write нет.

Пример вывода текста:

```
a=100
Print a + "%" ;на экран выйдет 100%
```

Пример ввода значения и вывода его на экран:

```
a=Input("Введите число ")
Print "число а равно" + a
```

Пункт2.8. Управляющие конструкции.

Управляющие конструкции предоставляют средства для построения сложных программ, способных проверять условия и реагировать на

изменения значений входных данных во время работы, а также повторять некоторые команды программы несколько раз.

Пункт 2.8.1. Проверка условий.

Иногда на истинность требуется проверить некоторое выражение, которое даёт либо истинный, либо ложный результат.

Например:

```
a=1
```

```
b=0
```

```
Print a=b
```

На экран выйдет значение 0 (ложь), так как $a \neq b$. Если b было равно 1, то на экран вышло бы 1 (истина).

Команда If проверяет выражение на истинность или ложность, затем выполняет (не выполняет) блок программного кода. Существуют различные формы If.

Первая форма:

Если истина, то выполняется команда1 программы в одну строку, иначе не выполняется:

If выражение Then команда1

Пример:

```
a=Input("Введите число")
```

```
If a=1 Then Print "Один"
```

Вторая форма:

Если истина, то выполняется команда1 программы в одну строку, иначе код2 в одну строку:

If выражение Then команда1 Else команда2

Пример:

```
a$=Input("Введите символ")
```

```
If a$="z" Then Print "Один" Else Print "Другой символ"
```

Третья форма - позволяет выполнять несколько команд.

Если истина, то выполняется целый блок команд, иначе ничего не выполнится:

**If выражение Then
команда1
команда2**

...
End If

Пример:

```
a=Input("Введите число")
If a<10 Then
Print "Число меньше 10"
WaitKey()
End If
```

Четвёртая форма - позволяет выполнять несколько команд при истинном или ложном значениях:

If выражение Then
команда1
команда2

...
Else
команда1
команда2

...
End If

Пример:

```
a=Input("Введите число")
If a<10 Then
Print "Число меньше 10"
WaitKey()
Else
Print "Число больше 10"
WaitKey()
End If
```

Пятая форма позволяет создать сложное условие с несколькими вариантами. Для этого используется Elseif, где после него указывается новое выражение. Выглядит это так:

If выражение1 Then
команда1
команда2

...
ElseIf выражение2
команда1
команда2

...
ElseIf выражение3

```
команда1
команда2
...
...
Else
команда1
команда2
...
End If
```

Пример:

```
a=Input("Введите число")
If a=1 Then
Print "Один"
Else If a=2
Print "Два"
Else
Print "Больше или равно 3"
End If
```

Еще существует особая форма Select. Это более упрощенная форма, чем использование Else If. Структура данной конструкции выглядит так:

```
Select переменная
Case значение1
команда1
команда2
...
Case значение2
команда1
команда2
...
...
Default
команда1
команда2
...
End Select
```

Default - выполнение команд, если все значения ложны. Пример:

```
a=Input("Введите число")
Select a
Case 1
Print "Один"
Case 2
```

```
Print "Два"
Case 3
Print "Три"
Default
Print "Больше или равно 4"
End Select
```

Пункт 2.8.2. Циклы.

Часто в программе требуется, чтобы команды повторялись несколько раз, поэтому были созданы циклы. Существует четыре вида цикла:

1. Цикл от начального значения до конечного с шагом (For...Next)
2. Цикл с предусловием (While...Wend)
3. Цикл с постусловием (Repeat...Until)
4. Бесконечный цикл (Repeat...Forever)

1) Цикл от начального значения до конечного с шагом (For...Next) задаётся с помощью For, затем указывается имя переменной с присвоенным ей начальным значением, затем To и конечное значение. После ставят Step и число - это шаг цикла, причем, если шаг равен 1, то Step можно опустить. Ниже указываются все команды, которые нужно повторить. Завершает тело цикла оператор Next - он переходит к команде For, а значение переменной увеличивается (уменьшается) с заданным шагом. И так повторяется, пока не достигнуто конечное значение. Конструкция выглядит так:

```
For перем=нач_знач To кон_знач Step шаг  
команда1  
команда2  
...  
Next
```

Пример программы с циклом For (выводится значение переменной i 10 раз):

```
For i=1 To 10
Print i+"Раз"
Next
```

2) Цикл с предусловием (While...Wend) задаётся с помощью команды While. Данный вид цикла работает, пока истинное условие некоторого выражения, записанного после While. Ниже указываются команды, а затем цикл завершается с помощью Wend. Команды внутри цикла должны изменять значение условия, иначе цикл будет бесконечным. Конструкция цикла While:

While выражение
команда1
команда2
...
Wend

Пример программного кода (цикл проработает 5 раз, пока истинное условие):

```
While s<=5  
s=s+1  
Wend
```

3) Цикл с постусловием (Repeat...Until) задаётся с помощью команды Repeat. Ниже указываются команды тела цикла, затем цикл завершается с помощью Until и выражения. Цикл работает пока ложное условие, причём, если условие будет истинное, то цикл сработает хотя бы один раз. Вид цикла:

Repeat
команда1
команда2
...
Until выражение

Пример программного кода:

```
Repeat  
s=s+1  
Until s>=5
```

4) Бесконечный цикл позволяет работать командам внутри себя бесконечно. Задаётся с помощью Repeat, затем указываются команды и завершается с помощью Forever.

Repeat
команда1
команда2
...
Forever

Чтобы выйти из цикла досрочно, используется команда **Exit**. Она работает во всех видах циклов.

Пункт2.9. Функции.

Функцией называется фрагмент программного кода, обладающий

уникальным именем и предназначенный для решения конкретной задачи. Каждая функция имеет своё собственное имя. Функция может быть вызвана многократно из любой области программы. Функция в результате выполнения возвращает некоторое значение. Задаётся функция с помощью Function, затем имя и параметры в скобках через запятую. Эти параметры могут быть любого типа. Завершается функция командой End Function. Внутри тела функции пишутся команды некоторой задачи. Чтобы функция вернула значение, используют Return, затем указывают значение или имя переменной, которая использовалась в функции. Вызывается функция указанием её имени и перечислением используемых значений используемых параметров в скобках. К сожалению, функции нельзя передать массив, как в некоторых языках программирования. Префикс определяет тип функции (% , # или \$. Причём, % можно опустить)

Структура функции такова:

```
Function имя[префикс]([параметр1],[параметр2],...)  
команда1  
команда2  
...  
Return значение  
End Function
```

Вызов функции осуществляется так:

```
имя([знач1],[знач2],...)
```

Рассмотрим применение функции на конкретном примере. Функция нахождения расстояния между двумя точками:

```
x1=Input("Введите значение x1 ")  
y1=Input("Введите значение y1 ")  
x2=Input("Введите значение x2 ")  
y2=Input("Введите значение y2 ")  
Print rast(x1,y1,x2,y2) ;вывод значения функции
```

```
Function rast#(x1,y1,x2,y2) ;функция  
r#=sqr((x2-x1)^2+(y2-y1)^2)  
Return r# ;функция возвращает значение переменной r  
End Function ;конец функции
```

Здесь выводится значение функции rast. В переменную r заносится значение расстояния и возвращается функции с помощью Return r.

Переменные, которые используются внутри функции - локальные, поэтому удаляются после выполнения функции. Их названия могут совпадать, но значения этих переменных будут разные. Также внутри функции можно

использовать глобальные переменные.

Замечание! Внутри функций нельзя объявлять другие функции, так как это приведет к ошибке.

Для функций допускается рекурсивный вызов, т.е. когда функция вызывает сама себя.

Пункт2.10. Процедуры.

Процедура в Blitz3D тоже самое, что функция. Процедуры обычно используют для решения определенных задач, в которых не обязательно возвращать значение. Также как и у функций можно использовать рекурсивный вызов.

Объявляется она также как функция:

```
Function имя([параметр1],[параметр2],...)  
команда1  
команда2  
...  
End Function
```

Вызов функции осуществляется так:

```
имя([знач1],[знач2],...)
```

И простой пример - вывести системное время с помощью процедуры:

```
proc("Сегодня") ;вызов процедуры  
Function proc(word$) ;объявление процедуры  
Print word$+" "+CurrentDate () + " и "+CurrentTime()+" времени"  
End Function
```

Пункт2.11. Строка констант.

Строка констант служит для хранения большого числа данных разного типа, которые будут считываться и присваиваться переменным. Перед перечислением значений задают имя метке для этой строки, ниже с помощью команды Data перечисляют значения через запятую.

Команда выглядит так:

```
.имя_метки_строки  
Data значение1, значение2, ...
```

Чтобы обратиться к метке строки перед считыванием данных, служит

команда Restore:

Restore имя_метки_строки

Считывание данных идёт последовательно, т. е. когда 1 элемент считан, указатель строки данных переходит к следующему элементу при новом считывании. Итак, после того, как получили доступ к строке данных, для считывания используют Read и имя переменной, которой присвоится значение элемента строки данных:

Read имя_переменной

Пример:

```
. constants
Data 1,2,"String"
Restore constants
Read z
Print z
Read z1
Print z1
Read z2$
Print z2$
```

Пункт2.12. Массивы.

Массивом называется совокупность элементов одного типа. Массив имеет своё собственное имя и может быть одномерным, двумерным и т.д. Каждый элемент имеет свой определённый индекс или несколько индексов в зависимости от измерения массива. Индексация начинается с 0. Объявляется он с помощью Dim, затем указывается имя массива со знаком его типа (\$-строковый, %-целый, #-дробный), а в скобках количество элементов для каждого измерения. Внимание, элементы массива могут быть только одного типа.

Объявляется массив так:

Dim имя(размер1,[размер2],...)

Присвоение значения осуществляется так:

имя(индекс1,[индекс2],...)=значение

Рассмотрим применение массивов на конкретном примере:

```
Dim arr%(10) ;создание одномерного массива из десяти элементов целого типа
For i=1 to 10 ;цикл от 1 до 10
arr(i)=Rand(-10,10) ;присвоение случайного целого числа в диапазоне от -10 до 10 i
элементу
```

```
Print i+" Элемент равен "+arr(i) ;вывод его значения
Next ;конец тела цикла
```

Массивы можно объявить в одну строку через запятую:

```
Dim a(10), b$(100,2), t$(20)
```

В последних версиях Blitz3D появился другой способ объявления массивов. Теперь их можно объявить с помощью Local или Global (как переменные), но вместо круглых скобок () используются квадратные []. Присвоение по индексу тоже в квадратных скобках. Их также можно объявлять в процедурах. Массивы такого вида могут быть только одномерные! Т.е. так:

Local имя[размер]

или

Global имя[размер]

```
Local a[10]
For i=1 To 10 a[i]=i
Print a[i]
Next
```

Пункт2.13. Банки.

Банк - выделенная область памяти, где хранятся значения разного типа с определённым адресом. Размер банка может быть изменён и он может быть удалён в любой момент. Создаётся банк с помощью CreateBank(число), где в скобках указывается его размер в байтах.

Объявление банка:

имя>CreateBank(размер)

Изменяется размер банка так:

ResizeBank имя,новое_знач

Удаляется банк следующим образом:

Freebank имя

Теперь о записи и чтении значений из банка. В банк можно заносить значения типа Byte(0..255) - 1 байт, Short(0..65535) - 2 байта, Integer(-2³¹..2³¹-1) - 4 байта, Float(с плавающей точкой) - 4 байта.

Для заноса значения типа Byte пользуются командой PokeByte, затем указывают имя банка и номер байта, с которого заносится значение, далее

само значение. Выглядит это так:

PokeByte имя_банка,номер_байта,значение

Аналогично для типа Short:

PokeShort имя_банка,номер_байта,значение

Для типа Integer:

PokeInt имя_банка,номер_байта,значение

Для типа Float:

PokeFloat имя_банка,номер_байта,значение

Теперь о чтении данных из банка.

Для чтения значения типа Byte из банка служит функция:

PeekInt(имя_банка,номер_байта)

Чтение значения типа Short:

PeekShort(имя_банка,номер_байта)

Чтение значения типа Integer:

PeekInt(имя_банка,номер_байта)

Чтение значения типа Float:

PeekFloat(имя_банка,номер_байта)

Внимание! Нумерация номера байтов в банке начинается с 0!

Пример записи и чтения из банка(выделяется 11 байт в памяти для 4 значений всех типов: 1 байт + 2 байта + 4 байта + 4 байта):

```
bank=CreateBank(11)
PokeByte bank,0,255
PokeShort bank,1,65535
PokeInt bank,3,-2^31
PokeFloat bank,7,9.99999
Print PeekByte (bank,0)
Print PeekShort (bank,1)
Print PeekInt (bank,3)
Print PeekFloat (bank,7)
FreeBank bank
```

Другие команды:

Размер банка. Функция возвращает размер банка в байтах:

BankSize(имя)

Копирование содержимого банка в другой:

CopyBank

имя_банка1,номер_байта1,имя_банка2,номер_байта2,колво_байтов

Где имя_банка1 - банк, с которого копируют; номер_байта1 - с какого байта считывают; имя_банка2 - банк, в который копируют; номер_байта2 - с какого байта записывают; колво_байтов - сколько байтов копируют.

Пример(копируется 6 байтов из банка 1 в банк 2 с позиции 1 для типов Short и Int):

```
bank1=CreateBank(11)
bank2=CreateBank(6)
PokeByte bank1,0,255
PokeShort bank1,1,65535
PokeInt bank1,3,-2^31
PokeFloat bank1,7,9.99999
CopyBank bank1,1,bank2,0,6
FreeBank bank1
Print PeekShort(bank2,0)
Print PeekInt(bank2,2)
FreeBank bank2
```

Пункт2.14. Типы.

Типы - это совокупность переменных различных типов объединенных в одну группу. Типы еще называют структуры или записи. Кто работал в C или Pascal знают, что это такое. Переменные типов могут хранить числовые и строковые значения. Чтобы начать работать с типом, нужно сперва объявить его. Это делается с помощью команды Type, затем указывают имя типа. Ниже после команды Field указывают имена переменных, которые будут использоваться в типе. Внимание! Типы нужно объявлять только в главной программе, нельзя объявлять их в процедурах или функциях.

Объявление типа выглядит так:

Type имя_типа

Field имя_перем1,имя_перем2,...

End Type

Теперь, когда тип объявлен, можно работать с его переменными. Когда тип объявлен, в нём еще ничего нет. Для работы с элементами типа используется указатель. Чтобы создать новую запись в типе, используют команду New. Сначала указывают имя указателя, потом с точкой имя типа, затем знак присвоения, команду New и снова имя типа.

Вот так это выглядит:

указатель.имя_типа = New имя_типа

Чтобы присвоить значение переменной типа, пишут имя указателя, затем знак "\" и имя переменной, используемой в типе:

указатель\имя_перем = значение

Итак, мы создали первую запись и присвоили значения переменным типа первой записи. Чтобы создать следующую запись, заново используют команду New. После этого указатель типа перейдет на следующую запись и можно присваивать значения новой записи. Чтобы отслеживать элементы каждой записи типа используется цикл с командой Each. Этот цикл проработает столько раз, сколько записей в типе.

Такой цикл выглядит так:

For указатель.имя_типа = Each имя_типа

команда1

команда2

...

Next

Чтобы удалить текущую запись в типе, используют Delete:

Delete указатель

Чтобы удалить все записи в типе, используют Delete Each:

Delete Each имя_типа

А вот пример программного кода с использованием типов(структура имеет имя struct1, в ней используются две переменные: var1, var2; метка - h) :

Type struct1

Field var1,var2

End Type

For i=1 to 10

h.struct1 = New struct1

h\var1=i

h\var2=Rand(0,100)

Next

For h.struct1= Each struct1 ;цикл пройдет 10 раз

print h\var1 + " " + h\var2 ;выйдут значения 2 переменных каждой записи

Next

Delete Each struct1

Существуют четыре команды, которые позволяют управлять указателем в

типе: First, Last, After, Before.

Чтобы поместить указатель в начало типа, используют First:

указатель.имя_типа = First имя_типа

Чтобы поместить указатель в конец типа, используют Last:

указатель.имя_типа = Last имя_типа

Чтобы передвинуть указатель вперед, используют After:

указатель = After указатель

Чтобы поместить указатель назад, используют Before:

указатель = Before указатель

Пример программы:

```
Type all
Field a
End Type
For z=1 To 4
    ukaz.all= New all
    ukaz\a = z
Next
ukaz.all = First all
Print ukaz\a
ukaz = After ukaz
Print ukaz\a
ukaz.all = Last all
Print ukaz\a
ukaz = Before ukaz
Print ukaz\a
```

Чтобы переместить текущую запись в другое место, используют Insert, дополнение к предыдущему программному коду:

```
Insert ukaz After First all
```

В полях структур также могут присутствовать массивы. Описывается он как массив с квадратными скобками:

Type имя_типа

Field имя_массива[размер]

End Type

К примеру:

```
Type TMytype
```

```
Field arr[10],num  
End Type
```

Тогда создание записи и обращение к ней будет выглядеть так:

```
Hndl.TMytype=New TMytype  
Hndl\arr[0]=35 ;присвоение значения массиву в типе  
Hndl\arr[3]=11
```

Кроме переменных и массивов, в типе может находиться ссылка на другой тип. Это необходимо для создания связанных типов:

```
Type имя_типа1  
Field перемен1,...  
End Type
```

```
Type имя_типа2  
Field указатель1.имя_типа1,перемен2,...  
End Type
```

Создание записи и присвоение значений элементам первого и второго типа выглядит так:

```
указатель2.имя_типа1=New имя_типа1 ;создание записи во втором  
типе  
указатель2\перемен2=значение ;присвоение значения переменной во  
втором типе  
указатель2\указатель1=New имя_типа2 создание записи в первом  
типе через второй  
указатель2\указатель1\перемен1=значение присвоение значения  
переменной в первом типе
```

Чтобы было лучше понять этот механизм, рассмотрим следующий пример. Программа демонстрирует связанные типы:

```
Type tsecond ;второй тип  
Field z# ;его переменная z#  
End Type
```

```
Type tfirst ;первый тип  
Field x%,y%,ab.tsecond ;его переменные x%, y% и ab.tsecond - ссылка на второй тип  
End Type
```

```
For i1=1 To 3  
q.tfirst=New tfirst ;создание записи в первом типе  
q\x%=Rand(1,9) ;присвоение значений переменным x и y  
q\y%=Rand(1,9)
```

```

q\ab=New tsecond ;создание записи во втором типе через первый
q\ab\z#=Rnd(10,99) ;присвоение значения переменной z#
Next

For q.tffirst=Each tfirst
Write q\x%+" "+q\y+" "+" " ;вывод значений x и y
Write q\ab\z+" " ;вывод z#
Print " "
Next

```

Также можно создавать массив типов:

```

Type имя_типа
Field перемен1,...
End Type
Dim указатель.имя_типа(размер1,[размер2],...)

```

А также вторым способом (для одномерных массивов через Local или Global):

```

Type имя_типа
Field перемен1,...
End Type
Global указатель.имя_типа[размер1]

```

или

```

Type имя_типа
Field перемен1,...
End Type
Local указатель.имя_типа[размер1]

```

Пример для массива структур:

```

Type TArr ;структура
Field name$,el1
End Type

```

Local w.TArr[4];массив w из 4х структур

```

For i=0 To 4
For i1=0 To 3 ;каждый элемент массива содержит 4 записи
w[i]=New TArr
w[i]\name$="TArr["+i+"] "+"Record "+i1

```

```

w[i]\ell=Rand(0,9)
Next
Next

For z.TArr=Each TArr ;вывод всей структуры с другим указателем
Print z\name$+" "+z\ell
Next

```

Имеются особые команды, позволяющие вернуть адрес текущей записи или установить указатель по заданному адресу в типе. Это команды **Handle()** и **Object()**. Handle() возвращает значение от 1 до максимального значения (зависит от количества записей в типе).

Чтобы получить адрес текущей записи, служит команда:
Handle(указатель.имя_типа)

Чтобы установить указатель по нужному адресу, нужна команда:
указатель2.имя_типа=Object.имя_типа(адрес)

Программа выводит адрес и значения текущей записи:

```

Type Qtype
Field num%,z#
End Type

For i=0 To 10
e.Qtype=New Qtype
e\num=Rand(0,9)
e\z#=Rnd(10,99)
Next

For e.Qtype=Each Qtype
Print Handle(e.Qtype)+" "+e\num+" "+e\z#
Next

```

Пункт2.15. Операции со строками.

В этом пункте мы будем производить различные операции со строками. В Blitz имеется несколько возможностей для работы со строками:

Вернуть символы длины n слева:
Left(имя_строки,n)

Вернуть символы длины n справа:

Right(имя_строки,n)

Вернуть символы с m символа длины n:

Mid(имя_строки,m,n)

Заменить один символ на другой:

Replace(имя_строки,"символ1","символ2")

Показать номер символа с n номера:

Replace(имя_строки,"символ1",n)

Показать длину строки:

Len(имя_строки)

Удалить пробелы перед и после предложением:

Trim(имя_строки)

Отступ слева с n длиной строки:

LSet(имя_строки,n)

Отступ справа с n длиной строки:

RSet(имя_строки,n)

Значение строки в шестнадцатеричном виде:

Hex(имя_строки)

Значение строки в бинарном виде:

Bin(имя_строки)

Пример:

a\$="Пример простой программы" ;строковая переменная

b\$="255543.457547" ;строковая переменная

Print a\$;вывод на экран значения

Print Left(a\$,6) ;вывод 6ти символов слева

Print Right(a\$,9) ;вывод 9ти символов справа

Print Mid\$(a\$,8,7) ;вывод 8ми символов начиная с 7го

Print Replace(a\$,"п","р") ;замена символа "п" на "р"

Print "Длина строки - " + Len(a\$) + " символов" ;вывод длины строки

Print b\$;вывод на экран значения

Print Hex\$ (b\$) ;вывод значения строки в шестнадцатеричном виде

Print Bin\$ (b\$) ;вывод значения строки в двоичном виде

Пункт2.16. Файлы.

В этой главе вы узнаете, как записывать и считывать данные из файлов. В Blitz3D существует три режима работы с файлами: режим записи, режим чтения и режим записи-чтения. Режим записи позволяет открыть и занести значения в файл, а режим чтения считать значения из файла. При работе с файлом присваивается переменная-идентификатор, которая характеризует его. Чтение и запись происходит последовательно, путём перемещения указателя (аналогично в банках). В режиме записи файл создаётся автоматически, если он не существует на диске. Если перед записью в файле присутствовали значения, то они перезаписываются на новые. Внимание! Если прекращается работа с файлом в одном из режимов, то надо его закрыть, а затем можно его открыть в новом режиме.

Файл для записи открывают с помощью команды WriteFile и в скобках указывают путь в кавычках:

идентификатор_файла=WriteFile("путь")

Файл для чтения открывают с помощью команды ReadFile:

идентификатор_файла=ReadFile("путь")

Файл для записи/чтения открывают с помощью команды OpenFile:

идентификатор_файла=OpenFile("путь")

Закрыть файл:

CloseFile идентификатор_файла

Запись значений разных типов в файл (только в режиме для записи и записи/чтения).

Занести значение типа Byte:

WriteByte(идентификатор_файла, значение)

Занести значение типа Short:

WriteShort(идентификатор_файла, значение)

Занести значение типа Integer:

WriteInt(идентификатор_файла, значение)

Занести значение типа Float:

WriteFloat(идентификатор_файла, значение)

Занести значение типа String:

WriteString(идентификатор_файла, "значение")

Занести строку текста (запись текста строками, т.е. после каждой записи переход вниз) :

WriteLine(идентификатор_файла, "строка")

Занести некоторое число байт из банка в файл с некоторой позицией:

WriteBytes имя_банка,идентификатор_файла,позиция,число_байт

Чтение из файла значений разных типов (только в режиме для записи и записи/чтения).

Считать значение типа Byte:

имя_перем=ReadByte(идентификатор_файла)

Считать значение типа Short:

имя_перем=ReadShort(идентификатор_файла)

Считать значение типа Integer:

имя_перем=ReadInt(идентификатор_файла)

Считать значение типа Float:

имя_перем=ReadFloat(идентификатор_файла)

Считать значение типа String:

имя_перем=ReadString(идентификатор_файла)

Считать строку текста из файла:

имя_перем=ReadLine(идентификатор_файла)

Считать некоторое число байт из файла в банк с некоторой позицией начального байта:

ReadBytes имя_банка,имя_файла,позиция,число_байт

Работа с указателем файла. Значение начальной позиции указателя файла равно 0.

Проверка конца файла (функция возвращает 1, если конец файла и 0, если не конец файла):

Eof(идентификатор_файла)

Текущая позиция указателя файла (функция возвращает значение номера байта в данной позиции):

FilePos(идентификатор_файла)

Перенос указателя файла в нужную позицию:

SeekFile(идентификатор_файла,значение)

Пример программы на запись/чтение данных:

f=WriteFile("1.txt") ;открытие файла для записи

WriteByte(f,255) ;запись байтового значения
WriteInt(f,6500) ;запись целого значения
WriteFloat(f,0.1) ;запись вещественного значения
WriteString(f,"file") ;запись строкового значения
CloseFile(f) ;заккрытие файла
f=OpenFile("1.txt") ;открытие файла для чтения
Print ReadByte(f) ;печать байтового значения
Print ReadInt(f) ;печать целого значения
Print ReadFloat(f) ;печать вещественного значения
Print ReadString(f) ;печать строкового значения
CloseFile(f) ;заккрытие файла

Работа с файлами и папками.

Размер файла (функция возвращает размер файла в байтах):

FileSize(идентификатор_файла)

Проверка на существование файла (функция возвращает 0, если файл не существует; 1, если он существует и 2, если это папка):

FileType(идентификатор_файла)

Копирование файла:

CopyFile "путь1","путь2"

Удаление файла:

DeleteFile("путь")

Открытие папки для чтения:

идентификатор_папки=ReadDir("путь")

Заккрытие папки для чтения:

CloseDir(идентификатор_папки)

Считать имя файла из текущей открытой папки (считывание имен файлов идет по алфавиту):

NextFile\$(идентификатор_папки)

Текущий рабочий путь папки по умолчанию:

CurrentDir\$()

Изменить рабочий путь папки по умолчанию:

ChangeDir\$()

Удаление папки:

DeleteDir("путь")

Глава3. Расширяем возможности.

Итак, в предыдущей главе вы узнали основы языка, без знания которых невозможно дальнейшее обучение Blitz3D. В данной главе рассказывается о других функциях, которые расширяют возможности. Их знание тоже необходимо.

Пункт3.1. Подключение к файлам проекта.

Blitz3D позволяет разбить проект программы на несколько файлов, что облегчает программисту работу. Благодаря этому он не сможет заблудиться в своём огромном программном коде. При разбиении программы на несколько файлов учитывается то что, они будут подключаться к главному файлу, а вызов осуществляется как раз в главном файле.

Подключают части с помощью команды Include:

Include "имя файла"

Вызов осуществляется в любой области программы. Можно все объявленные функции записать в отдельный файл, чтобы их можно было использовать и в других проектах.

Пункт3.2. Преобразование типов.

С помощью специальных функции можно преобразовать значение одного типа в другой. Например: целый в вещественный, строковый в целый и др.

Существуют следующие функции преобразование типов:

Преобразование значений типа Float и String в тип Integer (строковый тип преобразуется тогда, когда в нем будет только число, иначе же функция возвратит 0):

Int(значение)

Преобразование значений типа Integer и String в тип Float:

Float(значение)

Преобразование числового значения в строковый:

Str(значение)

Пример программного кода:

```
a1$="4"
b1%=6
c1#=2.5
a2%=Int(a1$)
b2#=Float(b1%)
c2$=Str(c1#)
Print a1$ + " " + a2%
Print b1% + " " + b2#
Print c1# + " " + c2$
```

Пункт 3.3. Математические функции.

В этом пункте узнаете, как находить значение синуса, косинуса, логарифма и других математических функций.

Квадратный корень числа:

Sqr(значение)

Нахождение значений тригонометрических функций по углам в градусах.

Значение синуса:

Sin(угол)

Значение косинуса:

Cosin(угол)

Значение тангенса:

Tan(угол)

Значение арксинуса:

Asin(угол)

Значение арккосинуса:

Acos(угол)

Значение арктангенса:

ATan(угол)

Абсолютная величина числа(модуль):

Abs(значение)

Экспонента (e^x):

Exp(значение)

Натуральный логарифм ($\ln(x)$):
Log(значение)

Десятичный логарифм ($\lg(x)$):
Log10(значение)

Sign(x):
Sgn(значение)

Округление числа к нижней границе:
Floor#(значение)

Округление числа к верхней границе:
Ceil#(значение)

Генерирование случайного целого числа в промежутке от начального до конечного значения:
Rand(нач_знач,кон_знач)

Генерирование случайного дробного числа в промежутке от начального до конечного значения:
Rnd(нач_знач#,кон_знач#)

Установка генератора случайного числа, перед генерированием чисел нужно установить его, чтобы при каждом запуске генерировались различные числа:
SeedRnd(значение)

Пример по генерированию случайных чисел:

```
SeedRnd(Millisecs());генерирует случайные числа по системному времени  
Print Rnd(0,100)
```

Пункт3.4. Работа с устройствами ввода.

В пункте приведены функции, которые работают с устройствами ввода: клавиатурой, мышью, джойстиком.

Работа с клавиатурой.

Проверка нажатия клавиши. Возвращает 1, если нажата определённая с кодом клавиша или 0, если не нажата:

KeyHit(код_клавиши)

Проверка нажатия и удерживания клавиши с кодом:

KeyDown(код_клавиши)

Таблица кодов клавиш клавиатуры:

Клавиша	Код	Пояснение
1	2	
2	3	
3	4	
4	5	
5	6	
6	7	
7	8	
8	9	
9	10	
0	11	
Минус (-)	12	На основной панели
Равно (=)	13	
Пробел	14	
Tab	15	
Q	16	
W	17	
E	18	
R	19	
T	20	
Y	21	
U	22	
I	23	
O	24	
P	25	
[26	
]	27	
Enter	28	На основной панели
Левый Ctrl	29	
A	30	
S	31	

D	32	
F	33	
G	34	
H	35	
J	36	
K	37	
L	38	
Точка с запятой (;)	39	
Аппостроф (')	40	
Grave	41	
Левый Shift	42	
Обратный слеш (\)	43	
Z	44	
X	45	
C	46	
V	47	
B	48	
N	49	
M	50	
Запятая (,)	51	
Точка (.)	52	На основной панели
Слеш (/)	53	На основной панели
Правый Shift	54	
Умножить (*)	55	На числовой панели
Левый Alt	56	
Пробел	57	
Capital	58	
F1	59	
F2	60	
F3	61	
F4	62	
F5	63	

F6	64	
F7	65	
F8	66	
F9	67	
F10	68	
NumLock	69	
Scroll Lock	70	
NumPad 7	71	
NumPad 8	72	
NumPad 9	73	
Минус (-)	74	На числовой панели
NumPad 4	75	
NumPad 5	76	
NumPad 6	77	
Плюс (+)	78	На числовой панели
NumPad 1	79	
NumPad 2	80	
NumPad 3	81	
NumPad 0	82	
Точка (.)	83	На числовой панели
F11	87	
F12	88	
Enter	156	На числовой панели
Правый Ctrl	157	
Делить (/)	181	На числовой панели
Print screen	183	
Правый Alt	184	
Pause Break	197	Панель со стрелками
Home	199	Панель со стрелками
Стрелка вверх	200	Панель со стрелками
Page Up	201	Панель со стрелками
Стрелка влево	203	Панель со стрелками

Стрелка вправо	205	Панель со стрелками
End	207	Панель со стрелками
Стрелка вниз	208	Панель со стрелками
Next	209	Панель со стрелками
Insert	210	Панель со стрелками
Delete	211	Панель со стрелками
Левая Windows	219	Основная панель
Правая Windows	220	Основная панель
Power	222	
Sleep	223	
Wake	227	

Проверка нажатия любой клавиши и возвращает ASCII код клавиши:

GetKey()

Остановка программы до тех пор пока не нажата любая клавиша и возвращает код клавиши:

WaitKey()

Присвоить всем функциям проверки нажатия 0, т.е. отменить нажатие клавиш:

FlushKeys()

Работа с мышью и курсором.

Проверка щелчка кнопки мыши с кодом кнопки. При щелчке функция возвратит 1, иначе 0:

MouseHit(код_кнопки)

Проверка нажатия и удерживания кнопки мыши с кодом:

MouseDown(код_кнопки)

Проверка нажатия любой кнопки мыши:

GetMouse()

Остановка программы до тех пор пока не нажата любая кнопка мыши и возвращает код кнопки:

WaitMouse()

Присвоить всем функциям проверки нажатия 0, т.е. отменить нажатие кнопки мыши:

FlushMouse()

Вернуть координаты курсора мыши по оси x (в пикселях):

MouseX()

Вернуть координаты курсора мыши по оси y (в пикселях):

MouseY()

Вернуть число поворотов колёсика мыши(если вверх - число положительное, иначе отрицательное):

MouseZ()

Вернуть скорость изменения курсора мыши по оси x (в пикселях):

MouseXSpeed()

Вернуть скорость изменения курсора мыши по оси y (в пикселях):

MouseYSpeed()

Вернуть скорость изменения поворотов колёсика мыши(если вверх - число положительное, иначе отрицательное):

MouseZSpeed()

Работа с джойстиком.

Проверка типа устройства. Функция возвращает 0, если джойстик не подключен; 1, если джойстик цифровой и 2, если джойстик аналоговый:

JoyType([порт_джойстика])

Проверка нажатия кнопки джойстика:

JoyHit(код_кнопки,[порт_джойстика])

Проверка нажатия и удерживания кнопки джойстика:

JoyDown(код_кнопки,[порт_джойстика])

Проверка нажатия любой кнопки джойстика:

GetJoy([порт_джойстика])

Останавливает программу пока не нажата любая кнопка джойстика и возвращает код кнопки:

WaitJoy([порт_джойстика])

Вернуть координаты рукоятки джойстика по оси x (от -1 до 1):

JoyX#()

Вернуть координаты рукоятки джойстика по оси y (от -1 до 1):

JoyY#()

Вернуть координаты рукоятки джойстика по оси z (от -1 до 1):

JoyZ#()

Вернуть направление рукоятки джойстика по оси x (-1 - слева, 1 - справа):

JoyXDir()

Вернуть направление рукоятки джойстика по оси y (-1 - снизу, 1 - сверху):

JoyYDir()

Вернуть направление рукоятки джойстика по оси z (-1 - вдавлен, 1 - выпущен):

JoyZDir()

Отменить нажатие кнопок джойстика:

FlushJoy()

Пункт 3.5. Системные функции.

В пункте приведён список системных функций.

Возвращает путь системных папок, если значение в кавычках : "systemdir" - путь к папке System32, "windowsdir" - путь папки Windows, "tempdir" - путь к папке Temp, "appdir" - путь, откуда запущено текущее приложение:

SystemProperty (значение)

Показывает сообщение ошибки и останавливает программу (в скобках - текст сообщения):

RuntimeError("текст")

Присваивает имя окну приложения (есть необязательный параметр (0 или 1) - показывать сообщение перед закрытием):

AppTitle("текст",[значение])

Приостановка приложения:

Stop

Завершение приложения:

End

Вывести значение в модуле отладки:

DebugLog("текст")

Пункт3.6. Таймер.

Иногда в программе требуется использовать таймер, который, например, отсчитывал бы время с интервалом в одну секунду. В этом пункте приведён пример таймера, а также других функций, которые работают с системным временем.

Функции времени:

Текущее время:

CurrentTime\$()

Текущая дата:

CurrentDate\$()

Текущее системное время в миллисекундах:

Millisecs()

Функция задержки программы на некоторое время (значение в миллисекундах):

Delay(значение)

А теперь рассмотрим пример таймера с использованием команды **Millisecs()**:

```
Repeat ;цикл Repeat...Until
```

```
If MilliSecs() > timer + 1000 ;если системное время больше переменной time + 1000(1 секунда), то
```

```
x=x+1 ;увеличиваем счётчик
```

```
timer=MilliSecs() ;заново заносим системное время
```

```
EndIf ;конец условия
```

```
print x ;вывод x
```

```
Until KeyHit(1) ;до тех пор пока не нажата клавиша Esc
```

```
End ;Выход
```

Глава4. Работа со звуковыми файлами и видео.

Данная глава посвящена работе со звуками и видео. В ней описывается, как загружать, воспроизводить и удалять звуки.

Пункт4.1. Работа со звуками.

В Blitz3D можно использовать звуковые файлы в двух режимах: использовать как фоновый звук и загружать его в оперативную память для многократного использования. Загружаемые или воспроизводимые звуки присваиваются переменные-идентификаторы, в которые заносится адрес звуков.

Использование звука в фоновом режиме. Воспроизведение ведётся напрямую с жесткого диска. Также звук можно остановить, сделать паузу/возобновить. Воспроизведение происходит с помощью такой команды (в кавычках путь к файлу) :

перем_звук=PlayMusic("файл")

Воспроизвести CD трек (первый параметр - номер трека, второй необязательный параметр - режимы воспроизведения: 1 - воспроизвести 1 раз, 2 - воспроизвести за циклено, 3 - воспроизвести до конца диска) :

перем_звук=PlayCDTrack(трек,[значение])

Режим загрузки звука в оперативную память:

перем_звук=LoadSound("путь")

Воспроизвести звук:

PlaySound перем_звук

Зациклить воспроизведение звука:

LoopSound перем_звук

Изменить частоту дискретизации звука (значение в герцах):

SoundPitch(перем_звук,значение)

Громкость звука (значение дробное от 0 до 1):

SoundVolume(перем_звук,значение)

Баланс звука (значение дробное от -1 до 1):

SoundPan(перем_звук,значение)

Удалить звук из оперативной памяти:

FreeSound идентификатор_звук

Команды относящиеся к обоим режимам. Работа с каналом звука.

Пауза звука:

PauseChannel перем_звук

Возобновление звука:

ResumeChannel перем_звук

Остановить звук:

StopChannel (перем_звука)

Изменить частоту дискретизации канала звука (значение в герцах):

ChannelPitch(перем_звука, значение)

Громкость канала звука (значение дробное от 0 до 1):

ChannelVolume(перем_звука, значение)

Баланс канала звука (значение дробное от -1 до 1):

ChannelPan(перем_звука, значение)

Функция проверки воспроизведения звука (возвращает 1, если воспроизводится звук или 0, если нет) :

ChannelPlaying(перем_звука)

Пример (воспроизводится фоновый звук и управляется его воспроизведение с помощью клавиш влево и вправо. Воспроизводится загружаемый звук с помощью клавиши вверх):

```
background=PlayMusic("1.mp3")
sound=LoadSound("2.mp3")
While ChannelPlaying(background)
If KeyHit(200) Then PlaySound sound
If KeyHit(203) Then PauseChannel background
If KeyHit(205) Then ResumeChannel background
Wend
FreeSound sound
End
```

Глава5. 2D Графика.

Вы переходите к разделу работы с графикой. Данная глава посвящена двумерной графике.

Пункт5.1. Графический режим.

Blitz3D может работать в двух графических режимах: двумерном и трехмерном. Так как данная глава посвящена двумерной графике, то поговорим о ней, а о трехмерной в следующей главе. Каждый графический элемент (точка, линия, окружности и изображение) имеет свои координаты на плоскости. Прямоугольная область, которая отображается на экране, называется графическим полем (viewport). Единицей координат является

пиксель. Начало координат на экране расположено в верхнем левом углу экрана, а размер поля равен разрешению экрана.

Пункт 5.1.1. Подключение 2D графического режима.

Перед тем, как начать работать с графикой, нужно подключить графический режим. В пункте приведены команды, которые работают в графическом режиме. При подключении графического режима указывают размер окна, глубину цвета, которые поддерживает видеокарта. Стандартными разрешениями являются: 640x480, 800x600, 1024x768.

Подключение графического режима. Первое число - разрешение по горизонтали, второе - по вертикали, третье - глубина цвета в битах, а четвертое: 1 - полноэкранный, 2 - оконный, 3 - масштабируемое окно. Значение 0 для всех - значение по умолчанию. Команда выглядит так:
Graphics ширина,высота,[глубина],[режим]

Пример команды:

Graphics 800,600,32,1

Отключение графического режима:

EndGraphics

Установить позицию и размеры графического поля:

Viewport x,y,ширина,высота

Вернуть текущий размер экрана по ширине:

GraphicsWidth()

Вернуть текущий размер экрана по вертикали:

GraphicsHeight()

Вернуть текущую глубину цвета:

GraphicsDepth()

Функции, получающие сведения от драйвера видеокарты.

Размер видеопамяти:

TotalMem()

Размер доступной видеопамяти:

AvailMem()

Вернуть число всех доступных видеорежимов:

CountGfxModes()

Вернуть размер по ширине по порядковому номеру режима (значение от 1):
GfxModeWidth(значение)

Вернуть размер по высоте по порядковому номеру режима (значение от 1):
GfxModeHeight(значение)

Вернуть глубину цвета по порядковому номеру режима (значение от 1):
GfxModeDepth(значение)

Проверяет, существует ли режим с такими значениями. Первое значение - ширина, второе - высота, третье - глубина цвета:
GfxModeExists(ширина,высота,глубина)

Пример. Вывести список всех видеорежимов:

```
For i=1 To CountGfxModes()  
Print GfxModeWidth(i)+" "+GfxModeHeight(i)+" "+GfxModeDepth(i)  
Next
```

Вернуть число видеодрайверов:
CountGfxDrivers()

Показать имя видеодрайвера по номеру:
GfxDriverName\$(значение)

Использовать видеодрайвер по его номеру:
SetGfxDriver(значение)

Пример. Вывести список всех драйверов:

```
For k=1 To CountGfxDrivers()  
Print GfxDriverName(k)  
Next
```

Пункт 5.1.2. Графические буферы.

При выводе на экран, кадр формируется в графическом буфере. Существует три графических буфера: фронтальный, задний, буфер изображения. О буфере изображения речь будет в пункте 5.4. Фронтальный буфер - буфер, в котором формируется кадр выводимый на экран. Задний буфер - это буфер, в котором записывается предыдущий сформированный кадр. При выводе во фронтальном буфере видны мерцания, так как кадры формируются напрямую. Чтобы избежать это, используют двойную буферизацию. При двойной буферизации кадр, сформированный во

фронтальном буфере, перемещается в задний, и, пока формируется другой во фронтальном буфере, на экран выводится кадр с заднего буфера. По умолчанию стоит фронтальный буфер. Чтобы установить буфер, служит команда Set Buffer в комбинации с которой подключаются фронтальный или задний буфер.

Чтобы установить фронтальный буфер, служит команда:

SetBuffer FrontBuffer()

Установить задний буфер позволяет команда:

SetBuffer BackBuffer()

Чтобы на экран прорисовывалось изображение из заднего буфера, используют команду:

Flip

Теперь поговорим о других параметрах буферов. Чтобы сохранить изображение из буфера в файл (сделать скриншот в формате bmp), служит SaveBuffer. Значение буфер может быть BackBuffer() или FrontBuffer():

SaveBuffer(буфер, "файл")

Загрузка изображения из файла в буфер:

LoadBuffer(буфер, "файл")

Пункт 5.2. Рисование фигур.

Итак, переходим к основной части графики. В главе показано, как рисовать фигуры, а также в следующей приведён пример с использованием двойной буферизации.

Рисование фигур.

Точка с координатами на экране:

Plot x,y

Отрезок с координатами от начальной до конечной точки:

Line x1,y1,x2,y2

Прямоугольник с координатами от начальной точки, затем ширина и высота, а последнее значение - делать закрашенным или нет (0 или 1):

Rect x,у,ширина,высота,[залитый]

Эллипс с координатами от начальной точки, затем ширина и высота, а

последнее значение - делать закрашенным или нет (0 или 1):

Oval x,y,ширина,высота,[залитый]

Работа с экраном.

Очистка экрана:

Cls

Сделать задержку до тех пор, пока не прорисуются кадр:

VWait

Пункт5.3. Цвет, экран.

Работа с цветом. В Blitz3D с цветами работают по структуре RGB. Т.е. имеется три цвета: красный, синий и зелёный, которые при смешивании дают новый цвет. Каждому цветокомпоненту соответствует значение от 0 до 255. При рисовании по-умолчанию стоит белый цвет (255,255,255).

Чтобы применить новый цвет рисования, служит команда Color, где указываются значения компонента от 0 до 255:

Color красный, зеленый, синий

Функции, которые возвращают текущее значение компонента цвета.

Текущий красный:

ColorRed()

Текущий зелёный:

ColorGreen()

Текущий синий:

ColorBlue()

Получить значения цвета пикселя по его координате:

GetColor x,y

А вот пример использования двойной буферизации. Рисует линии разного цвета на экране. Сформированное изображение выведется на экран при помощи команды Flip:

Graphics 800,600,16

SetBuffer BackBuffer()

SeedRnd Millisecs()

Repeat

Color Rnd(255),Rnd(255),Rnd(255)


```
Line Rnd(800),Rnd(600),Rnd(800),Rnd(600)
Flip
Until GetKey()
End
```

Пункт 5.4. Работа с изображениями.

В подпунктах объясняется, как загружать и выводить изображения, загружать анимационные изображения, работать с буфером изображения, проверять изображения на столкновения различных видов.

Пункт 5.4.1. Простые изображения.

При загрузке картинки функция загрузки присваивает значение переменной, которую называют идентификатор (дескриптор), если изображение не загрузилось, то присваивает 0. Можно загружать изображение следующих форматов: bmp, jpg, png, tga, tiff, psx. Каждое изображение имеет точку координат.

Загрузка происходит так:

перем=LoadImage("файл")

Вывод изображения на экран:

DrawImage перем,х,у

Замостить на экране выводимое сообщение:

TileImage перем,х,у

Вывод изображения на экран без реагирования на Cls:

TileBlock перем,[х],[у]

Копировать изображение:

перем2=CopуImage(перем1)

Удалить изображение:

FreeImage(перем)

Функция сохраняет изображение по пути и возвращает 1, если сохранено:

перем=SaveImage(перем,"путь")

Сделать точку координат изображения по центру:

MidHandle перем

Сделать точку координат для всех изображений по центру:
AutoMidHandle

Изменить координаты точки изображения:
HandleImage *перем,х,у*

Вернуть координаты точки изображения по оси X:
ImageXHandle(*перем*)

Вернуть координаты точки изображения по оси Y:
ImageYHandle(*перем*)

Вернуть размер по ширине:
ImageWidth(*перем*)

Вернуть размер по высоте:
ImageHeight(*перем*)

Пример. Загружается изображение и выводится на экран:

```
Graphics 640,480
fen=LoadImage("Fencer.jpg") ;загружает изображение
DrawImage fen,180,100 ;рисует изображение
WaitKey()
Cls ;очищает экран
End
```

Пункт5.4.2. Буфер изображения.

Чтобы создать изображение и нарисовать в нем что-нибудь, нужно использовать буфер изображения. Итак, все по порядку. Сперва создаётся изображение размером *pxm*, затем активируется буфер по этому изображению и рисуются в нём (буфере) пиксели. Далее активируются буферы экрана, и выводится данное изображение.

Создание изображения размером *m* на *n* (если нужно анимационное, то можно указать число кадров):

перем=CreateImage(m,n,[кадры])

Активация буфера изображения:
SetBuffer ImageBuffer(*перем,[кадр]*)

Пример:

```
img=CreateImage(100,100)
SetBuffer ImageBuffer(img)
```

```
Color 255,0,0
Oval 0,0,100,100
Color 0,255,0
Rect 20,20,60,60
SetBuffer BackBuffer()
DrawImage img,150,150
Flip
WaitKey()
```

Пункт 5.4.3 Анимационные изображения.

В Blitz3D анимационные изображения представляют собой файл, в котором кадры располагаются в ряд вплоты по горизонтали. А вывод на экран осуществляется по одному кадру. Например, дана полоса размером 100x10, которая содержит 10 кадров, тогда каждый кадр будет размером 10x10.

Загрузка происходит с помощью команды LoadAnimImage. Первый и второй параметр - размер ОДНОГО кадра, третий - с какого кадра начинается анимация (обычно с 0), четвертая - общее количество кадров на полосе. Итак:

перем=LoadAnimImage("путь",ширина,высота,начало,всего_кадров)

Вывод осуществляется с помощью той же команды DrawImage, только еще указывается номер выводимого кадра изображения:

DrawImage перем,х,у,кадр

Рассмотрим анимацию на конкретном примере:

```
Graphics 640,480
fen=LoadAnimImage("Ships.jpg",149,149,0,16) ;загружает анимационную картинку
Repeat
x=x+1 ;счётчик
If x>15 Then x=0 ;если x>15,то
;х присваиваем 0, чтобы кадр стал нулевым (анимация будет повторяться бесконечно, и чтобы
не была ошибка, если кадр будет превышать больше 15)
DrawImage fen,180,100,x ;рисует анимационную картинку
Delay 250 ;задержка на 250 миллисекунд
Cls
Until KeyHit(1)
End
```

Пункт 5.4.4. Проверка на столкновение изображений.

Иногда требуется проверить столкновение двух изображений. Например, курсор с кнопкой. Все функции, проверяющие на столкновение возвращают

0 или 1. Существует два вида проверки: метод столкновения прямоугольных областей и метод столкновения сложных форм.

1) Метод столкновения простых прямоугольных областей, т. е. их пересечение.

Пересечение изображения, как прямоугольника, с виртуальной прямоугольной областью (x,y - текущая позиция картинки, x1 и y1 - позиция прямоугольной области с определённой шириной и высотой):

ImageRectOverlap (перем,x,y,x1,y1,ширина,высота)

Пересечение изображения с другим изображением, как прямоугольник с прямоугольником:

ImagesOverlap (перем1,x1,y1,перем2,x2,y2)

Проверка на пересечение двух прямоугольников:

RectsOverlap (x1,y1,ширина1,высота1,x2,y2,ширина2,высота2)

Пример программы на пересечение двух изображений. Загружаются два изображения. Изображение №2, с которым будет происходить пересечение первого, создаётся случайными координатами x и y. Координаты изображения №1 меняются от положения курсора. При пересечении изображение №2 заново меняет координаты:

```
SetBuffer BackBuffer()  
image1=LoadImage("1. bmp")  
image2=LoadImage("2. bmp")  
Rand MilliSecs()  
x=Rand(20, 380)  
y=Rand(20, 280)  
Repeat  
DrawImage image2,x,y  
DrawImage image1,MouseX(),MouseY()  
If ImagesOverlap(image1,MouseX(),MouseY(),image2,x,y) Then x=Rand(20, 380) : y=Rand(20, 280)  
Flip  
Cls  
Until KeyHit(1)  
End
```

2) Метод столкновения сложных форм.

Например, даны два изображения, в одном из которых нарисован круг, в другом - квадрат. Оба нарисованы на чёрном фоне. Тогда столкновение будет происходить круг с квадратом. Только маска изображений должна быть чёрной, иначе они будут сталкиваться как квадраты.

Столкновение двух изображений (если изображение не анимационное, то кадр равен 0):

ImagesCollide(перем1,х1,у1,[кадр],перем2,х2,у2,[кадр])

Столкновение изображения с прямоугольной областью:

ImageRectCollide(перем1,х1,у1,[кадр],х2,у2,ширина,высота)

Пример на столкновение. Создаётся изображение и в буфере изображения рисуется окружность. Рисуется квадрат и изображение. Координаты квадрата меняются от положения курсора. При их столкновении происходит закрытие программы. Определите внимание на то, что столкновение происходит не как квадрат с квадратом:

```
Color 0,255,0
image=CreateImage(50,50)
SetBuffer ImageBuffer(image)
Oval 0,0,50,50
SetBuffer BackBuffer()
Repeat
Rect 100,100,50,50
DrawImage image,MouseX(),MouseY()
If ImageRectCollide(image,MouseX(),MouseY(),0,100,100,50,50) Then End
Flip
Cls
Until KeyHit(1)
End
```

Пункт5.5. Текст в графическом режиме.

До этого мы пользовались командами Print и Write, которые выводят текст по строке. Но существует команда Text, которая позволяет выводить текст по заданным координатам. Также существуют функции, работающие с шрифтом текста.

Вывод текста по координатам (необязательные параметры в скобках - выравнивание по центру соответствующих координат):

Text х,у,значение,[центр_х],[центр_у]

Загрузка шрифта. Параметры в скобках: первый - имя шрифта, второй - размер шрифта, третий - жирный (1 или 0), четвёртый - курсив (0 или 1), пятый - подчёркнутый. Если загрузка не удалась, то функция возвращает 0:
перем=LoadFont("имя",размер,жирный,курсив,подчёркнутый)

Установить текущим шрифт:

SetFont(перем)

Удалить шрифт:

FreeFont(перем)

Вернуть размер высоты символов шрифта:

FontHeight()

Вернуть размер ширины символов шрифта:

FontWidth()

Пример:

```
f=LoadFont ("courier",40,False,False,True)
SetFont f
Color 255,0,255
Text 50,50,"Шрифт Courier"
WaitKey()
FreeFont f
End
```

Пункт 5.6. Работа с пикселями.

В главе расписаны команды, работающие с пикселями экрана. В программе есть два вида команд. Первый вид медленный, но работает без блокировки буфера, второй - скоростной, но требует блокировки графического буфера.

Первый тип команд.

Прочитать значение цвета пикселя по координате:

перем=ReadPixel(x,y,[буфер])

Нарисовать пиксель по координате и цвету в формате argb (первый канал - прозрачность от 0 до 255):

WritePixel(x,y,rgb,[буфер])

Скопировать пиксель:

CopyPixel(x1,y1,буфер1,x2,y1,[буфер2])

Второй тип команд.

Блокировка буфера:

LockBuffer буфер

Разблокировка буфера:

UnLockBuffer буфер

Прочитать значение цвета пикселя по координате:

перем=ReadPixelFast(x,y,[буфер])

Нарисовать пиксель по координате и цвету в формате argb:
WritePixelFast(x,y,rgb,[буфер])

Скопировать пиксель:
CopyPixelFast(x1,y1,буфер1,x2,y1,[буфер2])

Небольшой пример для второго типа. Обратите внимание на то, что сначала делают блокировку буфера:

```
LockBuffer FrontBuffer()  
CopyPixelFast 125,244,FrontBuffer(),10,136  
UnlockBuffer FrontBuffer()
```

Глава6. Работа с видеофайлами.

Видео файлы в играх, например, используются как заставки. Итак, в Blitz3D проигрываются форматы Avi, Mpg и другие.

Открыть видеофайл (начинается звуковое воспроизведение файла):
перем=LoadMovie("файл")

Вывести изображение видеофайла:
DrawMovie(перем, x,y,[ширина],[высота])

Закреть видеофайл:
CloseMovie перем

Функция, которая возвращает 1, если файл воспроизводится и 0, если нет:
MoviePlaying(перем)

Размер видео по ширине:
MovieWidth(перем)

Размер видео по высоте:
MovieHeight(перем)

Пример. Воспроизводится видео, пока не нажата ESC или не перестал воспроизводиться файл:

```
Graphics 800,600  
SetBuffer BackBuffer()  
file=OpenMovie("D:\Terminator3.avi")  
Repeat  
Cls  
DrawMovie(file,0,0,GraphicsWidth(),GraphicsHeight())
```

```
Flip  
Until KeyHit(1) Or (Not MoviePlaying(file))  
CloseMovie(file)  
End
```

Глава7. 3D Графика.

Итак, мы приступаем к новой главе, в которой вы научитесь работать в трёхмерном режиме. Как и в предыдущих главах будет затронуто несколько тем, в каждой из которой будут объясняться различные команды и примеры.

Пункт7.1. 3D графический режим.

Перед тем как начать работать в графическом режиме, нужно активировать его с помощью команды Graphics3d. Все параметры этой команды такие же как у Graphics. После того как установили режим, создаётся пустое 3х-мерное пространство.

Graphics3D ширина,высота,[глубина],[режим]

Пример команды:

```
Graphics3D 800,600,32,1
```

Аналогично, отключение графического режима:

EndGraphics

Для наилучшей производительности рекомендуется использовать задний буфер.

Дополнительные команды.

Эмуляция 32 битного цвета в 16 битном режиме (True/False (1/0)):

Dither значение

Пункт7.2. Камера.

Камера - это объект, который позволяет отображать все объекты в пространстве. Без неё мы ничего не увидим. Для создания камеры служит функция CreateCamera, которая при создании камеры возвращает переменной адрес, указывающий на камеру. В параметрах может присутствовать дополнительный параметр, который указывает на координатную взаимосвязь с другим объектом-родителем. Например,

камера может зависеть от куба и, если двигается куб, то за ним двигается камера.

Создание камеры:

> перем=CreateCamera([родитель])

У камеры есть свойство дистанции прорисовки, т.е. от и до куда видны объекты, прорисовываемые перед камерой. По-умолчанию стоит от 1 до 1000:

> CameraRange перем,начало,конец

Зум камеры:

CameraZoom перем,значение

Область, в которой выводиться изображение в камере на экран. По-умолчанию область равна всему экрану. x, y - координаты начала прорисовки:

CameraViewport перем,x,y,ширина,высота

Цвет пространства (по-умолчанию 0,0,0):

CameraClsColor перем,крас,зел,син

Режим тумана. Создает у камеры эффект окружающего тумана (True/False (1/0)):

CameraFogMode перем,значение

Цвет тумана:

CameraFogColor перем,крас,зел,син

Дистанция начала и конца тумана:

CameraFogRange перем,начало,конец

Функция, которая проверяет, находится ли объект в поле зрения камеры. Возвращает 0 или 1:

EntityInView(перем_объекта,перем_камеры)

Самая главная команда. Прорисовывает на экран изображение в текущей камере:

RenderWorld

В Blitz3D возможно использование нескольких камер, которые должны отображаться на экране с помощью команды CameraViewport с

комбинацией их положений на экране, чтобы были видны обе камеры. Сам пример на создание и использование камеры будет рассмотрен в следующем пункте, когда мы будем работать с геометрическими объектами.

Пункт 7.3. Прimitives.

К примитивам в Blitz3D относятся бесконечная плоскость, LOD ландшафт, куб, сфера, конус, цилиндр, собственный объект. Все перечисленные относятся к классу Mesh. Каждый объект состоит из треугольников, каждый из которых в свою очередь состоит из трёх вершин. При создании, объект помещается в начальной координате 0,0,0. При создании объекта, переменной возвращается адрес, указывающий на данный объект. Как при создании камеры.

Создание бесконечной плоскости:

перем=CreatePlane()

Создание куба:

перем=CreateCube([родитель])

Создание сферы. Сегменты от 3 до 100. Определяет количество треугольников в сечении. По-умолчанию 8.:

перем=CreateSphere([сегменты],[родитель])

Создание цилиндра. Сегменты от 3 до 100. Определяет количество треугольников в сечении. По-умолчанию 8. Основание: 1 или 0, если 0, то будет труба:

перем=CreateCylinder([сегменты],[родитель],[основание])

Создание конуса. Сегменты от 3 до 100. Определяет количество треугольников в сечении. По-умолчанию 8. Основание: 1 или 0, если 0, то основание отсутствует:

перем=CreateCone([сегменты],[родитель],[основание])

Пример:

Graphics3D 640,480 ;3D графический режим

cam=CreateCamera() ;создание камеры

cube=CreateCube() ;создание куба

PositionEntity cube,0,0,4 ;позиция объекта

RenderWorld ;прорисовывает мир

Flip ;выводит изображение из буфера на экран

WaitKey() ;Останавливает программу до тех пор пока не нажата любая клавиша

End ;выход

Пункт 7.4. Работа с поверхностью тел.

После того, как мы создали тела, требуется изменить их внешние свойства, например, изменить цвет, надеть текстуру, сделать прозрачным. Ведь по-умолчанию, созданное тело белого цвета.

Пункт 7.4.1. Цвет тела, прозрачность и др.

Цвет тела (интенсивность каждого цвета от 0 до 255 (от -255 до 0 - для маски)):

EntityColor *перем,крас,син,зел*

Прозрачность тела (вещественное значение от 0 до 1):

EntityAlpha *перем,#значение*

Светимость тела (вещественное значение от 0 до 1). Работает при наличии источника света:

EntityShininess *перем,#значение*

Свойства смешивания свойств(0 - нет текстуры, 1 - по-умолчанию, 2 - часть тела, которая на фоне пространства - прозрачна, 3 - часть тела, попадающая на объекты - прозрачна):

EntityBlend *перем,значение*

Эффекты тела (Допустимые значения: 1 - полная яркость, 2 - цвет тела вместо цвета кисти, 4 - отсутствует сглаживание, 8 - не зависит от тумана, 16 - двусторонняя поверхность. Можно применить несколько эффектов сложив соответствующие значения):

EntityFX *перем,значение*

Пример. Создаётся полупрозрачную несглаженную сферу красного цвета:

```
Graphics3D 640,480
cam=CreateCamera()
sp=CreateSphere()
EntityColor sp,255,0,0
EntityAlpha sp,0.5
PositionEntity sp,0,0,5
EntityFx sp,4
RenderWorld
Flip
WaitKey()
End
```

Пункт 7.4.2. Текстура тела.

Чтобы тело имело сложную цветовую гамму, используют текстуры. Текстуры загружаются из файла изображений и могут быть многократно надеты на различные тела.

Загрузка текстуры. При удачной загрузке, переменной возвращается адрес на данную текстуру:

перем=LoadTexture("файл",[флаг])

Загрузка анимационной текстуры. Смотри пункт 5.4.3 для сравнения:

перем=LoadAnimTexture("файл",флаг,ширина,высота,начало,всего_ка

Удалить текстуру:

FreeTexture перем

Надеть текстуру на тело:

EntityTexture перем_тела, перем_текстуры,[кадр]

Изменить масштаб текстуры:

ScaleTexture перем,#ширина,#высота

Сдвинуть текстуру:

PositionTexture перем,горизонталь,вертикаль

Повернуть текстуру:

RotateTexture перем,угол

Функция, возвращающая ширину текстуры:

TextureWidth(перем)

Функция, возвращающая высоту текстуры:

TextureHeight(перем)

Пример. Загруженная текстура натягивается на куб:

Graphics3D 640,480 ;3D графический режим

cam=CreateCamera() ;создание камеры

cube=CreateSphere() ;создание сферы

PositionEntity cube,0,0,4 ;позиция объекта

tex=LoadTexture("texture.bmp") ;загрузка текстуры

EntityTexture cube,tex ;натягивание текстуры на объект

RenderWorld ;прорисовывает мир

Flip ;обновляет графический буфер

WaitKey() ;Останавливает программу до тех пор пока не нажата любая клавиша

End ;выход

Создание собственных текстур. Принцип аналогичен созданию собственных изображений (пункт5.4.2), только используется текстурный буфер.

Создать текстуру:

перем=CreateTexture(ширина,высота,[флаг],[кадр])

Активировать режим текстур:

SetBuffer TextureBuffer(перем)

Пример. Создаётся текстура с изображением круга и натягивается на куб.

```
Graphics3D 800,600
```

```
cam=CreateCamera()
```

```
cube=CreateCube()
```

```
PositionEntity cube,0,0,5
```

```
tex=CreateTexture(128,128)
```

```
SetBuffer TextureBuffer(tex)
```

```
Color 0,255,0
```

```
Oval 0,0,128,128
```

```
EntityTexture cube,tex
```

```
SetBuffer BackBuffer()
```

```
RenderWorld
```

```
Flip
```

```
WaitKey()
```

End В Blitz3D допускается режим мультитекстур, т.е. на один объект можно наложить несколько текстур.

Активировать режим мультитекстур (значение True/False (1/0)):

HWMultiTex значение

Возвращает имя файла текстуры:

перем\$=TextureName\$(перем_текстуры)

Пункт7.4.3. Кисти.

Кисти - это инструменты, которые позволяют рисовать на поверхностях объектов тени и блики. Кисти придают объектам глянцевый вид и делают более объёмным. Для полного эффекта должен присутствовать один и более источников света.

Создать кисть. При успешном создании, функция возвращает адрес переменной. В скобках может присутствовать значение цвета кисти:

перем=CreateBrush([кр,син,зел])

Загрузить кисть. При успешном создании, функция возвращает адрес переменной:

перем=LoadBrush("файл")

Удалить кисть:

FreeBrush перем

Цвет кисти (каждый цвет: 0-255):

BrushColor перем,крас,зел,син

Прозрачность кисти (вещественное значение от 0 до 1):

BrushAlpha перем,#знач

Яркость кисти (вещественное значение от 0 до 1):

BrushShininess перем,#знач

Применить текстуру с кистью:

BrushTexture перем_кисти,перем_текстуры,[кадр]

Пример:

```
Graphics3D 800,600
SetBuffer BackBuffer()
l=CreateLight() ;источник света
LightColor l,255,0,0
RotateEntity l,45,45,0
cam=CreateCamera()
sphere=CreateSphere(50)
PositionEntity sphere,0,0,5
brush=CreateBrush(255,255,25)
BrushShininess brush,1
PaintEntity sphere, brush
RenderWorld
Flip
WaitKey() : End
```

Пункт7.5. Движение тел.

Почти всегда вам понадобится, чтобы тело изменяло положение в пространстве. Нужно будет изменить координаты, изменить масштаб, повернуть на некоторый угол, заставить двигаться или вращаться. Любое тело создаётся в точке 0,0,0 и повёрнуто в каждой оси на 0 градусов. По-умолчанию, тело создаётся с масштабом 1,1,1.

Изменить положение тела в пространстве:

PositionEntity *перем,х#,у#,z#*

Повернуть тело в пространстве (значения в градусах, вращение происходит по оси):

RotateEntity *перем,х#,у#,z#*

Изменить масштаб тела в пространстве:

ScaleEntity *перем,х#,у#,z#*

Следующие команды должны работать в циклах:

Движение тела:

MoveEntity *перем,х#,у#,z#*

Вращение тела:

TurnEntity *перем,х#,у#,z#*

Установить цель на другое тело:

PointEntity *перем1,перем2*

Движение тела по координатным осям. (При повороте тело все равно будет двигаться прямо):

TranslateEntity *перем,х#,у#,z#*

Пример на движение:

```
Graphics3D 640,480 ;3D графический режим
SetBuffer BackBuffer() ;задний буфер
cam=CreateCamera() ;создание камеры
cube=CreateCube() ;создание куба
PositionEntity cube,0,0,4 ;позиция объекта
Repeat ;цикл Repeat...Until
TurnEntity cube,1,1,1 ;вращает тело
RenderWorld ;прорисовывает мир
Flip ;вывод с буфера на экран
Until KeyHit(1);конец цикла (пока не нажата Esc)
End ;выход
```

Пункт7.6. Управление телами. Родитель. Точка вращения.

Часто требуется удалять, скопировать, делать видимыми/невидимыми объекты.

Удалить объект из памяти:

FreeEntity *перем*

Удалить все объекты, текстуры и кисти из памяти:

ClearWorld

Сделать объект невидимым (он остаётся в памяти):

HideEntity *перемен*

Сделать объект видимым:

ShowEntity *перемен*

Скопировать объект:

перемен2=CopyEntity(перемен1)

Каждому объекту можно присвоить индивидуальное имя:

NameEntity *перемен, строка\$*

Каждый объект может зависеть координатами от другого. Объект, от которого зависят другие объекты, называется родителем. А зависимые объекты - детьми. Если присвоить объекту родителя, то начало координат у "ребёнка" будет находиться в родителе. А если родитель будет менять положение в пространстве, то ребёнок тоже будет менять положение, но относительно родителя не будет. У родителя будут мировые координаты, а у ребёнка - местные.

Присвоение объекту родителя:

EntityParent *перемен_ребёнок, перемен_родитель*

Функция, возвращающая число детей у родителя:

CountChildren(перемен_родитель)

Функция, возвращающая имя родителя у ребёнка:

GetParent(перемен_ребёнок)

Функция. Получить адрес переменной ребёнка по индексу (порядковому номеру):

GetChildren(перемен_ребёнок, индекс)

Функция. Получить адрес переменной ребёнка по индивидуальному имени:

FindChildren(перемен_ребёнок, строка\$)

Пример. Создаётся три тела: шар и два куба. Кубы являются детьми шара, т.е. шар - родитель. Одному из кубов присваивается индивидуальное имя. Зная только родителя, определить адрес каждого ребёнка двумя способами:


```

Graphics3D 800,600
parent=CreateSphere()
children1=CreateCube()
children2=CreateCube()
NameEntity children2,"Vasya" ;присвоить имя
EntityParent children1,parent ;присвоить родителя (1)
EntityParent children2,parent ;присвоить родителя (2)
Text 4,5,"Children1: "+children1+" Children2: "+children2 ;вывести адреса детей
Text 4,15,GetChild(parent,1) ;определить адрес ребёнка по индексу
Text 4,25,FindChild(parent,"Vasya") ;определить адрес ребёнка по имени
WaitKey() : End

```

Можно создавать особый объект, который называется точкой вращения. Это невидимая точка в пространстве, которая служит как точка вращения для других объектов.

Эта команда создаёт точку вращения и присваивает переменной её адрес:
перем=CreatePivot([родитель])

Пример. Создаётся точка вращения и шар, который является ребенком для точки вращения и будет вращаться относительно неё:

```

Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
aim=CreatePivot(); создаётся точка вращения
PositionEntity aim,0,0,20
sph=CreateSphere(8,aim); родитель шара - точка вращения
PositionEntity sph,0,10,0
Repeat
TurnEntity aim,0,0,1; вращение точки вращения
RenderWorld
Flip
Until KeyHit(1)
End

```

Пункт7.7. Статус тел в мире.

Иногда нужно знать, где находятся объекты, на сколько градусов они повернуты, определить расстояние между двумя объектами и др. Для этих целей существуют особые функции.

Функция, возвращающая координату тела по оси X:

EntityX#(перем)

Функция, возвращающая координату тела по оси Y:
EntityY#(перем)

Функция, возвращающая координату тела по оси Z:
EntityZ#(перем)

Функция, возвращающая отклонение тела от оси X (градусы):
EntityPitch#(перем)

Функция, возвращающая отклонение тела от оси Y (градусы):
EntityYaw#(перем)

Функция, возвращающая отклонение тела от оси Z (градусы):
EntityRoll#(перем)

Функция, возвращающая ширину тела:
MeshWidth#(перем)

Функция, возвращающая длину тела:
MeshDepth#(перем)

Функция, возвращающая высоту тела:
MeshHeight#(перем)

Функция, определяющая расстояние между телами:
EntityDistance#(перем1,перем2)

Функция, возвращающая имя объекта:
EntityName\$(перем)

Функция, возвращает класс объекта (Её результатом является одно из следующих строковых значений: Pivot, Light, Camera, Mirror, Listener, Sprite, Terrain, Plane, Mesh, MD2, BSP):
EntityClass\$(перем)

Пункт 7.8. Создание своих тел.

В Blitz3D можно создавать собственные объекты вида Mesh. Сначала создаётся тело, затем поверхность, которая будет содержать в себе треугольники. Далее создаются вершины тела по координатам. По созданным вершинам создаются треугольники.

Создание тела Mesh:

перем=CreateMesh([родитель])

Создание поверхности для Mesh:

перем_пов=CreateSurface(перем)

Удаление поверхности. Если значения для вершин и/или треугольников True, то они удаляются с поверхности, а False - не удаляются:

ClearSurface перем,[вершины],[треугольники]

Создание вершины для поверхности:

перем_вер=AddVertex(перем_пов,x,y,z)

Создание треугольника (создаётся по трём вершинам):

AddTriangle(перем_пов,перем_вер1,перем_вер2,перем_вер3)

Функция, возвращающая количество треугольников на поверхности:

CountTriangles(перем_пов)

Функция, возвращающая количество вершин на поверхности:

CountVertices(перем_пов)

Функция, возвращающая количество поверхностей на объекте типа Mesh:

CountSurfaces(перем_объекта)

Функция, возвращающая указателю поверхность, взятую с объекта по индексу (номеру) поверхности(номер должен быть от 1). Т.е. создаётся новая поверхность, взятая с другого объекта:

перем_пов=GetSurfaces(перем_объекта,индекс)

Пример. Создаётся объект, поверхность которого состоит из четырех вершин и двух треугольников:

```
Graphics3D 800,600
```

```
cam=CreateCamera()
```

```
obj=CreateMesh() ;создаётся объект
```

```
so=CreateSurface(obj) ;создаётся его поверхность
```

```
v1=AddVertex(so,-1,0,0) ;создаётся вершина поверхности
```

```
v2=AddVertex(so,1,1,0)
```

```
v3=AddVertex(so,0,3,0)
```

```
v4=AddVertex(so,-1,2,0)
```

```
AddTriangle(so,v3,v2,v1) ;создаётся треугольник на поверхности по вершинам
```

```
AddTriangle(so,v4,v3,v1)
```

```
PositionEntity obj,0,0,5
```

```
RenderWorld
```

```
WaitKey()
```

End

Пункт 7.9. Загрузка объектов.

В Blitz3D можно загружать объекты, созданные на других программах. Загружаются объекты типа Mesh: 3DS, B3D и X; а также формата MD2 и BSP. Формат 3DS - формат 3D Studio Max, X - формат DirectX, B3D - формат Blitz, MD2 - формат анимационных моделей из Quake2, а BSP - формат уровней из Quake3. Если у объектов формата 3DS, X и B3D имеются текстуры, то они будут загружаться автоматически при загрузке данных объектов. Только эти текстуры должны находиться в той же папке, где находятся сами объекты. Объект типа MD2 чаще используется при анимации персонажей. Для него текстуры не загружаются, поэтому их надо загружать самим. MD2 не относится к объектам типа Mesh.

Загрузка объектов формата X и 3DS (при успешной загрузке, переменной возвращается адрес объекта):

перем=LoadMesh("файл",[родитель])

Загрузка объектов формата MD2 (при успешной загрузке, переменной возвращается адрес объекта):

перем=LoadMD2("файл",[родитель])

Пример. Загружается объект "house.3ds":

```
Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
m=LoadMesh("house.3ds")
ScaleEntity m,0.001,0.001,0.001
PositionEntity m,0,0,10
RenderWorld
Flip
WaitKey()
End
```

загрузка BSP уровней осуществляется так (инт_света# - интенсивность окружающего света, по-умолчанию 0):

перем=LoadBSP("файл",[инт_света#],[родитель])

Оттенок BSP уровня:

BSPAmbientLight перем, кр#, зел#, син#

Включить/выключить световые карты окружения в уровне (знач 0/1 или False/True):

BSPLighting перем, знач

Пример. Загружается уровень "q3dm12.bsp":

```
Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
level=LoadBSP("q3dm12.bsp",.75 )
ScaleEntity level,0.01,0.01,0.01
PositionEntity level,0,0,10
RenderWorld
Flip
WaitKey()
End
```

Пункт 7.10. Анимация объектов.

Почти в любой игре вы встречали анимацию, например, бег героев, врагов, вращение винта вертолѐта и др. И в Blitz3D существуют специальные команды, которые позволяют воспроизводить анимацию 3D объектов. Анимационные объекты имеют "кадры" или определённые фиксированные координаты отдельных частей объекта в разные моменты времени. Теперь поговорим об анимации объектов различных форматов. У объектов форматов 3DS и X анимируют компоненты, состоящие из примитивов, но не анимируют по вершинам и треугольникам. Но формат MD2 позволяет анимировать по отдельным вершинам и треугольникам, поэтому его используют для анимации персонажей и животных.

Загрузка анимационного объекта в формате 3DS и X (при успешной загрузке, переменной возвращается адрес на объект):

перем=LoadAnimMesh("файл",[родитель])

Воспроизвести анимацию для форматов 3DS и X. Допустимые значения для параметра режим: 1 - однократная анимация, 2 - зацикленная анимация, 3 - воспроизведение вперѐд-назад, 0 - анимации нет. Скорость - параметр скорости анимации. Кадр - значение начального кадра. Задержка - пауза перед анимацией:

Animate перем,[режим],[скорость#],[кадр],[задержка#]

Функция, возвращающая время анимации объекта:

AnimTime#(перем)

Проверка на воспроизведение анимации. Функция возвращает 1, если объект анимирует и 0, если нет:

Animating(перем)

Загрузка анимационного объекта в формате MD2. Загружается такой же командой, как и неанимационный объект:

перем=LoadMD2("файл",[родитель])

Воспроизвести анимацию MD2. Допустимые значения для параметра режим: 1 - однократная анимация, 2 - зацикленная анимация, 3 - воспроизведение впрёд-назад, 0 - анимации нет. Скорость - параметр скорости анимации. Начанльный кадр - значение начального кадра. Конечный кадр - значение конечного кадра. Задержка - пауза перед анимацией:

**Animate перем,[режим],[скорость#],[начальный_кадр],
[конечный_кадр],[задержка#]**

Функция, возвращающая время анимации MD2 объекта:

MD2AnimTime#(перем)

Проверка на воспроизведение анимации MD2 объекта. Функция возвращает 1, если объект анимирует и 0, если нет:

MD2Animating(перем)

Чтобы воспроизводилась анимация объектов, должна присутствовать в цикле команда UpdateWorld (значение параметра скорость по-умолчанию равно 1):

UpdateWorld [скорость#]

Пункт 7.11. Столкновение объектов.

Созданные объекты при движении не будут взаимодействовать друг с другом, так как не установлены для них столкновения. Объекты будут проходить насквозь друг друга, поэтому чтобы избежать таких ситуаций, используют столкновения.

Установка режима столкновений. Используется команда Collisions, которая имеет множество параметров. Первый параметр - тип объекта (целое значение от 1 до 999), который будет сталкиваться. Вторым параметром - тип объектов (целое значение от 1 до 999), об которые будут сталкиваться. Третий параметр - метод столкновения: 1 - сфера к сфере, 2 - сфера к многограннику, 3 - сфера к кубу. Четвёртый параметр - реакция сталкиваемого объекта: 1 - объект останавливается при столкновении, 2 - объект скользит при столкновении, 3 - объект скользит с трением.

Общий вид команды:

Collisions тип1,тип2,метод,реакция

Чтобы объекты могли сталкиваться, нужно им применить тип (значение от 1 до 999):

EntityType перем,значение

Чтобы работала анимация, в цикле используют (та же команда для анимации объектов):

UpdateWorld

Примеры.

1 метод: метод с остановкой. Создаётся шар и конус. Тип шара - 1, а конуса - 2. В команде Collisions устанавливается первый параметр равен 1, т.е. сталкивается объект с типом 1 - это шар. Второй параметр равен 2 - тип конуса, об который будет сталкиваться шар. Третий параметр равен 2 (сфера об многогранник), лучше использовать такой метод, потому что конус - это многогранник. Последний параметр равен 1, значит, шар при столкновении остановится.

```
Graphics3D 640,480 ;3D графический режим
cam=CreateCamera() ;создаёт камеру
sphe=CreateSphere() ;создаёт шарик
PositionEntity sphe,4,0,8 ;позиция шарика
EntityType sphe,1 ;тип шарика
cone=CreateCone() ;создаёт конус
PositionEntity cone,-3,0,8 ;позиция конуса
EntityType cone,2 ;тип конуса
Collisions 1,2,2,1 ;устанавливает столкновения
While Not KeyHit(1)
MoveEntity sphe,-0.05,0,0
UpdateWorld ;обновляет анимацию и столкновения объектов
RenderWorld ;прорисовывает мир
Flip ;отображает графический буфер
Wend ;конец цикла
End ;выход
```

2 метод: метод со скольжением. Пример ничем не отличается от первого, только последний параметр команды Collisions равен 2.

```
Graphics3D 640,480 ;3D графический режим
cam=CreateCamera() ;создаёт камеру
sphe=CreateSphere() ;создаёт шарик
PositionEntity sphe,4,0,8 ;позиция шарика
EntityType sphe,1 ;тип шарика
cone=CreateCone() ;создаёт конус
```

```

PositionEntity cone,-3,0,8 ;позиция конуса
EntityType cone,2 ;тип конуса
Collisions 1,2,2,2 ;устанавливает столкновения
While Not KeyHit(1)
MoveEntity sphe,-0.05,0,0
UpdateWorld ;обновляет анимацию и столкновения объектов
RenderWorld ;прорисовывает мир
Flip ;отображает графический буфер
Wend ;конец цикла
End ;выход

```

3 метод: метод со скольжением и трением. Последний параметр команды Collisions равен 3.

```

Graphics3D 640,480 ;3D графический режим
cam=CreateCamera() ;создаёт камеру
sphe=CreateSphere() ;создаёт шарик
PositionEntity sphe,4,0,8 ;позиция шарика
EntityType sphe,1 ;тип шарика
cone=CreateCone() ;создаёт конус
PositionEntity cone,-3,0,8 ;позиция конуса
EntityType cone,2 ;тип конуса
Collisions 1,2,2,3 ;устанавливает столкновения
While Not KeyHit(1)
MoveEntity sphe,-0.05,0,0
UpdateWorld ;обновляет анимацию и столкновения объектов
RenderWorld ;прорисовывает мир
Flip ;отображает графический буфер
Wend ;конец цикла
End ;выход

```

Дополнительные команды:

Отменить все столкновения:

ClearCollisions

Сбросить столкновение для определённого объекта:

ResetEntity(перем)

Установить радиус столкновения для объекта (вещественное значение) с методом столкновения сфера к сфере. Если значение меньше 1, то объект будет частично проходить другой, больше - не доходить до его поверхности.:

EntityRadius перем,знач#

Установить границу столкновения для объекта с методом столкновения

сфера к кубу: x,y,z - координаты относительно объекта, а ширина, длина, высота - размер границы:

EntityBox **перем,х#,у#,z#,ширина#,высота#,длина#**

Функция, проверяющая произошло ли столкновение. Возвращает 1, если объекты столкнулись и 0, если нет. переменная - сталкиваемый объект, а тип - тип всех объектов, с которыми сталкивается объект:

EntityCollided(перем,тип)

Функция, возвращающая тип объекта:

GetEntityType(перем)

Функция, возвращающая количество произошедших столкновений в мире:

CountCollisions(перем)

Пример на проверку столкновений. Если первый куб столкнулся со вторым, то выходит сообщение:

```
Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
cub1=CreateCube()
PositionEntity cub1,-4,0,8
EntityType cub1,125
cub2=CreateCube()
PositionEntity cub2,4,0,8
EntityType cub2,347
Collisions 347,125,1,1
Repeat
MoveEntity cub2,-0.1,0,0
UpdateWorld
RenderWorld
Flip
Until EntityCollided(cub2,125)
RuntimeError "Кубики столкнулись"
End
```

Пункт7.12. Свет.

Источники света используются для того, чтобы объекты в пространстве не казались плоскими, а также чтобы использовать различные эффекты. В Blitz3D допускается одновременно использовать 8 источников света.

Создание источника света, при создании переменной присваивается адрес источника света. Необязательный параметр определяет тип света: 1 -

направленный свет, 2 - точечный свет, 3 - свет в виде луча(конуса):
перем = CreateLight([тип],[родитель])

Цвет света. Значения для красного, зелёного и синего от 0 до 255. Если значения будут от -255 до 0, то цвет света будет в негативе:
LightColor перем,кр,зел,син

Расстояние распространения точечного света (вещественное значение):
LightRange перем,знач#

Угол для лучевого (конусоидального) света. Первое значение - внутренний угол луча, а второе значение - внешний угол луча. По-умолчанию внешний угол равен 90, а внутренний 0.:
LightConeAngles перем,внут_угол#,внеш_угол#

Пример. Создаётся сфера, которая освещается точечным зелёным светом, распространяющимся на 5 едениц:

```
Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
sph=CreateSphere(16)
PositionEntity sph,0,0,8
light=CreateLight(2)
LightColor light,0,0,255
LightRange light,5
PositionEntity light,0,10,8
RenderWorld
Flip
WaitKey
End
```

Пункт 7.13. Спрайт.

Спрайт - это двухмерное изображение загруженное, в трёхмерное пространство. Оно представляет из себя полупрозрачный квадрат с текстурой, т.е. плоскость, состоящая из двух треугольников. Спрайты используют как частицы пламени, дыма, искр и многих других эффектов.

Команда загружает спрайт и возвращает переменной адрес на спрайт. Первый параметр - имя файла, из которого загружается изображение для спрайта, второй - вид спрайта, третий - переменная объекта родителя. Второй параметр должен принимать следующие значения: 1 - полупрозрачный спрайт, 2 - затемненный, 4 - непрозрачный:
перем = LoadSprite("файл",[вид],[родитель])

Можно изменить размер (масштаб) спрайта с помощью:
ScaleSprite *переменная, ширина, высота*

Поворот спрайта на некоторый угол:
RotateSprite *переменная, значение#*

По-умолчанию созданный спрайт будет всё время поворачиваться на камеру, этот метод придаёт спрайту эффект объёмности. Но иногда нужно, чтобы спрайт не поворачивался на камеру, поэтому существует команда **SpriteViewMode**. Параметр этой команды принимает следующие значения: 1 - спрайт поворачивается на камеру, 2 - спрайт не поворачивается на камеру, 3 - спрайт поворачивается на камеру только по оси у:
SpriteViewMode *переменная, значение*

Изменить координатную точку отсчета у спрайта:
SpriteHandle *переменная, x, y*

Можно создать спрайт без изображения:
переменная = CreateSprite([родитель])

Пример. Загружается спрайт, повернутый на 45 градусов, и не поворачивается на камеру:

```
Graphics3D 800,600
SetBuffer BackBuffer()
cam=CreateCamera()
sprite=LoadSprite("sprite.bmp",1) ;загружается полупрозрачный спрайт
PositionEntity sprite,0,0,5
SpriteViewMode sprite,2 ;спрайт не поворачивается на камеру
RotateSprite sprite,45 ;спрайт повернут на 45 градусов
RenderWorld
Flip
WaitKey()
End
```

Пункт 7.14. Озвучивание 3D объектов.

Существует эффект, позволяющий загружать звуковой файл, который будет воспроизводиться определённым объектом. Также существует слушатель - это объект, который слышит звук, воспроизводимый другим объектом. По мере приближения (отдаления) объекта относительно слушателя, будет увеличиваться (уменьшаться) громкость воспроизводимого звука. Пример: слушателем является камера, а объектом, воспроизводящем звук, - космический корабль. Пролетая мимо камеры, будет меняться громкость

звука.

Сперва надо задать слушателя для некоторого объекта. Это делается с помощью команды `CreateListener`. При успешном создании слушателя, функция возвращает его адрес. Первый параметр - переменная объекта, который будет слушателем. Второй параметр - коэффициент громкости (по-умолчанию 1). Если он меньше 1, то звук, доносящийся от объекта, будет громче, в зависимости от расстояния. Тише, если больше 1. Третий параметр - значение эффекта Доплера (по-умолчанию 1). Четвёртый параметр - искусственное вычисление расстояния (по-умолчанию 1):

`CreateListener(перем,[парам2#],[парам3#],[парам4#])`

Загрузить 3D звук и вернуть адрес переменной:

`перем=Load3DSound("файл")`

Воспроизвести 3D звук объектом:

`EmitSound перем_звука,перем_объекта`

Конкретный пример для наглядного представления. Камера - слушатель, куб - источник звука. По мере приближения куба к камере, будет меняться громкость звука:

```
Graphics3D 640,480
cam=CreateCamera()
x=CreateCube()
CreateListener (cam,0.5,1) ;создаём слушателя
s=Load3DSound("fly.wav") ;загружаем 3D звук
PositionEntity x,2,0,-100
LoopSound s ;делаем звук зацикленным
EmitSound s,x ;объект x воспроизводит звук s
While Not KeyHit(1)
MoveEntity x,0,0,1
RenderWorld
Flip
Wend
End
```

Заключение.

Данный учебник создавался по материалу помощи в Blitz3D v1.98. В будущем ожидается выход дополнительных тем и исправлений для этого учебника.
