Appendix 1 Summary of mode and colour facilities

Text facilities available in different modes

mode	colours available	characters per line	lines
0	2	80	32
1	4	40	32
2	16	20	32
3	2	80	25
4	2	40	32
5	4	20	32
6	2	40	25
7	Teletext display	40	25

Graphics facilities available in different modes

mode	colours available	graphics resolution
0	2	640 x 256
1	4	320 x 256
2	16	160 x 256
4	2	320 x 256
5	4	160 x 256

Note that there no graphics facilities in modes 3, 6 and 7.

Memory requirements for different modes

mode	memory requirements
0	20K
1	20K
2	20K
3	16K
4	10K
5	10K
6	8K
7	1K

Overall colour range

There are sixteen actual colours available (on the Model A or B). These colours are numbered from 0 to 15.

Actual colour numbers and corresponding colours

colour number	colour name
0	black
1	red
2	green
3	yellow
4	blue
5	magenta
6	cyan
7	white
8	flashing black-white
9	flashing red-eyan
10	flashing green-magenta
11	flashing yellow-blue
12	flashing blue-yellow
13	flashing magenta-green
14	flashing cyan-red
15	flashing white-black

Colour codes in different modes

In each mode colours are referred to by code numbers from 0 upwards (using COLOUR for text colour and GCOL for graphics colour). The background colour is set by adding 128 to the required code number. The code numbers for a mode can be made to refer to any combination of actual colours (using VDU 19). There is an initial or default setting for each mode which specifies the colour that you get if you do not use VDU 19.

2 colour modes (MODES 0,3,4,6)

colour code numbers		default actual colours	
foreground	background	colour	number
0	128	black	0
1	129	white	7

4 colour mode (MODES 1 and 5)

colour code numbers		default actual colours	
foreground	background	colour	number
0	128	black	0
1	129	red	1
2	130	yellow	3
3	131	white	7

In the 16 colour mode (MODE 2) the colour codes are initially set to the corresponding actual colour numbers.

Appendix 2 **Bits, bytes and hex**

For the majority of straightforward progranuning applications, the user of the BBC micro need not concern himself with the details of how things like numbers and strings are represented inside his computer, but for some advanced applications a more detailed knowledge of the internal representation of information is required.

Bits

All information stored in a modern digital computer is held in the form of 'binary digits'. In this context, the word 'binary' means 'having two possible values', and a binary digit can thus be set to one of two possible values. We usually abbreviate the term binary digit to 'bit'.

When we write a bit on paper, we represent its two possible values as 0 or 1. Inside a computer, a bit might be represented by a magnetic field lying in one of two possible directions, or by an electronic voltage that can be positive or negative. The programmer, however, need not concern himself with the practicalities of representing a bit electronically or magnetically. When he needs to think in terms of the binary representation of information, he can think entirely in terms of ones and zeros.

With one bit, we can represent only two possible values, 0 or , and in fact scane of the information in our BBC computer is coded using only one bit. For example, in MODE 4, one bit is used to code the colour of each pixel on the screen. Each pixel can be one of two colours, colour 0 or colour 1.

Bit patterns

Bits are usually organised into groups or 'patterns'. With a group of two bits, each bit can one of two values giving 2x2 possible different patterns.

first bit	second bit	bit pattern
0	0	00
0	1	01
1	0	10
1	1	11

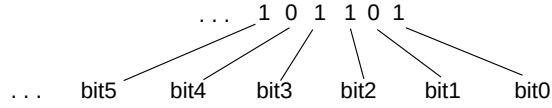
A two-bit pattern is used to code the colour of each pixel an the screen in a four colour mode such as MODE 5.

With three bits, there are 2x2x2 possible different patterns and so on:

no. of bits in pattern	example	no. of possible differnt patters	
1	0	2	
2	10	4	= 2x2
3	011	8	= 2x2x2
4	1010	16	= 2x2x2x2
5	10100	32	= 2x2x2x2x2
6	011010	64	= 2x2x2x2x2x2
7	1101001	128	= 2x2x2x2x2x2x2
8	11000101	256	= 2x2x2x2x2x2x2x2

Bit numbering

The bits in a bit pattern are usually referred to by numbering thenm from zero upwards from right to left, bit0, bit1, bit2 and so on.



Bytes

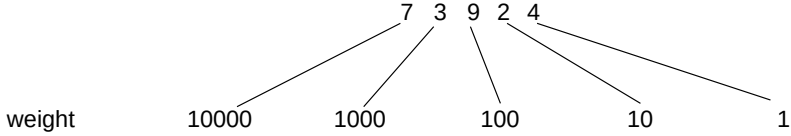
A group of 8 bits is called a 'byte'. One 'word' on your BBC micro contains one byte or one 8-bit pattern. The entire store that is accessible to the user consists of 16,384 words or bytes on a Model A and 32,768 words or bytes on a Model B. We usually quote storage capacity in 'K' where:

$1K = 1024 \ (1024 = 2^{10})$

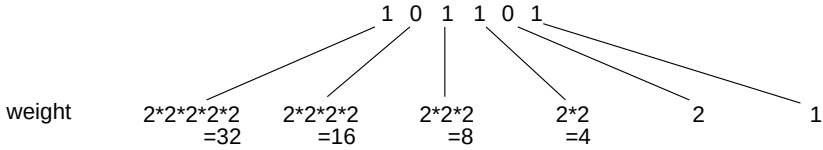
Because we are working on a binary system, everything is organised behind the scenes in powers of 2. Thus we say that a Model A has 16K bytes of store, i.e. 16*1024 bytes or 16*1024*8 bits.

8-bit integers

When we use a group of decimal digits to represent a non-negative integer, each digit has a weight that is a different power of 10. For example, with 5-digits:



When we use a bit-pattern to represent a non-negative integer, only two values are available for each digit, so we give each digit a weight that is a power of 2. For example, with a 6-bit pattern we might have:



We use a full byte to represent an integer in this way, we have:

binary	decimal
00000000 =	0
00000001 =	1
00000010 =	2
00000011 =	3
.	.
.	.
.	.
01111110 =	126
01111111 =	127
10000000 =	128
.	.
.	.
.	.
11111110 =	254
11111111 =	255

We saw earlier that there are 256 different B-bit patterns and they can be used in this way to represent integers in the range 0 to 255. Because it contributes least weight to an integer, the rightmost bit, bit0, is usually called the least significant bit and the leftmost bit is called the most significant.

8-bit positive and negative integers

If we want to use bytes to represent both positive and negative integers, we have to define a different correspondence between the available bit-patterns and the values they represent. The representation normally used is kown as '2s complement' representation. A detailed description of this is beyond the scope of this book, but the next table shows how a byte would be used to represent negative as well as positive integers. The bit-patterns that were previously used to represent positive integers from 128

up to 255 are now usod in the same order as before to represent the negative integers from -128 up to -1. In particular, -1 is represented by a bit-pattern that consists entirely of ones. This representation for negative numbers may seem rather strange, but it has many advantages when the computer is doing calculations that involve positive and negative numbers.

binary	decimal
.	.
10000000 =	-128
10000001 =	-127
10000010 =	-126
.	.
.	.
.	.
11111110 =	-2
11111111 =	-1
00000000 =	0
00000001 =	1
00000010 =	2
.	.
.	.
.	.
01111110 =	126
01111111 =	127

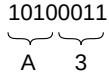
Note that you cannot tell by looking at a bit-pattern what sort of information it is being used to represent. This is determined by the context in which it is used and by the way it is processed by the circuits of the computer. For example, the same bit pattern might be used in different contexts to represent an integer or a character code.

Hexadecimal notation

When we are working with bit-patterns, it becomes very tedious having to write long sequences of ones and zeros when we want to specify a particular bit-pattern. We could abbreviate a byte by writing it as the equivalent positive decimal number, such as 179, but it is not at all obvious if we write 179 that we are talking about the bit-pattern 10110011. When we want to abbreviate a bit-pattern in a way that is not too far removed from its binary form, it is usual to write it in 'hexadecimal' notation (or hex for short). The bit-pattern is first divided into groups of four bits. There are 16 possible different patterns of four bits and each of these possible patterns can be represented by a single 'hexadecimal digit' as follows:

4-bit pattern	hexadecimal digit	4-bit pattern	hexadecimal digit
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

We can thus write the bit-pattern 10100011 in hex as A3:



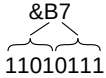
In BBC BASIC, we can write numbers in a program in hex if we precede the number by the symbol '&'. Thus we write &B3. Here are some other examples of bytes and the corresponding hex and decimal numbers:

byte	hex	decimal
00011111	&1F	31
00101110	&2E	46
01101001	&69	105
11111111	&FF	255

Note that &69 is quite different from decimal 69 which would be represented by the bit-pattern:

01000101 = &85

Because one hexadecimal digit corresponds to four binary digits, it is easy to visualize the bit-pattern corresponding to a hexadecimal number (provided that we are familiar with the sixteen patterns that correspond to the sixteen hex digits). Thus, for example, &B7 is easily visualized as:



and &FA is easily visualized as:

32-bit numbers

A numeric variable in BASIC occupies four computer words which contain four bytes or 32 bits. A number stored in such a variable is coded as a pattern of 32 bits. The way in which a 32-bit pattern is used to represent positive and negative integers is a simple extension of the 8-bit 2s complement representation introduced earlier. Note in particular that -1 is represented by a pattern of 32 ones. Details of how real numbers are coded as bit-patterns are beyond the scope of this book.

Logical operations on bit-patterns

The various logical plotting modes selected by GCOL (Chapter 2) use logical operations on bit-patterns when plotting new information on the screen. For this reason alone, some knowledge of these operations is necessary. The logical operators AND, OR, EOR and NOT treat the values to which they are applied as bit-patterns and operate on the individual bits of those patterns. A detailed knowledge of how these operations work is occasionally useful in advanced programing applications.

When a logical operation is applied to a bit-pattern or to a pair of bit-patterns, the individual bits are handled separately in creating the resultant bit-pattern. AND, OR and EOR are each applied to a pair of bit-patterns of the same length and the result is another bit-pattern of the same length. NUT is applied to a single bit-pattern and the result is another bit-pattern of the same length. We shall illustrate the behaviour of the logical operations on bytes, but they will behave in exactly the same way on shorter or longer bit-patterns.

AND

Each bit in the new pattern is the result of 'anding' the two bits in the same position in the two given bit-patterns according to the following table:

bit1	bit2	bit1 AND bit2
0	0	0
0	1	0
1	0	0
1	1	1

Thus, for example:

```
       byte1         10110100
       byte2         01100101

  byte1 AND byte2    00100100
```

OR
Each bit in the new pattern is the result of 'oring' the two bits in the same position in the given bit-patterns according to the following table:

bit1	bit2	bit1 OR bit2
0	0	0
0	1	1
1	0	1
1	1	1

Thus, for example:

```
       byte1        10110100
       byte2        01100101

  byte1 OR byte2    11110101
```

EOR
Each bit in the new pattern is the result of 'exclusive oring' the two bits in the same position in the given bit-patterns according to the following table:

bit1	bit2	bit1 EOR bit2
0	0	0
0	1	1
1	0	1
1	1	0

The name of the operator derives from the fact that it 'excludes' the case where both bits to which it is applied are 1. Thus, for example:

```
       byte1        10110100
       byte2        01100101

  byte1 EOR byte2   11010001
```

NOT
Each bit in the new bit-pattern is the result of 'negating' the same bit in the given bit-pattern. NOT produces the 'logical inverse' of the given bit-pattern by changing Os to 1s and 1s to Os.

bit	NOT bit
0	1
1	0

Thus, for example:

byte	10110100
NOT byte	01001011

Representation of TRUE and FALSE

In BBC BASIC, the value TRUE is represented by a bit-pattern containing nothing but ones and FALSE is represented by a bit-pattern containing nothing but zeros. When these values are stored in numeric variables, they look like the numeric values -1 and 0.

Appendix 3 Characters, ASCII codes, control codes and Teletext codes

ASCII codes

A character is stored inside the computer as an integer that zccupies 8 bits or one byte. There is an internationally agreed standard set of codes for the commonly used characters. These are the ASCII codes (American Standard Code for Information Interchange). The next table contains a list of the common visible characters together with their ASCII codes in decimal and hex.

ASCII characters and their codes

decimal code	hex code	char	decimal code	hex code	char	decimal code	hex code	char
32	&20		64	&40	@	96	&80	£
33	&21	!	65	&41	A	97	&61	a
34	&22	"	66	&42	B	98	&62	b
35	&23	#	67	&43	C	99	&63	c
36	&24	$	68	&44	D	100	&64	d
37	&25	%	69	&45	E	101	&85	e
38	&26	&	70	&46	F	102	&66	f
39	&27	'	71	&47	6	103	&67	g
40	&28	(	72	&48	H	104	&68	h
41	&29	)	73	&49	I	105	&69	i
42	&2A	*	74	&4A	J	106	&6A	j
43	&2B	+	75	&4B	K	107	&6B	k
44	&2C	,	76	&4C	L	108	&6C	I
45	&2D	-	77	&4D	M	109	&BD	m
46	&2E	•	78	&4E	N	110	&6E	a
47	&2F	/	79	&4F	0	111	&8F	a
48	&30	0	80	&50	P	112	&70	p
49	&31	1	81	&51	Q	113	&71	q
50	&32	2	82	&52	R	114	&72	r
51	&33	3	83	&53	8	115	&73	a
52	&34	4	84	&54	T	116	&74	t
53	&35	5	85	&55	U	117	&75	u
54	&36	6	86	&56	V	118	&76	v
55	&37	7	87	&57	W	119	&77	w
56	&38	8	88	&58	X	120	&78	x
57	&39	9	89	&59	Y	121	&79	y
58	&3A	:	90	&5A	Z	122	&?A	z
59	&3B	;	91	&5B	[	123	&7B	{
60	&3C	<	92	&5C	\	124	&7C	;
61	&3D	=	93	&5D	]	125	&7D	j
62	&3E	>	94	&5E	*	126	&7E	~
63	&3F	?	95	&5F	_			

Control codes

A number of the 256 available character codes are reserved for special purposes on the BBC computer. Sending one of these codes to the display hardware by using a PRINT or a VDU statement has a special effect. These codes are usually referred to as 'VDU drivers'. Note that some of the codes must always be followed by a fixed number of additional codes or 'parameters'. If these are omitted, the next few characters printed will be taken as the missing parameters.

Sumary of VDU codes

decimal	hex	parameters	effect
0	0	0	Does nothing
1	1	1	Send a character to printer only
2	2	0	Switch on printer output
3	3	0	Switch off printer output
4	4	0	Separate text and graphics cursors
5	5	0	Join text and graphiescursors
6	6	0	Enablevntidrivers
7	7	0	Beep
8	8	0	Move cursor backonespace
9	9	0	Move cursor forwardonespace
10	&A	0	Move cursor down one line
11	&B	0	Move cursor up one line
12	&C	0	CLS (clear text screen)
13	&D	0	Move cursor to start of current line
14	&E	0	Page mode on
15	&F	0	Page mode off
16	&10	0	CLG (clear graphics screen)
17	&11	1	COLOUR c
18	&12	2	GCOL l,c
19	&13	5	New actual colour for colour number
20	&14	0	Restore default actual colours
21	&15	0	Disable VDU drivers
22	&16	1	MODE m
23	&17	9	Create user-defined character shape
24	&18	8	Define graphics window
25	&19	5	PLOT k,x,y (2 bytes for x, 2 for y)
26	&1A	0	Restore default windows
27	&1B	0	Does nothing
28	&1C	4	Define text window
29	&1D	4	Define graphics origin
30	&1E	0	Hove text cursor to top left
31	&1F	2	WINE x,y
127	&7F	0	Backspace and delete

These codes can also be sent frcan the keyboard by typing a CONTROL character - hold down the CTRL key and type the character. For example, codes 1 to 26 correpond to CONTROL-A to CONTROL-Z.

Teletext control codes

In Teletext mode (MODE 7), a number of special effects can by switched on and off by displaying special control codes. Remember that one of these control codes appears as a space

on the screen and that its effect lasts only tor the current screen line.

Teletext control codes for MODE 7

code	controls	effect
129	colour	text characters in red
130	colour	text characters in green
131	colour	text characters in yellow
132	colour	text characters in blue
133	colour	text characters in magenta
134	colour	text characters in cyan
135	colour	text characters in white
136	flash	set flashing on current line
137	flash	clear flashing on current line
140	char. ht.	single height characters
141	char. ht.	double height characters
145	graphics	graphics characters in red
146	graphics	graphics characters in green
147	graphics	graphics characters in yellow
148	graphics	graphics characters in blue
149	graphics	graphics characters in magenta
150	graphics	graphics characters in cyan
151	graphics	graphics characters in white
152	special	supress display (hide)
153	special	normal graphics (not separated)
154	special	separated graphics
156	colour	reset background colour to black
157	colour	background colour = current foreground

Teletext graphics characters

The Teletext (MODE 7) graphics characters consist of 2x3 patterns of foreground and background colour. There are two numeric codes for each of the graphics character shapes. After a line of text has been switched to graphics mode by one of the graphics codes in the previous table, the ASCII characters with codes 32 to 63 and 95 to 126 are displayed as graphics characters. (The codes from 64 to 94 are displayed as normal ASCII characters, i.e. numeric digits and capital letters.)

These ASCII codes provide a convenient way of printing a string of graphics characters. A PRINT statement in a program can contain a string of the corresponding ASCII characters, and, providing an appropriate code precedes them on the output line, they will be displayed as graphics characters.

The graphics shapes that replace the normal ASCII characters are given in the next table. Note that there is no lower code for a solid block of foreground colour.

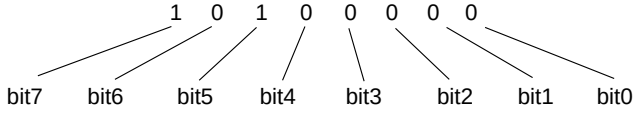
Graphics characters that replace the normal ASCII characters

decimal code	hex code	ASCII char.	graphics char.	decimal code	hex code	ASCII char.	graphics char.
32	&20	space		95	&5F	_	
33	&21	!		96	&60	`	
34	&22	"		97	&61	a	
35	&23	#		98	&62	b	
36	&24	$		99	&63	c	
37	&25	%		100	&64	d	
38	&26	&		101	&65	e	
39	&27	'		102	&66	f	
40	&28	(		103	&67	g	
41	&29	)		104	&68	h	
42	&2A	*		105	&69	i	
43	&2B	+		106	&6A	j	
44	&2C	,		107	&6B	k	
45	&2D	-		108	&6C	l	
46	&2E	.		109	&6D	m	
47	&2F	/		110	&6E	n	
48	&30	0		111	&6F	o	
49	&31	1		112	&70	p	
50	&32	2		113	&71	q	
51	&33	3		114	&72	r	
52	&34	4		115	&73	s	
53	&35	5		116	&74	t	
54	&36	6		117	&75	u	
55	&37	7		118	&76	v	
56	&38	8		119	&77	w	
57	&39	9		120	&78	x	
58	&3A	:		121	&79	y	
59	&3B	;		122	&7A	z	
60	&3C	<		123	&7B	{	
61	&3D	=		124	&7C	|	
62	&3E	>		125	&7D	}	
63	&3F	?		126	&7E	~	

The other codes for graphics characters are 160 to 191 and 224 to 255. To print graphics characters using these codes, the VDU statement can be used, or CHR$ can be used to construct a string containing graphics codes. The advantage of these higher codes is that the order in which the codes correspond to the graphics shapes is more systematic. A program (or programmer) can more easily calculate a code for a given shape. We label the six cells in a graphics character as follows:

bit0	bit1
bit2	bit3
bit4	bit6

These numberings correspond to the bits in the one byte character code as follows:



In the higher codes for graphics characters, bit5 and bit7 we always set to 1. The remaining bits are set to 1 for foreground colour and too for background colour in the corresponding cell. Thus, given the bit values that specify a shape, the code for the required character can be calculated by

```
bit0 + bit1*2 + bit2*4  + bit3*8 + bit4*16 + bit6*64
                        + 32 + 128
```

There is no such simple expression for calculating the lower codes.

Appendix 4 Matrix notation and multlplication

In Chapter 3 we have made use of matrix notation in linear transforms. We say that a point (x,y) transforms to a point (xt,yt):

```
xt = ax + by
yt = cx + dy
```

Given that all our transformations are of this form we can say that the transform T can be represented by the matrix:

```
[ a  d
  b  d ]
```

Now using matrix notation to represent the above operation we rewrite the equations in the form:

```
(xt, yt) = (x, y) [ a  c
                    b  d ]
```

On the right hand side we are multiplying a row matrix (representing a single point in two-dimensional space) by a 2x2 matrix. The equation specified in the matrix notation is identical in every respect to the non-matrix form of the equation. To obtain xt from the matrix formwemultiply the row matrix (x,y) by the first column:

```
xt    = (x, y)   [ a  .
                   b  .  ]
      = ax + by
```

and to obtain yt from the matrix form we multiply the row vector by the second column:

```
yt    = (x, y)   [ .  c
                   .  d  ]
= cx + dy
```

The other context in which we used matrix multiplication was to concatenate transforms together.

```
T  =   T1*T2
   =   [ a c    [ e g
         b d ]    f  h ]
```

- [(ae + cf) (ag + ch)
  (be + df) (bg + dh)]

- [p r
  q s]

p is formed by taking the sum of the products of the entries in the first row of T1 with the first column in T2. q is formed by taking the sum of the products of the entries in the second row in T1 with the first column in T2. Inspecting the other two entries r and s will show how these are similarly derived. In the general case:

C = A*B

each entry Cij of the product is the sum of the products of the entries of the ith row of A with the corresponding entries of the jth column of B. We could easily write a procedure to multiply two 3x3 matrices together and this follows. In Chapter 3 we multiplied matrices together manually.

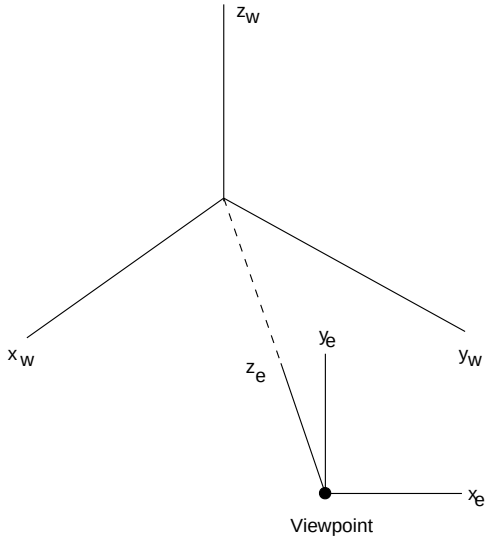
```
100  DEF PR0Cmatmult
110    FOR i = 1 TO 3
120      FOR j = 1 TO 3
130        INPUT A(i,j)
140      NEXT j
150    NEXT i

160  FOR i = 1 TO 3
170    FOR j = 1 TO 3
180      INPUT B(i,j)
190    NEXT i
200  NEXT j

210  FOR i = 1 TO 3
220    FOR j = 1 TO 3
230      sum = 0
240      FOR k = 1 TO 3
250        sum = sum + A(i,k)*B(k,j)
260      NEXT k
270      C(i,j) = sum
280      NEXT j
290    NEXT i
300  ENDPROC
```

Here we have used the usual convention when handling matrices - the first subscript is the row number, the second subscript is the column number (not to be confused with the convention for handling screen coordinates). Note that the each matrix must be typed in row-wise.

Appendix 5 **The viewing transformation**

The viewing transformation, V, transforms points in the world coordinate system into the eye coordinate system:

(xe, ye, ze, 1) = (xw, yw, zw, 1)V

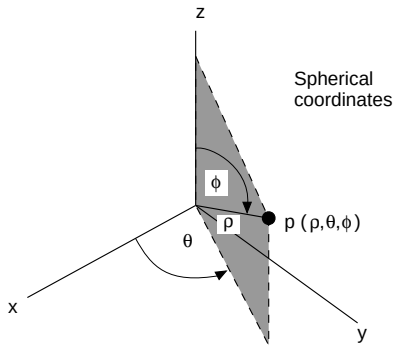


A viewpoint is given as a set of three coordinates specifying the viewpoint in the world coordinate system. An object described in the world coordinate system is viewed from this point along a certain direction. In the eye coordinate system, the z-axis points towards the world system origin and the x-axis is parallel to the x-y plane of the world system. It is standard to adopt a left-handed convention for the eye coordinate system. In the eye coordinate system the x and y-axes match the axes of the display system and the ze direction is away from the viewpoint (into the display screen). World coordinates are normally right handed systems so that in the computation of a net transformation matrix for the viewing transformation we would include a conversion to a left-handed system.

We can now specify the net transformation matrix as a series of translations and rotations that take us from the

world coordinate system into the eye coordinate system, given a particular viewpoint. These steps will be given as separate transformation matrices and the net transformation matrix resulting from the product will simply be stated. If you are unhappy with the derivation you can of course skip it and accept the final result - the net transformation matrix required for a viewing transformation.

Now the viewing transformation is best specified using spherical instead of cartesian coordinates. We specify a viewpoint in spherical coordinates by giving a distance from the origin (rho) and two angles (theta and phi).



These are related to the viewpoint's cartesian coordinates as follows:

$Tx = \rho \sin\phi \cos\theta$
$Ty = \rho \sin\phi \sin\theta$
$Tz = \rho \cos\phi$

Another fact we require in this derivation is that to change the origin of a system from (0, 0, 0, 1) to (Tx, Ty, Tz, 1) we use te transformation:

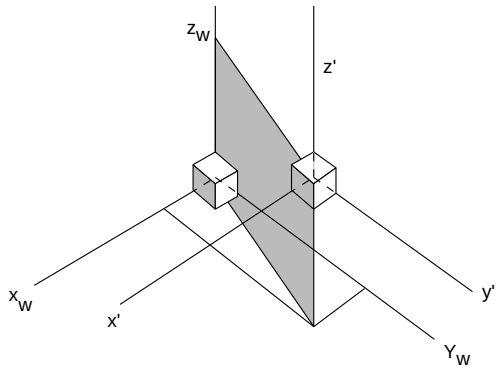
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Tx & -Ty & -Tz & 1 \end{bmatrix}$$

Note that this is the inverse of the transformation that would take a point from (0, 0, 0, 1) to (Tx, Ty, Tz, 1).

The four transformations required to take the object from a world ccxzrdinate system into an eye coordinate system are:

(1) Translate the world coordinate system to (Tx, Ty, Tz), the position of the viewpoint. All three axes remain

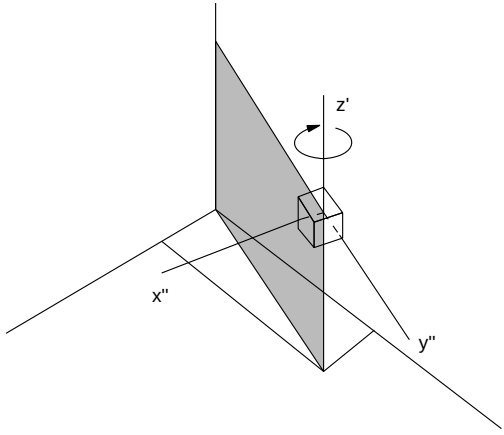
parallel to their counterparts in the world system.



The cube in the diagram is not an object that is being transformed, but is intended to enhance an interpretation of the axes. Using spherical coordinate values for Tx, Ty, and Tz the transformation is:

$$T1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\rho\cos\theta\sin\phi & -\rho\sin\theta\sin\phi & -\rho\cos\phi & 1 \end{bmatrix}$$

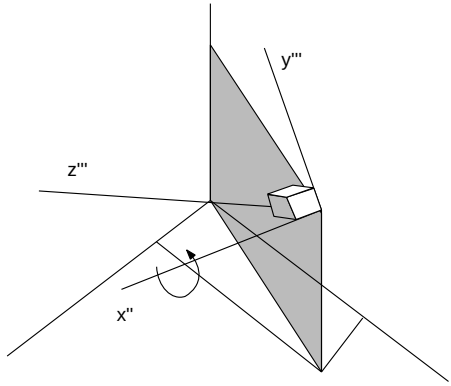
(2) The next step is to rotate the coordinate system through (90 degrees - theta) in a clockwise direction about the z'-axis. The rotation matrices defined in Chapter 3 were for counter-clockwise rotation relative to a coordinate system. The transformation matrix for a clockwise rotation of the coordinate system is the same as that for a counter-clockwise rotation of a point relative to the coordinate system. The x''-axis is now normal to the plane containing rho.



$$T2 = \begin{bmatrix} \sin\rho & \cos\theta & 0 & 0 \\ -\cos\theta & \sin\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(3) The next step is to rotate the coordinate system (180 degrees - phi) counter-clockwise about the x'-axis. This makes the z'''-axis pass through the origin of the world coordinate system.

$$T3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & -\cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(4) Finally we convert to a left-handed system as described bove.

$$T4 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying these together gives the net transformation matrix required for the viewing transformation.

$$V = T1*T2*T3*T4 = \begin{bmatrix} -\sin\theta & -\cos\phi & -\cos\theta\sin\phi & 0 \\ \cos\theta & -\sin\theta\cos\phi & -\sin\theta\sin\phi & 0 \\ 0 & \sin\phi & -\cos\theta & 0 \\ 0 & 0 & r & 1 \end{bmatrix}$$

where

$$(xe\ \ ye\ \ ze\ \ 1) = (xw\ \ yw\ \ zw\ \ 1)*V$$