Jello and the Art of 3D

Mark Moxon introduces the first of a short series on 3D graphics that will turn your mind to jelly.

3D graphics is one of those subjects that people get hooked on, like politics and soap operas. I m hooked, and in this short series I hope to show that the basic techniques used to display objects in 3D are not as difficult to grasp as many think. The subject of 3D graphics was touched on back in RISC User Volume 1 Issues 5, 6 and 7, and I hope to go beyond the examples given there to show how even more can be achieved.

I ll also present a program that shows just how much fun the subject can be; and do remember that 3D graphics is fun, even to people who suffer from technofear. I shan t go into great detail about how the program has been written, but instead I ll describe the methods used in more general terms, and for those of you who really want to know how these ideas have actually been implemented, the magazine disc will contain fully annotated listings with copious comments to explain more of what is going on. Unfortunately the comments are far too copious to fit in the listing in the magazine, so the programs here are somewhat terse, but the fuller versions are well worth obtaining if you want to know more.

THE EXAMPLE PROGRAM: JELLO

The example program given here is the first part of the whole Jello application. This first part displays a correctly shaded three dimensional icosahedron which can be rotated with the mouse; an icosahedron is the shape that revolves on the screen at the start of Neighbours, for all you soap freaks (see figure 1). I m going to call this shape a Jello, because by the end of the series I ll have added code to make this shape bounce around inside a box, and wobble as if it was made of jelly (which is quite a sight, I can tell you!). The program is entirely in assembly language, which is necessary to achieve the speed needed to display the shapes in real time.

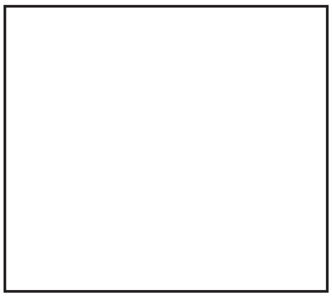
Figure 1. Hello Jello

To enter the program, you should proceed as follows. First, create a directory called !Jello, and inside that (double-click on !Jello while holding down Shift) create a directory called SourceCode. Now create an Obey file called !Run inside the !Jello directory containing the following lines:

```
WimpSlot -min 64K
Run <Obey$Dir>.!RunImage %*0
```

and then create another Obey file called !Compile within the SourceCode directory containing the following:

```
Dir <Obey$Dir>
Run <Obey$Dir>.JelloMain
```

Next, type in the listings given here (eight in all) and save them in the SourceCode directory under the names given in their first lines. Take care with the line numbers for JelloMain, as lines will be added to this program in the next two parts. Then set the currently selected directory to SourceCode and double-click on the

CreateData program to create a file called MathsData. Finally, double-click on !Compile to compile the code, which is saved as !RunImage in the !Jello directory. This method of assembling applications was described in the article Managing Assembler in RISC User 5:9, and it allows the rather large source code to be split up into lots of manageable parts.

It would be prudent to say a word about trapping typing errors in this system. If you have made a typing error in one of the programs passed to the LIBRARY command at the start of JelloMain (the so-called library programs), then the line number given in the error will be meaningless (and will, in fact, always be 2340, the last line in JelloMain). To help with error checking, change line 1080 in JelloMain to:

```
1080 FOR pass%=4 TO 7 STEP 3
```

and an assembly listing will be shown during compilation. Then you can put a PRINT statement at the start of each function in each library program to print out the name of the routine, e.g. in ClearScrn you could put:

```
35 PRINT "**** ClearScrn ****"
```

and from the assembly listing it should be easy to track down which library routine has the error, and which line it is in. Note the line number reported will still be wrong, but the information displayed should be enough to find the error.

When you ve managed to compile the program successfully, try double-clicking on the Jello application. You should get a red Jello in the centre of the screen that you can rotate with the mouse. Magic!

PROGRAM OVERVIEW

Although I shan t go into great detail about the intricacies of the Jello program quite yet, it is worth pointing out the functions of the various parts of the code. The program has been split into a number of parts to ease the writing process and to clarify the overall structure. The eight programs presented in this first part have the following functions:

JelloMain - forms the main part of the application. It contains the co-ordinate data for the Jello (lines 670 to 890), palette data (lines 1020 to 1040), the assembler locations used to store variables (lines 1110 to 1810), the initialisation code (lines 1850 to 2110) and the main loop (lines 2120 to 2340).

CreateData - creates a file called MathsData which contains tables of values for calculating trigonometric functions. Simply looking up a value in a table is far quicker than calculating a value in assembler, and speed is of the essence in 3D graphics.

Adr - contains a macro to avoid problems when using the assembler ADR directive over a large range (as described in Hints and Tips, Volume 5 Issue 4).

```
 10 REM        >JelloMain
 20 REM Program    Jello Source Program
 30 REM Version    A 1.18
 40 REM Author     Mark Moxon
 50 REM RISC User October 1992
 60 REM Program    Subject to Copyright
 70 REM            Not Public Domain
 80 :
 90 :
100 ON ERROR REPORT:PRINT" at line ";E
RL:END
110 :
120 LIBRARY "Adr"
130 LIBRARY "ClearScrn"
140 LIBRARY "DrawJello"
150 LIBRARY "RotateJell"
160 LIBRARY "Rotation"
170 LIBRARY "ScreenJell"
180 :
 300 zdist%=512*256
 330 PRINT "Please wait...compiling"
 340 PROCassemble
 350 PRINT "Please wait...initialising
data":PROCinitialise
 360 OSCLI("Save ^.!RunImage "+STR$~cod
e%+" "+STR$~(recip+off%+12780*4))
 370 *SetType ^.!RunImage Absolute
 380 END
 390 :
 400 DEFPROCinitialise
 410 off%=code%-&8000
 420 OSCLI("Load MathsData "+STR$~(sin+
off%))
 430 FOR I%=0 TO 11:READ x,y,z
 440 xj!(off%+I%*4)=INT(x*256)
 450 yj!(off%+I%*4)=INT(y*256)
 460 zj!(off%+I%*4)=INT(z*256+zdist%)
 470 NEXT I%
 480 FOR I%=0 TO 19:FOR J%=0 TO 3
 490 READ jsurf?(off%+I%*4+J%)
 500 NEXT J%:NEXT I%
 580 FOR I%=0 TO 15:READ rgb
 590 IF I%=0 THEN r=0:g=0:b=0
 600 IF I%=1 THEN r=&FF:g=&FF:b=&FF
 610 IF I%>1 THEN r=rgb:g=0:b=0
 620 rcol!(off%+I%*4)=r
 630 gcol!(off%+I%*4)=g
 640 bcol!(off%+I%*4)=b
 650 NEXT I%:ENDPROC
 660 :
 670 DATA    0, 12.3, 25.2
 680 DATA  24.3, 12.3,  6.5
 690 DATA  15.0, 12.3,-20.2
 700 DATA -15.0, 12.3,-20.2
 710 DATA -24.3, 12.3,  6.5
 720 DATA    0, 28.0,    0
 730 DATA  15.0,-12.3, 20.2
 740 DATA  24.3,-12.3, -6.5
 750 DATA    0,-12.3,-25.2
 760 DATA -24.3,-12.3, -6.5
 770 DATA -15.0,-12.3, 20.2
 780 DATA    0,-28.0,    0
 790 :
 800 DATA 0, 1, 5, 5, 1, 2, 5, 5
```

```
 810 DATA 2, 3, 5, 5, 3, 4, 5, 5
 820 DATA 4, 0, 5, 5, 1, 0, 6, 6
 830 DATA 6, 7, 1, 1, 2, 1, 7, 7
 840 DATA 7, 8, 2, 2, 3, 2, 8, 8
 850 DATA 8, 9, 3, 3, 4, 3, 9, 9
 860 DATA 9,10, 4, 4, 0, 4,10,10
 870 DATA 10, 6, 0, 0, 7, 6,11,11
 880 DATA 8, 7,11,11, 9, 8,11,11
 890 DATA 10, 9,11,11, 6,10,11,11
 900 :
1020 DATA &00,&00,&66,&77,&88,&88,&99
1030 DATA &99,&AA,&AA,&BB,&BB,&CC,&DD
1040 DATA &EE,&FF
1050 :
1060 DEF PROCassemble
1070 vdu=&100:DIM code% &10000
1080 FOR pass%=4 TO 6 STEP 2
1090 P%=&8000:O%=code%
1100 [OPT pass%:FNmain:FNcls
1110 .main1P EQUB 1:EQUW -1:EQUW -1
1120 EQUW 361:EQUW 361:ALIGN
1130 .main2P EQUD 149:EQUD 148:EQUD -1
1140 EQUW 0:EQUB 0:.mouse EQUB 3
1150 .mousex EQUW 0:.mousey EQUW 0
1160 .bank  EQUD 1:.bank1 EQUD 0
1170 .bank2 EQUD 0
1250 .xrot  EQUD 0:.yrot EQUD 0
1310 FNrotate_jello
1380 .xt    EQUS STRING$(12*4,CHR$0)
1390 .cenxt EQUD 0
1400 .yt    EQUS STRING$(12*4,CHR$0)
1410 .cenyt EQUD 0
1420 .zt    EQUS STRING$(12*4,CHR$0)
1430 .cenzt EQUD zdist%
1440 .xj    EQUS STRING$(12*4,CHR$0)
1450 .cenx  EQUD 0
1460 .yj    EQUS STRING$(12*4,CHR$0)
1470 .ceny  EQUD 0
1480 .zj    EQUS STRING$(12*4,CHR$0)
1490 .cenz  EQUD zdist%
1500 .xjs   EQUS STRING$(12*4,CHR$0)
1510 .yjs   EQUS STRING$(12*4,CHR$0)
1520 .xjs5  EQUS STRING$(12*4,CHR$0)
1530 .yjs5  EQUS STRING$(12*4,CHR$0)
1560 .rcol  EQUS STRING$(16*4,CHR$0)
```

```
1570 .gcol  EQUS STRING$(16*4,CHR$0)
1580 .bcol  EQUS STRING$(16*4,CHR$0)
1590 .jsurf EQUS STRING$(20*4,CHR$0)
1610 .draw:STMFD R13!,{R14}
1640 FNscreen_jello:FNdraw_jello
1660 LDMFD R13!,{PC}
1670        EQUS STRING$(255,CHR$0)
1680        EQUS STRING$(255,CHR$0)
1690        EQUS STRING$(255,CHR$0)
1700        EQUS STRING$(255,CHR$0)
1710 .stack EQUD 0
1720 .sin   EQUS STRING$(250,CHR$0)
1730        EQUS STRING$(110,CHR$0)
1740 .cos   EQUS STRING$(250,CHR$0)
1750        EQUS STRING$(250,CHR$0)
1760        EQUS STRING$(250,CHR$0)
1770        EQUS STRING$(250,CHR$0)
1780        EQUS STRING$(250,CHR$0)
1790        EQUS STRING$(250,CHR$0)
1800        EQUS STRING$(250,CHR$0)
1810      ALIGN:.recip:]
1820 NEXT pass%:ENDPROC
1830 :
1840 DEF FNmain
1850 [OPT pass%:.main:FNadr(13,stack)
1860 STMFD R13!,{R14}:SWI vdu+22
1870 SWI vdu+137:MOV R2,#0
1880 FNadr(3,rcol):FNadr(4,gcol)
1890 FNadr(5,bcol):.main1:SWI vdu+19
1900 MOV R0,R2:SWI "OS_WriteC"
1910 SWI vdu+16:LDR R0,[R3,R2,LSL#2]
1920 SWI "OS_WriteC"
1930 LDR R0,[R4,R2,LSL#2]
1940 SWI "OS_WriteC"
1950 LDR R0,[R5,R2,LSL#2]
1960 SWI "OS_WriteC":ADD R2,R2,#1
1970 CMP R2,#15:BLE main1:MOV R0,#112
1980 MOV R1,#1:SWI "OS_Byte"
1990 MOV R0,#113:MOV R1,#2
2000 SWI "OS_Byte":ADR R0,main2P
2010 ADR R1,bank1
2020 SWI "OS_ReadVduVariables"
2030 SWI vdu+23:SWI vdu+1:SWI vdu+0
2040 SWI vdu+0:SWI vdu+0:SWI vdu+0
2050 SWI vdu+0:SWI vdu+0:SWI vdu+0
2060 SWI vdu+0:SWI vdu+29
```

```
2070 SWI vdu+(640 MOD &100)
2080 SWI vdu+(640 DIV &100)
2090 SWI vdu+(512 MOD &100)
2100 SWI vdu+(512 DIV &100):MOV R0,#21
2110 ADR R1,main1P:SWI "OS_Word":.main2
2120 SWI "OS_ReadEscapeState":BCS main3
2130 SWI "OS_Mouse":CMP R0,#0
2140 MOVLT R0,#360:CMPGT R0,#360
```

```
 10 REM >CreateData
 20 :
 30 SYS "Hourglass_On":DIM data% 55000
 40 P%=data%:[OPT 2
 50 .sin EQUS STRING$(250,CHR$0)
 60      EQUS STRING$(110,CHR$0)
 70 .cos EQUS STRING$(250,CHR$0)
 80      EQUS STRING$(250,CHR$0)
 90      EQUS STRING$(250,CHR$0)
100      EQUS STRING$(250,CHR$0)
110      EQUS STRING$(250,CHR$0)
120      EQUS STRING$(250,CHR$0)
130      EQUS STRING$( 10,CHR$0)
140    ALIGN:.recip:]
150 FOR I%=0 TO 699
160 sin!(I%*4)=INT(SIN(RAD(I%))*(2^8))
170 NEXT I%
180 FOR I%=1 TO 12799
190 recip!(I%*4)=INT((2^18)/I%)
200 NEXT I%
210 OSCLI("Save MathsData "+STR$~sin+"
```

```
 10 REM >Adr
 20 :
 30 DEF FNadr(r,l)
 40 IF l>P% THEN
 50 [OPT pass%
 60 ADD r,PC,#(l-P%-8)MOD&100
 70 ADD r,r,#(l-P%-8)DIV&100<<8:]
 80 ELSE
 90 [OPT pass%
100 SUB r,PC,#(P%+8-l)MOD&100
110 SUB r,r,#(P%+8-l)DIV&100<<8:]
```

```
10 REM >ClearScrn
20 :
30 DEF FNcls
40 [OPT pass%
50 .cls:MOV R2,#0:MOV R3,#0:MOV R4,#0
60 MOV R5,#0:MOV R6,#0:MOV R7,#0
70 MOV R8,#0:MOV R9,#0:.cls1:]
80 FOR loop%=1 TO 32:[OPT pass%
90 STMIA R0!,{R2-R9}:]:NEXT loop%
100 [OPT pass%:CMP R0,R1:BLT cls1
```

```
10 REM >DrawJello
20 :
30 DEF FNdraw_jello
40 [OPT pass%:MOV R14,#0
50 .dj1:FNadr(1,jsurf)
60 LDRB R3,[R1,R14,ASL#2]!
70 LDRB R4,[R1,#1]:LDRB R5,[R1,#2]
80 FNadr(12,xjs5)
90 LDR R6,[R12,R3,ASL#2]
100 LDR R7,[R12,R4,ASL#2]
110 LDR R8,[R12,R5,ASL#2]
120 ADD R12,R12,#yjs5-xjs5
130 LDR R9,[R12,R3,ASL#2]
140 LDR R10,[R12,R4,ASL#2]
150 LDR R11,[R12,R5,ASL#2]
160 SUB R0,R8,R6:SUB R1,R10,R9
170 MUL R1,R0,R1:SUB R0,R7,R6
180 SUB R2,R11,R9:MUL R2,R0,R2
190 SUBS R0,R1,R2:BLE dj2:SWI vdu+18
200 SWI vdu+0:MOV R0,R0,ASR#26
210 CMP R0,#2:MOVLT R0,#2:CMPGT R0,#15
220 MOVGT R0,#15:SWI "OS_WriteC"
230 FNadr(10,xjs):ADD R11,R10,#yjs-xjs
240 LDR R1,[R10,R3,ASL#2]
250 LDR R2,[R11,R3,ASL#2]
260 MOV R0,#4:SWI "OS_Plot"
270 LDR R1,[R10,R4,ASL#2]
280 LDR R2,[R11,R4,ASL#2]
290 MOV R0,#4:SWI "OS_Plot"
300 LDR R1,[R10,R5,ASL#2]
```

```
10 REM >RotateJell
20 :
```

```
30 DEF FNrotate_jello
40 [OPT pass%:.rotate_jello
50 STMFD R13!,{R14}:FNadr(1,xj)
60 FNadr(2,xt):LDR R3,xrot
70 RSB R3,R3,#360:LDR R4,yrot
80 RSB R4,R4,#360:MOV R0,#12
90 .rj1:LDR R6,[R1,#yj-xj]
100 LDR R7,[R1,#zj-xj]:LDR R5,[R1],#4
110 SUB R7,R7,#zdist%
120 FNrot(8,9,10,11,12,5,7,4)
130 FNrot(8,9,10,11,12,6,7,3)
140 ADD R7,R7,#zdist%
150 STR R6,[R2,#yt-xt]
```

```
10 REM >Rotation
20 :
30 DEF FNrot(s,d1,d2,d3,d4,x,y,a)
40 [OPT pass%:FNadr(s,sin)
50 LDR d1,[s,a,LSL#2]!
60 LDR d2,[s,#360]:MUL d3,x,d2
70 MUL d4,y,d1:MUL d2,y,d2
80 MUL d1,x,d1:MOV d3,d3,ASR#8
90 ADD x,d3,d4,ASR#8:MOV d2,d2,ASR#8
```

```
10 REM >ScreenJell
20 :
30 DEF FNscreen_jello
40 [OPT pass%:MOV R0,#0:FNadr(5,xt)
50 FNadr(6,recip)
60 .sj1:LDR R2,[R5,#yt-xt]
70 LDR R3,[R5,#zt-xt]:LDR R1,[R5],#4
80 BIC R3,R3,#&30
90 LDR R4,[R6,R3,LSR#4]:FNadr(7,xjs)
100 MUL R9,R1,R4:MOV R9,R9,ASR#13
110 STR R9,[R7,R0,ASL#2]!:MUL R9,R2,R4
120 MOV R9,R9,ASR#13
130 STR R9,[R7,#yjs-xjs]:LDR R9,cenzt
140 SUB R3,R3,R9:ADD R3,R3,#256*256
150 BIC R3,R3,#&30
160 LDR R4,[R6,R3,LSR#4]:FNadr(8,xjs5)
170 MUL R9,R1,R4:MOV R9,R9,ASR#6
180 STR R9,[R8,R0,ASL#2]!:MUL R9,R2,R4
190 MOV R9,R9,ASR#6
160 STR R7,[R2,#zt-xt]:STR R5,[R2],#4
170 SUBS R0,R0,#1:BGE rj1
180 LDMFD R13!,{PC}:]:=""
```