

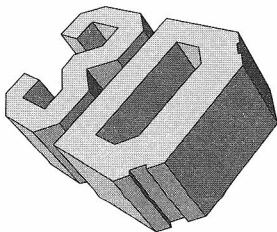
# AMOS



USER GUIDE

MANDARIN

# AMOS



by



## Voodoo Software

© 1991 Europress Software Ltd.

Programming and design

Anthony Wilkes

Richard Lewis

Manual Authors

Anthony Wilkes

Richard Lewis

Project Manager

Richard Vanner

Demos and game

Nick Harper

Write to Europress Software for help with defective discs or other initial problems:  
Customer Services, Europress Software Ltd, Europa House, Adlington Park,  
Macclesfield SK10 4NP.

**MANDARIN**

Mandarin is the entertainment  
label for Europress Software Ltd

**EUROPRESS**  
SOFTWARE

Manual typeset by *Visual Eyes* Stockport

No material may be reproduced in whole or part without written permission from Europress Software. While every care has been taken to ensure this product is correct, the publishers cannot be held legally responsible for any errors or omissions in the manual or software. If you do find any, please tell us!

# Super-charge your creations!

Now you can make your AMOS programs run like lightning with **AMOS Compiler**. Typically most programs will **double in speed** – some commands are more than **5 times faster**. These spectacular results will have everyone thinking you've programmed in machine code!

In addition, AMOS Compiler comes with an incredible compactor which will squash the size of your programs by up to 80% (**60% compression on average**) – and in a matter of seconds! This means that your programs take up less disc space – and they load faster too.

With the incredible speed increases and program compaction the AMOS Compiler will give your programs that professional edge.

## AMOS Compiler at work on the AMOS Sprite Editor:

- It compiles in less than two minutes – or just 14 seconds if you have a hard disc!
- Compresses in 8 seconds from 147k to 86k – that's 58.5% compression... and it runs much faster too!

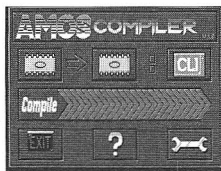
## Inside the Compiler box:

AMOS Compiler, AMOS language updater, AMOS Assembler and eight demonstration programs which show off the power of the Compiler. The comprehensive and easy-to-use manual will provide you with all the assistance you'll need to develop lightning-fast software of your dreams.

*If you're amazed by AMOS you'll be astounded by AMOS Compiler.*

*AMOS Compiler – £29.99 from all good software retailers*

## AMOS Compiler in use



Compiling your programs is simplicity itself: Click on **COMPILE** then select source and destination – that's all there is to it!



The welcome disc will enable you to activate the AMOS Compiler and the AMOS updater in seconds.

*"I truly believe you can write a commercial game in AMOS with the AMOS Compiler – now my programs are running too fast!"*

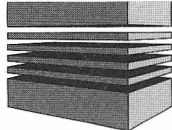
– Amiga Computing

MANDARIN



AMOS Compiler written  
by François Lionet  
© 1991 Mandarin/Jaww.

EUROPRESS  
SOFTWARE

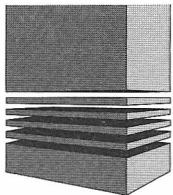


# Contents

<b>1: Welcome</b> .....	<b>1</b>
Making a disc back-up .....	2
<b>2: In the beginning</b> .....	<b>3</b>
<b>3: Quick start</b> .....	<b>5</b>
How to use this manual.....	6
<b>4: Updating to the latest version of AMOS</b> .....	<b>7</b>
<b>5: Installing the 3D extension</b> .....	<b>11</b>
<b>6: The Object Modeller</b> .....	<b>13</b>
Introduction .....	13
Loading the modeller .....	13
Getting to know OM .....	14
The OM screen .....	14
Selecting shelves .....	15
Selecting parts of a block.....	18
Gluing blocks together .....	19
Saving objects.....	21
Customising blocks .....	21
Pulling lines.....	21
Pulling points.....	22
Groups .....	23
Surface detail .....	26
<b>7: The object modeller tools</b> .....	<b>29</b>
Part 1 Primary Commands .....	29
Problem objects .....	39
Part 2 Block Commands.....	41
Surface anchor points .....	43
Pulling rules.....	43
Selecting the sensitivity of the pull tool .....	45
Part 3 Group Commands.....	47
Part 4 Surface Detail.....	56
Attaching a surface to a face.....	57
Positioning the surface .....	58
Attaching a surface to a 2D block .....	59
Re-using surfaces .....	60
Copying surfaces between objects and within objects.....	60
<b>8: 3D Programming</b> .....	<b>63</b>
Part 1 The 3D World.....	63
Space .....	63
The double buffered display.....	66
Angles .....	67
local coordinate system.....	68



	The viewpoint .....	69
	Choosing the best coordinate system .....	69
Part 2	The AMOS commands .....	71
	Positions .....	71
	Angles .....	71
	Objects .....	72
	The display .....	72
	The Redraw Loop .....	72
	Loading and removing objects .....	73
	Invoking objects .....	75
	Object movement commands .....	75
	Reading an object's position .....	77
	Changing the attitude of objects .....	77
	Bearing and range .....	79
	Pointing an object .....	80
	Converting coordinates .....	81
	Object visibility .....	82
	Collision detection .....	82
	Animation .....	84
	Surface animation .....	86
	Backgrounds .....	87
	Memory .....	89
	<b>9: Hints &amp; Tips .....</b>	<b>91</b>
	<b>Appendix A: Copying OM .....</b>	<b>95</b>
	<b>Appendix B: File structure .....</b>	<b>97</b>
	<b>Appendix C: The Utilities .....</b>	<b>99</b>
	<b>Appendix D: AMOS Errors .....</b>	<b>103</b>
	<b>Glossary .....</b>	<b>105</b>



# 1: Welcome to the world of 3D

There is something magic about a 3D game. It is as if there is an infinity of space to explore just behind your monitor screen. With AMOS 3D we have tried to give you the means to get behind the screen and create exciting and imaginative worlds of your own.

AMOS is already by far the most powerful languages for 16-bit graphics and with 3D we are literally adding another dimension.

We hope you will like 3D. It is based on one of the most sophisticated systems yet to be developed and contains many features that are totally new.

Yet 3D is no harder to program than sprites and backgrounds. It uses the familiar AMOS commands specially adapted for 3D and optimised for the best possible performance.

At the heart of 3D is the powerful 3D object modeller (OM). With this fascinating tool you can design 3D objects as complex as those in any 3D game, using a simple set of mouse controls and a fully interactive display. You can even add pictures to the faces of your objects that you can animate under program control.

We have had a fantastic year designing AMOS 3D . We can't wait to see your programs. Now back up your discs and good luck with 3D.

Anthony Wilkes

Richard Lewis

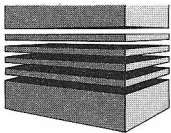
## Making a disc backup

Before proceeding any further, it's vital that you make an immediate back-up copy of the enclosed discs. This will allow you to play around with the package to your heart's content, without the risk of destroying something vital. The duplication procedure for a disc is extremely straightforward.

- 1 Start up your Amiga using your usual Workbench disc.
- 2 Prepare a formatted disc to hold your copy of the disc. Use the Initialise option from the Workbench.
- 3 Take the original disc and slide up the write-protect tab at the top right hand corner. A small hole should now be visible in the corner of the disc. This will stop your Amiga writing to the master disc and will protect it from any unfortunate mistakes during the duplication process.
- 4 Insert the master disc into the Amiga's internal drive and select its icon with the mouse.
- 5 a) If you've got just a single drive, click on the Duplicate disc option from the Workbench menu. You can now copy the disc by simply following the prompts as they are displayed on the screen.  
b) If you've two or more drives, place your blank disc into the external drive. Now drag the icon representing the master disc directly over your target disc icon and release the mouse button. This will initiate an automatic back-up of the disc.
- 6 After the disc has been successfully duplicated, the new copy will be assigned a name such as *Copy of AMOS 3D*. It's vital to change the label back to the original version immediately, as the enclosed software occasionally refers to the various discs by name. Remove the original disc and select the Rename option from the Workbench menu. You can now delete the offending text using the normal cursor keys.

**Note:** If you are uncertain about any of these steps, you are recommended to carefully read through the relevant section in the Amiga Users' manual supplied with your computer. This contains a detailed breakdown of the entire procedure.

**Important:** The Object Modeller disc must be left so that it is read/writable. The system needs to update a small file on the disc every time it is run, so please ensure you don't write protect this disc. If you do, you'll be asked to unprotect the OM disc so that it can continue its load.



## 2: In the beginning

Computer graphics, and especially 3D, has a short history. Although the first computers were developed in the late 40s and early 50s, notably at Cambridge and Manchester in Europe and at MIT in the USA, computer graphics had to wait until the 60s and the pioneering DEC 340 display.

By the standards of the early pioneers, graphics of any sort required very powerful machines and very large memories. The memory-mapped raster display, the type used exclusively in modern personal computers and workstations, took even longer. In the early days no-one could afford (or even address) enough memory for a modern bitmap and the earliest examples required a room full of magnetic store.

From very early on it had been recognised that, given enough power and storage, computers could be programmed to model the laws of optics and generate perspective views of objects in a simulated 3D space. The 3D line drawn cube rotating in real time with hidden lines suppressed, exhibited at an early MIT conference, was an impressive sight. But it was a long haul from there to the flight simulator displays of the late 70s with fully shaded landscapes and thousands of polygons refreshed at 30 frames a second.

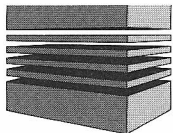
During the 80s as cheap personal computers entered the High Street, the world of computer graphics became accessible to a much wider public. The early 8-bit, and even today's 16-bit machines were no match for the special-purpose hardware developed by companies like Evans and Southerland to run commercial simulations. In many ways 3D graphics had to be redeveloped from scratch. In the main stream of commercial 3D development hardware and software development go hand in hand. The big 3D engines are based around sophisticated displays with much of the 3D number crunching built into the hardware. On an Amiga or an ST all the work, or nearly all, must be done by the processor.

The best 16-bit 3D software gets results by exploiting every possible shortcut and by using a whole variety of tricks to save processing time. Only cut-throat competition between games programmers has made this possible and it has all happened over the last few years. The developers responsible for the graphics behind the best 3D games have concentrated on one thing above all: Speed.

3D itself is not especially hard. You need high-school maths and a good textbook. The trouble is that the results you would get using traditional methods would be far too slow.

No-one today wants to go and make a cup of coffee while they wait for the next frame. Because of this it's not surprising that the 16-bit 3D of recent years has been written in highly optimised assembler, and not structured so as to be usable by the programming public.

Added to this, the companies who undertook expensive 3D development have been keen to protect their investment by keeping their code under wraps. This is why it is not until now that 3D tools aimed at programmers have generally become available. AMOS 3D aims to change all that. With this package, anybody capable of a little programming can create 3D games and other applications. And even non programmers can design 3D objects.



# 3: Quick start

By the time you're reading this, you'll be raring to go! Don't be too impatient though. You'll need to install the new AMOS 3D commands onto a working **copy** of your AMOS programs disc. The whole process takes about 10 minutes, but thankfully it only needs to be done once. Here's a quick run-down of the general technique.

- Make an immediate back-up of the two discs supplied. If you're unsure about this, full instructions can be found in chapter 1.
- Now place your new copy of the AMOS 3D Installation disc into the internal drive, and boot up your Amiga in the normal way. After a few seconds, the AMOS Installation program will be executed, and you'll be presented with the following screen:

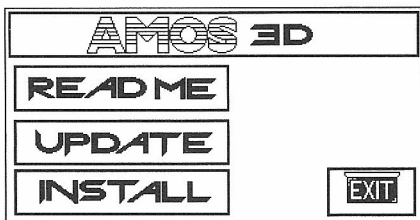


Figure 3.1

## Read me

Displays a complete explanation of the 3D installation along with a full list of any updates or changes to 3D. It's a good idea to read this carefully before continuing. (Use the mouse to move up and down the text file.)

## Update (see next chapter)

Updates your current version of AMOS Basic to the latest version.

**Note:** AMOS 3D will only work with versions of AMOS Basic from 1.3 upwards. So if you're still using an earlier version, it's essential to upgrade immediately using the updater included.

**WARNING: Don't update the original AMOS Programs disc!** The updater will automatically delete any existing .AMOS programs to make room for the new version. Use a back-up copy instead.



## **Install** (see chapter 5)

Installs the 3D extension onto a copy of the AMOS Programs disc. It also installs example programs and objects onto the disc. This will allow you to boot from a single disc with everything installed and ready to go. As mentioned previously, 3D should only be installed on versions of AMOS from 1.3 onwards.

## **Booting the Object Modeller**

To run OM, simply insert the AMOS 3D Object Modeller disc into the internal drive. Switch on the computer and the program will load and run automatically. After a short delay the OM title screen will appear. Hit any key to begin your modelling session.

## **How to use this manual**

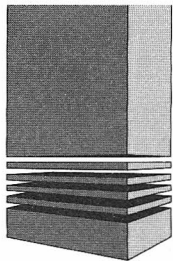
The main documentation for 3D is divided into two parts. First a tutorial and instructions for using the Object Modeller OM (chapters 6 and 7) and then a detailed account of the Td (Three Dee) commands (chapter 8).

A glossary is included to explain any unfamiliar terms.

Whether you are an experienced programmer or not, we suggest that you start with the Object Modeller. Make a few objects and take a look at the examples. This will give you a good idea of the possibilities.

Once you have a feel for 3D objects read Chapter 8 entitled *3D Programming*. This explains 3D coordinates and gives all the background you need to use the Td commands.

Once you have experimented a little with the language extension we suggest that you read Chapter 9 on *Hints and Tips*. It could save you a lot of programming time.



## 4: Updating to the latest version of AMOS

The AMOS 3D commands can only be used with AMOS V1.3 or higher. There's no need to worry if you've got a previous version. We've helpfully included a free update to AMOS along with this product. If you've been using the original AMOS V1.1 all this time, you should notice a number of significant improvements (see the *What's new in AMOS* supplement).

Here's the procedure in detail, for updating to the new version:

- The first job you must do is create a copy of your AMOS programs disc (see *Making a back-up*). Once you've created a new AMOS Programs disc and have tested it works, you'll be ready to update this disc.
- Place the AMOS Installation disc into the internal drive and turn on your Amiga. The 3D installation program will automatically load into memory and the following welcome screen will be displayed:

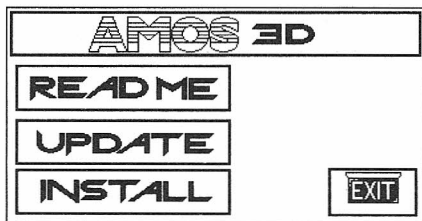


Figure 4.1

- Select the Update button with the left mouse button.
- The AMOS Updater program will be loaded and a new screen will be displayed:

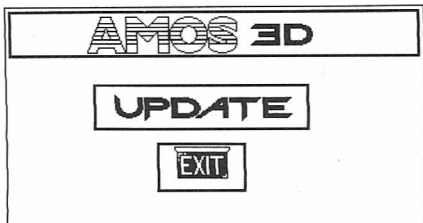


Figure 4.2

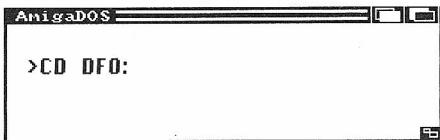
- The process of updating is completely automatic. If you want to update to a hard disc though, you'll have to update a floppy disc first and then copy across the relevant files (see below).
- **WARNING!** The updater will automatically delete any non-essential program files from the backed-up AMOS disc! So it's vital to ensure that you've made a **copy** of the AMOS programs disc and are not using the original AMOS master.
- Start the updater by pressing the left mouse button when the pointer is over the *Update* icon.
- You'll be prompted to insert the copy of the AMOS programs disc (the one you prepared earlier) into any available drive. If you've an external drive, you should keep the Installation disc in Df0: and place the program disc into the external drive. This will avoid the need to do any laborious disc swapping. (For unexpanded A500 users, there are only 12 disc swaps during this process.)

Your AMOS program disc will be updated to the new version of AMOS. Once the update is complete, select the EXIT button.

Before continuing, you are strongly recommended to make an immediate back-up of the new system onto a fresh disc. See *Making a back-up* for more details.

## Installing AMOS onto a hard disc

- Follow the above procedure so that you have a copy of the new AMOS on a floppy disc.
- Boot up your Amiga from your hard disc as normal
- Enter the CLI or SHELL.
- Insert your new copy of AMOS into the internal drive.
- Set the directory to the internal drive:



- You can now type the following line from the CLI prompt (note the space between AMOS1.3 and the filename INSTALL.AMOS. If a version higher than 1.3 is now supplied then change the 1.3 to the correct version number.



- AMOS will load into memory and you'll be presented with the hard disc installation program automatically. This program will be requesting you to select an option from the main menu.
- Pull down the *Installation* menu by pressing the **right** mouse button and highlighting the Installation text.
- From the resulting menu list, move the mouse over the *AMOS Master disc* entry. This will produce another menu list associated with installing files from the AMOS1.3 master to the hard disc.
- Select the *Just Install AMOS* option from this menu.
- When a file selector appears and requests you to select the **drive** you wish AMOS installed onto, first use the right mouse button to change the directory list into a **drive** list.
- From the **drive** list that appears, select the one you want AMOS installed onto by highlighting the drive name and clicking on the **left** mouse button.
- The files on the selected drive will now be listed in the file selector. When you're happy you've selected the right drive, click with the left mouse button on the file selector's **OK** button.
- Now wait for the installation to finish.
- When the installation is complete, automatic amendments to the hard disc startup-sequence can be performed. Unless you know what this means (read below) we strongly advise that you avoid this option and select **QUIT** at this point.

- If you want to append a new assign command to your startup-sequence, then use the file selector to find the startup-sequence you wish to add the line to, select it with the left mouse button and click on **OK**.

Inserting an assign means that every time you boot your hard disc, the use of AMOS: at the start of a filename will inform AmigaDos of where AMOS is located on the hard disc. For example:

**assign AMOS\_DATA: WORK:AMOS\_Discs/Data\_disc**

Will set up AmigaDos so that use of AMOS\_DATA: will always look on the hard disc partition WORK: and inside the folder path AMOS\_Discs/Data\_disc. In other words:

**dir AMOS\_DATA:**

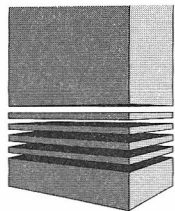
is the same as typing:

**dir WORK:AMOS\_Discs/Data\_disc**

A word of warning though, you must be sure the startup-sequence you select is used by the system on boot up. You must also be sure it doesn't finish its work before it reaches the new assign at the end of the startup-sequence.

If you're unsure about the whole idea simply select Quit as this option is not essential.





## 5: Installing the 3D extension

Now that you have an AMOS programs disc containing the new version, you'll need to install a special extension of commands onto it. This extension adds on new commands to AMOS (similar to the Compactor, Requester and Serial extensions).

- If you haven't already upgraded to the latest version of AMOS, now's the time to do so. Jump back to the previous chapter on *Upgrading to the latest version of AMOS*.

The installation procedure is very straightforward:

- If using floppy discs, place a **copy** of your Installation disc into the internal drive, and re-boot your Amiga. The *Welcome* program will now load into memory.
- Hard disc users should boot from their hard disc, insert the Installation disc into DF0: and double click on the disc icon. When the window has displayed the programs on the disc, double click on the *Welcome* icon.

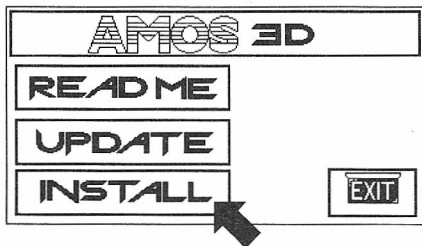


Figure 5.1

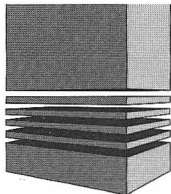
- You are now ready to install the AMOS 3D extension file onto your new AMOS program disc.
- Select the Install icon with the mouse and click once on the left button. This will automatically load a separate Installation program from the disc.



Figure 5.2

- Now click on the Install button.
- Floppy disc users will be asked to insert their new AMOS programs disc into any drive.
- Hard disc users are presented with a file selector, from this you must double click on the AMOS1.3 file (or higher version) located on the hard disc. This then informs the installer where to do the installation.
- Follow the prompts at the base of the screen for a successful install.
- The installer will also create a folder called objects and a number of AMOS programs. You don't need these to use AMOS 3D but they will show you what's possible and are referenced within the manual. For now keep them on the disc until you're ready to create your own objects and programs.

*clean for TaskPrint*



# 6: The Object Modeller

## Introduction

Object modeller (OM) provides all you need to design 3D objects. These objects can be saved onto disc and loaded into your AMOS programs. The objects you create can be just as complex and interesting as those in any 16 bit game. OM will also create objects that can be drawn quickly, both by the modeller itself and by the AMOS Td (Three Dee) commands.

As an extra, to make your objects just that bit unusual OM provides a way of adding pictures to the faces of your designs. Lastly, the disc library of examples provides a wealth of original objects illustrating both the simple and the more advanced aspects of object design.

This chapter is a tutorial style, step-by-step guide to OM which takes you through the steps involved in creating an object and saving it to disc. The following chapter gives a full description of each of the OM controls.

## Loading the Modeller

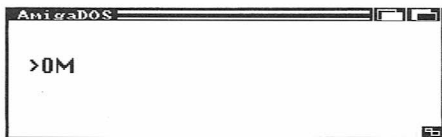
We begin by loading OM. Place the OM disc into the internal drive and reset (or switch on) your computer. After a minute or two the intro screen will appear. Press any key to reveal the OM panel and above it a dark space with a row of small shapes along the top.

## Loading the OM from the Workbench

To run OM, close down any running programs and double click on the OM icon. After a short delay The OM title screen will appear. Hit any key to begin your modelling session.

## Loading OM from the CLI or Shell

To launch OM from the CLI or the shell, make the directory containing OM the current directory, close down any running programs and type



After a short delay The OM title screen will appear. Hit any key to begin your modelling session.

## Memory

OM is a very large program and needs about 480k to operate. So if you're short of memory, load it on its own.

## Related files

OM requires a number of files in addition to the OM program itself. These files are located in the OM directory. When you make copies of OM, make sure you copy the whole directory.

We recommend that you use a separate directory to store your objects and keep the OM directory clear of other files, this will make copying OM easier in some circumstances. As explained more fully in appendix A on Copying OM, each directory that is used to store OM should contain a special file called 'ID' containing a unique two character identifier. Every time you create a new OM directory you should run the supplied program SID and choose a unique identifier when requested. You can change the identifier at any time, simply by running SID again. If you don't do this OM will create an ID file for you but the identifier will not be unique and this may lead to problems when it comes to copying objects.

## Getting to know OM

The best way of getting to know any software is by using it. You may be daunted by the idea of making 3D objects, after all, even professional software houses usually stick to 2D sprites and backgrounds. We think you will be pleasantly surprised by OM and to prove it we will make a complete 3D object right away. The object we are about to make is a 3D version of the letter T.

Designing on object with OM is a little like building using a set of bricks. OM provides a small number of basic shape types which you can stick together to build more complex forms. OM is more powerful than a building set though because each basic shape can be stretched and moulded, sized and mirrored giving you access to building blocks of every conceivable sort. As you will see, blocks can also be decorated using OM's unique surface detail feature.

## The OM screen

The OM screen is divided into two parts. The upper section, the object display area, is your window onto the 3D world. This is where objects are assembled and decorated. The lower portion is devoted to OM's controls and the tools which you use to interact with the display.

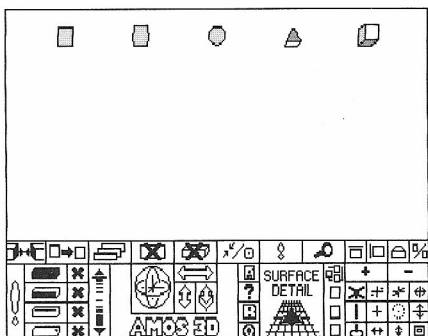


Figure 6.1

## The display area

The display area (in default mode) is divided into 12 *shelves*. Each shelf can hold one object. The top row of five shelves are known as the system shelves and hold the five basic shape types (see below). These basic shapes are known as *blocks* (even though some of them are simply flat surfaces). The next row, the *user shelves*, provide somewhere to put work in progress. They can contain your custom blocks, half finished objects and so on. The remainder of the display is divided into two larger *work shelves*. All the modelling functions operate on these areas.

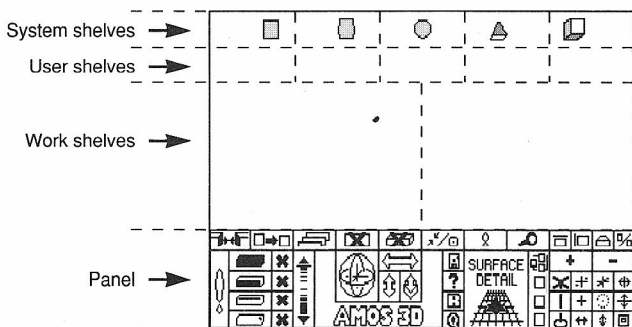


Figure 6.2

## Selecting shelves

One thing that may confuse you a little at first is that initially, except for the system blocks, the display area is completely blank. The shelves themselves are not shown in




any way. There is a reason for this which will become apparent as you get used to OM. To see where the shelves are, move the pointer into the display area, hold the left mouse button down and move the mouse around. You will see the shelves light up as you move through them. Get used to their positions.


Most of the OM tools work on a selected shelf, usually one of the work shelves. You tell OM which shelf (or shelves) to work on by selecting the one you want. To select a shelf simply move the pointer somewhere inside and click the left mouse button. OM draws a box around the shelf to show that it is selected. Most OM commands work on a single shelf, some work on a pair. No more than two shelves can be selected at a time. The work shelves are the only ones used for modelling, the others are simply for storage.

## Using the OM tools

OM provides three types of tool:

### Multi-shelf tools

An example of a multi-shelf tool is the Copy tool. To copy the contents of one shelf to another, click (left button) on the source shelf, click on the destination shelf and finally click on the  icon. Try this now as follows:

First select the right hand system block, the small cube. Now click on the *left hand work shelf* and then the  icon. You should see the cube appear in the work shelf. Notice that it looks larger than before. This is because objects displayed on the work shelves are moved closer so that you can work on them more easily. There is one other multi-shelf tool: The Unite tool. We shall come to this soon.

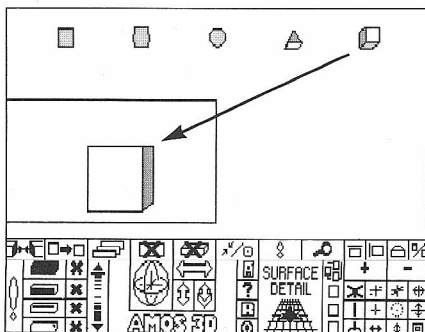





Figure 6.3  
Copying a block

## One-shot tools

These are by far the most common tools. A one-shot tool acts on the selected work shelf. You invoke it by selecting the shelf (if it's not already selected) and clicking on the icon. Zoom is a one-shot tool. Select the work shelf containing the cube and click the  icon. The display now changes to show a close-up view of the cube filling the whole display. Click  again to restore the display.




## Slide tools

A slide tool works just like a one-shot except that you hold the left button down over the icon while you slide the mouse about. The pointer stays stuck to the icon while some aspect of an object changes. The Rotation icon  is a slide tool. Make sure the work shelf containing the cube is still selected and hold the left button down over the rotation icon. As you move the mouse you will see the cube rotate. Spend a little time getting used to the Rotation tool. Try to look at the cube from every angle. This is the most frequently used tool.

## Squashing and stretching

Our next task is to turn our cube into the upright part of the letter T. Before we do this however we must align the cube so that it faces us straight on. There are special ways to do this but for the present the simplest way is to get a fresh copy of the cube from the system shelves, copying the block just as you did before. This time do not rotate the cube.

The next step is to turn the cube into a *group*. Groups are a way of combining several blocks so that one tool can work on them all at the same time. We will say more about groups later but for now we must turn the cube into a group of one block.

With the cube still selected, click with the right button on the  icon. Little dots should appear on all the vertices of the cube. Now we can use the rest of the tools in the group box on the right of the panel, and in particular the stretching tools  and . These are both slide tools and you will find them quite intuitive to use, they stretch the cube in the directions indicated by the arrows. With a little experimentation you should be able to turn the cube into an upright post.

Incidentally, when you use the stretching tools, be very careful about the orientation of the object you are stretching. You will see why if you try stretching blocks that lie at an odd angle.

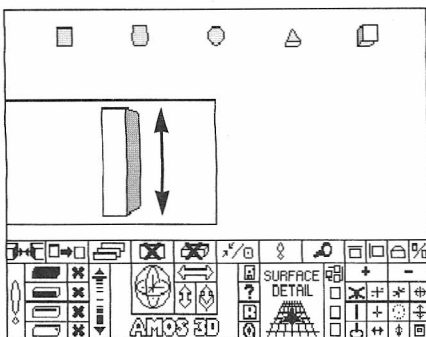


Figure 6.4  
Stretching a block

## Selecting parts of a block

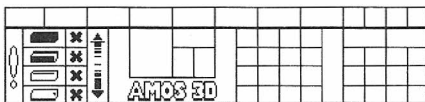


Figure 6.5  
Component selection icons

A block is made up from a number of faces, for example a cube has six. Each face is made up of four lines and every line has two end points or simply *points*. Many of the OM tools work on one or more of these components. See figure 7.2.2


Look in the component selection box in the lower left of the control panel. You will see four pairs of buttons with a long vertical button to either side. The four select buttons show a picture of a block with the different parts highlighted. These are the component selection buttons. Click on the second from the top, the face selector. You will see one of your block's faces highlight with a square (in some cases this square may be transparent).


Click again to select a different face. Try also selecting lines and points (the bottom two selection buttons). The only selection button that will have no effect is the top one, the block selection button. This is usually only used in objects containing more than one block.

Experiment with the selection buttons a little. You will notice that they only select visible parts of the block. To select a hidden part rotate the block so that you can see the part you want. You will find that it is now selectable. The de-selection buttons marked with an X, de-select the corresponding component. The remaining two buttons we will come to later. (For a complete explanation of component selection see Chapter 7.2, block selection tools.)

## Gluing blocks together

To complete our letter T we must add another block to lie across the top of the upright part. Multiple block objects are formed by gluing together the objects in the two work shelves. This can be done for a pair of blocks or for other multi-block objects.

First we select a face on each of the objects (one on each work shelf), then we select the two shelves in turn and click on the  button. The new object appears on the last shelf selected. You will see how this works when we join the two parts of our letter T.

Before we do this however we still have to make our second block. We do this using a neat trick: Copy the upright block to the right hand work shelf and turn it on its side. We now have the block we want and we can go through the *gluing* procedure. We wish to join the top of the tall block to the bottom of the horizontal block. Select the correct face of each block using the selection buttons (don't forget to select the shelf in each case). When both faces are highlighted, select first the right work shelf and then the left. Then click (left button) on the  icon on the far left of the control panel.

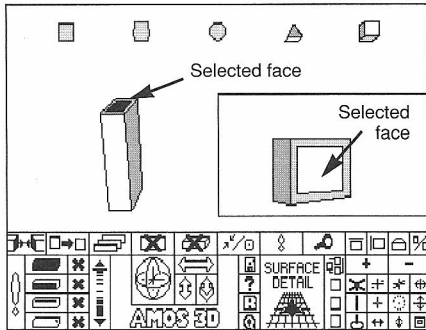

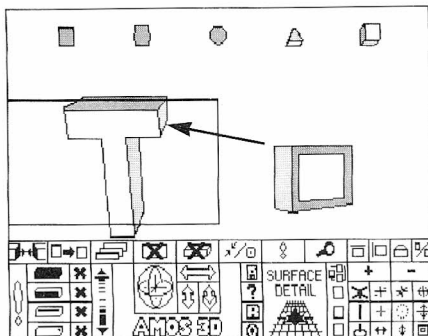




Figure 6.6  
Selecting the faces to be joined

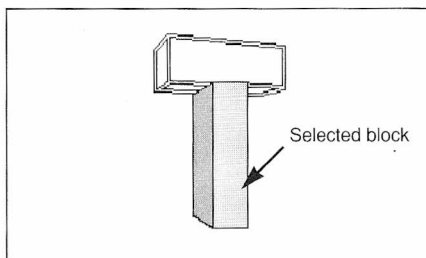
In fact there was no need to turn the top of the T on its side. When you glue objects together, OM automatically handles this for you.

The resulting two block object which appears on the left hand work shelf will probably look a little strange. Move it around with the rotation tool and see if you can tell what is wrong. The reason it looks strange is that OM is not yet drawing the blocks in any particular order. It is not yet taking account of the fact that one of the blocks is behind the other in most views. We tell OM to correct this by clicking on the  tool. After a second or two the blocks will be displayed correctly





**Figure 6.7**  
**Joining the blocks**

Before we save our letter T, we will complete the section on selecting components by trying out the Block Select tool. Make sure the work shelf containing the T is selected and click on the  icon. You will see that one of the two blocks is now a line drawing rather than a solid object; rather like the graphics in old 3D games. The other block, the solid one is the selected block. Now click on the  icon (opposite the block select button) to deselect the block.





**Figure 6.8**  
**Block selected**




Click  again. The situation is reversed. You might wonder why we show selected blocks in this way. The reason is that whatever the attitude of an object, the selected block is always visible; you can see it through the others. This makes editing easier in more complex objects. Actually there is an alternative way of highlighting selected components and we will come to this when we deal with surface detail later on. Now click on the  icon (the one opposite the block select button) to deselect the block.



## Saving objects

If all has gone well you should now have your first 3D object, the letter T like the one in Figure 6.7. To save the object ensure the object is still selected, click on the  button and when the file requester appears, type in a suitable name and click on the  button.

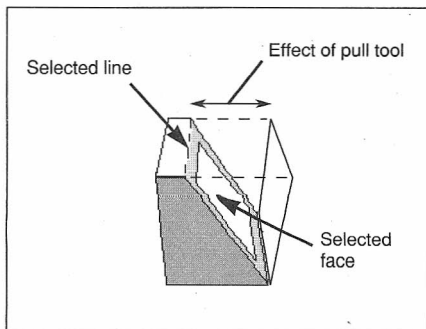
## Customising blocks

The number of interesting objects that you can create with the five basic shapes is of course limited. In this section you will learn to make new basic blocks out of those provided. Before you do this though, take a good look at the system blocks by copying them to a work shelf and moving them around. Look at them in zoom mode and try positioning them by clicking on the ,  and  tools. You will notice that only two of the blocks have depth. The three on the left have no thickness. They are especially useful for such things as the wings of spaceships.

## Pulling lines

We illustrate line pulling using the cube again. Copy this block to the left hand work shelf and select one of the faces. We will now *pull* that line.


The tall button to the right of the selection box is the Pull tool. It is a slide tool so you must keep the left button down over it while you move the mouse around. When you do this, do so gently using the tip of the icon

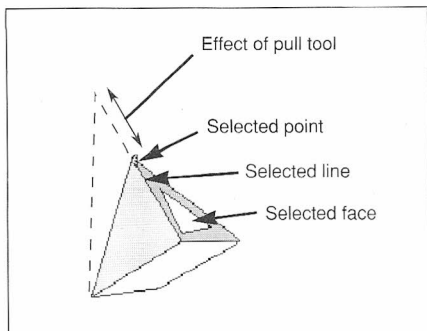


**Figure 6.9**  
**Pulling a line**

the effects can be quite dramatic. The sensitivity of this control is greater in the lower portion of the icon.

## Pulling Points

To illustrate point pulling use the five pointed shape, second from the right on the system shelves. Select one of the three pointed faces and then one of the lines touching the top of the pyramid. Now select the top vertex with the point selector. Try using the  tool again.



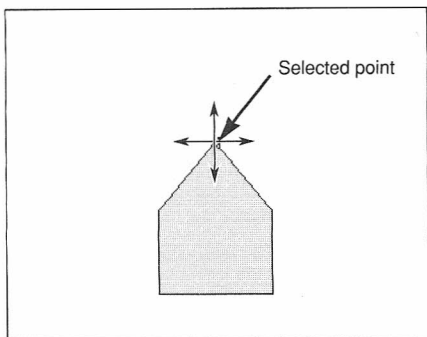
**Figure 6.10**  
**Pulling a point**

After some experimentation you may be confused about the rules for pulling lines and points in the 3D blocks. Here is an explanation:

- You can pull any line.
- You can only pull points in certain cases. OM does not let you pull points that have to be in a certain place to keep the faces flat. Blocks with bent faces are not allowed!
- To pull a line you must have both a face and a line selected (actually, selecting a line automatically selects a face but you will need to select faces to get access to all the lines). When the line moves it moves **in the plane of the face joined to the selected face by the selected line**. This may seem complicated but it's not. The best way to think of it is to imagine that the selected face is a door with the handle on the selected line. Pulling the line is like opening the door.

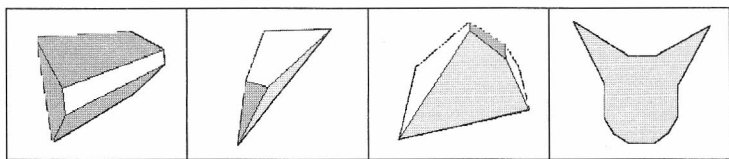
It will probably take you a short time to get used to these rules. With a little thought you can design any convex shape with the same number of faces, lines and points as the original block

Finally, try pulling one of the points in the 5 pointed flat shape on the left system shelf. This is the only way to change the shape of a 2D block. You will find that you can move the selected point anywhere within the plane of the block itself. See figure 6.11.



**Figure 6.11**  
Point pulling on a 2D block

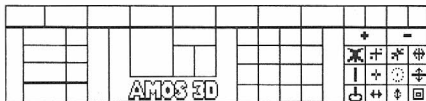
Look at the custom blocks in fig 6.12. See what a wide range of shapes you can make.



**Figure 6.12**  
'Some custom blocks'


## Groups


Earlier, when we were using the stretching tools we touched on the topic of groups. Now we will deal with the subject more fully. The group icons are located on the right of the panel.






**Figure 6.13**  
Group icons



So far we have learnt how to select individual blocks. When we are dealing with more complex objects containing several blocks we will need a way of working on more than one at the same time. Let's make a more complex object, the letter H. Can you see how this can be done?



We use the horizontal block that formed the top of our T again. It is still in the right work shelf with one of its faces already selected. Make sure the T shelf is selected and select the face which forms the bottom of the the upright strut. Now run through the gluing procedure again. Click on the right shelf, click on the left shelf and then on the  button.

When the new object appears, click on  to get the ordering right. Lastly rotate the resulting object so that it is the right way up for a letter H. Next, to give the H a more 3D look, let's introduce a small gap in between the blocks. We could do this by moving single blocks but because we are discussing groups let's try moving two blocks at a time, say the right vertical block and the cross member.

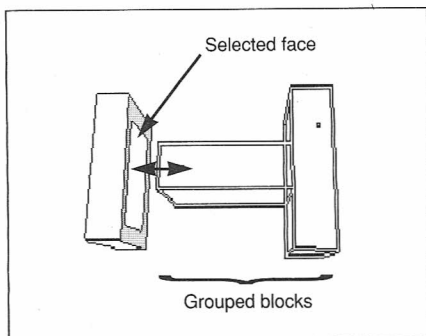
To do this we need a way of telling OM which two blocks we mean. We will do this by selecting the two blocks in turn and, as we do so, adding them to the group.

Select one of the two, say the vertical. Now click on  in the group box with the left mouse button. Then select the other block, the cross member and click  again. You will see that the points of the two blocks light up. We now have a group of two. Can you guess what the  button does? It removes a block from the group.

It's worth mentioning here that there is a shortcut to group selection which can save time when you are working with complex objects. If you click with the **right** mouse button on the  icon, all the blocks in an object get added to the group. The same goes for ; the whole group is cleared.

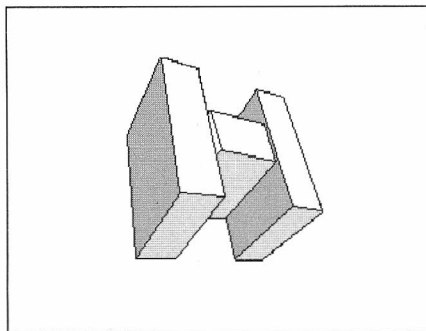
Each of the tools in the group box does something to the current group but many of them require another piece of information. The tool we are about to use, the  tool, moves a group away from or towards a selected face. We say that it moves the group *normal* to the face. The  tool is a slide tool but if you click on it now you will get a message complaining that there is no block or face selected.

Select the block not in the group and then the face with the cross member attached and try again. Gently move the mouse to the right. Do you notice what happens? The whole group moves away from the selected face.





**Figure 6.14**  
**Moving a group normal to a selected face**

Now, as an exercise, try making a similar gap between the cross member and the other vertical strut. We are aiming for the effect in figure 6.15.



**Figure 6.15**  
**The completed letter H**

While we are on the subject of groups, let's try some of the other group tools. Make sure you have a group and a face selected. Now try the  button. Once again this is a slide tool. It moves the group parallel to the selected face. This tool is particularly useful when you wish to position blocks after a join operation.

The  tool just below, provides the one further operation that is needed; it rotates the group in a plane parallel to the selected face. Now that you understand the concept of groups, read the reference section on the group toolbox in Chapter 7.3.

## Surface detail

We now come to a completely unique feature of OM – the surface detail toolbox. With surface detail you can design pictures to decorate the faces of your objects. To demonstrate, let's get a fresh cube onto the left hand work shelf (you can save your letter H if you wish).

Now using the face selector, select one of the faces. We are now ready to open the surface detail tool box. Do so carefully so that the tools don't spill out all over the floor. At the centre right of the panel is a square area marked Surface Detail. Click somewhere inside and notice what happens.

Firstly the surface detail area is replaced with 12 new buttons. These are the tools.

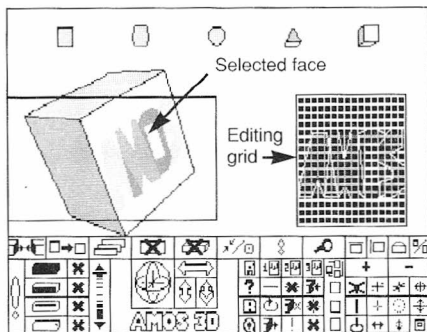






Figure 6.16  
A surface detail editing section

Secondly a criss-cross pattern appears in the display area replacing the right hand work shelf. This is the surface design matrix. Actually there is one more change: The square pattern that identifies the currently selected face of the cube disappears and instead the face is outlined with a faint dotted line. This is so that the surface details will not be obscured (you can switch between these two methods of face highlighting with the  button).

Let's design a simple detail. Move the mouse somewhere inside the matrix and drag it a little way with the left button down. When you release the button you should see a line appear reflecting your 'drawing' movement. Using this technique you will find that you can draw quite freely.

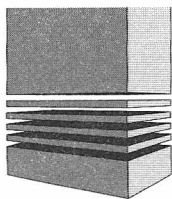
Now to design a surface detail. First of all let's clear the matrix. Click on the  button in the surface toolbox. Now draw a closed shape on the matrix. The shape must be closed, that is, it must join up exactly and enclose a region of the matrix with no gaps. If it is not closed OM will tell you when it attempts to attach the surface to a block.

Now all that remains is to decorate our face. Click on the  icon. The surface should appear on the selected face. You will notice that the picture on the block is somewhat different to the one on the matrix. It is solid and not a line drawing. This is how surface detail works. It's also why we had to be so careful about closing our shape.

The shape may also have a different orientation to the one on the matrix. Try clicking on the  icon. The shape will rotate. Now that you have a taste of surface detail, turn to part four of the next chapter for a full explanation of all the tools.

*That ends the OM tutorial. Of course we have only covered a tiny fraction of the things that you can do with OM. But now you have the general idea you will find that the next chapter contains plenty of explanation.*

*Just a final word. At Mandarin (and at Voodoo Software) we are very keen to see your objects as well as your programs. Feel free to send them to us along with any suggestions you may have for improving OM.*



# 7: The object modeller tools

This section is divided into four parts. Each part is concerned with a group of controls on the OM panel. At the beginning of each part is an illustration of the panel controls concerned. When you are more familiar with the commands, you will find the quick reference card useful as a reminder. Many of the buttons are in fact two tools in one. These are known as double commands. You use a different mouse button for each.

## 7.1: Primary Commands

The primary commands are those which operate on whole objects or shelves. Most are located on the top row of the panel.



Figure 7.2.2  
Primary icons

We discuss the use of these commands from left to right, top to bottom. In some cases the buttons invoke features such as surface detail which are dealt with in one of the other parts of this chapter. Some of the commands have effects which depend on which of the two mouse buttons is pressed. Unless otherwise specified, *clicking* means pressing the left mouse button.




### The Unite or gluing tool

This is a double command.

#### Left mouse button – Unite

**Purpose** This tool unites the objects in the two work shelves to yield a compound object.

**Method** Select the face on each of the objects where you want the join to take place. Select the two work shelves, one after the other. Click on the Unite button. The resulting compound object will appear in the last work shelf selected. The contents of the other work shelf will remain unchanged. Note that no object can contain more than 8 blocks. The effect of Unite can be undone using the  button.


**Comments** You will need to use the Precedence tool  after a Unite operation.



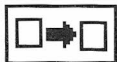
## Right mouse button – Unite and group

**Purpose** To unite two objects, order the blocks and select a group ready for positioning.

**Method** Proceed exactly as above. The effect is the same except that several additional operations are performed:

- The selected block and face on the destination shelf remain selected.
- The blocks added to the object from the source shelf are automatically selected as a group.
- The resulting object is automatically given correct precedence just as if the  tool had been used.

**Comments** When you unite with the right button OM performs all the operations necessary to prepare the new blocks for final positioning using the face relative movement tools. This is such a common next step that OM provides this sequence of commands in a single operation.



## Copy tool/Copy group tool

### Left mouse button – Copy

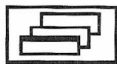
**Purpose** This tool copies the contents of one shelf to another, leaving the source object intact.

**Method** Click on the source shelf (the one you wish to copy *from*) and then the destination shelf (the one you want to copy *to*). Click on Copy.

**Comments** Copy always places an object on the target shelf in its unrotated attitude. If you want to copy an object at its current attitude, click on the rotation tool with the right mouse button first.

### Right mouse button – Copy group

The right mouse button may be used with the Copy tool to copy only the selected group. This is covered under Group Commands, chapter 7.3.



## Precedence/Culling tool

This is a double command.

### Left mouse button – Precedence


**Purpose** To arrange the blocks comprising an object in such a way that the object will be ordered correctly when viewed from any angle.

**Method** Click on the work shelf containing the object to be treated. Click on the Precedence icon with the **left** mouse button. This process can take a few seconds for complex objects.

**Comments** Occasionally you will find that this operation fails, with blocks that should be in front appearing behind and visa versa. The usual reason for this is that blocks have somehow become slightly embedded within one another. For more information on how to cure this problem, see **Problem Objects**.

### **Mounting blocks inside one another**

It is possible to order an object correctly with one block inside another (see the object *struct*). To see the inner block you can make windows in the outer block using surface detail. To make this work you must arrange for the inner block to have a lower block

number than the outer block. You can display the block number of each block by selecting it and using the  button.

When you unite a block to an object, the new block is given the block number zero. Using this fact you can arrange your blocks in the correct order before using (or re-using) the Precedence tool.

## **Right mouse button – Culling**

**Purpose** To compute surface, block and object culling depths for an object.

**Method** Click on this tool using the **right** button. An object can be unculled by reloading it into OM and saving it again.

**Comments** Culling is a way of speeding up drawing in an AMOS program when objects are so far away that detail, or the whole object, is small compared with the resolution of the screen. For each surface detail and block, OM calculates the distance beyond which that component is too small to be worth drawing. When you run the object in a 3D program the details on the surfaces and the blocks themselves will disappear, each at an appropriate depth. A culling depth is also calculated for the whole object. When it is displayed at a distance greater than this it will be replaced by a dot or a short line. In some types of program this can greatly speed up object processing.



## **Delete object tool**

**Purpose** Removes an object from a shelf.

**Method** Click on the shelf containing the object to be deleted and then click on the Delete button.

**Comments** The effect of Delete can be undone using Undo.




## **Delete block/delete group tool**

This is a double command.

## **Left mouse button – Delete block**

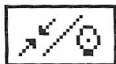
**Purpose** Removes the selected block from the object on the selected work shelf.

**Method** Click on the work shelf containing the object to be simplified. Click on the

 button in the selection box until the block you wish to remove is highlighted. Click on Delete Block. The effect of Delete Block can be reversed using Undo.

## Right mouse button – Delete group

**Comments** The tool may be used with the right mouse button to delete the current group. This is covered under Group Commands, chapter 7.3.




## Snap/Centre tool

This is a double command.

### Left mouse button – Snap

**Purpose** To tidy up an object by bringing nearby points together.

**Method** Select a block, a face, a line or a point or the whole object (by selecting only the shelf). Click on the Snap icon using the Left mouse button. Snap normally works only on the component selected. If no component is selected, Snap works on the whole object.

**Comments** Because Snap brings points within a certain distance of each other together, you can make Snap more or less sensitive by changing the size of the object with the  tool (see 7.3 *Group Commands* below). Some care is needed with this tool however since it takes no account of the flatness of faces. If you make an object very small and then Snap you may get an object with bent faces. This will not necessarily display correctly. Because Snap can introduce small errors it is a good idea to wait until your object is complete before using it. There is a temptation to use Snap whenever you notice some small discrepancy in an object – don't! Or at least if you do, save a copy of the unSnapped object first.

### Right mouse button – Centring

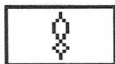
**Purpose** To centre an object at a selected point or at its centre of gravity.

**Method** Click on the icon with the Right mouse button. If a point on the object is selected the object is centred about that point. Otherwise OM calculates the Centre Of Gravity of the whole object and makes this the object's centre. The centre of gravity of an object is a point, usually but not always inside the object where all the object's mass acts (of course our 3D objects only have notional mass). If you actually made the object out of some real material and you had a way of balancing it on a spike, the centre of gravity would be the point at which it just balances.

**Comments** The position of an object's centre is important for two reasons. Firstly it is the point about which the object rotates, either under the control of the rotation tool or under control of an AMOS program. Secondly the centre of an object is the point whose coordinates you specify when you use one of the TD positioning commands.

Note that you may not always want an object to be centred about its centre of gravity. If you want an object to swing around a point at one of its extremities for example, that is where its centre should be. You can use the Group commands (with the whole object as a group) to perform centring about non centre of gravity points which are not vertices of the object. If you use one of these tools to move an entire object the centre stays where it is.

Note that the XYZ commands below do not move the centre of an object. They simply move the existing centre to a different point on the shelf; a subtle but important difference!



## Undo tool

### Purpose

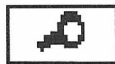
To undo the effect of the copy, unite, delete and undo commands. Both of the work shelves and the user shelves have a 'memory' of the last operation performed.

### Method

Click on the work shelf to be undone. Click on Undo.

### Comments

Clicking again on the icon will restore the undone object. Clicking a third time will perform the undo again and so on.



## Zoom tool

### Purpose

Causes the whole display area to be devoted to the object on the selected work shelf.

### Method

Click on the work shelf to be zoomed. Click on the zoom icon.

### Comments

Zoom is provided for two reasons. Firstly it lets you take a much closer look at the object you are working on. Secondly it speeds up the system by removing the other objects on the display area. The XYZ tools can be used in zoom mode to move the object about and to bring it closer (or further away). Use of the Zoom tool will not affect the position of the object in regular display mode. Zoom is also useful in situations where you want to look squarely at an object, rather than from the side.



## XZ-Align tool

This is a double command.

### Purpose

Used to align a face of an object parallel to the XZ-plane. See the glossary for a description of 3D axes.

### Method

Select a work shelf and then one of the faces of the object on the shelf.

Click on the icon. The left mouse button causes the selected face to point up. The right button makes it face down. If no group is selected the whole object is aligned, otherwise the alignment affects only the grouped blocks. (see Group Commands)

**Comments** One important use of this tool is in conjunction with the Stretching and Symmetry tools. The former can be used to stretch a group of blocks either horizontally or vertically. If the object being stretched is not in the correct attitude, it can produce unwanted effects. The Align tools allow you to select the correct attitude accurately. You can also use these to make two faces parallel. To do this you must define two groups and align them separately with one of the axes.



## YZ-Align tool

This is a double command.

### Right mouse button – Quick mode

**Purpose** Works in exactly the same way as the XZ-Align tool. The same comments apply. Left and right buttons produce opposite effects.

**Method** Select a work shelf and then one of the faces of the object on the shelf. Click on the icon.

**Comments** See the comments for the XZ-Align tool.



## XY-Align tool

This a double command.

**Purpose** Works in exactly the same way as the XZ-Align tool. The same comments apply. Left and right buttons produce opposite effects.

**Method** Select a work shelf and then one of the faces of the object on the shelf. Click on the icon.

**Comments** See the comments for the XZ-Align tool.



## Highlight mode tool

**Purpose** To select the highlighting method used by OM to show selected components.

**Method** Click on the icon to change mode. The diagram in figure 7.2.2 shows how components are highlighted when selected. In the default mode all surface detail is suppressed when components are selected.

**Comments** Alternate mode is suitable for surface detail editing where the selected face must not be obscured. Otherwise default mode provides clearer information. If you want to look at a complete object without de-selecting components, switch temporarily to Alternate mode.






## File tool

This is a double command.

### Purpose


To load or save an object.

### Method

Click on the shelf containing the object to be saved, or the shelf into which you wish to load an object. Now click on the File tool to bring up a standard filename dialogue. At the top of the dialog is a path box containing the current directory. Initially this will be the directory from which OM was launched. The path may be edited freely. Typing ENTER or clicking on the disc icon causes the new directory to be read and displayed in the list box. Filenames may be selected either by clicking in the list box or by typing in the filename box (or both). Once the path box and the filename box are correct click on  to load or  to save.  cancels the operation.

Directories are indicated in the list box by an asterix against the directory name. A double click loads the new directory. At the top of the list is a special directory *\*Parent*. Double clicking on this loads the directory containing the current one.

### Right mouse button – Quick mode

There is nothing more irritating than waiting for a file dialog to read a large directory when you know what you want or when you want to change directory immediately. If you click on the File icon with the right button instead of the left directory reading is suppressed and the dialog appears immediately. You can always force a directory read with the  button.

### Comments

When you save an object make sure that it is facing the way you want it to face in your AMOS program. Of course you can rotate it under program control but you will find objects much easier to control if they all have the same standard (unrotated) attitude.

The same applies to centring. Make sure that the rotation tool swings your object about an appropriate point. For details of centring see the Snap/Centring tool above. If you want your object culled, be sure to click the culling button before each save.

### File structure

Objects are multi file entities. The details are explained in appendix B on File Structure. When you save an object OM checks that all related .3DT and .3DS files are present in the same directory and adds them if they are not. Because of this the File tool provides the safest way of copying objects from one disc or directory to another. For an alternative, see appendix C describing the three Utility programs OL, PRUNE and SID.

### Disc Errors

OM uses AmigaDos and will respond with standard Intuition messages to any errors, for example non existent devices. For this

reason OM switches to the workbench or CLI screen during certain operations.

Sometimes you may notice a brief flash during file operations. This is perfectly normal.



## Info tool

**Purpose**  
**Method**

To display technical details of the object on the selected work shelf.  
Hold the left button down over the icon for as long as you wish information to be displayed.

**Comments**

Some of the AMOS commands allow points within an object to be manipulated directly from within a program. Other commands make use of other object components such as faces and blocks. The Info tool displays the information needed to identify these units. According to which components have been selected within the current work shelf, the following data will be displayed:

- R:r** The *radius* of the object. This is actually the radius of the circumsphere; the smallest sphere which will contain the object. It provides a very useful guide to the size of the object.
- B:b** The block number *b*
- F:f** The face number *f* within the selected block
- L:l(len)** The line number *l* within the selected face. The number in parentheses is the length of the line. This can be very useful prior to a Join operation.
- P(p1,p2)** The point number *p1* within the selected block. The point number *p2* relative to the whole object.

The animation command uses *p2* to identify points. If you want to place blocks inside one another you will need to know the block number *b*.



## Surface detail tool

**Purpose**

To put OM into surface mode and display the surface detail editing window. The icons in the surface detail panel box become active.

**Method**

Only the object on the left work shelf may be the subject of this operation. Click anywhere on the panel marked Surface Detail. The panel will be replaced by a set of surface editing tools.

**Comments**

For a full description of surface detail editing see part 7.4.



## Colour combination tool

This is a double command.

<b>Purpose</b>	To select the colour combination for a block. It does not change the palette in any way; it simply allows you to select which faces take which colours out of the OM palette. There are up to 10 possible combinations which vary according to the type of block selected.
<b>Method</b>	Select a work shelf containing an object. Select one of the blocks. Click on the icon repeatedly until the block shows in the desired colour combination. The left button moves forward through the sequence, the right button backwards.
<b>Comments</b>	The purpose of this function is to let you select the desired contrast between faces and blocks. Once an object has been designed which looks correct in the OM palette, a suitable palette can be designed from within the AMOS program which uses the object or using OM's RGB colour drafting tools. Note that objects use only colours 8, 9, 10, 11, 12, 13 and 14 leaving other colours free for backgrounds and other effects. When you click on this icon the combination number and the three colour numbers are displayed. These are the colour numbers of the block's faces. Any surface detail will also be drawn in a combination of these colours. If the block is one of the 2D blocks which have only two faces, the third of the three displayed numbers should be ignored. (The third colour of a block is actually always the logical OR of the first two). Of course you can make objects containing more than three colours by using several blocks.



## RGB tools

<b>Purpose</b>	To allow the amount of red, blue or green comprising a given colour to be adjusted. They also allow the colour number associated with object components to be ascertained.
<b>Method</b>	Click on a face of any object in the display area. Now hold the left button down over one of the RGB icons. As you move the mouse from side to side the intensity of the associated colour component changes.
<b>Comments</b>	While the left button is down over one of the RGB controls OM displays the following information:
<b>C:c</b>	The colour number <i>c</i> of the colour being adjusted
<b>R:r</b>	The amount of red <i>r</i> in the mixture (0 to 15)
<b>G:g</b>	The amount of green <i>g</i> in the mixture (0 to 15)
<b>B:b</b>	The amount of blue <i>b</i> in the mixture (0 to 15)

This information may be used in an AMOS program to generate a suitable palette.





## Reset tool

- Purpose** Causes OM to be reset to its initial state.
- Method** Click on the icon.
- Comments** When you reset OM all work not previously saved will be lost. OM will ask you to confirm that you want to reset the system.



## Quit tool

- Purpose** This tool implements an advanced software removal algorithm. The OM software is carefully removed and each byte is thoroughly cleaned.
- Method** Click on the icon.
- Comments** You will be prompted to confirm that you wish to leave OM.



## Rotation Tool

This is a double command.

- Purpose** To rotate the object on the currently selected work shelf.
- Method** Hold the left button down over the icon and move the mouse. A left/right movement rotates the object about the y-axis, an up/down movement rotates it about the object's x-axis. Click with the right button to redefine the object's axes after a rotation operation.
- Comments** It is well worth understanding clearly what happens when you use the rotation tool.

### Left mouse button

With the left button OM uses a system known as Euler Angles. To visualise Euler Angles imagine placing your object in the centre of a gramophone's turntable. Moving the mouse from side to side rotates the turntable. Moving the mouse up and down tilts the object up and down, but always about the same axis *relative to the object*. If you rotate the turntable through a right angle and then move the mouse up and down the effect would be to rotate the object in the plane of the screen.

In other words objects rotate about their own axes and not the fixed screen axes. Using the left button alone, it is not possible to see the object in every possible attitude. That's where the right button comes in.

### Right mouse button

The right button lets you redefine an object's axes. Whatever the current attitude of an object, pressing the right button over the rotation icon

causes its axes to be defined parallel to the screen axes. In other words OM now considers the object to be unrotated in its current attitude. You can see this if you copy the object to another shelf; it will appear in exactly the same orientation.

Whenever you save an object OM does this for you; the object is saved in its current attitude.



## XYZ Tools

These are double commands.

### Left mouse button

**Purpose** To move the centre of a shelf in the direction indicated by the one of the six arrows, left, right, up, down, towards the observer and away from the observer.

**Method** Click on one of the arrows.

**Comments** Each click on one of the XYZ icons moves the shelf a standard distance in the indicated direction. You are free to move shelves as far as you wish in any direction. You should avoid bringing them too close though as this may prevent correct drawing of the object.

The tool moves the shelf and with it the object. It does not change the position of the object's centre relative to the object, to do that use the Centring tool. Note that there is no visual difference between increasing the size of an object and moving it closer.

### Right mouse button

If you hold the Right button down over one of the XYZ icons you can move objects around continuously, rather than in jumps. All four of the XY icons behave the same way and move the shelf parallel to the screen. The two Z icons translate mouse up-down movements into depth changes in the shelf.

## Problem objects

Sometimes, when you are modelling an object you may find it hard to make the blocks appear with the correct precedence. A block may appear in front when it should be behind, even after using the Precedence tool.

If this happens to you, the reason will probably be that the faces you have *glued* together using the Unite tool have somehow become slightly embedded within one another. The precedence tool will not work properly unless the blocks are separate, that is either butted up exactly or with a gap between them. The faces must not be broken.

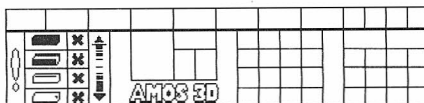
To solve the problem move suspect blocks a little way apart and try the precedence tool again. Once you have found the problem you can bring the blocks together again taking care not to push one inside the other.

Remember that OM does not complain if you move blocks through one another (modelling certain objects would be hard otherwise). In fact there are some cases when you actually **want** one block to be inside another.

This problem tends to crop up more often with objects that you have been working on for a long time. Although OM is very accurate, many modelling operations can introduce small errors, sometimes making the faces of blocks slightly bent. OM constantly attempts to remove these *cumulative* errors but it can not always do so, especially on very small blocks. If a face looks a little bent, or *wrong* in some other way, try remodelling that block on its own. You can always isolate individual blocks using Group Copy and Group Delete (see chapter 7.3).

## 7.2: The Block level tools

The tools described in this section are located on the lower left of the OM panel.



**Figure 7.2.1**  
Component selection and pull tools

These tools are grouped together because they all have something to do with the components of objects.

### Component selection tools

The four component selection icons each depict a block with a different component highlighted.


	No component selected	Block selected	Face selected	Line selected	Point selected
Normal Mode					
Alternate Mode					

**Figure 7.2.2**  
Selected components

To the right of each of the selection icons is a smaller icon bearing the symbol . These are the de-selection controls. They undo the effects of their corresponding selectors.

All four selection tools work in similar ways. They cause the indicated component on the

selected work shelf to be highlighted. As you click on the selectors OM cycles through all the possibilities for that type of component within a higher level selected component.

OM provides two alternative ways of indicating selected components. The default mode is usually the most appropriate. However if you are working on surface detail, or you wish to see the object with only minimal highlighting, you can change mode by clicking on the  icon. This is located among the primary tools and not in the block level box. For a description of the two modes see the Highlight tool in chapter 7.1.



## Block selector

Some operations require a block to be selected. If the object contains only one block there will be no visible effect of doing so. If there is more than one block, one of them - the selected block - is shown as a solid, the remainder being drawn as outlines. This allows you to see the whole of the selected block even if there are other blocks in the way.

As you click repeatedly on the block selector OM cycles through the blocks, selecting each in turn. Stop at the block you wish to select.



## Face selector

The face selector cycles through the visible faces in the selected block highlighting each in turn. If no block is currently selected, OM selects one for you. In default mode the face is highlighted with a square or triangular panel. If the selected block is one of the 2D flat types, only one face can be visible at a time. In this case OM does not highlight the selected face.



## Line selector

The line selector cycles through the lines bordering the selected face highlighting each in turn. If no block or face is currently selected, OM selects them for you. The selected line is indicated with a broken line drawn over it. Because of colour contention it is sometimes difficult to see the selected line. If this happens, rotate the object to a new attitude until you see the broken line.



## Point selector

This is a double command.

On 3D blocks the Point selector flips between the end points of the selected line, indicating the points with a small arrow. If no block, face or line is currently selected, OM selects them for you. On 2D flat blocks the selector cycles through each of the points.

## Surface anchor points

The point selector has another use which is dealt with in the section on surface detail. If you use the right button instead of the left, OM designates the selected point as a surface anchor point. See chapter 7.4, Surface Detail.



### Pull tool

This provides a means of altering the shape of a block by *pulling* a line or a point. If OM were a 2D object editor this would be easy. As it is, a 2D mouse and screen cannot provide sufficiently good visual feedback to let you know how a line or a point is moving in 3D space. The depth dimension confuses the situation. Additionally, OM has to ensure that all faces remain flat. This places restrictions on certain operations.

To pull a point or line proceed as follows:

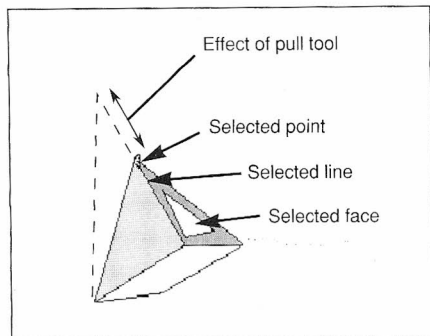
- Select the component you wish to pull (line or point)
- Click on the upper part of the pull icon and hold the left button down
- Gently move the mouse
- Release the left button when you have affected the desired change.

The rules governing point and line pulling are given below. To make interesting shapes you need to plan a sequence of operations. The method may seem a little awkward at first. Persevere. With a little practice you will get the hang of the controls. They provide enough flexibility to make any convex shape with a given number of points. Look at the example objects. You will be surprised at the wide variety of shapes that can be made from the simple basic types provided. Here are the rules:

## Pulling rules

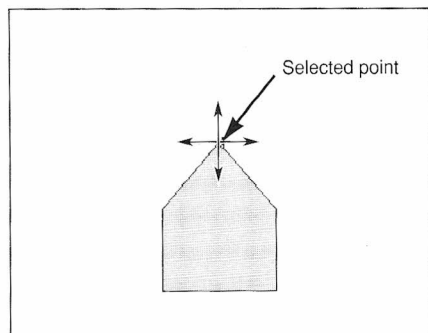
### Point pulling

- You can only pull points in certain cases. OM does not let you pull points that have to be in a certain place to keep faces flat. Blocks with bent faces are not allowed! It is impossible for this reason to pull a single point in the eight-pointed basic cube. Edit eight pointers by pulling their lines.
- When you **can** pull a point of a 3D block it always moves along the selected line, produced if necessary. When you pull one of the points of a 2D (flat) block it moves freely in the plane of the block.



**Figure 7.2.3**  
**Point pulling on a 3D block**

- Point pulling is the only way of changing the shape of flat blocks.

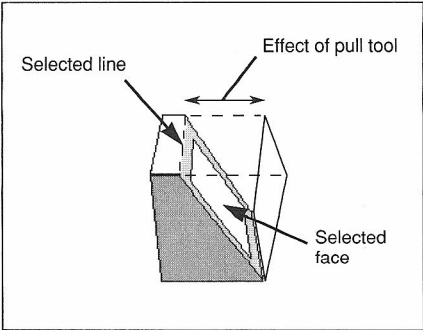


**Figure 7.2.4**  
**Point pulling on a 2D block**

### Line pulling

- Any line on a five or eight-pointed block can be pulled.
- When you select a line in a selected face you identify two faces uniquely. The first is the selected face, the second is the face attached to the selected face by the selected line. We will call these the primary and secondary faces respectively.

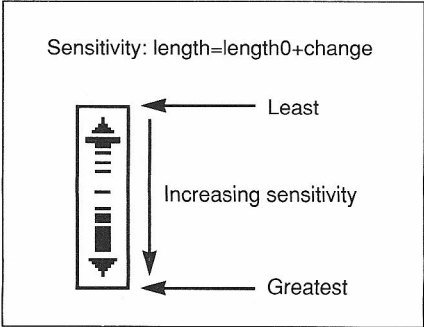
When you pull the line it moves *in the plane of the secondary face*. The best way to think of this is to imagine that the selected face is a door with the handle on the selected line. Pulling the line is like opening the door.



**Figure 7.2.5**  
**Pulling a line**

## Selecting the sensitivity of the pull tool

Sometimes you will want to make a big change to a point or line. Other times you will want a small accurate change. The sensitivity of pull depends on the part of the pull icon you click over. At the top the sensitivity is least; you can move one unit at a time. Further down the icon the sensitivity increases. The sensitivity you have selected is displayed (see below).



**Figure 7.2.6**  
**Pull sensitivity**



When you pull a point or a line the information displayed is very useful for making symmetrical blocks. If the block is one of the 3D blocks the information is as follows:

<b>Sensitivity</b>	is the pull sensitivity you have selected. The number is the size of the smallest change the operation will resolve.
<b>length</b>	is the length of the produced or truncated line .
<b>length0</b>	is the length of the produced or truncated line prior to the pull operation
<b>change</b>	is the amount by which the produced or truncated line has changed

If you want to perform two or more equal pulls so as to make a block symmetrical, **change** in both cases should be the same.

If you are pulling a point of a 2D (flat) block the information is the same except each of the numbers above are coordinates of the form (x,y). These tell you the coordinates of the point as it changes.





## Undo tool

If you make a mistake during a pull operation, this tool will undo it. A second click on Undo restores the 'mistake'.

## 7.3: Group commands

Grouping is a way of identifying a selection of the blocks comprising an object. Sometimes a group will be just one block that you want to single out for manipulation, sometimes several blocks and sometimes you will want to turn a whole object into a group. A group is different from a selected block. In fact to use some of the group tools you will need both.

With the exception of the group defining tools  and , the commands described below work only on the selected group. You may define one group on each of the two work shelves.

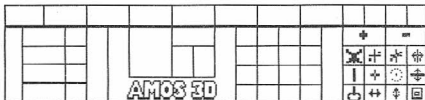




Figure 7.3.1  
Group icons

### Group highlighting



OM indicates which blocks belong to the current group by drawing a small dot at each vertex (corner point) of each block in the group. These dots show through any obscuring blocks.

#### **Selecting a group**



These are double commands.

You set up a group on the selected work shelf using the  and  commands. These work in two ways:

#### **Left mouse button**

If you click on  with the left mouse button any selected block is added to the current group. If you click on  with the left button any selected block is deleted from the current group.

#### **Right mouse button**

If you click on  with the right button, OM makes a group out of all the blocks in the object. If you click on  with the right button the group is cleared completely. Once you have selected a group you are ready to use the group commands.

## Face relative movement tools

One of the most difficult aspects of object design is judging the relative movement of blocks. You may think that you have achieved the effect you want from one point of view, only to find that it's wrong from another.


Modelling objects with OM is very like modeling with clay or some other material. Of course when you are making a physical object you see it in 3D. Your hands can feel the shape and you can get a real feel for the changes you are making. OM does its best to give you as much visual feedback as possible but on a 2D screen there are bound to be limitations.


Experience has shown that the best way of expressing a change in the relative position of blocks is by reference to some part of the object itself. With a 3D object it is not very precise to say 'move that block a bit to the left'. It's much better to say something like 'slide that block over that face' or 'lift the block off that face'.

This is the purpose of the face relative tools. They let you specify changes relative to something you can see (another part of the object) rather than relative to the screen.

Suppose that you have just glued two blocks together with the Unite tool (see chapter 7.1). Unite butts the selected faces of the selected blocks up against one another. OM can't know exactly where the glued object should go so it puts it in the middle somewhere. You will probably want to adjust the exact position, perhaps by sliding the blocks over one another or by raising one *above* the other. This is where the three face relative tools come in. They provide the three operations you will need to get the position and angle of the glued object exactly right.

To slide one over a face of the other use 



To raise or lower one over a face of the other (or **normal** to it) use 

To rotate one **in the plane of** a face of the other use 

Of course these tools are useful in many situations, not just after a Unite. They can be used at any time, whenever you need to move one or more blocks relative to an object.

Before using the face relative tools OM needs to know two things.

- Which blocks to move
- Which face you wish to move relative **to**; the anchor face.

You indicate which blocks you wish to move by turning them into a group using the  and  buttons.

You indicate the anchor face by selecting it using the selection tools. (Note that the anchor face may be any face, even one on the group you intend to move)

Once you have a group and a face selected you are ready to use the face relative tools.

When you are changing the relative position of groups you will find it easier to judge the effect of your changes if you frequently alter the attitude of your object with the rotation tool. Objects come to life when they rotate, and complex shapes become much easier to grasp.



## Face relative Slide tool

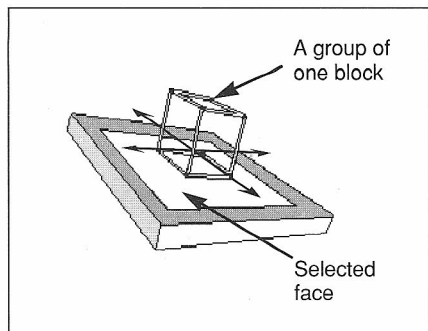
**Purpose**  
**Method**

To move a group parallel to the selected face.

Select a group and an anchor face. Now hold down the left mouse button over the Slide icon and move the mouse.

**Comments**

Because an object can be in any orientation, the direction of mouse movement does not always produce movement in quite the same direction on the screen. Experiment to find the best attitude for your object.



**Figure 7.3.2**  
**Slide tool**



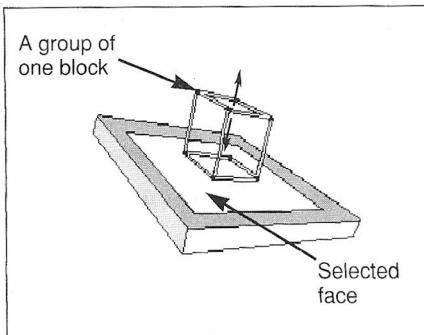
## Face relative Normal tool

**Purpose**

To move a group normal to the selected face. Normal means *at right angles to*.

**Method**

Select a group and an anchor face. Now hold down the left mouse button over the Normal icon and move the mouse.



**Figure 7.3.3**  
**Normal tool**



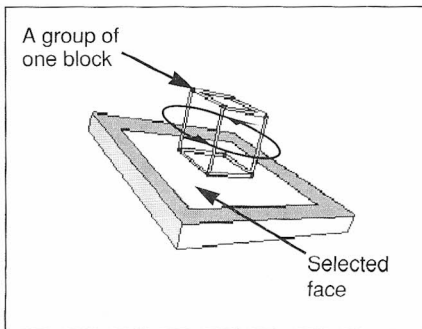
## Face relative Rotation tool

**Purpose**  
**Method**

To rotate a group in a plane parallel to the selected face.  
Select a group and an anchor face. Now hold down the left mouse button over the Rotation icon and move the mouse.

**Comments**

Because of the amount of calculation involved, this operation is slower than most. Move the mouse very gently until you get the feel of the command.



**Figure 7.3.4**  
**Rotation tool**

## Axis relative movement commands

These commands also move and rotate groups, but this time they do so relative to the plane of the screen. It is best to use these tools in Zoom mode (see Part 1) which allows you to look at an object head on rather than from one side.




### XY group displacement

- Purpose** To move a group parallel to the XY plane.
- Method** Hold the left button down over this icon and move the mouse left, right, up and down.
- Comments** This command is most useful when used in conjunction with the Alignment tools.





### Z group displacement

- Purpose** To move the group parallel to the Z-axis.
- Method** Hold the left button down over the  icon and move the mouse left and right.
- Comments** This command is most useful when used in conjunction with the Alignment tools.



### Set rotation centre

- Purpose** To set a centre for the  tool. The group rotates in a plane parallel to the screen.
- Method** Hold the left button down over this icon and move the mouse. A set of cross hairs will appear and move following the mouse.
- Comments** This command does not affect the object in any way. It merely sets a centre for the axis relative Rotation tool.  
Don't get confused between this tool and the **centre**  tool in part 7.1. The centre set with this tool only applies to the Axis relative rotation tool below.



### Axis relative rotation tool

- Purpose** To rotate a group in a plane parallel to the screen using the centre of rotation set with the set centre tool above.

- Method** Select a centre with the set centre tool, then hold down the left mouse button over this tool and move the mouse, left and right.
- Comments** Because of the amount of calculation involved, this operation is slower than most. Move the mouse very gently until you get the feel of the command.

## The symmetry tools

The purpose of these tools is to invert a group either vertically or horizontally. They do not make groups symmetrical. That can be done in other ways. A typical use for the symmetry tools is to make objects where the block(s) on one side are a mirror image of those on the other, like the wings of an aeroplane. A second use of the command undoes the effect of the first.



### Vertical symmetry tool

- Purpose** To turn a group into a mirror image of itself in the vertical direction.
- Method** With a group selected, click on this icon.
- Comments** This command is most useful when used in conjunction with the alignment tools.

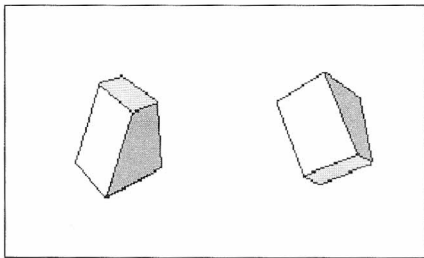


Figure 7.3.5  
Vertical Symmetry



### Horizontal symmetry tool

- Purpose** To turn a group into a lateral mirror image of itself.
- Method** With a group selected, click on this icon.
- Comments** This command is most useful when used in conjunction with the alignment tools.

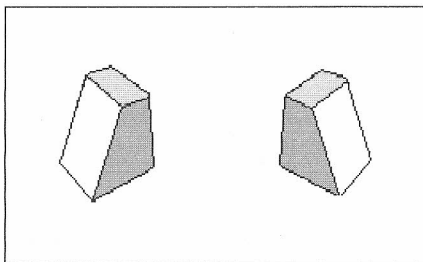


Figure 7.3.6  
Horizontal Symmetry

## Stretching tools

Sometimes the best way to shape a block or a group of blocks is to stretch it either vertically or horizontally. This is the easiest way of making regular six-sided figures and certain sorts of pyramids. It works well when the object is stretched along a main axis. If you stretch a group at an arbitrary attitude you will often get very odd effects which are difficult to correct accurately.

Another sort of stretching tool is the Sizing tool (see below). This stretches along all axes simultaneously, which is the same thing as making it larger. The Stretching tools work both ways, that is by moving the mouse from right to left you can compress as well as stretch.

Compressing an object or making it smaller with the Sizing tool has a side effect which can alter the shape of your object. Suppose you have an eight-pointed block 100 X 100 X 30. Now imagine making the object a quarter of its original size with the Sizing tool. The dimensions will now be 25 X 25 X 7.5. Since OM works in integer arithmetic (this makes it much faster) the 7.5 would be rounded down to 7.0.

Now imagine using the sizing tool again to bring the block back to its original size, that is, multiplying each dimension by 4. You would now have a block 100 X 100 X 28, not quite the original shape. You will only get this effect if you make the object smaller, let go of the mouse button and then make it larger as a separate operation. While you hold the button down OM works from the original coordinates.

**Warning:** OM will let you compress a group to nothing and then beyond. If you do this you create a very strange object that has no analog in reality. One property of such inside out objects is that the laws of perspective are reversed. The parts which are ostensibly further away look larger than the closer ones. What is really happening is that the faces that OM draws are actually the faces that would be hidden in an ordinary object. By all means use this effect if you wish; it can be stomach curdling, but expect the unexpected.





## Horizontal stretching tool

- Purpose** To stretch a group along the x-axis.
- Method** Hold the left button down over this icon. Move the mouse to the right to stretch, to the left to compress.
- Comments** This command is most useful when used in conjunction with the Alignment tools.



## Vertical stretching tool

- Purpose** To stretch a group along the y-axis.
- Method** Hold the left button down over this icon. Move the mouse to the right to stretch, to the left to compress.
- Comments** This command is most useful when used in conjunction with the Alignment tools.

## Sizing



## The 3 axis stretching or sizing tool

- Purpose** To make a group larger or smaller.
- Method** Hold the left button down over this icon. Move the mouse to the right to stretch, to the left to compress.

## Aligning groups

The Alignment tools are not exclusively group tools. If no group is defined they align the whole object on the selected work shelf as described in Part 1. If a group is defined, these three tools will operate on the group only. The main use for this is in lining up blocks or groups of blocks to make them fit together.

Suppose you have an object in which there are blocks that you wish to butt up against one another, or in which you want to make the faces of two blocks parallel. Usually you will achieve this by gluing blocks together with the Unite tool. On occasion though, you may want to do so without splitting up the object and regluing.

You can do so by selecting the blocks in turn as groups and aligning them separately against one or other of the axis pairs using the Align tools. There are many useful variants of this method, for example placing groups of blocks at right angles to one another. For a description of the Aligning tools see Part 1 of this chapter.

## Copying and deleting groups

The copy and delete block tools as described under Primary Commands may also be used to copy and delete groups as follows.



### Copy group tool

**Purpose**

To copy a group of blocks to another shelf.

**Method**

With a group selected, click first on the source shelf and then on the destination shelf.

Then click on the copy tool button with the **right** mouse button.

**Comments**

The blocks comprising the group will be copied just as they were in the original object. The resulting object will be displayed unrotated, in its base attitude. If you want to preserve the current attitude, click on the Rotation tool with the right mouse button first.



### Delete group tool

**Purpose**

To delete a group of blocks.

**Method**

With a group selected, click on the icon with the **right** mouse button.

## 7.4: Surface detail

OM's surface detail feature is one of its most powerful facilities. With it you can not only add pictures to the faces of your objects, you can also add transparencies which allow you to place windows and holes in objects. With judiciously placed surface detail you can turn even a single block into a complex structure.

Many of the example 3D objects use this technique to create highly complex looking objects from three or four blocks. Surface detail is fast too. You can achieve much greater speed with decorated objects than you could by using lots of blocks. If the feature is used effectively the resulting objects can look as if they contain dozens of basic shapes.

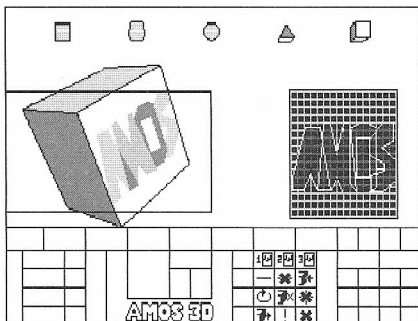


Figure 7.4.1  
Surface detail icons

OM is brought into surface editing mode with the large surface detail button. The object on the left work shelf is selected for surface editing and a square editing grid appears to the right. Most of the regular OM controls remain active and an additional set, to the lower right of the panel becomes active.


In the following it is assumed that surface detail mode has been selected.

### Component selection

When the faces of objects are selected using the face selection tool, OM places what is actually a simple surface detail on the selected face and suppresses all other surface detail. During surface detail editing it is not appropriate to highlight selected faces in this way and the alternative highlighting method is used instead (see figure 7.2.2).

In this mode, which is automatically selected during surface detail editing, selected faces are shown by means of a faint dotted outline. The simplest way of decorating a face is as follows:

- Select the face you wish to decorate

- Draw a design consisting of closed edges on the editing window
- Click on the  icon to attach your picture to the face.

There are many variants of this procedure as you will appreciate when you understand the function of the controls in the surface detail panel box.

## Closed edges


Surface details consist of filled shapes. When you design a surface you do so by outlining shapes on the editing grid. Because OM fills your shapes when it attaches them to faces, your line drawings must not contain gaps where the colour can leak out. OM will not attach a surface that contains unclosed edges. Closed edges must be drawn with lines of the **same colour**.

## Drawing on the editing grid

In the following we refer to the tools in the surface editing box to the lower right of the panel. The first three icons in the box are virtual colour selectors. These are not colours in the ordinary sense. The effect of a given virtual colour on a face will vary according to the 'background' face colour. Shapes of different colours can overlap and there is a wide range of possible effects. One of the virtual colours will yield a transparent area. The other two will produce colours taken from the colours on the other faces of the same block. The best way to find out how a given virtual colour behaves in a particular situation is to experiment. Once you have selected a virtual colour, you can draw lines between grid intersection points as follows:

- Place the pointer over the grid intersection where the line is to start
- Hold the left button down and move to the the part of the grid where the line is to end
- Release the left button
- Repeat the process until you have enclosed an area.

## Attaching a surface to a face

This is easy. Simply click on the  icon. Your drawing will be filled and attached to the selected face of the object on the left work shelf.

## Surface complexity

When you attach a new surface to a face OM displays the following message:

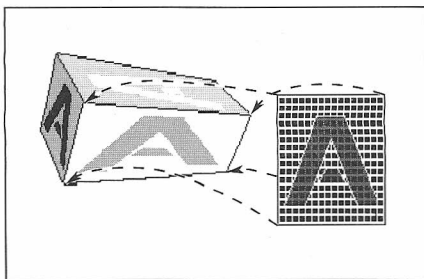
### Surface Complexity N

where **N** is a number. This number is a measure of the amount of work that 3D has to do every time it draws the surface. Low numbers mean little work and high numbers mean more. The complexity number can be a useful guide to the speed of your object and hence that of any program containing it.


Complexity is not related only to the number of lines or points in a surface and relatively small changes to a surface can often significantly reduce its complexity.

## Positioning the surface

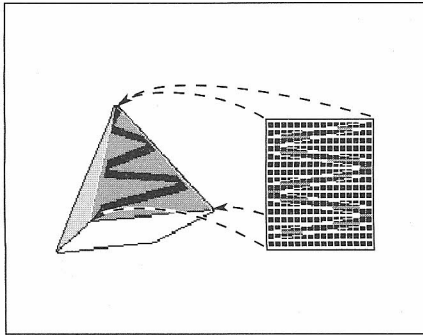
Surfaces can be added to any face but see below for special discussion of flat blocks. In the case of a four-pointed face, it is easy to see how the drawing on the grid is mapped onto the face. It is as if the grid were made of elastic and simply stretched over the face.



**Figure 7.4.2**  
**Stretching a surface over a 4 point face**

Whatever the shape of the face, the design will be stretched to fit. When you click on the transfer icon OM uses the four corners of the face as **anchor points** and attaches one corner of the grid to each. Because of this there are four possible orientations for the design. You can cycle through these by clicking on the  icon.

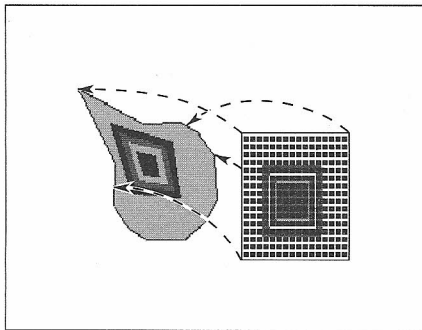
When you attach a surface to a three-pointed face things are not so simple because there is a point left over. What OM actually does is to attach two corners of the grid, the top left and the top right, to a single point of the face. This distorts the design. A square for example turns into a triangle. With a little practice you will learn how to compensate for this distortion when you design your surface.



**Figure 7.4.3**  
**Stretching a surface over a 3 point face**

This is easier than it sounds. It is a simple matter to modify a surface until you achieve the effect you want. When you use the orientation tool, the surface rotates as before, but this time there are only three positions. Whatever the position, it is always the top corners of the grid that are collapsed into a single point.

### **Attaching surfaces to 2D (flat) blocks**



**Figure 7.4.4**  
**Stretching a surface over a 2D face**

The two faces of a 2D block can be selected like any other face but in this case the face is not highlighted. The reason for this is that only one can be visible at a time. When you use the **attach** tool for a 2D block, OM chooses which four points to use as anchors. In some cases you will want to use a different set of anchor points. You can do this as follows:

- 1 Select the face to receive the surface
- 2 Use the point selector to select the first of the four anchor points
- 3 Click on the point selector again but this time using the **right mouse button**. OM will display a message to confirm that the anchor point has been correctly designated.
- 4 Repeat steps (2) and (3) until you have nominated all four points. Two of the points may be the same if required.

## Re-using surfaces



Surfaces live in the same folder as your object and have the extensions .3ds. When you save an object OM automatically saves all its surfaces. You might wonder why surfaces are not saved as part of objects. The reason is that objects and surfaces exist in a many-to-many relationship. The same surface can be used many times on the same object and many times over different objects. An object can have many surfaces. A single surface can be attached to many objects.

OM never keeps more than one instance of a given surface in memory at the same time. The same applies to AMOS. This means that objects use much less memory than they otherwise would. You will find that surfaces generally consume very little memory. They are also very fast.

## Copying surfaces between objects and within objects

It is good practice to exploit 3D's ability to use the same surface in many places. The same surface can look utterly different when it is attached to a different shaped face and you will also save a lot of memory and disc space (as well as design time).

Suppose you have an object containing a surface that you would like to use again. Proceed as follows:

Copy your new object to one of the User shelves. Now load the object containing the surface you wish to use and select the face sporting the surface. Next select surface detail editing mode and click on  to copy the surface to the editing grid. Finally, copy the new object to the left work shelf, select the face and click on  to mount the surface.

## The surface toolbox

When you enter surface detail mode by clicking on the large icon, the tools in the surface toolbox become active. As follows:



**Line colour selection/  
colour flip tools**

## Left mouse button

These are double commands.

**Purpose**  
**Method**

These buttons select the virtual colour of the lines drawn into the editing grid.  
Click on one of the buttons



selects virtual colour 1



selects virtual colour 2



selects virtual colour 3

## Right mouse button

**Purpose**  
**Method**

To swap the virtual colours of lines already on the editing grid.  
Click with the right button on one of the three icons.



swaps virtual colours 2 and 3



swaps virtual colours 1 and 3



swaps virtual colours 1 and 2

**Comments**

As mentioned above, a shape drawn with a particular line colour on the editing grid will produce an effect on the target face which depends on the block colours. Sometimes you will find that this is not the effect you want. Instead of changing the virtual colour of every line on the grid, use these tools, swapping colours until you have the effect you want.





## Line editing selectors

**Purpose**

To cause subsequent lines drawn into the editing grid to be added to the detail or deleted from it.

**Method**

To draw new lines click on the  button. To delete existing lines click on the  button. They remain in force until a new mode is selected.



## Transfer tool

**Purpose**  
**Method**

To add the detail on the editing grid to the selected face of the object.  
Select a face. Click on the icon.



**Comments** This tool checks to ensure that the edges formed by the lines on the editing grid are closed. Each closed edge must be built out of lines of the same colour. If the edges are not closed OM will display a message and leave the target face unchanged.



## Edit tool

**Purpose** To transfer the detail on the selected face of the object to the editing grid.  
**Method** Select a face of the object containing a surface detail. Click on the icon.  
**Comments** Some combinations of block colours and virtual surface line colours are incompatible. In such circumstances, no surface will be visible.



## Surface attitude tool

**Purpose** To rotate the surface on the selected face in jumps of approximately a right angle.  
**Method** Select a face sporting a surface detail. Click on the icon.



## Grid clear tool

**Purpose** To clear the surface detail editing grid (but not the selected face).  
**Method** Click on the icon.



## Surface removal tool

**Purpose** To remove a surface detail from the selected face (but not from the editing grid).  
**Method** Select a face. Click on the icon.



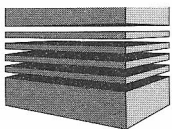
## Line Undo tool

**Purpose** To delete the last drawn line(s) from the editing grid.  
**Method** Click on the icon.



## Quit Surface detail tool

**Purpose** To end a surface editing session. This button removes the surface toolbox and grid, ending the surface editing session.



# 8: 3D Programming

The objects you create in OM come to life when they are brought under program control with the AMOS 3D extension. We suggest you read this chapter in the order it is presented so that you approach your programming in the correct way. You must first understand how the 3D world works, this will make using the commands easier. Refer to the glossary for more explanation of unfamiliar words.

## 8.1: The 3D World

### Introduction

In this section we introduce the basics of 3D programming. 3D graphics has a reputation for being difficult. We hope you'll agree with us that actually it's no harder than working with sprites and backgrounds. In AMOS 3D we have provided a set of commands that give you the power of 3D without all the maths and bit crunching. With 3D you can concentrate on writing programs and let us take care of the details.

To get the best out of 3D though, you will need to know some geometry. We start with some basic concepts and a little terminology.

The programs TD\_Simple.AMOS, TD\_Loop.AMOS and TD\_View.AMOS which are used as examples throughout this manual can be found on the AMOS 3D disc you prepared during the installation procedure.

### Space

Most people know how to read a graph. It has two *axes* (Figure 8.1.1), the x-axis and the y-axis. The point where the axes meet is called the *origin*. The two axes, which are marked with a scale, allow you to identify any point on the graph.

Any point can be reached by starting at the origin, travelling a distance along the x-axis and then another distance at right angles, parallel to the y-axis. These two distances are called the x and y coordinates of the point. They are written (x,y). Figure 8.1.1 shows a few points with their coordinates marked (notice that some of the coordinates are negative). A graph like the one in Figure 8.1.1 is an example of a *coordinate system*.

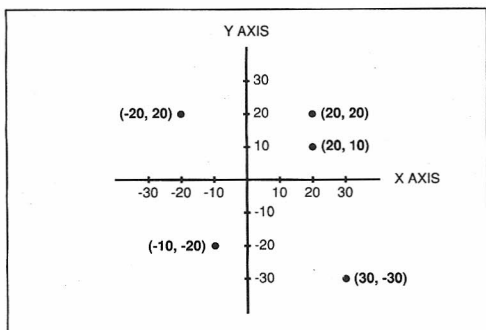


Figure 8.1.1

If you have programmed any 2D graphics you will have used a coordinate system, the *screen coordinate system* or simply *screen coordinates*. Often, screen coordinates have the origin in the top left corner with the y-axis increasing down the screen. This is appropriate because of the way screen memory is laid out (see Figure 8.1.2).

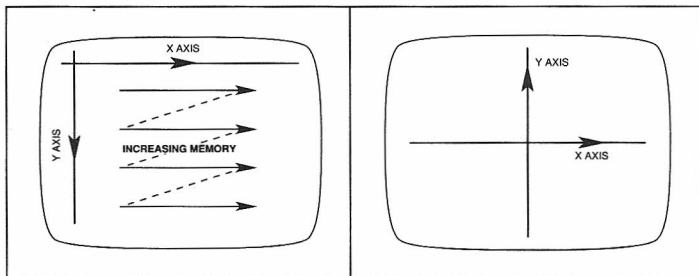


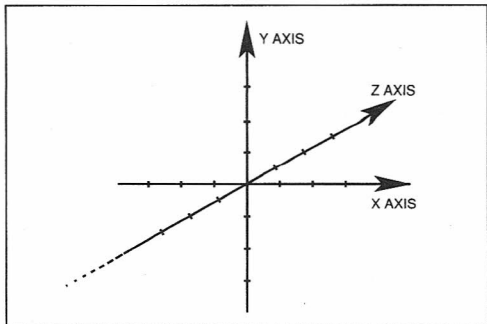
Figure 8.1.2

Figure 8.1.3

In 3D graphics we use a screen coordinate system too, the one shown in Figure 8.1.3. Notice that the origin is roughly in the centre, with x and y axes that *go negative* as well as positive. Notice also that the y-axis increases in the upward direction like the conventional graph of Figure 8.1.1.

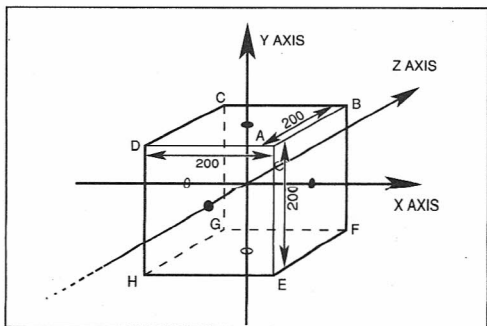
2D coordinate systems like Figure 8.1.3 are fine for flat pictures. For 3D though we need something more to represent depth; an extra dimension. We call this dimension the *Z dimension* and we measure it as you would expect, along the z-axis.

We now come to our first problem. How do we draw a 3D graph on two dimensional paper? Well the answer is that we can't. To draw properly in 3D we would need to draw inside a tank, perhaps filled with treacle. The best we can do on a flat page is to make a perspective drawing of a 3D graph, like the one in Figure 8.1.4. As you will see, this 3D coordinate system is exactly like our 2D graph, except that it has one more axis. This is zero at the origin and increases as you travel into the distance. In the other direction, out of the paper, the z-axis becomes negative.



**Figure 8.1.4**

With this 3D coordinate system we have a way of naming any point in space. To do so we use three numbers: x, y and z, written  $(x,y,z)$ . Now look at Figure 8.1.5. This shows a 3D coordinate system with a cube at its centre.



**Figure 8.1.5**

The cube is 200 units on each side and the eight corners or *vertices* of the cube are marked with letters of the alphabet. Can you work out the 3D coordinates of the vertices? The answers are:

A=(100,100,-100)  
B=(100,100,100)  
C=(-100,100,100)  
D=(-100,100,-100)  
E=(100,-100,-100)  
F=(100,-100,100)  
G=(-100,-100,100)  
H=(-100,-100,-100)

This coordinate system is called the *world coordinate system*. It is where we build our 3D world. We look at this world through a window, the computer screen.

We are now ready to look at a simple 3D program. Load the program *TD\_Simple.AMOS*.

## The double buffered display

In 3D graphics we usually use a double buffered display. This consists of two screens, one for showing the graphics and a hidden one where we draw the next frame. Once we have a new picture, we swap the screens over, displaying the new one and use the old display screen to prepare the next frame. You don't have to do things this way but it does give the best results because you never see graphics as they are being drawn. To set up a double buffered display we use the following two lines of Basic:

```
Double Buffer  
Autoback 0
```

These are the first two lines of the program. The next line is a Td LOAD command. This loads the object *disc*. Once you have loaded an object definition like the disc you can have as many *instances* of it as you wish. You could have half a dozen discs on the screen, all based on one object.

To build an object instance you use the Td OBJECT command. Td OBJECT is the fourth line of the example program. It tells 3D to build an object based on the definition. It also tells 3D where to place it in world coordinates and what attitude to give it.

As you will see the Td OBJECT command is followed by eight parameters. The first is the object number (1 in this case). You can choose any number for this between 1 and 20. It is used to refer to the object in other commands. The second tells 3D to base object 1 on the disc we have just loaded. The next three numbers are simply the object's position in world coordinates, x,y and z. The last three are the attitude. We will discuss these later.

Now all that remains to do is to set up an appropriate colour palette, draw the object to the hidden screen and then swap screens to make it visible. The commands:

```
Palette ,,,,,,,,$fff,$00f,$777  
Td CLS
```

## Td REDRAW Screen Swap

do this.

Now run the program. You should see something like Figure 8.1.6.

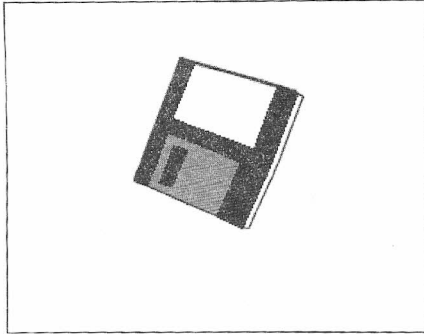


Figure 8.1.6

## Setting up a moving 3D display

The *TD-Simple.AMOS* program can easily be turned into a loop that constantly redraws the disc and swaps screens. To see any change though you would need to move or rotate the object in between frames.

Load the program *TD\_Loop.AMOS*. This is the basic redraw loop used to create a moving display. The *REM* statement tells you where to move your object(s). If you run this program now all you will see is a disc in the middle of the screen. Press Control-C to quit the endless display loop.

## Angles

In 3D you can rotate objects about each of the axes *x*, *y* and *z*. The commands that do this rotate objects not about the world axes but about a set of axes based on the centre of the object. This *local* coordinate system is a little like the local coordinate system described below. For now, imagine that our disc is like the cube in Figure 8.1.5 sitting in the middle of its coordinate system. You can rotate about the *x*-axis, the *y*-axis and the *z*-axis. We call these angles *A*, *B* and *C*.

The units used to describe angles in 3D are a special sort called VRUs (Voodoo Rotation Units). They are explained more fully in the sections on *Positions* and *Angles*. VRUs divide up the circle into 65,536 divisions. 90 degrees for example is the same as 16384 VRUs.

Let's add a statement to the *TD\_Loop.AMOS* program to rotate the disc about the *x*-axis. Just below the *REM* statement which says 'Move your objects here' add the command:

Td ANGLE 1,A,0,0  
A=A+1000

If you run the program now you will see the disc rotate. The four parameters following Td ANGLE are the object number (as defined in the Td OBJECT command) and then the three angles A, B and C. The program works by increasing A by 1000 each time the object is displayed.

At this stage you should be ready to try some of the other movement and angle commands. Try replacing the Td ANGLE command above with other commands such as Td MOVE. You will find the Td commands quite like the Sprite commands. The only real difference is that there is an extra dimension.

One problem you will probably encounter is that of losing objects. 3D space is big, much bigger than a 2D screen. In 3D space it's easy to lose objects, and, as you will find out, just as easy to get lost yourself! Consequently it is important to limit the distances you work with. We will be saying more about this later on.

## The local coordinate system

So far we have described the basic coordinate system used in 3D graphics: the world coordinate system. Now we must learn about one more: the *local coordinate system*. To see why we need another coordinate system let's consider a typical example. Suppose you're in the cockpit of the fighter aircraft A in Figure 8.1.7. An enemy plane B comes into view. You line it up in your sights and fire.

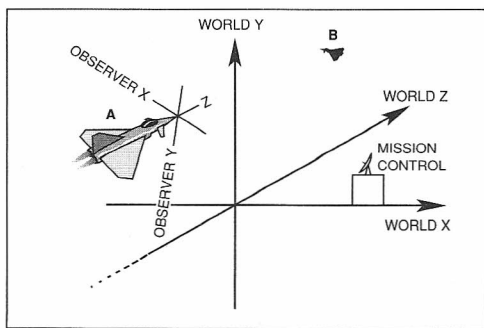


Figure 8.1.7

Let's relate that to our world coordinate system. Each of the objects, your plane and the enemy, will have a position  $(x,y,z)$ . We might imagine a missile firing system which takes both sets of coordinates and computes a path. This would be very inconvenient. Instead it would be better to tell our missile system something like: *target straight ahead, range 15 miles*. This also fixes the missile's position, but in a more appropriate way.

When we look through our sights we are in fact using a coordinate system, but this system is based around *our* position. Its origin is our craft and its z-axis is a line pointing straight ahead. The x and y axes are the same as the sighting lines on our sights. Obviously this is a much better coordinate system so far as you are concerned. (Of course mission control on the ground might prefer to think in terms of world co-ordinates; your local coordinate system has little meaning for them).

This new system is called the *local coordinate system* and it is worth noting that the coordinates of the same point in world coordinates and in local coordinates may bear little relation to one another. Your plane may be pointing anywhere. Your z-axis could be pointing in the same direction as the world x-axis, or if you are in the middle of a nose dive, your z-axis would be pointing in the same direction as the world's y-axis. In many situations the local system will be pointing at a crazy angle.

## The viewpoint

In AMOS 3D you can define up to 20 objects at different positions in world coordinates. One of those objects, object 0 is special; it is your own viewpoint. You can move your viewpoint around just like any other object. Whatever it *sees* you see.

Let's try another example. Load the program *TD\_View.AMOS*. This program lets you fly around the 3D world using the mouse. Movement in this demo is controlled as follows:

Mouse up/down	Fly forwards and backwards in the directions you're pointing
Mouse left/right	Turn on a point either left or right
Mouse up/down holding left mouse button	Moves you up and down
Right mouse button	Faces you directly at the spinning disc. Useful if you get lost or confused!

Try running the program. You should be able to swing the viewpoint round by moving the mouse from side to side and zoom in on the disc by moving the mouse away from you. We sometimes use a special name for the local co-ordinate system based on the viewpoint; we call it the *observer co-ordinate system* or just *observer co-ordinates*.

As you get more advanced you will find that there are many interesting things that you can do with the viewpoint. For example you can have two viewpoints by changing object zero's position and generating two separate views of the same world at the same time. See *TD\_View2.AMOS*.

## Choosing the best coordinate system

Much of the time you will be dealing in world coordinates. The positions of objects are defined using this system and in many programs these are all you will need except for controlling the viewpoint. Local coordinates become useful when you want to make something happen at some position relative to the viewpoint or another object.

To make a 3D shoot'em-up interesting it is best to make sure that attacking ships appear from the direction that the player is looking. You might decide to make a ship



attack from a point straight ahead, say at (0,0,10000) in local co-ordinates. You would convert to world co-ordinates and then use Td OBJECT.

3D provides a family of commands to do all the conversions you need. You can convert from world to local (Td VIEW), local to world (Td WORLD) and even world to screen coordinates. In fact 3D allows you to convert to and from the coordinate system of any object, not just the viewpoint. When you use Td WORLD or Td VIEW you can specify the object number of any object, not just object zero.

Now that you understand co-ordinate systems you are all set to read the next section, the AMOS commands. At Voodoo we have been really impressed with the creativity of AMOS users. We are expecting to see some fantastic 3D programs. Please take the trouble to send us your creations. Good luck!

## 8.2: The AMOS commands

AMOS 3D provides a comprehensive set of commands for handling the 3D objects either from the example object library or created by Object Modeller (OM). All the commands begin with the letters Td.

### Positions

All distances are measured in VLU's (Voodoo length units). VLU's do not correspond to any particular physical length. But to give you an idea of scale, a typical object might be 500 to 2000 VLU's across. The standard cube in OM is 360 VLU's on each side.

Objects can be positioned and moved anywhere within a box of 16,000,000 VLU's square. The box is centred around the world coordinate origin (0,0,0) and so the most distant object from the origin would not be further away than (8000000,8000000,8000000). In practice it is best to keep objects quite close to the world origin. If your program uses objects that are fixed in world coordinates this is the natural thing to do.

If you are working with objects which are all moving, the size of the world need not prevent you from writing programs in which you appear to travel for enormous distances. In this kind of 3D work it is usual to *normalise* all positions every few hundred frames. To do this you simply subtract the coordinates of the viewpoint (object 0) from all the objects (including object 0). This simply moves the whole scene to a new location and has no visual effect at all. It merely keeps the numbers manageable. You will find that when you are free to roam in 3D space the numbers can get out of hand quite quickly!

### Angles

These are measured in VRUs or Voodoo Rotation Units. The reason we don't use more normal units such as degrees or radians is to allow fine accuracy while at the same time keeping all numbers in integer form. Decimal (or floating point) numbers are much slower to work with and 3D graphics needs to be *fast*.

One complete circle (or 360 degrees) is 65536 VRUs. This may seem a funny number but it's not. It is the largest number that can be held in one *word* of memory. The following formulae convert from VRUs to degrees and back again:

To convert degrees to VRUs:            multiply by 182

To convert VRUs to degrees:           divide by 182

To specify a direction in 3D space you need at least two angles. To specify an object's attitude (including object 0, the viewpoint) completely, you need three angles. These describe rotation about the X, Y and Z axes. We denote these three angles by the letters A, B and C.

For example to place an object in the world at (1000,2000,3000) pointing straight up we would use:

```
Td OBJECT 1,"object-name",1000,2000,3000,16384,0,0
```

The first parameter is the object number, the second is its name. The next three are its x, y and z coordinates and the last three are its angles, A, B and C.

## Objects

Objects are created using Object Modeller (OM). The data stored when you save an object under OM is called an *Object Definition*. Object definitions are packed tight to save space. When you use the Td commands to display an object, 3D uses the object definition to create a new structure in memory called an *Object Instance*. It is this structure that you address when you use all the other Td commands.

An object instance is a version of the object optimised for speed. You can have several object instances for a single object definition. For example, suppose you design a 3D missile. You will probably want to be able to display more than one missile at a time. To do so you only need load the missile once. After that you can create as many instances as you wish. These are quite independent of each other, but they are all based on the same object definition.

The command that loads an object is Td LOAD. The command that creates the instance is Td OBJECT. When you create an instance you give it an object number. After that, you always refer to the object by its number. This gives you a unique way of addressing different instances of the same object. An object number can be any number you choose between 1 and 20. Object 0 is the viewpoint.

## The display

The view from the viewpoint (object 0) can be displayed on a 16 colour screen up to 256 lines high, see the end of this section for more information on object colours.

You can also make use of multiple screens on the Amiga - see the AMOS Screen Open command. Any 3D objects that you define will be drawn only when you use the Td REDRAW command. When choosing a screen height, bare in mind that this will affect the speed of the 3D system. Bigger screens mean slower graphics although in many situations there may not be a very great difference. If you're a PAL user (256 lines), it's best to keep to a 200 line height. This will allow NTSC users to see your creations.

## Td SCREEN HEIGHT *(Set the screen height for 3D drawing)*

Td SCREEN HEIGHT n

Example:

```
Td SCREEN HEIGHT 130
```

n is in raster lines.

## The Redraw loop

All 3D programs contain a redraw loop. This is a sequence of instructions which sets up a double buffered display and repeatedly redraws all objects. For an explanation of double buffering, see your main AMOS manual.

## **Td REDRAW**     *(Draw all current visible 3D objects)*

Td REDRAW

This draws all current visible 3D objects and any background. You must explicitly tell 3D to refresh the display. All your calculations and object movements must be done in a loop. Here is a typical sequence of instructions for setting up a moving display:

```
RedrawLoop:  
Double Buffer  
Autoback 0  
Repeat  
    Rem Do all your calculations and object positioning here  
    Wait Vbl : Rem This command is optional  
    Td CLS  
    Td Redraw  
    Rem You can draw on top of the 3D objects here  
    Screen Swap  
Until False
```

The Wait Vbl command is sometimes required to prevent flicker in simple programs. You will find that in programs with several objects the Wait Vbl can be dispensed with.

## **Td CLS**     *(Clear the 3D display area with extra speed)*

Td CLS

This command is a fast screen clear for the part of the current screen specified in Td Screen Height. If required you can use this command to avoid erasing any 2D graphics (such as a control panel) below the 3D display.

## **Loading and removing objects**

3D objects, even simple ones, are complex structures. They contain much more than a simple list of points. The structure of an object is described in appendix B but here we must mention that each object is built up from up to three types of disc file: Object, template and surface. The only one of these files that you need to know about is the object file.

3D will load any other files as necessary. The only reason we mention this here is that 3D must know where to find all the files. By default they are held in the directory *objects* which must be in the same directory as your *AMOS\_System* directory.

If you wish to change the name or location of your object directory use the Td DIR command.

## **Td DIR**     *(Set the object directory name)*

Td DIR folder\$

This tells 3D to look in *folder\$* for object files. Naturally the string must be a valid pathname.

Example:

### **Td Dir "Otherobjects"**

Tells 3D to look for its objects in the directory *Otherobjects*, on the current drive.

If you use the AMOS compiler you will need to know about another file called *c3d.lib*. This is the 3D run-time library and contains most of the graphics system. If you distribute compiled programs to your friend the AMOS\_System directory of your disc *must* contain *c3d.lib* or 3D will not work.

### **Td LOAD**     *(Load the named object)*

Td LOAD file\$

Loads the named object. The name should be the same as the one you chose when you designed the object. This command only loads the object. Nothing is displayed. To display a loaded object use Td OBJECT and Td Redraw.

Note that although you supply a single name, 3D may load several files. This is completely automatic. See appendix B for an explanation of the different file types.

### **Td CLEAR ALL**     *(Remove any loaded objects)*

Td CLEAR ALL

Removes any instances of the loaded objects, then removes all the objects. If you have been loading many objects and are no longer using some of them, use this command and reload the ones you need. This will ensure that the maximum amount of memory is free.

Note the difference between Td CLEAR ALL and simply using Td KILL to kill all the object instances. The latter command does not remove the object definitions.

### **Td KEEP ON/OFF**     *(Store or keep loading objects)*

Td KEEP ON  
Td KEEP OFF

While developing large AMOS-3D programs involving lots of objects it is often frustrating waiting for the objects to load each time the program is run.

To speed up development, Td Keep On tells 3D to keep your objects in memory once

they are loaded. Objects will remain in memory for use by any 3D program until a Td Clear All or Td Keep Off instruction is issued.

When using Td Keep On remember that every object you load will use up valuable memory. Even if you start editing a different program, any objects loaded after a Td Keep On will still be present.

Td Keep Off tells 3D not to keep objects in memory, but to load them each time a program is run. Td Keep is Off when 3D starts.

## Invoking objects

### Td OBJECT *(Create an object)*

Td OBJECT *n,name,x,y,z,A,B,C*

Creates an object instance based on a previously loaded object definition. You choose a number *n* between 1 and 20 to refer to the object instance. You also supply details of its starting position and attitude.

<i>n</i>	the object number (between 1 and 20, your choice)
<i>name</i>	the name of the object
<i>x,y,z</i>	the world coordinates of the object's starting position
<i>A,B,C</i>	the attitude of the object (see Angles above)

No objects will be drawn until you execute the Td REDRAW command. Remember that object zero is the viewpoint.

### Td KILL *(Remove an object)*

Td KILL *n*

Removes an object instance. *n* is the object number supplied when the instance was created using Td OBJECT. This command only removes the instance, not the object definition. To remove all instances and all object definitions use Td CLEAR ALL.

## Object movement commands

There are two basic ways of moving an object. You can either place an object at an absolute location in world coordinates or you can specify a change in its position. Aside from Td OBJECT which sets an object's initial position, there is only one command to set absolute coordinates, Td MOVE. The remainder of the movement commands are *object relative*.

For example Td MOVE REL lets you specify a change to be made to an object's current position. This change would be made every time the command was executed. The alternative form of Td MOVE uses the same type of movement string as the ones used to move sprites. If you are unfamiliar with these, consult chapter 14 of your AMOS manual and read the the section entitled *STOS compatible animation commands*.

Like the AMOS sprite commands many of the Td movement commands work on one coordinate at a time. When you use sprites you supply one movement for the x direction

and one for  $y$ . In 3D work there is an extra dimension  $z$  so you will usually need a command for  $x,y$  and  $z$ .

## **Td MOVE**     *(Move an object)*

Td MOVE  $n,x,y,z$

This moves object  $n$  to the absolute position  $(x,y,z)$  in world coordinates.

Example:

**Td Move 4,100,100,3000**

## **Td MOVE REL**     *(Move an object relative to its current position)*

Td MOVE REL  $n,dx,dy,dz$

This command operates in a similar fashion to Td Move. The movement it applies though, is relative to the object's current position. For example:

**Td Move Rel 2,0,100,0**

This will move object 2 a hundred VLU's upwards. If the same command is executed again the object will move another 100 VLU's. If you place a command like this in your main redraw loop (see above) it will have the effect of moving the object in the  $y$  direction with a constant speed. Of course the object would only move up the screen if your viewpoint is *behind* it and pointing in the  $(0,0,0)$  direction. If your viewpoint is above for example and you are looking down onto the object it will appear to be coming straight at you.

## **Td FORWARD**     *(Move an object forwards)*

Td FORWARD  $n,d$

This moves object  $n$  forward  $d$  VLU's each time it is executed. If you place a Td FORWARD command in your main object loop, object  $n$  will move forward with constant speed. The direction *forward* is the direction that the object is pointing. When you design an object using OM, you should save it front forward, pointing straight at you. The attitude of an object when it is saved defines the *forward* direction.

Td FORWARD can be very useful. Because it always moves an object in the direction it is pointing, you can make objects execute smooth turns simply by changing the attitude gradually using Td ANGLE or Td ANGLE REL.

your object closely. However be warned, 3D collision detection is not easy for 3D to do and it takes time. The more zones you have the slower your program will run.

Zones are useful for more than just collision detection. If you define a very large zone around two objects you can use them to detect whether they have come within a certain range. This can help with strategy routines.

One point to remember when using zones is that 3D can only check the zones once per frame, that is, once each time you call Td COLLIDE. If your objects are moving so fast that they pass through one another between one frame and the next, Td COLLIDE might not register an overlap. If you run into this problem, make your zones bigger.

Don't forget that object 0 is the viewpoint. You can set zones around the viewpoint just like any other object.

## **Td SET ZONE** *(Define a zone)*

Td SET ZONE *n,zone,x,y,z,r*

This command defines a invisible spherical zone around an object.

<i>n</i>	the object number
<i>x, y,z</i>	the position of the centre of the zone.
<i>zone</i>	the zone number – 0 for the first zone, 1 for the second and so on.
<i>r</i>	the zone radius

Because the zone is defined relative to the object, we use a local coordinate system centred on the object. To understand this, think of your object as sitting at the centre of a set of 3D axes. Now choose *x, y* and *z* so that the zone surrounds the part of the object you want to be sensitive to collisions. If you are just setting a single zone for each object, *x, y* and *z* should probably all be zero.

When you rotate an object using one of the Angle commands 3D automatically rotates the centres of all the zones as well.

## **=Td COLLIDE** *(Detect a collision)*

This function has two forms:

=Td COLLIDE(*n1,n2*)

This tells you whether objects *n1* and *n2* have collided. If they have the function returns *n2*. Otherwise it returns -1.

=Td COLLIDE(*n*)

This tells you whether any object has collided with object *n*. If there has been a collision the function returns the number of the object it collided with. Otherwise it returns -1. If you have several objects this command is equivalent to calling the first form of Td COLLIDE once for each object other than object *n*. Expect it to take longer. It is wasteful



Td ANGEL REL n,dA,dB,dC

This command changes an object's current attitude by dA, dB and dC. If you place this command in your main redraw loop the object will rotate smoothly.

## Reading an object's attitude

**=Td ATTITUDE** (Return an object's attitude)

=Td ATTITUDE A(n)

=Td ATTITUDE B(n)

=Td ATTITUDE C(n)

Td ATTITUDE is a function. You supply an object number *n* (the one you supplied in Td OBJECT) and Td ATTITUDE returns an angle A,B or C. The angle the function returns depends on which form you use.

## String Commands

Objects can be moved and rotated using string commands similar to those available to AMOS sprites. Consult your AMOS manual for the syntax of these strings. One thing to note about the 3D string commands is that, unlike sprites, it is inappropriate to change the positions and angles of objects under interrupts. The 3D string commands change the positions and angles of objects every time Td Redraw is called.

**Td MOVE** (Set up an animation movement string)

Td MOVE X n,string

Td MOVE Y n,string

Td MOVE Z n,string

This acts on object *n* and applies the movement command in *string*. The movement string follows the same rules as those for sprites. There is a separate command for x, y and z.

Much of the AMAL animation language is inappropriate and only a subset applies. See STOS compatible animation commands in chapter 14 of your AMOS manual.

Example:

```
Td Object 1,"cube",0,0,2000,10000,10000,10000
```

```
Td Move Z 1,"(1,-100,18)(1,100,18)L"
```

```
Rem Place your redraw loop here
```

**Td ANGLE** (Set up an angle animation string)

Td ANGLE A n,angle\$

Td ANGLE B n,angle\$  
Td ANGLE C n,angle\$

Like Td MOVE, there is a separate command for A, B and C. Td ANGLE applies the changes specified by the movement string *angle\$* to either A, B or C.

## Bearing and range

In a good deal of 3D work it is necessary to know the bearing and/or range between one point in space and another. The points might be the centre of an object, the viewpoint or something else.

The *bearing* of one point from another is the direction of the second point as seen from the first. The *range* is the distance between them. For example, suppose you want to program a fire and forget missile. You will know which object you want to shoot down and you will know where you want the missile to start. You still need to know which direction to point it in and perhaps how fast to make it travel. The bearing of the target from the missile's launch position gives you the direction. You might also use the range (the distance between the two) to calculate a suitable speed for the missile.

To specify a bearing in 3D space you only need two angles, not three. To see why, imagine yourself on a rotating gun turret. The turret moves like a swivel chair. The angle of swivel is one of the angles, B. The other angle A is the angle of elevation, that is the angle between the ground and your line of sight to the target.

Once you have these two angles it's easy to make your missile fly in a path to the target; just use Td ANGLE A,B,0 and then Td FORWARD.

### =Td BEARING *(Return a bearing and range)*

There are two forms of this function. Each of the forms, returns either A, B or R (the range), just like Td ATTITUDE. However there is a further refinement that you can use to save time. Whenever you use one of the forms of the function to return A, B or R, Td BEARING actually works out all three and remembers the others (It has to do this because they are all interdependent).

To find out what the others were you simply use Td BEARING again but this time without any parameters. In other words Td BEARING A on its own returns A, as it was the last time Td BEARING was used in full.

The same goes for Td BEARING B and Td BEARING R. The bearing/range calculation is quite a long one so this is well worth doing. The results A,B and R are also calculated by the command Td FACE (below). After using Td FACE you can read A,B or R just as if you had just called Td BEARING.

The first form of this function is:

=Td BEARING A(n1,n2)  
=Td BEARING B(n1,n2)  
=Td BEARING R(n1,n2)

This returns the bearing/range of object  $n2$  from object  $n1$ . Either  $n1$  or  $n2$  can be object 0, the viewpoint.

=Td BEARING A( $n,x,y,z$ )

=Td BEARING B( $n,x,y,z$ )

=Td BEARING R( $n,x,y,z$ )

This returns the bearing/range of the point ( $x,y,z$ ) in world coordinates from object  $n$ . (Remember that object 0 is the viewpoint).

All bearings can be worked out with just one call to the Td Bearing function. Example:

**B1=Td Bearing A(2,3)**

**B2=Td Bearing B**

**B3=Td Bearing R**

**=Td RANGE**     *(Return only the range between two objects)*

=Td RANGE( $n1,n2$ )

This function just returns the range between two objects  $n1$  and  $n2$ , that is the distance between them. It does not calculate any angles. If you need the range and the bearing between two objects, use Td BEARING (which calculates both) instead.

## Pointing an object

**Td FACE**     *(Point an object at another)*

Td FACE is just like Td BEARING. It calculates A, B and R between two objects (or an object and a point). However Td FACE also rotates the first object so that it points to the second object (or point). You can get exactly the same effect as Td FACE by using Td BEARING to find A and B, and then using Td ANGLE.

There are two forms of the command:

Td FACE  $n1,n2$

This points object  $n1$  at object  $n2$ .

Td FACE  $n,x,y,z$

This points object  $n$  at the point ( $x,y,z$ ).

After using Td FACE you can read A,B or R just as if you had just called Td BEARING. (see Td BEARING)

# Converting between coordinate systems

AMOS 3D makes use of three coordinate systems. These are:

- The World Coordinate System.
- Local Coordinate Systems.
- The Screen Coordinate System.

In many situations you will find that you need to convert coordinates from one system into another. For example, suppose you want to make an attacking ship come out of the distance straight ahead of the player. You will know where you want the ship to appear, say 10,000 VLU's in front of the viewpoint; ie (0,0,10000) in local coordinates.

However, when you start a new object using Td OBJECT you specify the world coordinates not the local coordinates. You will need a way of converting between the two. If you think about this conversion you might come to the conclusion that all you would need to do is to add the position of the viewpoint (object 0) to (0,0,10000). This would not be correct because it does not take into account the attitude of the viewpoint.

Fortunately 3D provides functions for doing this (as well as other coordinate system conversions). The conversion functions must be called once for each coordinate x,y and sometimes z. Like Td BEARING you can leave out the parameters after the first call. This is faster for the same reasons.

## **=Td SCREEN** *(Convert world coordinates to screen coordinates)*

```
=Td SCREEN X(x,y,z)
=Td SCREEN Y(x,y,z)
```

This takes the coordinates of a point (x,y,z) in world coordinates and converts them to AMOS screen coordinates.

3D will work out both X and Y values automatically when you call one of these functions. So the quickest way to calculate X and Y is to do the following:

```
SCX=Td Screen X(10,10,1000)
SCY=Td Screen Y
```

## **=Td WORLD** *(Convert local coordinates to world coordinates)*

```
=Td WORLD X(n,x,y,z)
=Td WORLD Y(n,x,y,z)
=Td WORLD Z(n,x,y,z)
```

This takes a point (x,y,z) expressed in local coordinates relative to object n and converts it to world coordinates. The object n does not have to be the viewpoint (object 0). It can be any object. Because of this you can use Td WORLD to get the world coordinates of any point as seen from object n's point of view. For example you could use it to write a routine that makes debris from a ship's engines trail behind it.

As with Td Screen, this command will also generate all X, Y and Z values with just one call. For example:

ZW=Td World Z(5,x5,y5,z5)

XW=Td World X

YW=Td World Y

## =Td VIEW *(Convert world coordinates to local coordinates)*

=Td VIEW X(n,x,y,z)

=Td VIEW Y(n,x,y,z)

=Td VIEW Z(n,x,y,z)

This is the opposite of Td WORLD. It takes a point in world coordinates and converts it to local coordinates, relative to object *n*. This function is most often used with object zero, the viewpoint (hence its name). It can be used with any object.

All three values are worked out with a call to any of these functions. For example:

ZV=Td View Z (3,x3,y3,z3)

YV=Td View Y

XV=Td View X

## Checking an object's visibility

As any graphics programmer knows, speed is all important. In 3D graphics especially you won't want to waste a microsecond drawing anything not absolutely necessary.

3D objects take a long time to process and it takes some time even for 3D to decide that an object can't be seen and therefore should not be drawn. The Td VISIBLE function gives you an easy way of telling whether an object, or any part of it is on screen. If it's not you may be able to delete it and save 3D time. It is surprisingly easy to leave objects hanging around after they have served their useful purpose. **Don't!**

## =Td VISIBLE *(Return whether an object is visible)*

=Td VISIBLE(n)

This function returns 1 if object *n* is visible and 0 if it's not.

## Collision detection and zones

3D collision detection works using zones. A zone is a sphere whose centre is attached to an object. To set up collision detection between two objects you first use Td SET ZONE, at least once for each object. Once you have done this you can use Td COLLIDE to find out whether the two zones have overlapped.

For many purposes a single zone centred on each object is enough. However most objects do not have a very spherical shape and you may want your collision detection to be more accurate. For this reason 3D lets you define many zones around each object, each with its own centre and radius.

If you use several zones you can make a compound zone that hugs the shape of

your object closely. However be warned, 3D collision detection is not easy for 3D to do and it takes time. The more zones you have the slower your program will run.

Zones are useful for more than just collision detection. If you define a very large zone around two objects you can use them to detect whether they have come within a certain range. This can help with strategy routines.

One point to remember when using zones is that 3D can only check the zones once per frame, that is, once each time you call Td COLLIDE. If your objects are moving so fast that they pass through one another between one frame and the next, Td COLLIDE might not register an overlap. If you run into this problem, make your zones bigger.

Don't forget that object 0 is the viewpoint. You can set zones around the viewpoint just like any other object.

## **Td SET ZONE** *(Define a zone)*

Td SET ZONE *n,zone,x,y,z,r*

This command defines a invisible spherical zone around an object.

<i>n</i>	the object number
<i>x, y,z</i>	the position of the centre of the zone.
<i>zone</i>	the zone number – 0 for the first zone, 1 for the second and so on.
<i>r</i>	the zone radius

Because the zone is defined relative to the object, we use a local coordinate system centred on the object. To understand this, think of your object as sitting at the centre of a set of 3D axes. Now choose *x, y* and *z* so that the zone surrounds the part of the object you want to be sensitive to collisions. If you are just setting a single zone for each object, *x, y* and *z* should probably all be zero.

When you rotate an object using one of the Angle commands 3D automatically rotates the centres of all the zones as well.

## **=Td COLLIDE** *(Detect a collision)*

This function has two forms:

=Td COLLIDE(*n1,n2*)

This tells you whether objects *n1* and *n2* have collided. If they have the function returns *n2*. Otherwise it returns -1.

=Td COLLIDE(*n*)

This tells you whether any object has collided with object *n*. If there has been a collision the function returns the number of the object it collided with. Otherwise it returns -1. If you have several objects this command is equivalent to calling the first form of Td COLLIDE once for each object other than object *n*. Expect it to take longer. It is wasteful

to use this form of the command if some of your objects are nowhere near object *n* and therefore won't collide with it. It takes 3D as long to decide that a pair of objects have not collided as it does to register a collision.

## **=Td ZONE**     *(Return information about a zone)*

- =Td ZONE X(*n*,*z*)     Returns the x coordinate of the zone's centre in world coordinates.
- =Td ZONE Y(*n*,*z*)     Returns the y coordinate of the zone's centre in world coordinates.
- =Td ZONE Z(*n*,*z*)     Returns the z coordinate of the zone's centre in world coordinates.
- =Td ZONE R(*n*,*z*)     Returns the zone's radius

This function returns information about zone *z* on object *n*. Td ZONE can be very useful if you want to draw zone circles around objects when you are debugging your program.

## **Td DELETE ZONE**     *(Remove a previously defined zone)*

Td DELETE ZONE on, *zn*

This command removes zones set up using the Td Set Zone instruction. If *zn* is positive or zero the command will remove zone number *zn* from the object number *n*, if zone number *zn* does not exist then an AMOS error will occur.

If *zn* is negative 3D will remove all the collision zones from an object. No error will occur if no zones exist.

## **Animation**

In 3D we use the word *animation* to mean something that changes the appearance of an object, not simply movement. 3D provides commands to perform two completely different types of animation: Shape animation and surface animation.

### **Shape animation**

Shape animation changes the shape of an object by moving one or more of the vertices. These are the corners of the blocks that make up an object. We will refer to them as *points*.

When you design an object using OM you can use the Selection tools to select the points of each block, one after another (in some cases you will need to select a block, face and line first). When a point is selected you can use the Info tool to display the point number. You will see something like:

**P(3,10)**

This tells you that the point you have selected is point 10 (don't worry about the first number). The most useful command Td ANIM REL lets you grab a point and pull it into a new position. You don't specify the new position; instead you say how much you would like it moved.

For example if your object is the pyramid of Figure 6.10, you can pull the top vertex up a little by specifying a change of (0,50,0). To pull it left you would use (-50,0,0) and so on.

Of course by using Td ANIM you can end up with some very strange looking objects.

You can also easily produce objects that don't *work*. For example if you move only a single point on a cube you will bend at least one of the faces. This will confuse 3D and you may get unexpected results. OM applies rules to keep faces flat but there are no such checks on Td ANIM REL. 3D does exactly what you ask, however silly. When you design an animation it's a good idea to try it out under OM first.

A great many good effects can be obtained by moving all the points in a particular block. For example you could make a hatch slide open on a spaceship. To do this you must use the same Td ANIM command for each point.

## **Td ANIM REL**     *(Apply a change to a point - relative to the points position)*

Td ANIM REL *n,p,x,y,z,finished\_flag*

This command applies a change (called a *delta*) to point *p* of object *n*. The change to be applied is specified by *x*, *y* and *z*. The change is applied to the object as it was saved under OM. If you have rotated the object in your program Td ANIM REL will rotate the change too so that the effect on the object is the same.

Often, animation effects involve changing several points. If you are using several Td ANIM REL commands in one place (or in a loop) you can save 3D time by using *finished\_flag*. If *finished\_flag* is 0, 3D expects more ANIMs. When *finished\_flag* = 1, 3D assumes that there are no more ANIMs coming and processes all the points together. In other words, *finished\_flag* should be zero except on the last Td ANIM REL command. Note that every object has its own *finished flag*.

Actually, if you are about to change the attitude of the object before the next Td REDRAW you can keep *finished\_flag* zero even on the last point. This will also save a little time.

One word of warning: Td ANIM REL is there because we want you to be able to animate objects. However not everything that you try will work, especially if you are making big changes to an object. The best way to find out what you can and can't do is to experiment.

## **Td ANIM**     *(Apply a change to a point)*

Td ANIM *n,p,x,y,z,finished\_flag*

This command moves point number *pn* in object *n* to coordinates *x,y,z*. The *finished\_flag* parameter is the same as the flag in Td Anim Rel above.

## **Td ANIM POINT**     *(Return the position of a point)*

=Td ANIM POINT X(*n,pn*)

=Td ANIM POINT Y(*n,pn*)

=Td ANIM POINT Z(*n,pn*)

These three functions return the X,Y and Z coordinates of animation point *pn* in object *n*. Before you animate an object, its points will always be in the position they were in when



the object was saved by OM, regardless of the object's rotation.

X,Y and Z are co-ordinates in the object's local system; they don't change when you move or rotate the object, only when you change its shape.

## Surface animation

Surface animation is much more like the sort of animation you use on sprites. It provides a way of changing the surface detail on a face, 'on the fly' and under program control. If you show a sequence of surfaces in quick succession you can give the appearance of movement.

This feature can be used to create all sorts of interesting effects. For example you could have a damage surface detail that gets attached to the wing of a ship when it gets hit with a missile. This could be a transparent shape with a jagged outline, so as to look as if a hole has been blown in the wing.

Transparent animated surface detail can also be used to open windows or portholes on objects to reveal something inside. You could also add a horrible grinning face to a 3D creature.

Let's discuss the first example, the damage surface detail. We will suppose that you have already designed your ship and a suitable damage surface detail to go on the wing. First of all we will need a way of identifying the face we want to animate. To do this under OM, select the face using the Face Selection tool and click on the Info icon. You will see something like:

**R:1000 B:2 F:3**

This tells us that the face we have selected is face 3 of block 2 (the first number is the object's radius which we don't need here). Now we know where to put our damage detail.

The next stage is to store the damage surface away where it can be accessed by your program. This is done by attaching it to a simple object, say a cube (which can be used to store up to six surfaces). We will never actually display the cube; we will simply use it to hold the surface until we need it. When you do this be sure to make a note of the number of the face holding the surface. Now we can save both objects (the ship and the cube) and quit OM.

Now back to your 3D program. Obviously you will be loading the ship, but you must also Td LOAD the cube. Once the cube is loaded the damage surface is accessible to 3D. To attach the damage surface to your ship, use Td SURFACE with the block and face information you noted earlier. As soon as you do this the surface will appear.

## Td SURFACE *(Copy a surface)*

Td SURFACE name1,b1,f1 to n2,b2,f2,rt

This is the surface copying command.

<i>name1</i>	the name of the source object (the one containing the surface to be copied)
<i>b1</i>	the block number within name1
<i>f1</i>	the face number within b1

<i>n2</i>	the object number of the destination object
<i>b2</i>	the block number within <i>n2</i>
<i>f2</i>	the face number within <i>b2</i>
<i>rt</i>	Rotation angle. Legal values range between 0 and 3.

Notice that the source object is referred to by its name. This allows surfaces to be copied from objects that are merely loaded, not displayed, so as to save memory. The destination object is referred to by its object number, the one you choose in the Td OBJECT command.

Td Surface can be used to do most surface animation, but one additional command may be needed for flat blocks. If you apply a surface detail to a flat block using Td Surface, 3D will automatically fix the four surface detail anchor points so that they are evenly distributed around the block (see the Object Modeller Surface Detail section for an explanation of surface anchor points). If you want to control which anchor points are used for surface detail use the command:

## **Td SURFACE POINTS** *(Set surface detail anchor points)*

Td SURFACE POINTS *p0,p1,p2,p3*

Specifies that point numbers *p0,p1,p2,p3* are to be used as anchor points for all future surface animation on flat blocks. Note that if you specify a point that does not exist in the block, the error will only be detected when you try to apply the surface using Td Surface.

If you want 3D to go back to fixing the surface points automatically, use the command:

## **Td SURFACE POINTS OFF** *(Clear currently defined anchor points)*

Td SURFACE POINTS OFF

After this instruction any previously selected points will be forgotten and 3D will automatically place surfaces evenly on flat blocks. Of course any surfaces you have already attached will continue to use the designated anchor points.

## **Backgrounds**

### **Td BACKGROUND** *(Displaying a background)*

This command lets you place a background behind all your 3D objects. Backgrounds come alive with 3D objects in front of them. You can also use this command to create a landscape or horizon.

Td BACKGROUND *source,x1,y1,width,height to x2,y2 [,plane]*

*source*            screen number containing the image(s)

x1,y1	x,y coordinates of the image in the source screen
width	width of the image
height	height of the image
x2,y2	screen coordinates of the image on the current AMOS screen. You can use this to move the image around, 3D will handle the clipping for you.

### Optional parameter

plane	Specifies the destination screen bitplane to start drawing the background. If omitted <i>plane</i> is 0.
-------	--

The destination screen is always the currently selected AMOS screen (usually the screen on which you are drawing your 3D objects). The number of bitplanes drawn by Td Background is read automatically from the source screen. Each bitplane takes some time to draw, so if you can get away with using a 2,4 or 8 colour background (1,2 and 3 bitplanes respectively) your programs will run much faster. See the Screen Open section of your AMOS manual to find out how to open a screen with 2,4 or 8 colours.

Usually you will want to draw backgrounds behind 3D objects, in order to do so use the Td Background command **after** the Td Redraw instruction.

If you want to draw an image in front of 3D objects, for example the circular windscreen of a spaceship, use Td Background **before** the Td Redraw. The image must contain only colours 8-15 and 0, you will be able to see your 3D objects where the image is colour 0.

The optional plane parameter tells 3D to start drawing a background at a given bitplane. Naturally you can't tell Td Background to draw to more bitplanes than there are in the destination screen.

## Background colours

3D draws objects in colours 8-14. Colour 15 is reserved for your special graphics (for example sights). Background can be drawn in any colours.

If you draw in colours 8-15 before you use Td REDRAW, 3D will place objects behind and not overwrite your image. In other words, bitplane 3 is used as a mask by 3D. The same applies to backgrounds.

## Display problems

When programming in 3D you may sometimes find that what appears on the screen is not what you expected. If this happens your problem could be one of the following.

## Interleaving objects

When you design objects under OM you have the freedom to place blocks wherever you like. You could have two letter U shapes, made out of eight pointed blocks which fit inside each other for example. We would say that these shapes are *interleaved*.

With whole objects this is not possible. Each one must be drawn as a whole by 3D. They must not be mixed up together. If you are uncertain, there is a simple test that will tell you whether two objects are interleaved:

Imagine wrapping the objects up tightly in cellophane so that it stretches in flat planes

over all the protruding vertices. If you did this to a letter T for example you would get a roughly triangular shape. Now ask whether or not the cellophane surrounding the two objects will touch. If it does the objects are interleaved and you may get unwanted effects at certain angles.

## Object precedence

When you design an object under OM and use the precedence tool to order the blocks, the modeller prepares all the information necessary to generate correct views of the object at any attitude. Without this 3D would not know which blocks to draw in front and which behind (actually 3D has to chop up blocks too sometimes!). This information depends on the precise position of each block and takes some time to prepare. When you display several objects under AMOS, 3D also calculates the precedence between whole objects. To keep 3D running fast, a more approximate method is used to order the objects than the one used to order the blocks within objects. This method is based on the centre of gravity of each object. If the centre of one object is further away than the centre of another the first is drawn behind the second.

Remember that the centre of an object is the point about which it rotates. It is also the *handle* that you use to position objects using the Td positioning commands.

The centre can be positioned anywhere inside or outside the object by using either OM's Centring tool or the Group movement tools.

Occasionally you can run into situations where this ordering method produces incorrect results. If you come across this problem, with an object appearing in front when it should be behind, consider the positions of the objects' centres. You will probably find that the object which visually should be in front actually has its centre behind.

You can get around the problem by giving one of the objects a centre position which guarantees that it will appear behind when it should. Remember that the centre can be anywhere, outside the object as well as inside.

## Memory

3D needs at least 90k of memory. The basic memory is allocated (if it's not allocated already) by the first use of a 3D command. This 90k is actually very little for a 3D system with the power of 3D, Nonetheless there will be times when you will want to release the storage for other things. You can do this using Td QUIT.

**=Td QUIT**     (*Release memory*)

=Td QUIT

Unload the 3D extensions along with all objects and release all 3D memory.

**Td ADVANCED**     (*Access to the object structures*)

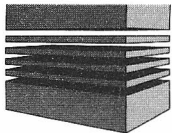
TD ADVANCED

Td ADVANCED is provided for advanced programmers who wish to experiment with the actual 3D object structures in memory. The Td Object command builds such a structure

called an *Object Frame* for each instance. It contains a basic block of data defining the instance including a pointer to each block structure known (as the Layers). Td ADVANCED can also be used to obtain the base address of 3D's static data area.

=Td Advanced n

If  $n$  is zero the function returns the 3D data segment address. Otherwise it returns the address of the Frame for object  $n$ . Needless to say we can't predict the results of monkeying with the 3D structures.



# 9: Hints & Tips

This section contains advice on 3D programming and how to make the most of the OM and TD commands. The information contained here will help you to make your 3D programs fast and to keep them interesting. Some of the topics covered are quite advanced and will not concern everybody. It is not hard to create interesting demos and games using 3D but of course, to mimic the best 3D games takes patience and experience.

## Speed

Good 3D graphics should be as fast as possible. Regardless of whether your application is actually a fast moving arcade game or not you should do everything you can to keep the frame rate up (the number of pictures per second). This will make your graphics smoother and more enjoyable to look at.

3D graphics takes time. It takes 3D a certain amount of time to draw every block of every 3D object. It also takes some time just to decide not to draw an object or a block that is off screen. The actual amount of time taken depends very much on the circumstances. Here are some facts:

## Distant objects

If an object is a long way from the viewpoint you may see little more than a dot. Nonetheless, 3D will still have a lot to do and this will take some time. If your application contains sequences where objects get very small, use depth culling, which is a way of stripping objects of unnecessary detail when this is small compared with screen resolution. OM can set up depth culling for you.

## Objects in the middle distance

Lots of applications contain objects that are seen in the middle distance, that is where they cover a smallish area of the screen. These are much faster than objects that are close up. The time taken to draw an object is related to the screen area it covers. Bear in mind that if you halve the distance of the viewpoint to an object, its screen area will increase by 4.

Ideally objects should spend most of their time in the middle distance, only occasionally getting close to the viewpoint. Don't let too many objects come close at a time.

## Close objects

These take the longest amount of time to draw. They are objects that cover most of the screen and require lots of clipping. This all takes time. Of course it adds drama to a game when the objects come close enough to touch. However it is often even more dramatic if they don't stay there for too long. Even if your frame rate slows down a little as a ship rushes past the viewpoint, you will still retain the impression of smooth graphics as it speeds up again quickly.

## Invisible objects

Objects can be invisible because they are outside the field of vision or because they are obscured by other objects. In the first case, 3D can often tell quite early on as it processes an object that it won't be visible and can save most of the calculation (this is known as *first rejection*). If it's only just outside the field of vision though *first rejection* may not be so easy. If you know that an object is not going to be seen, move it well out of the way (or kill it completely).

In the second case, where one object obscures another, there is less that 3D can do. In the worst cases 3D may draw the more distant object completely only to cover it up again with the closer one! One of the best ways to waste time is by having several objects close up which obscure one another.

## Object complexity

The time it takes to render an object rises steeply with its complexity. A six-block object may take more than twice the time of two three-block objects. You should look on blocks as being like gold dust. Use the absolute minimum and when you do decide to use an extra block get full value out of it. For example avoid objects where blocks have many points and lines in common. Objects often look complex and interesting according to how *busy* they are. Try to design objects with lots of visible faces. Actually, with a little thought you can use surface detail transparencies to increase the apparent complexity of objects with very few blocks. Surface detail is very fast especially if it's simple. Look at the example objects and take them apart. See how economically the designer has used his blocks.

## The Compiler

When you first start writing a 3D game you will probably find the speed of 3D sufficient. After all the 3D code itself is among the fastest there is. As your program grows though you will probably find that things start to slow down. AMOS is a very fast BASIC but it still has an enormous amount to do. Every statement has to be interpreted each time it is executed. That is where the compiler comes in. It won't speed up the actual 3D drawing but it will cut down drastically on that interpretation overhead, and a lot more besides. Aside from that, be sure that you are using the fastest AMOS commands. Use integers rather than floating point (3D itself uses integers throughout). If you need slow trigonometric functions like Sin and Cos create a table and use that rather than the functions themselves.

## Keeping a game busy

In a 2D game you know where you stand. If you draw a sprite you usually know that the player can see it. 3D objects are not like that – you have to make sure they can be seen. It's easy to arrange beautiful flight paths and attack sequences for your ships, only to find that the player is miles away or looking in the other direction.

The answer is to bring the mountain to Mohammed and not the other way round. Many arcade style games contain a routine which generates new characters as others die. A 3D version of this type of routine should find out the position and attitude of the viewpoint and generate new objects somewhere that they will be seen, perhaps

appearing out of the distance or from one side. A good technique is to check all your objects regularly to see whether they are visible or likely to become so.

If an object is miles away kill it off and generate a new one, or better still, simply move it in one jump to a position just off stage. It is often very effective to have your objects tend to head for the viewpoint when they're not doing anything else. This will at least keep them in range.

If you want to give an impression that 3D space is filled with objects you don't actually have to do this. All you have to do is fill the portion of space that the player can see. In other words, don't imagine that you can set up a 3D world full of objects for the player to explore. No home computer is anything like powerful enough for this. Some games give this impression but they do it by the sort of trickery we have been discussing.

## Housekeeping

All games have to do some housekeeping. Numbers must be kept in range, dead entries deleted from arrays and so on. A typical piece of 3D housekeeping is to check every so often that objects have not wandered off into the infinity of 3D space. This sort of routine does not need to be called every frame. Before you put code into your redraw loop ask yourself whether it really needs to be executed that often. If the code is more than a statement or two it's probably much cheaper to increment a counter and do a job once every few frames. A lot of games would be faster if this technique was used more often.

## Smooth movement

Although 3D is fast, no 3D system can be as fast as 2D graphics such as sprites and scrolling backgrounds. Smooth movement depends on many things but a display will always look jerky if objects move too far across the screen between one frame and the next.

3D objects look great coming out of the distance. They can also look good passing across the field of vision. It is bad practice though to have objects executing a path which is largely on screen and in which they are displayed at very different positions in consecutive frames. Even if the frame rate is high this will look jerky because the eye is not getting enough information to construct the illusion of movement.

Another very important point that has considerable bearing on smoothness concerns how you handle velocity. In 3D graphics every frame takes a different amount of time to draw. If you make an object move by adding the same amount to its position every frame it will constantly appear to be changing speed. Remember that  $distance = velocity * time$ . To obtain smooth constant velocity you should time each frame and use a number proportional to this for your frame by frame position change. Even this is not quite accurate because you cannot know how long a frame will take until it is over. Nonetheless the method does usually produce good results.

## Flight paths

The following is a very useful technique for generating interesting flight paths for objects in 3D space. Suppose you want to program a dog fight between two ships A and B (one might be the viewpoint). Start the objects off some distance from one another and give them some reasonable velocity in a random direction using the TD Forward command in



your redraw loop. Every 10 frames or so take the bearing of A from B. Now, over a few frames gradually adjust B's attitude to that it ends up facing A. Do exactly the same thing the other way around for the other object, but not on the same time scale, say every 13 frames. With a little experimentation and adjustment you can achieve some very graceful effects. Add a little randomisation to the time intervals, velocities and so on for variety.

## Normalisation

Normalisation is a way of keeping a game's action from straying too far away from the work origin. This is sometimes necessary if you want to keep the numbers reasonable and not stray out of 3D's world. Normalisation depends on the fact that an arrangement of 3D objects is completely unaffected if everything is shifted in space to a different location (of course the viewpoint must be shifted too). Normalisation can be done as often as you like but every 100 frames or so is usual. Simply subtract the coordinates of the viewpoint  $x$ ,  $y$  and  $z$  from the position of every object and set the viewpoint's position to  $(0,0,0)$ .

## Viewpoint control

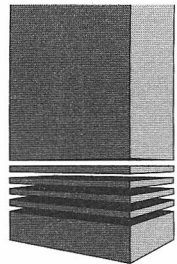
This is the method you use to control the position and attitude of the viewpoint with the mouse or joystick, and it can be a tricky business. Unless you are an expert don't attempt to write a control system which gives the operator freedom to point and travel in any direction. A little experimentation will show you why.

Limit the view angle. Allow the viewpoint to rotate about the  $y$ -axis freely but prevent it from looking straight up or down or nearly so. Instead, move the viewpoint up and down physically. This will give an intuitive feel to the controls. Think carefully about how you want the mouse to affect the viewpoint's attitude. Should a movement of the mouse to the left turn the view through a certain angle and then stop? Or should it start the view spinning until the mouse moves back?

In fact neither of these is very satisfactory. A jerk on the mouse should start the viewpoint spinning but the rotation should be heavily *damped* and not carry on forever. For best results, use a combination. Move the viewpoint through an angle directly and give it a little damped rotational velocity.

## Relative velocities

A problem that you may run into concerns the relative velocities of objects and the viewpoint. If the viewpoint is allowed to travel too fast it will be uncontrollable and objects will flash by before you have time to react. If the viewpoint can't move fast enough it will seem to take forever to get anywhere. You should think carefully about the distances you work with. A typical object should be possible to reach in a reasonable time at a speed which does not cause it to flash by once you get there.



# Appendix A

## Making copies of OM

OM is a complex program and consists of more than just a single program file. The OM disc will autoboot, but it can be used as an auxiliary disc when you have booted from your favourite system disc. OM can also be copied to another directory on a floppy or on a hard disc.

If you wish to make another autoboot OM disc, simply copy the disc in the usual way. Otherwise, here is what to do:

### Copying OM to a directory on another floppy or hard disc

- Make a new directory on the destination disc and copy everything in the OM directory of the OM disc to it. If you wish you can also copy the example object directory *examples* which is located within the OM directory. From the CLI the command:

```
AnigaDOS
>copy OM:om to dirname
```

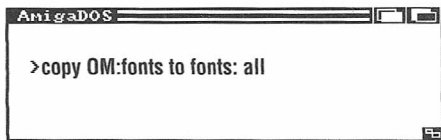
will copy OM alone. *dirname* is the name of your new OM directory. The command:

```
AnigaDOS
>copy OM: to dirname all
```

will copy the examples directory as well.

- Add the OM font to your fonts directory. The OM font is called *xfont* and is located in the fonts directory of the OM disc. A font consists of a file, in this case *xfont.font* and a directory containing a file for each point size. In this case the directory is called *xfont* and the only file within it is called '8'.

It is essential that you copy the font; OM won't work without it! If you are using one or more floppy drives, your boot disc will contain a directory called *fonts*. This will have the device name `FONTS:` assigned to it. From the CLI, the command

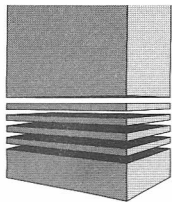
A screenshot of a command prompt window titled "AmigaDOS". The window has a standard graphical interface with a title bar, a maximize button, and a close button. The main area of the window contains the command prompt text: ">copy OM:fonts to fonts: all".

will copy the font.

## Setting the ID

There is one further thing that you must do every time you make a new OM directory, whether by copying the whole disc or by installing OM in a hard or floppy directory.

To make sure that OM doesn't get confused about surface files (see Appendix B - File Structure) you must also run the utility program `SID` (or `SetID`). AFTER you have copied the OM directory, make it the current one, type `SID` and enter a unique two character identifier when requested. This identifier will form the first two characters of all surfaces generated with that copy of OM.



# Appendix B

## File structure

If you are using 3D at all seriously, or you wish to swap objects with friends, you should read the following.

The example objects and the objects that you design yourself appear on the disc directory as single files. When you load an object you supply the filename and OM or AMOS does the rest. In fact objects are made up of several components, each with its own disc file. If you are interested in the reason for this it is explained below.

There are three types of file in all. Object files, Template files and Surface files. You can tell which are which by the three character extension which follows the file names; these are **.3DO** for objects, **.3DT** for templates and **.3DS** for surfaces.

When you refer to an object in 3D you don't need to supply the extension. 3D adds it for you (this is a little like many word processors which add an extension like **.DOC** to the end of file names). You also don't need to know about templates or surfaces as 3D takes care of all that too. The only fact that you should be aware of is that the surface and template files must be in the directory containing your objects. If you copy an object you will have to copy its surfaces and templates too. A simpler way to copy an object is to load it under OM and then save it again to another disc or directory.

### Surface files (.3DS)

These contain the surface details that you design to decorate the faces of your objects. There is one surface file for each unique picture that you design. When you copy a surface from one object to another OM uses the same surface file for both objects.

### Template files (.3DT)

The purpose of these is a little less obvious. They contain details of the structure of the basic building blocks. There is one for each of OM's basic block types and there are others which go with some of the example objects. Once again, when you grab a block from an existing object to use in a new one 3D shares the template rather than creating a duplicate. Templates describe basic block structure. When you customise a block using the OM tools, you don't affect the template itself, only the way it is used in your object.

Here is a summary of the three types of file:

Type	File extension	Contents
Object	.3DO	All information having to do with the object as a whole, along with the names of all associated files.
Template	.3DT	Information relating to an individual block type. There is a reference to a template for every block in an object.

Surface .3DS A description of an individual surface detail. There is a reference to a surface for each decorated face in an object.

As you can see, the information describing an object may be spread over several files. One might well ask why this is not held in a single file; the reason is as follows:

The relationship between objects and object files is 'one to one', that is there is one object file for every object and visa versa. The corresponding relationship between objects and surfaces however is many to many. For example an object can refer to many surfaces and many objects can refer to the same surface. The same goes for templates. It would be wasteful to use up disc space and memory by holding the same surface or template over and over again for different objects (or the same object). For this reason 3D keeps only one copy of each. When you load an object under OM or AMOS, 3D checks to see whether it already has a copy of its surfaces and templates before loading them. Of course the same block or surface may not look the same in different objects but to 3D they are essentially the same.

The file structure of objects has important implications when it comes to copying or deleting objects because template and surface files must also be copied for the object to be complete. This is the reason for the OL (Object Look) utility which asks for an object name and lists its templates and surfaces so that these can be copied or deleted. An alternative way of copying an object is to load it into OM and then save it again to another disc or directory.

## Surface names

It would be very tiresome if, every time you saved an object you had to invent names for all its surfaces. It would also be annoying if you had to supply a name every time you created or changed a surface. It would also be unsatisfactory to name surfaces after their objects (perhaps with an extension) because many objects can have the same surface (for example the damage surface detail described under Td Surface). Instead OM generates names for surfaces based on sequential numbering. These have the form

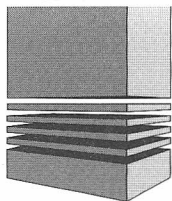
**<2-character-identifier><sequence-number>.3DS**

Both the two character identifier and the current sequence number is held in the OM directory in a file called *id*. The file contains the following

**dirid(xx),surgen(yy)**

where xx is your identifier and yy is the sequence number of the last surface saved.

**WARNING:** If you wish to swap objects with someone who also has a copy of 3D, or you simply want to keep several OM directories it is important to use different identifiers. This can be done either by editing the file or by using the supplied utility **SID** (Set Identifier).



# Appendix C

## The utilities

3D comes with three utilities to help you maintain your Object files. These programs are all stored in the C directory of the OM disc and should therefore be run from the CLI. It will help you to understand the utilities better if you first read the appendix on File Structure.

### OL (Object Look)

OL accepts an object name, examines the .3DO file and reports the names of the templates and surfaces associated with it. The template and surface files will have the extensions .3DT and .3DS respectively. When you delete an object, don't delete its templates and surfaces unless you are certain that they are not referenced by any other object. Templates are few in number and should not be deleted. Unreferenced surfaces can be removed with the PRUNE utility.

OL must be run from the CLI in the directory containing the object (.3DO) file. You can follow the OL command directly with the name of the object you wish to examine, or simply type OL and have the program ask you for the object name. Example call from the CLI:

```
AmigaDOS
1> CD OM:om/examples
1> OL
object name : struct
Templates:   p8,p5
Surfaces:   s0.139.2,s0.139.4,s0.139.10
```

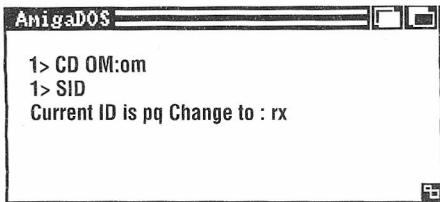
For the regular AMOS users, the same program can be found on your installed disc - Object\_Look.AMOS. You'll find this easier to use as it sports a handy AMOS file selector.

### SID (Set Identifier)

SID allows you to choose the two character identifier on which OM bases the surface names that it generates. It reports the current identifier and asks for a new one. It then updates the ID file.

The ID file is located in the OM directory and not in directories containing the objects themselves. It is very important that each copy of the OM directory contains a unique ID file. If two ID's are the same OM will get confused about which surfaces are which and may overwrite existing ones.

SID should be run from the CLI in the OM directory. Example call from the CLI (sets ID to rx):



```
AnigaDOS
1> CD OM:om
1> SID
Current ID is pq Change to : rx
```

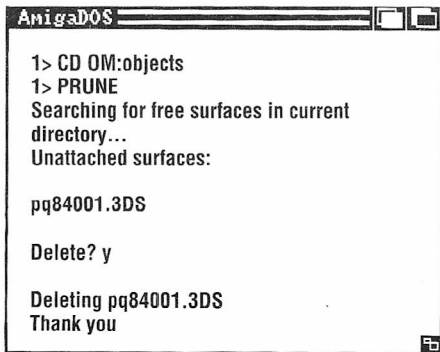
## PRUNE (Clean up an object directory)

In 3D, each surface detail has a separate .3DS file. If any of the surfaces belonging to a given object have been copied to other objects, the surface will become multi-user; it will be shared by several objects.

When an object is deleted its surfaces should be left intact in case they belong to other objects as well. Because of this, an object directory that has been in use for a time can accumulate 'orphan' surfaces that are not attached to any object. This can happen either because all associated objects have been deleted or because surfaces have been re-edited, leaving the old ones without an owner.

The PRUNE program looks at every object and surface in the current directory and cross references them to build a list of any unattached surfaces. If it finds any, it will ask whether to delete them. PRUNE will also list any objects that contain non-existent surfaces (see note below).

PRUNE should be run from the CLI in the directory containing the objects to be examined. Example call from the CLI:



```
AnigaDOS
1> CD OM:objects
1> PRUNE
Searching for free surfaces in current
directory...
Unattached surfaces:

pq84001.3DS

Delete? y

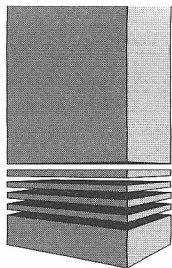
Deleting pq84001.3DS
Thank you
```

## **Missing surface files**

We all make mistakes and sometimes a surface can get deleted accidentally. If this happens, any object which uses it will fail to load, either under OM or via Td Load. There is no way of getting the deleted surface back but you can make the object loadable again by supplying a surface of the same name as the deleted one.

The best way to do this is to run PRUNE which will report all missing surfaces. Once you know the names of your missing surfaces you can copy any existing surface giving the new copy the name of the missing one. The object should now load with the copied surface in the place of the lost one.





# Appendix D

## AMOS 3D error messages

**3d background source screen is current screen:** The source screen for Td background can't be the current screen.

**AMOS screen not compatible with 3d:** 3D will only work with screens that are 200 pixels wide, at least as high as the 3D screen height and have at least 16 colours.

**Bad Object/Template/Surface file:** The specified file is either corrupted or is not a valid 3D file.

**Block does not exist:** You have specified a block number that does not exist in the given object.

**Can't change screen size while objects exist:** You must kill off all your objects before trying to change the size of the screen.

**Can't load 3d code:** The 3D extension can't find the main 3D code file *c3d.lib*. Check your AMOS .Env file and make sure *c3d.lib* is present in the AMOS\_System directory.

**Directory string too long:** The pathname you have supplied to the TD Dir command is too long.

**Face does not exist:** You have specified a face number that does not exist in the given object.

**Invalid 3d screen size:** The 3D screen height ranges from 1 to 256 lines.

**Invalid object number:** Valid object numbers range from 0 to 20. Object 0 is the viewpoint, it can't be created or killed.

**Not enough memory for 3D:** 3D has run out of memory.

**Object already exists:** You have tried to invoke an object which already exists.

**Object already loaded:** You have tried to load the same object twice. This message will never occur if Td Keep is On.

**Object does not exist:** You have specified the object number of an object which does not exist.

**Object file not found:** 3D can't find an object file you require. Check the object name is correct and, if you have changed the 3D object directory, check the directory you have specified is correct.

**Object not loaded:** The object name you have specified has not been loaded using Td Load

**Point does not exist:** The point you have specified does not exist in this object. Check the object number and the point number.

**Surface file not found:** 3D can't find an object's surface file. Use the OL utility to list out all the files the object uses.

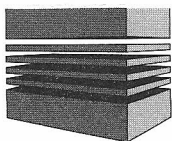
**Syntax error in string:** A movement or angle string is incorrect.

**Template file not found:** 3D can't find an object's template file. Use the OL utility to list out all the files the object uses.

**Too many objects:** There are too many objects loaded and 3D can't cope! This message should never occur, if it does check you haven't got lots of unused objects loaded.

**Too many planes for 3d background:** The background source screen has more planes than the current screen.

**Zone parameter(s) out of range:** The 3D collision zone you have specified is too big.



# Glossary

Throughout this manual we have tried to keep the language free of unnecessary jargon. All the same, 3D graphics is quite a technical subject and so we have provided this glossary to explain unfamiliar terms.

**2D:** A space with two dimensions. A flat piece of paper or a computer display is an example of a 2D space.

**3D:** A space with three dimensions. The physical world is an example of 3D space.

**ANGLES A,B,C:** These are the angles used in Td angle commands.

A is the angle about the x-axis

B is the angle about the y-axis

C is the angle about the z-axis

Angles are measured in VRU's (Voodoo Rotation Units)

**Axis:** A scale on a coordinate system used to measure distances in a particular direction. In 3D we use three AXES, the x-axis running left to right, the y-axis, drawn vertical and the z-axis pointing into the distance. The word is also used to describe a line (an imaginary one) about which a point or object is rotated.

**Block:** One of the basic shapes used to build 3D objects

**Coordinate system:** A set of AXES usually set at right angles to one another. A coordinate system gives you a way to express the position of points and objects.

**Depth culling:** A method of reducing the complexity of objects as they get further away. Culling is used in 3D to speed up the display of distant objects.

**Dimension:** The direction measured by a given axis in a coordinate system.

**Double buffering:** A method of graphics rendering in which two screens are used. One screen is displayed while a new scene is prepared on the other. When the new scene is complete, the hidden screen is displayed and drawing begins on the other and so on.

**Face:** One of the flat areas on a block.

**Graph:** A representation of a coordinate system, usually in two dimensions.

**Group:** One or more of the blocks comprising an object can be selected as a group. Many of OM's commands work on groups.

**Line:** A straight edge bordering a face.

**Local coordinates:** A set of coordinates based around a particular object. The Observer Coordinate System is an example of local coordinates.

**Object:** This word is used in the 3D manual to refer to the fundamental unit of 3D design. An object consists of blocks and surface details.

**Object component:** The parts of an object.

**Observer coordinates:** A coordinate system based on the 3D *camera*.

**Origin:** The point at which the axes meet in a coordinate system.

**Perspective:** A way of drawing 3D scenes on a flat surface like a piece of paper or a computer display. 3D generates *perspective views*. There are other ways of doing the same thing, for example archetechet's 'Orthographic' projection. Perspective is the most realistic because it is a perspective image that appears on the eye's retina.

**Point:** The place where two or more lines meet. In this context a point is the same as a vertex.

**Projection:** The result of transferring a point or object from a coordinate system into a system with a different number of dimensions.

**Rotation:** The process of moving a point or an object through an angle relative to an AXIS OF ROTATION.

**Shelf:** An area of the OM graphics screen used to hold objects.

**Surface detail:** A *picture* which can be attached to one or more faces. Each surface detailed designed with OM and attached to a saved object is written to a disc file which ends in the extension .3DS.

**Td:** The word Td (Three Dee) precedes all the AMOS 3D commands.

**Template:** A disc file containing information concerning the structure of a block. The names of all templates begin with the letter *p* or *f* (for flat), are followed by a number equal to the number of points in the shape and end with the extension .3DT. Every block in a 3D object refers to a template.

**Transformation:** A transformation is the calculation applied to a point to find its position in a different coordinate system or to find its position after an operation such as rotation.

**Translation:** Translation means *change in position*.

**Vertex / vertices:** Any place on an object where lines meet at a point, for example the corners of a cube.

**Viewpoint:** The point in 3D space from which 3D *views* its internal world. In 3D the viewpoint is an object like any other and has object number zero.

**VLU:** VLUs or Voodoo Length Units are used in 3D to measure lengths.

**VRU:** VRUs or Voodoo Rotation Units are used in 3D to measure angles. 65536 VRUs is the same as 360 degrees.

**World coordinates:** A coordinate system used to represent the whole 3D world.



## Total Map Editor For Your Amiga

Now you can design your own map based games like Rainbow Islands and Gauntlet.

TOME consists of a new powerful map editor and an AMOS extension which provides 27 new commands for AMOS.

First you can create the tiles from 16x16 up to 32x32 pixels in size using the MIni Art package included. Then use the icon-driven editor with its 51 main functions to create screen layouts or multi-screen maps up to 960 screens in size (which only occupy 64k of memory!).

Now you can call up AMOS and scroll around the maps using the 27 commands at your disposal.

TOME has been created by Aaron Fothergill, editor of the very successful AMOS Club. Please send a cheque or postal order for £24.99 (£19.99 for AMOS Club members) payable to Shadow Software at 1 Lower Moor, Whiddon Valley, Barnstable, North London EX32 8NW.



*This is not a MANDARIN product. Shadow Software are solely responsible for this product and its customer support.*

MANDARIN