Hands On DarkBASIC Pro

Volume 2

A Self-Study Guide to Games Programming

Alistair Stewart

Hands On DarkBASIC Pro

Volume 2

A Self-Study Guide to Games Programming

Alistair Stewart

DIGITAL SKILLS
Milton
Barr
Girvan
Ayrshire
KA26 9TY

www.digital-skills.co.uk

DarkBASIC Professional is produced by The Game Creators Ltd

Printed September 2006

Title : Hands On DarkBASIC Pro Volume 2

ISBN-10 : 1-874107-09-2
ISBN-13 : 978-1-874107-09-5

Other Titles Available:

Hands On DarkBASIC Pro Volume 1
Hands On Pascal
Hands On C++
Hands On Java
Hands On XHTML

TABLE OF CONTENTS

Chapter 30 3D Concepts and Terminology

Chapter 31 3D Primitives

Chapter 32 Texturing

Chapter 33 Cameras

Chapter 34 Lighting

Chapter 35 Meshes and Limbs

Chapter 36 Importing 3D Objects

Chapter 37 Screen Control

Chapter 38 Solitaire

Chapter 39 Advanced Lighting and Texturing

Chapter 40 Collisions

Chapter 41 Particles

Chapter 42 The Elevators Game

Chapter 43 Handling BSP Models

Chapter 44 Creating Terrain

Chapter 45 Using Matrices

Chapter 46 Manipulating Vertices

Chapter 47 Accessing Memory

Chapter 48 Open Dynamics Engine

Chapter 49 Vectors and Matrices

Chapter 50 Shaders

Chapter 51 Network Programming

Chapter 52 Using File Transfer Protocol

Chapter 53 Dynamic Link Libraries

Acknowledgements

I would like to thank all those who helped me prepare the final draft of this book.

In particular, Virginia Marshall who proof-read the original script and Michael Kerr who did an excellent job of checking the technical contents. Mark Armstrong researched all the difficult bits for me and produced almost as much in the way of notes as is in this final text.

Any errors that remain are probably due to the usual extra paragraphs I added after all the proof-reading was complete!

Thanks also to The Game Creators Ltd for producing an excellent piece of software - DarkBASIC Professional.

Many of the 3D models and textures are from The Game Creators Dark Matter 1 package and used with their kind permission.

Finally, thank you to every one of you who has bought this book. Any constructive comments would be most welcome.

Email me at *alistair@digital-skills.co.uk.*

Introduction

Welcome to the second volume of a book that I hope is a little different from any other you've come across before. Instead of just telling you about software design and programming, it makes you get involved. There's plenty of work for you to do since the book is full of exercises - most of them programming exercises - but you also get a full set of solutions, just in case you get stuck!

If you've worked your way through Volume 1, then you should have gained a good grounding in, not only DarkBASIC Pro, but also professional programming skills.

Most of Volume 2 is dedicted to 3D graphics but there a few other interesting topics such as network programming and how to create your own DarkBASIC commands.

Learn by Doing

The only way to become a programming expert is to practice. No one ever learned any skill by just reading about it! Hence, this is not a text book where you can just sit back in a passive way and read from cover to cover whilst sitting in your favourite chair. Rather it is designed as a teaching package in which you will do most of the work.

The tasks embedded in the text are included to test your understanding of what has gone before and as a method of helping you retain the knowledge you have gained. It is therefore important that you tackle each task as you come to it. Also, many of the programming exercises are referred to, or expanded, in later pages so it is important that you are familar with the code concerned.

What You Need

You'll obviously need a PC and a copy of DarkBASIC Pro.

At this stage you'll also need some programming skills and a basic knowledge of DarkBASIC Pro.

How to Get the Most out of this Text

Experience has shown that readers derive most benefit from this material by approaching its study in an organised way. The following strategy for study is highly recommended:

1. Read a chapter or section through without taking notes or worrying too much about topics that are not immediately clear to you. This will give you an overview of the contents of that chapter/section.

2. Re-read the chapter. This time take things slowly; make notes and summaries of the material you are reading (even if you understand the material, making notes helps to retain the facts in your long-term memory); re-read any parts you are unclear about.

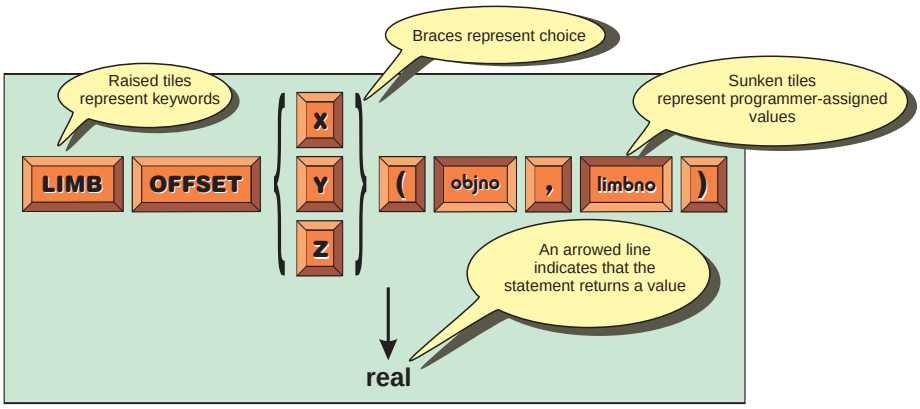
3. Embedded in the material are a series of activities. Do each task as you reach it (on the second reading). These activities are designed to test your knowledge and understanding of what has gone before. Do not be tempted to skip over them, promise to come back to them later, or to

make only a half-hearted attempt at tackling them before looking up the answer (there are solutions at the end of each chapter). Once you have attempted a task, look at the solution given. Often there will be important points emphasised in the solution which will aid higher understanding.

4. As you progress through the book, go back and re-read earlier chapters, since you will often get something new from them as your knowledge increases.

Syntax Diagrams

The format of each statement is explained using a syntax diagram. Raised tiles represent keywords of the language while sunken tiles are parts of the statement for which you are free to create your own values. Parts within square brackets are optional while braces represent a choice of options. Statements that return a value show this using an arrowed line and the type of value returned.



Line Continuation Symbol

Occasionally, a single programming instruction has to be split over two or more lines because of limited page width. In such cases the second line(and subsequent lines) begins with the ↳ symbol. For example, the instruction

```
POSITION OBJECT 2,OBJECT POSITION X(2),OBJECT POSITION Y(2)-0.1,OBJECT POSITION Z(2)+0.1
```

might appear as

```
POSITION OBJECT 2,OBJECT POSITION X(2),
↳OBJECT POSITION Y(2)-0.1,OBJECT POSITION Z(2)+0.1
```

In such cases you should enter the code as a single line when creating a DarkBASIC Pro program.

3D

3D Concepts and Terminology

The 3D World

Introduction

Welcome to the world of 3D. Of course, we can create great games in 2D - many people still consider 2D games like Space Invaders and Pac-Man to be some of the best games ever invented - but for sheer eye candy you really can't beat 3D.

In this chapter we'll get a broad view of the 3D world created by computers. We'll cover the basic concepts and define some of the terms. Many of these concepts will be explained in greater detail in later chapters as we discover how DarkBASIC Pro implements many of these ideas.

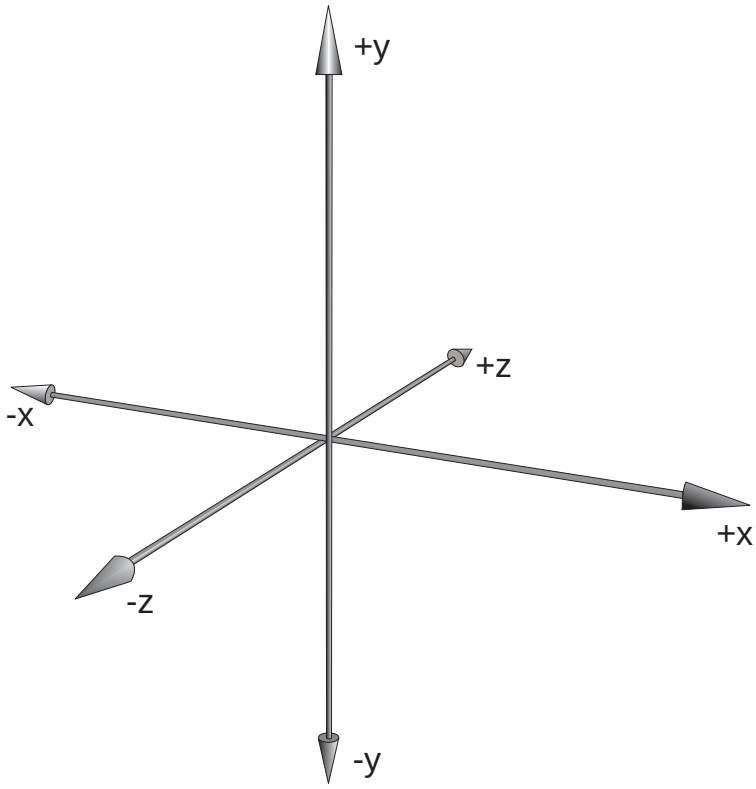
The Coordinate System

Axes

In a 3D world, just as in a 2D one, we need to identify the position of any point within that world. This we do using three axes (known as **world axes**) for reference. As before, we need *x* and *y* axes for width and height, but this time we also need a *z* axis to measure depth (see FIG-30.1).

FIG-30.1

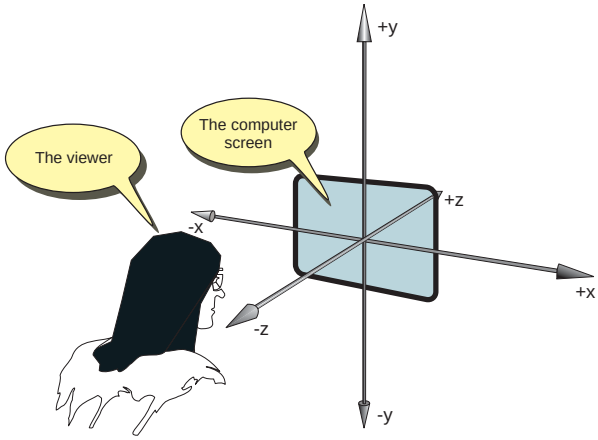
The Axes used in 3D



In the figure above, the axes have been skewed slightly to give a better perspective In reality the x-axis runs across the screen, the y-axis runs up and down, and the z-axis points directly out of the screen (-z) and into the screen (+z) (see FIG-30.2).

FIG-30.2

3D Axes and the Viewer

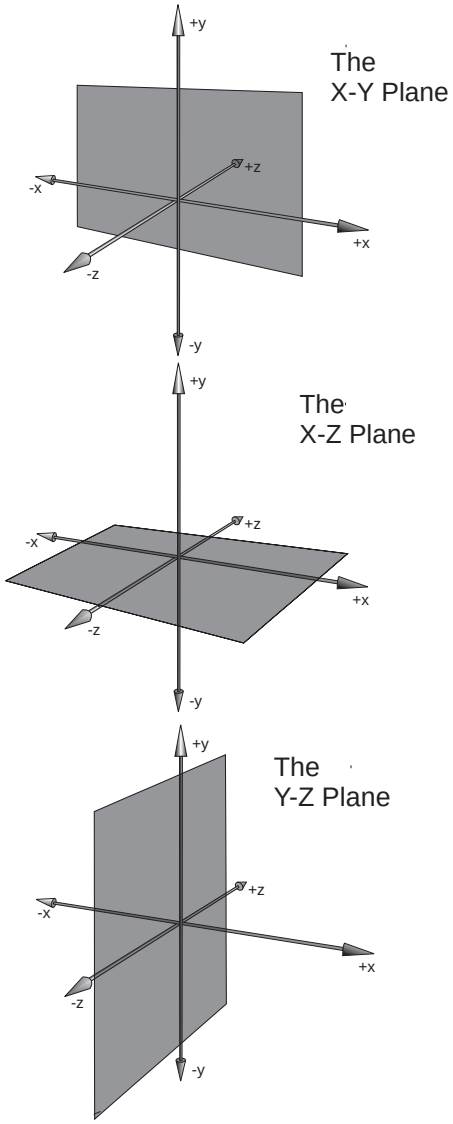


Planes

In mathematics, a **plane** is a flat surface with only two dimensions. 3D space has three main planes: the X-Y plane, the X-Z plane and the Y-Z plane (see FIG-30.3).

FIG-30.3

The Main 3D Planes



The X-Y plane has the x and y axes passing through its centre and, like every plane, expands to infinity in all directions. The X-Z plane has the x and z axes at its centre, and the Y-Z plane has the y and z axes at its centre.

These three planes are important since each divides space into two equally sized areas. The X-Y plane splits space with one half to the front, the other half to the back. The X-Z planes splits space into above and below sections, and the Y-Z plane splits space into left and right sections.

With all three planes in place, space is split into eight equally-sized sections. Each of these sections is known as an **octant**.

Of course, not all planes lie on axes; there are an infinite number of planes, some parallel to the main planes, others at angles to those planes, but it is the main planes that will be useful in many of the calculations required when determining the position of an object in 3D space.

Points

To specify the position of a point in 3D space we state its distance from the origin along all three axes in the order, x, y, z (see FIG-30.4).

FIG-30.4

Determining the Position of a Point in 3D Space

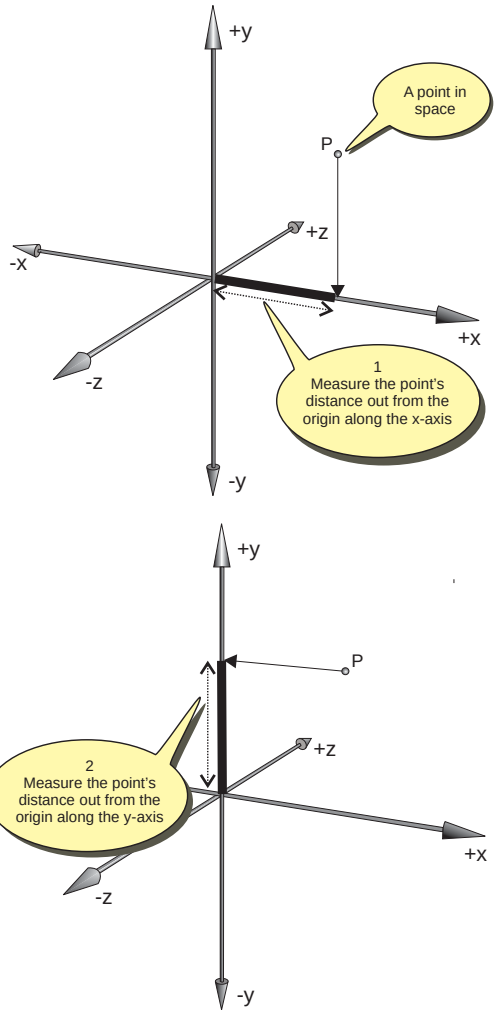
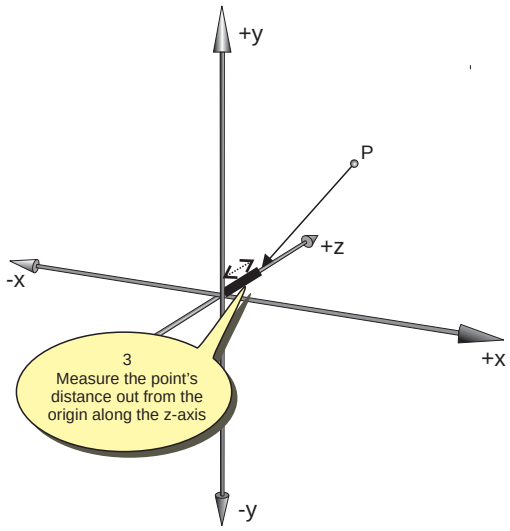


FIG-30.4
(continued)

Determining the Position of a Point in 3D Space



We might, for example, state that point *p* is at the position (8,12,5) meaning that point *p* is 8 units along the x-axis, 12 units along the y-axis and 5 units along the z-axis.

World Units

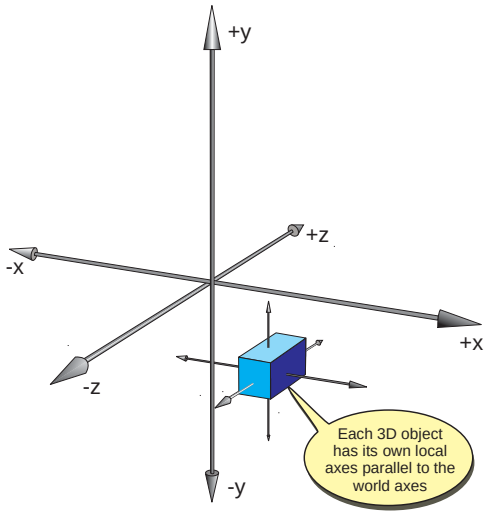
Distances are measured in units. These units have no relationship to real-life measurements such as centimetres or inches. Instead, objects are constructed in such a way as to be the correct size relative to other objects. For example, if we make a human character 6 units high, then a simple house might be 18 to 25 units high. Of course, if you wish, you can think of 1 unit being the equivalent of a real distance. The scale you choose will depend on the context; when creating a world with an ant as the main character, 1 unit might be equivalent to a millimetre, while a truly interstellar game might make 1 unit equivalent to 1 light year.

Local Axes

Every 3D object we create has its own **local axes**. These axes are (initially, at least) aligned to the world axes. FIG-30.5 shows a cuboid and its local axes.

FIG-30.5

Each Object has its Own Local Axes

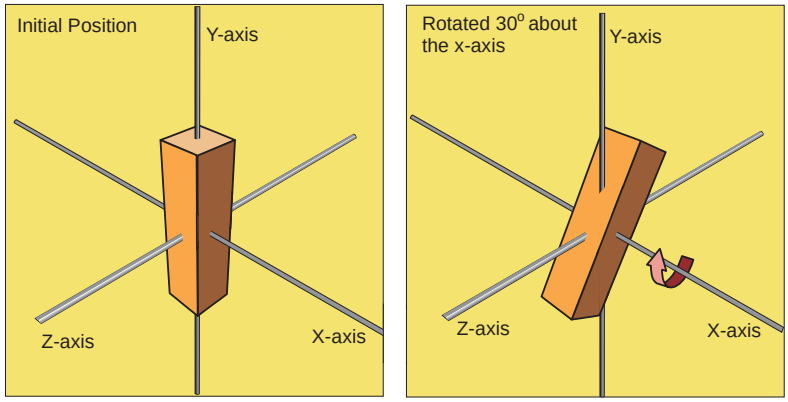


Rotation

An object can be made to rotate about its own, **local**, axes. In DarkBASIC Pro rotation is measured in degrees. For example, we might rotate an object 30 degrees about it's x-axis as shown in FIG-30.6.

FIG-30.6

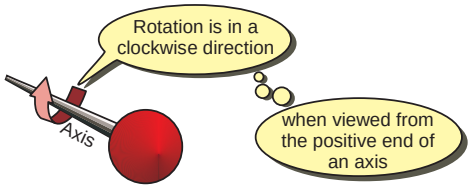
A Cuboid is Rotated 30^{o} about the x-axis



Rotation is performed in a clockwise direction when viewed down the positive end of an axis (see FIG-30.7).

FIG-30.7

Clockwise Rotation



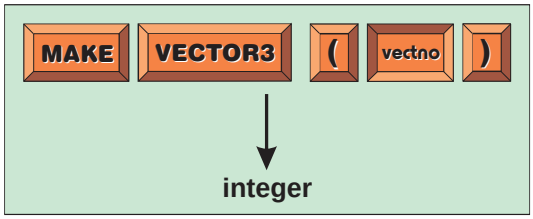
By specifying a negative angle of rotation, an object will rotate anti-clockwise.

3D Vectors

Although the purpose of this chapter is to describe basic 3D concepts, it's worth mentioning that DarkBASIC Pro allows the creation of a 3-element vector specifically for storing the coordinates of a point in 3D space. The vector is created using the MAKE VECTOR3 statement which has the format shown in FIG-30.8.

FIG-30.8

The MAKE VECTOR3 Statement



In the diagram:

vectno	is an integer value giving the ID to be assigned to the 3D vector being created.

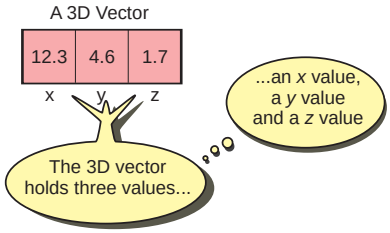
The statement returns 1 if the vector is created successfully; otherwise zero is returned. Usually we won't worry about the value returned and can create a 3D vector with a statement such as:

```
result = MAKE VECTOR3(1)
```

We can visualise a 3D vector object as shown in FIG-30.9.

FIG-30.9

A 3D Vector



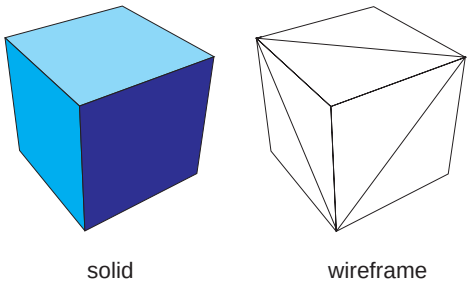
Many of the DarkBASIC Pro statements we'll encounter later make use of 3D vectors for storing results, so it's useful to give you this quick grounding in them at this early stage. We'll learn more on this subject in a later chapter.

Object Terminology

Just as 2D has a few basic shapes such as a line, a circle, a triangle and a rectangle, so we have a set of basic shapes (known as **primitives**) in 3D. These include the sphere, cylinder, cone, and cube. In FIG-30.10 we see and example of a cube.

FIG-30.10

A Cube - An Example of a Primitive

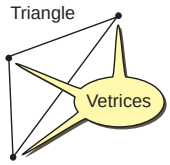


The cube is shown in two ways: **solid**, with shading caused by the light falling on its surface, and **wireframe** showing how the cube is constructed.

Polygon is the term used for a many-sided enclosed area. The simplest polygon (that is, the one with the least sides) is the triangle. The point where two lines of a polygon meet is known as a **vertex**. A triangle has three vertices (see FIG-20.11).

FIG-30.11

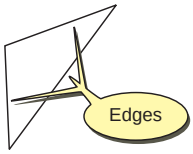
The Vertices of a Triangle



The line between two vertices is known as an **edge** (see FIG-30.12).

FIG-30.12

Edges



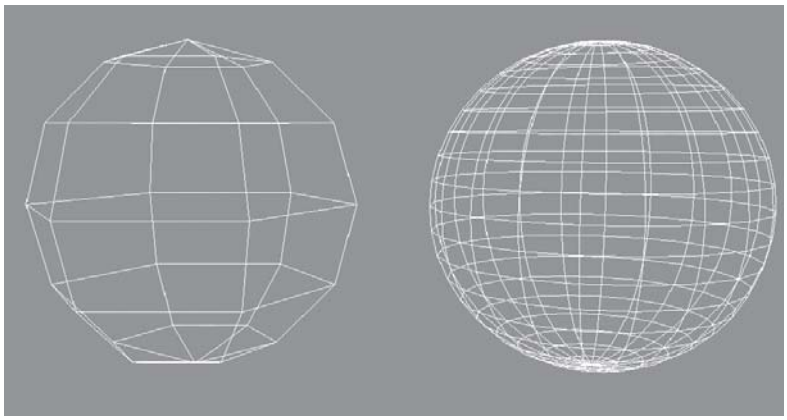
Every 3D shape in a game is constructed from polygons (normally triangles), as you can see from the wireframe version of the cube shown in FIG-30.10.

The greater the number of polygons used to create an object, the more detailed and realistic it will appear (see FIG-30.13). But there is a price to pay for greater detail

- higher processing requirements. As you increase the number of polygons that go to make up the objects in your scene, the harder your processor and video card need to work. Ask too much of your hardware, and screen updating will slow down. The number of times the screen is redrawn in one second is known as the **frame rate** and is quoted in **frames per second** (fps). If the frame rate falls much below about 20 fps, then your eyes will become aware of the screen refreshing and the picture will become jerky.

FIG-30.13

Varying the Polygons in a Sphere



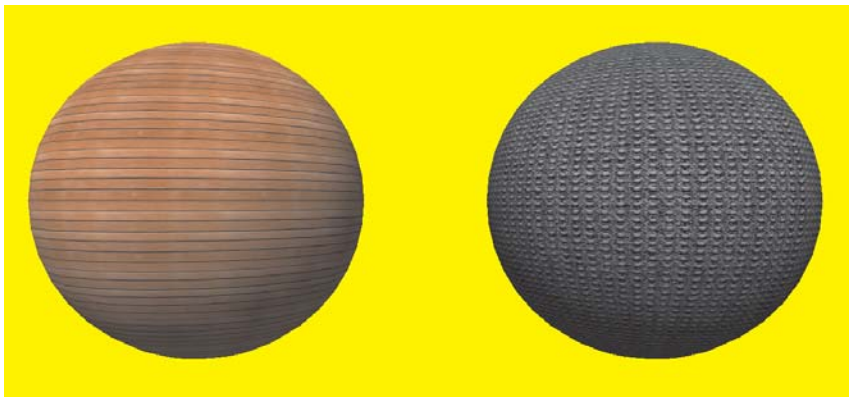
A Sphere with Few Polygons A Sphere with Many Polygons

Textures

In solid mode (as opposed to wireframe), a 3D object has a bland grey surface, but we can use an image wrapped around that object to give it a greater reality. By wrapping the image of riveted steel plate round a sphere, we can create the illusion of a metal ball. Wrap an image of wooden planks round the same sphere and we create a wooden ball (see FIG-30.14).

FIG-30.14

Adding Texture to a 3D Object



Images with an Alpha Channel

The image used to texture an object can be one of many different formats. For example, JPG and BMP files are often used, but sometimes we will see images stored in the PNG or TGA format.

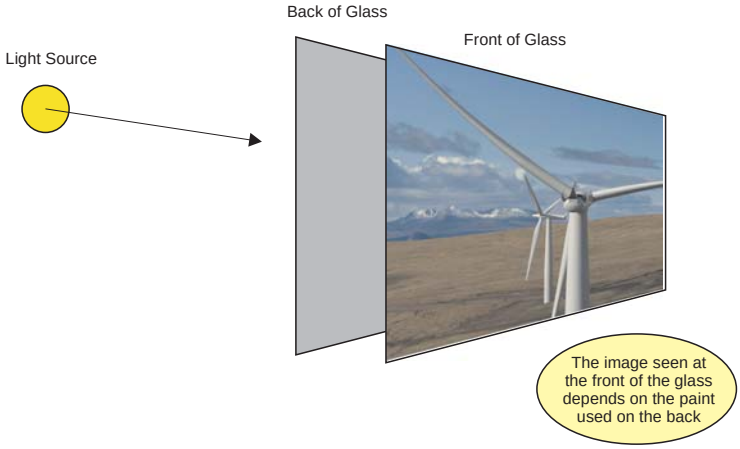
PNG and TGA files are amongst those formats capable of embedding an **alpha channel** within the image. An alpha channel affects how visible an image is and is probably best explained with an analogy.

Imagine you've just painted an image on a piece of glass and that the light illuminating the picture comes from behind the glass (see FIG-30.15) - like looking

out through a church's stained-glass window.

FIG-30.15

Perceived Image
Depends on the Backing



If we were to paint the back of the glass black, no light would get through and we wouldn't see the picture. If we used grey paint rather than black, then some light would get through. If we painted a pattern on the back of the glass using a mixture of black, dark grey, and light grey paint, the image would appear to have bright, dull and black areas depending on the paint on the back of the image.

This is how the alpha channel of an image works. As well as the basic red, green and blue elements (or **channels**) that go to make up the image, a fourth, alpha, channel is added. This is just another layer to the image which can only be shaded using greyscale colours (white through to black). Where black is used, the original image is unseen; where white is used the image appears at normal brightness (this is where the glass analogy falls down since it would be at its brightest with no paint on the back of the glass). Shades of grey give varying degrees of image brightness. FIG-30.16 shows original images, alpha channels, and the overall effects created.

FIG-30.16

Using an Alpha Channel

Cameras

The real world is a vast place, but with the help of television we can view any part of it - all we need is a camera. What the TV camera broadcasts we see on our screens. Move the camera and we see a different part of the world.

This is exactly how the 3D world we create within the computer works; what we see on the computer screen is the output from a virtual camera. The camera can be moved, just like a real camera, revealing different parts of our new 3D world. We can zoom the camera in or out allowing us to enlarge a distant object or show everything within a small space.

We can even use several cameras, switching between each to change what the user is seeing on the screen. Unlike real life, there's never any chance of seeing a camera in the view produced by a second camera - all virtual cameras are invisible!

DarkBASIC Pro creates and positions a single camera automatically at the start of every program that uses 3D objects. The exact position of the camera depends on the positioning of the 3D objects, since the camera normally places itself in order to see the objects that have been created. However, as the programmer, you can take complete control of the camera and thereby determine just exactly what appears on the screen.

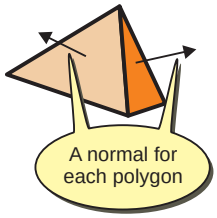
Lights

We can even set up the lights we want to use to illuminate our new world - just like placing lights on a movie set. By positioning various types of lights in just the correct positions, we can create any type of atmosphere we want - from dark and mysterious to bright and sunny. Like cameras in the 3D world, the lights are invisible, but the effects they create are not!

To help calculate the effect of lights on the individual polygons of a 3D object, a set of **normals** are maintained. A **surface normal** is a vector from the centre of a polygon perpendicular to the surface of that polygon. Every polygon in an object has an associated normal (see FIG-30.17).

FIG-30.17

Surface Normals



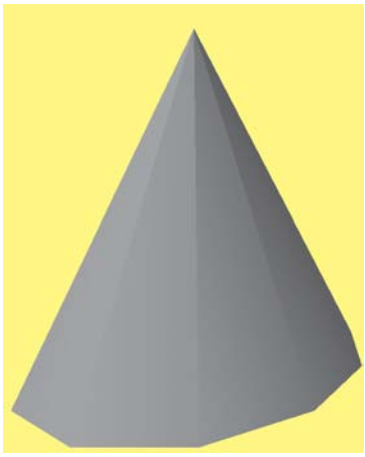
Normals are stored as mathematical expressions and are not part of the visible structure of the model.

In principal every polygon can have two surface normals: one on the top side and one on the bottom. However, often models only use a single normal - on the side facing outwards.

When using surface normals to calculate how an object should be lit, we sometimes get a rather faceted appearance, with an obvious jump in shading from one polygon to the next (see FIG-30.18).

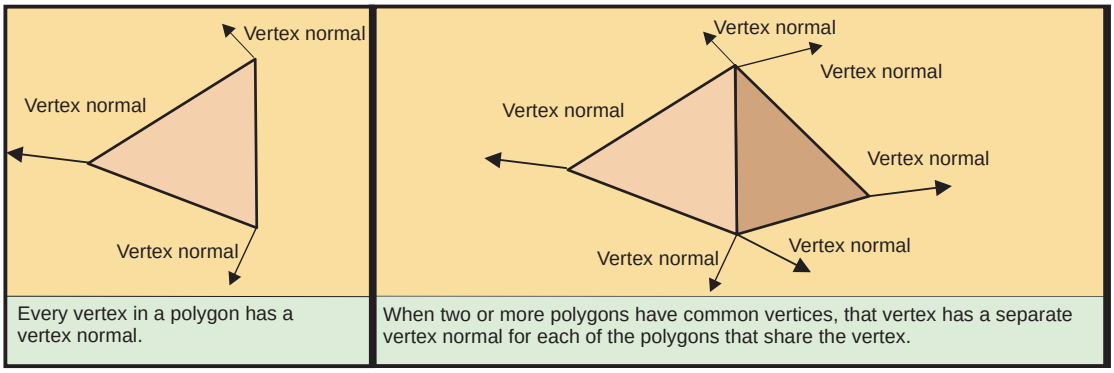
FIG-30.18

Visible Polygons



To solve this, **vertex normals** may be used. A vertex normal is created at every vertex of a polygon (see FIG-30.19).

FIG-30.19 Vertex Normals

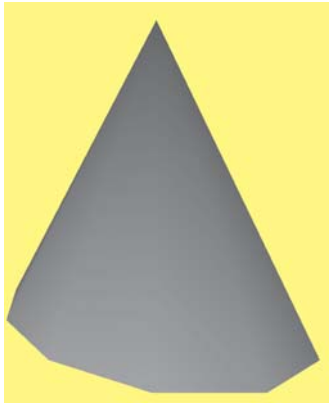


These vertex normals are calculated from the values of the two edges which meet at that vertex.

Using vertex normals creates a smoother lighting effect, but requires more calculations. You can see the effect produced in FIG-30.20.

FIG-30.20

Polygon Smoothing



Activity 30.1

Load and run the program *basic3D.exe*. This will demonstrate some of the basic concepts covered in this chapter.

(You can download this program, and all other files used in this text from **www.digital-skills.co.uk**)

As we'll see in the chapters that follow, DarkBASIC Pro has literally hundreds of commands designed to help us create a 3D world and manipulate the objects in that world.

Summary

- The 3D world uses three axes: x, y and z.
- 3D space is split into eight octants by the X-Y, X-Z and Y-Z planes.

- Space within the 3D world is measured in world units. These do not relate to real world units.
- A point in 3D space is defined by its distance along each of the axes.
- 3D objects have their own local axes.
- 3D objects can be rotated about their own local axes.
- Rotations are measured in degrees.
- Rotation is in a clockwise direction (as viewed from the positive end of the axis of rotation).
- DarkBASIC Pro provides 3D vector objects in which the coordinates of a point in 3D space can be stored.
- 3D objects are constructed from polygons.
- The simplest polygon is the triangle.
- The end of a line within a polygon is known as a vertex.
- The line between two vertices is known as an edge.
- More detailed objects require more polygons.
- Increasing the number of polygons used in a scene increases the load on the computer.
- When faced by a heavy load, the computer will output at a reduced frame rate.
- Images can be used to texture a 3D shape to increase realism.
- Some images can contain alpha channels which effect lightness when the image is used to texture a surface.
- Virtual cameras determine which parts of the 3D world are shown on the screen.
- Lights can be added to a scene to help create the desired atmosphere.
- The effects of lights on a surface are calculated using surface normals or vertex normals.
- Every polygon has an associated surface normal.
- A surface normal is a vector at right angles to its polygon.
- Using surface normals to calculate shading can result in a patchy effect.
- Every vertex of a polygon has an associated vertex normal.
- Vertex normals may be used to create smoother shading effects, but at the cost of more complex calculations.

.

81

3D Primitives

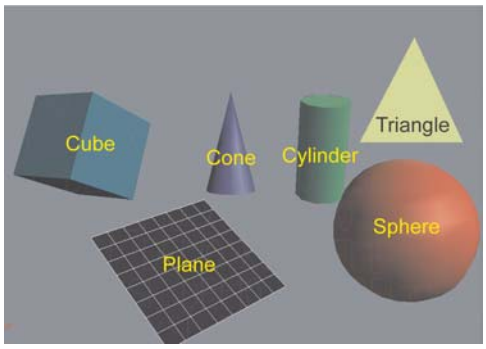
3D Primitives

Introduction

DarkBASIC Pro contains several statements for creating and manipulating 3D primitives such as spheres, cones and cubes. These statements are explained in detail below. A sample of the possible shapes is shown in FIG-31.1.

FIG-31.1

The 3D Shapes that can be Created in DarkBASIC Pro



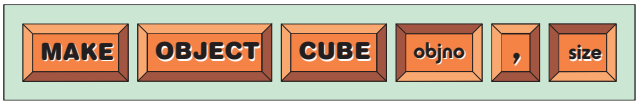
Creating a Cube

The MAKE OBJECT CUBE Statement

To create a cube on the screen, we use the MAKE OBJECT CUBE statement. Like sprites, every 3D object created must be given an identifying integer value (its ID). No two 3D objects within a program can be assigned the same ID. The size of the cube is also defined in this statement, which has the format shown in FIG-31.2.

FIG-31.2

The MAKE OBJECT CUBE Statement



In the diagram:

objno	is an integer value giving the ID to be assigned to the cube.
size	is a real value specifying the width, height and depth of the cube. This value is given in world units.

A typical usage of this statement might be:

```
MAKE OBJECT CUBE 1, 10
```

This would create a cube (with ID 1) which is 10 units wide, by 10 units high, by 10 units deep. FIG-31.3 shows a screen shot of the resulting cube.

FIG-31.3

A Cube in DarkBASIC Pro

This may not look too impressive as a 3D object, but that's because we're looking at the cube straight on and therefore can only see the front face of the object.

The cube shown above was created using the program given in LISTING-31.1.

LISTING-31.1

Creating a Cube

Statements such as COLOR BACKDROP and BACKDROP ON were covered in Volume 1.

```
REM *** Set display resolution and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON

REM *** Make the cube ***
MAKE OBJECT CUBE 1,10

REM *** End program ***
WAIT KEY
END
```

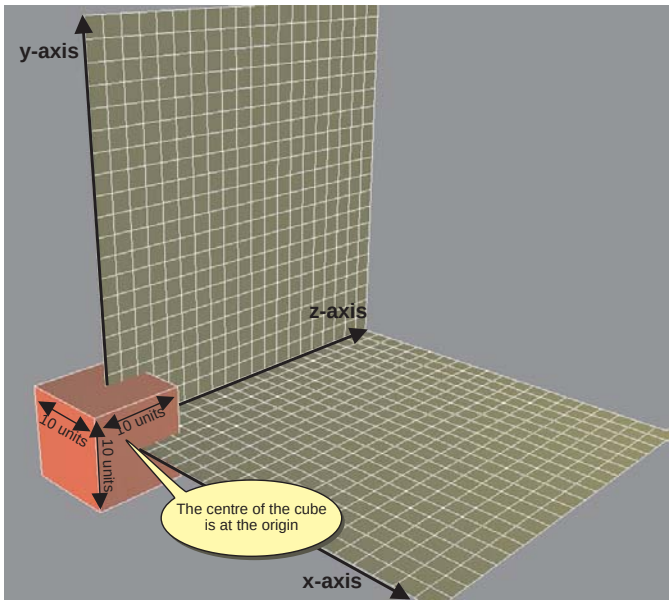
Activity 31.1

Type in the program in LISTING-31.1 (*object3D01.dbpro*) and check that you get the same display as shown above.

When any of the 3D primitives is first created, its centre is positioned at the origin. FIG-31.4 shows a model of what has been created by the program in LISTING-31.1. The 3 axes and parts of the XZ and YZ planes have been included to give a clearer picture of how the cube is positioned.

FIG-31.4

How the Cube is Positioned by the Program



Creating Other Primitives

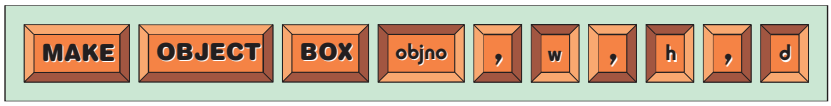
DarkBASIC Pro has a set of similar MAKE statements to create other basic 3D shapes. Like the cube, all of these objects are initially positioned with their centres at the origin. These statements are described below.

The MAKE OBJECT BOX Statement

The MAKE OBJECT BOX statement is similar to the MAKE OBJECT CUBE statement, but allows the three dimensions of the object to be set separately. The statement has the format shown in FIG-31.5.

FIG-31.5

The MAKE OBJECT BOX Statement



In the diagram:

objno	is an integer value giving the ID to be assigned to the box being created. No other 3D object in the program can be assigned the same value.
w	is a real value giving the width (x-dimension) of the box.
h	is a real value giving the height (y-dimension) of the box.
d	is a real value giving the depth (z-dimension) of the box.

For example, the line

```
MAKE OBJECT BOX 2, 10, 3.7, 12
```

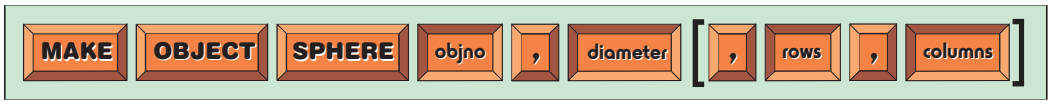
would create a box with ID 2 which is 10 units wide, by 3.7 units high, by 12 units deep.

The MAKE OBJECT SPHERE Statement

The MAKE OBJECT SPHERE statement creates a sphere of a specified diameter but offers extra options. The statement has the format shown in FIG-31.6.

FIG-31.6

The MAKE OBJECT SPHERE Statement



In the diagram:

objno	is an integer value giving the ID assigned to the sphere being created.
diameter	is a real number representing the diameter of the sphere.
rows	is an integer value specifying the number of *lines of latitude* making up the sphere.
columns	is an integer value specifying the number of *lines of longitude* making up the sphere.

The statement

```
MAKE OBJECT SPHERE 3,40.0
```

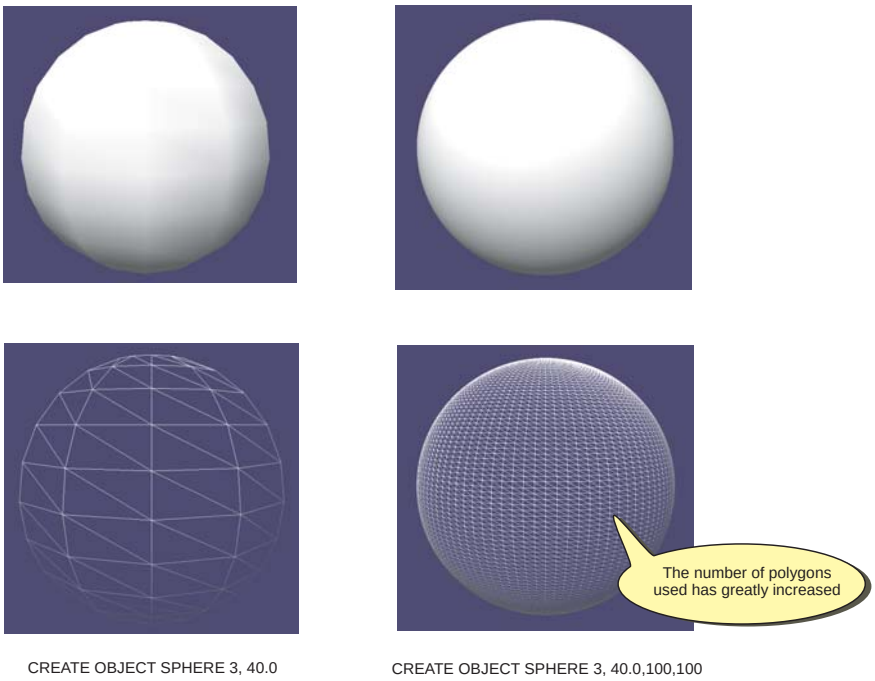
would create a sphere with a diameter of 40 units and assign it the ID number 3. However, the sphere produced is constructed from a relatively small number of polygons and hence its curve is not particularly smooth. By using the *rows* and *columns* values, we can control the number of polygons used to construct the sphere and thereby produce a more realistic effect. For example, the line

```
MAKE OBJECT SPHERE 3,40.0,100,100
```

would create a much smoother sphere. FIG-31.7 shows the difference between the default sphere and the more detailed one.

FIG-31.7

Creating a Smoother Sphere



However, there's a price to be paid for the more detailed sphere; the more polygons we use when creating any 3D shape, the more work the processor/video card needs to do and this reduces the frames per second that can be achieved.

Activity 31.2

Modify your previous program so that a standard sphere (diameter 10) is created instead of a cube.

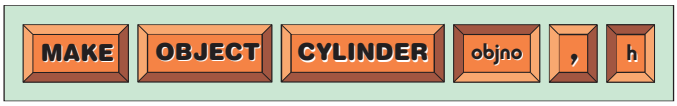
Modify the sphere to have 40 columns by 40 rows.

The MAKE OBJECT CYLINDER Statement

A cylinder of a specified height can be created using the MAKE CYLINDER OBJECT statement. The diameter of the cylinder's base automatically matches the height. The statement has the format shown in FIG-31.8.

FIG-31.8

The MAKE OBJECT CYLINDER Statement



In the diagram:

objno	is the integer value assigned to the cylinder being created.
h	is a real value giving the height and diameter of the cylinder.

For example, we could make a cylinder of height 31.5 units using the statement:

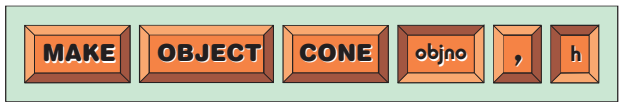
```
MAKE OBJECT CYLINDER 4,31.5
```

The MAKE OBJECT CONE Statement

The MAKE OBJECT CONE statement creates a cone of a specified height. The diameter of the base automatically matches the height. This statement has the format shown in FIG-31.9.

FIG-31.9

The MAKE OBJECT CONE Statement



In the diagram:

objno	is the integer value assigned to the cone being created.
h	is a real value giving the height of the cone and the diameter of its base.

For example, we could make a cone of height 10.1 units using the statement:

```
MAKE OBJECT CONE 5,10.1
```

Activity 31.3

Modify your previous program to display a cylinder of diameter 5.

Modify the program again to show a cone of the same height as the cylinder.

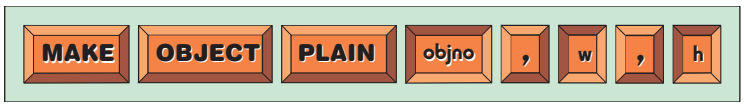
The MAKE OBJECT PLAIN Statement

A flat plane standing on the XY plane can be constructed using the MAKE OBJECT PLAIN statement which has the format shown in FIG-31.10.

FIG-31.10

The MAKE OBJECT PLAIN Statement

Note the spelling used in the instruction!



In the diagram:

objno	is the integer value assigned to the plane being created.
w	is a real value giving the width of the plane.
h	is a real value giving the height of the plane.

For example, we could create a plane which is 1000 units wide by 500 high using the line:

```
MAKE OBJECT PLAIN 6,1000.0,500.0
```

The centre of the plane will be located at the origin (see FIG-31.11).

FIG-31.11

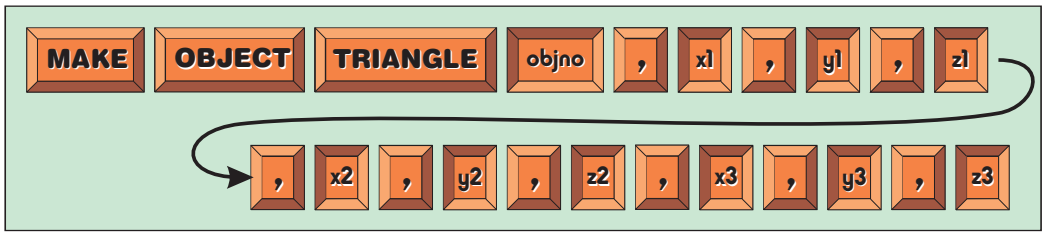
How a Plane is Positioned when First Created

The MAKE OBJECT TRIANGLE Statement

The simplest of all polygons, the triangle, can be constructed using the MAKE OBJECT TRIANGLE statement. The statement requires the positions of all three vertices to be supplied, so this statement contains a significant number of values, as shown in FIG-31.12.

FIG-31.12

The MAKE OBJECT TRIANGLE Statement



In the diagram:

objno	is the integer value assigned to the triangle being created.
x1,y1,z1	are real numbers representing the position of the first vertex.
x2,y2,z2	are real numbers representing the position of the second vertex.
x3,y3,z3	are real numbers representing the position of the third vertex.

The line

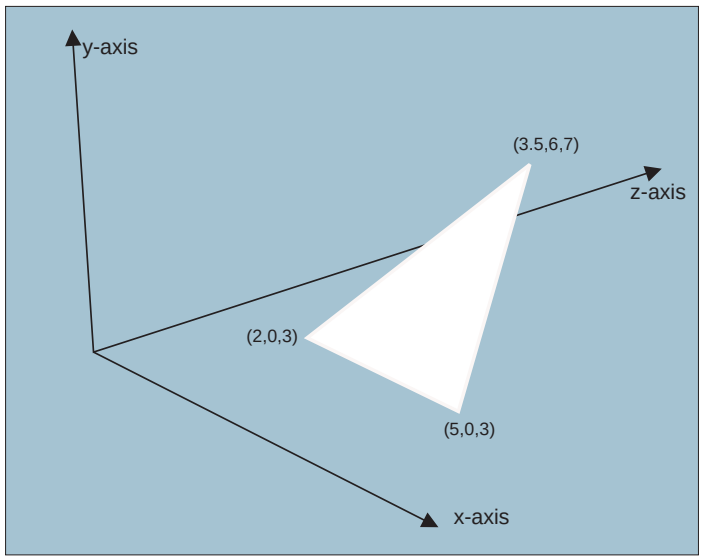
```
MAKE OBJECT TRIANGLE 7,  2,0,3,  5,0,3,  3.5,6,7
```

The additional spacing used within the instruction is used to highlight the various parameter groupings.

would create the triangle shown in FIG-31.13.

FIG-31.13

Creating a Triangle Object



Notice that, unlike any of the other objects, a triangle can be placed anywhere in 3D space.

Positioning an Object

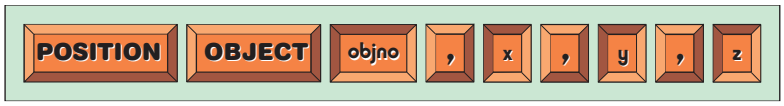
Other than the triangle, every object is created with it's centre at the point (0,0,0). However, once an object has been created, DarkBASIC Pro offers several ways of moving an object to another position.

The POSITION OBJECT Statement

One way to move an object is to use the POSITION OBJECT statement. The object is moved so that its centre is at the position specified. The statement has the format shown in FIG-31.14.

FIG-31.14

The POSITION OBJECT Statement



In the diagram:

objno	is the integer value previously assigned to the object.
x,y,z	are real values representing the position to which the object is to be moved. It is the centre of the object that is placed at this position.

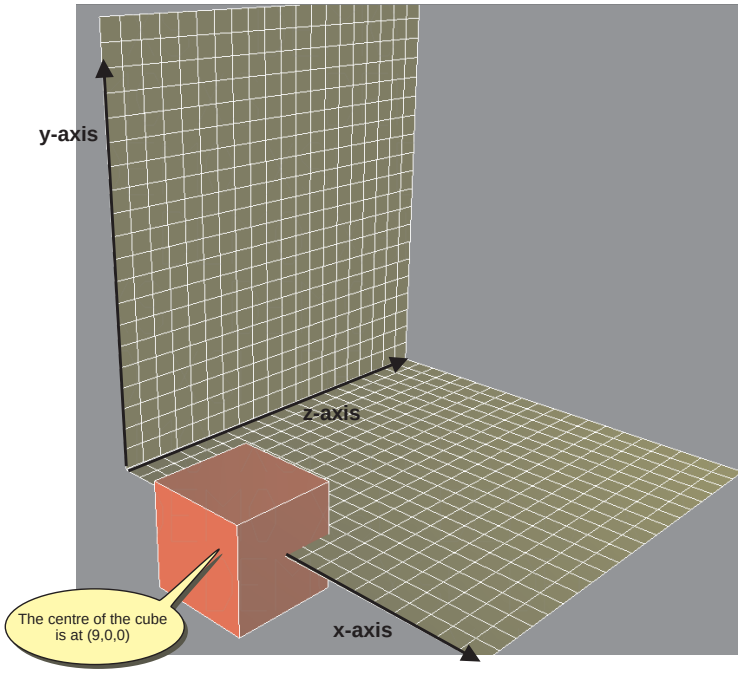
For example, if we wanted the centre of the cube we had created previously to be moved to position (9,0,0), then we would use the statement:

```
POSITION OBJECT 1,9,0,0
```

The result of executing this statement is shown in FIG-31.15.

FIG-31.15

The Result of Moving the Cube to (9,0,0)



LISTING-31.2 is a modification of the previous listing which moves the cube to position (9,0,0) after the user presses a key. The new lines have been highlighted.

LISTING-31.2

Moving the Cube

```
REM *** Set display resolution and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON

REM *** Make the cube **
MAKE OBJECT CUBE 1, 10

REM *** Move cube to (9,0,0) after key press ***
WAIT KEY
POSITION OBJECT 1,9,0,0

REM *** End program ***
WAIT KEY
END
```

Activity 31.4

Modify your previous program to match that given in LISTING-31.2.

Add the lines

```
REM *** Move the cube backwards ***
WAIT KEY
POSITION OBJECT 1, 9,0,30
```

so that the object is moved for a second time.

Notice that, in its final position, the cube looks smaller since it has now moved further away from our viewing position.

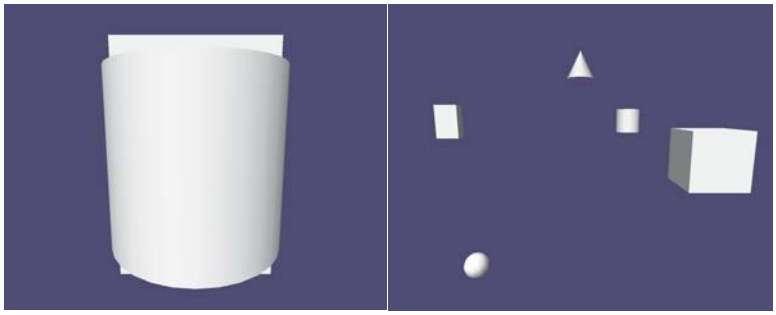
Activity 31.5

Write a program (*object3D02.dbpro*) to create the following objects, and then position each as specified:

Object Number	Object Type	Dimensions	Final Position
1	CUBE	4	9,0,0
2	BOX	10,15,5	-60,0,100
3	SPHERE	7	-30,-40,50
4	CYLINDER	12	25,0,120
5	CONE	12	0,25,100

Add a WAIT KEY statement between each move.

The screen should appear as shown below.



Screen at Start | Screen at End

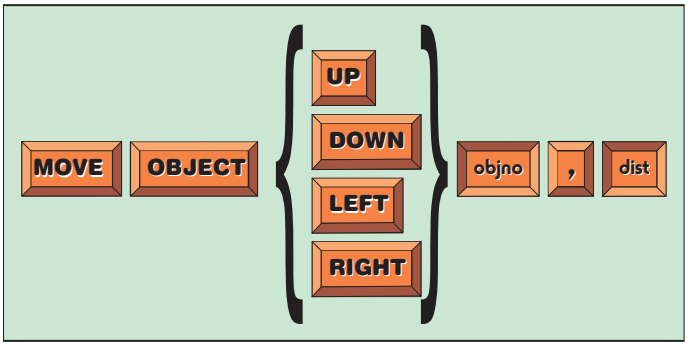
The MOVE OBJECT Statement

The MOVE OBJECT statement can be used to move an object a specified distance from its current location.

There are four possible directions available: RIGHT, LEFT, UP and DOWN. The statement has the format shown in FIG-31.16.

FIG-31.16

The MOVE OBJECT Statement



In the diagram:

UP, DOWN, LEFT, RIGHT
: One of these keywords must be used to indicate in which direction the object is to be moved.

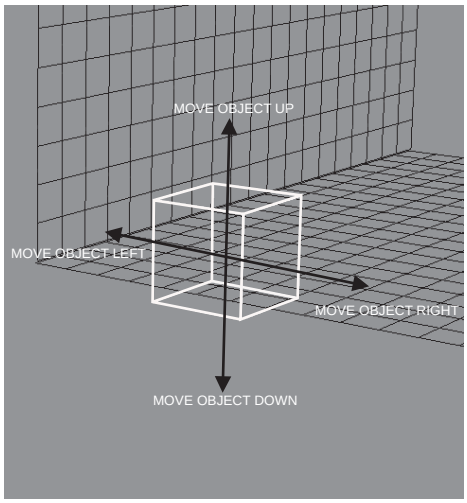
objno
: is an integer value giving the ID of the object to be moved.

dist is a real value giving the number of units the object is to moved.

The direction of movement for each option is shown in FIG-31.17.

FIG-31.17

Using the MOVE OBJECT Statement



Notice that there is no option to move the object backwards or forwards (i.e. along the z-axis). For example, object 1 could be moved 10.5 units to the right using the statement:

```
MOVE OBJECT RIGHT 1,10.5
```

Activity 31.6

Create a new program (*object3D03.dbpro*) containing a cube of size 10.

Use POSITION OBJECT to place the cube at (9,0,100).

Now move the cube 31.3 units to the right.

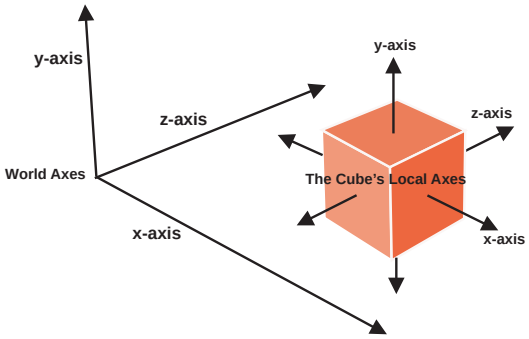
Place a WAIT KEY statement before each action.

Rotating Objects - Absolute Rotation

It is possible to rotate an object about one of its own **local axes**. FIG-31.18 emphasises the difference between the main (or world) axes and local axes.

FIG-31.18

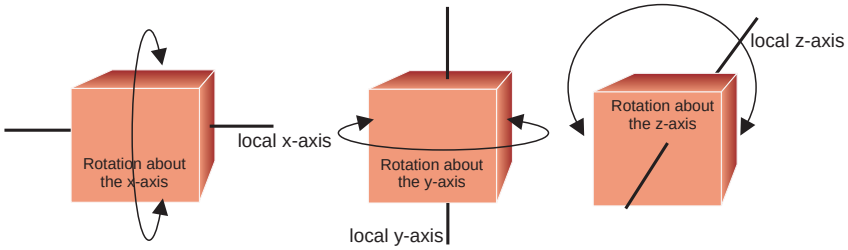
An Object's Local Axes



Possible rotations are shown in FIG-31.19.

FIG-31.19

Possible Rotations about Local Axes

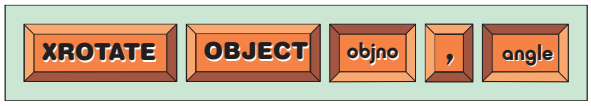


The XROTATE OBJECT Statement

This command causes an identified object to rotate to a specific angle about the object's local x-axis. Rotation is towards the viewer. The statement has the format shown in FIG-31.20.

FIG-31.20

The XROTATE OBJECT Statement



In the diagram:

objno	is the integer value specifying the object.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated.

For example, an existing cube could be rotated about the x-axis to 45^{o} using the line

```
XROTATE OBJECT 1, 45.0
```

FIG-31.21 shows the cube before and after rotation.

FIG-31.21

The Effect of the XROTATE OBJECT Statement

Activity 31.7

Write a program (*object3D04.dbpro*) which implements the following logic:

```
Set screen resolution to 1280 by 1024
Create a cube (40 units in size)
Move the cube to (0,0,100)
FOR degree := 1 TO 360 DO
    Rotate cube to degree° around the x-axis.
    Wait 1 millisecond
ENDFOR
```

Run the program and check that it performs as expected.

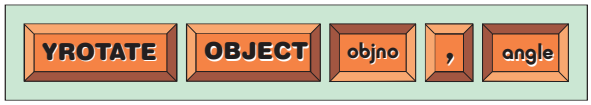
Modify the program so that the cube revolves in the opposite direction about the x-axis.

The YROTATE OBJECT Statement

To rotate an object about its local y-axis we use the YROTATE OBJECT statement which has the format shown in FIG-31.22.

FIG-31.22

The YROTATE OBJECT Statement



In the diagram:

objno	is the integer value previously assigned to the object.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated.

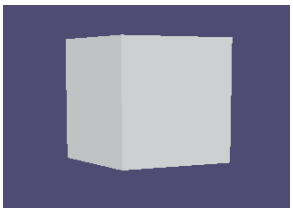
For example, a cube (ID 1) could be rotated about its y-axis to 60^{o} using the line:

```
YROTATE OBJECT 1, 60.0
```

FIG-31.23 shows the cube after a rotation to 60^{o}.

FIG-31.23

The Effect of the YROTATE OBJECT Statement

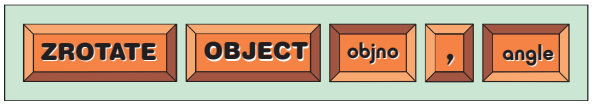


The ZROTATE OBJECT Statement

To rotate an object about its z-axis, we use the ZROTATE OBJECT statement which has the format shown in FIG-31.24.

FIG-31.24

The ZROTATE OBJECT Statement



In the diagram:

objno	is the integer value previously assigned to the object.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated.

For example, a cube (ID 1) could be rotated about the z-axis to 110^{o} using the line:

```
ZROTATE OBJECT 1, 110.0
```

FIG-31.25 shows the cube after a rotation to 110^{o}.

FIG-31.25

The Effect of the ZROTATE OBJECT Statement

The program in LISTING-31.3 revolves a cube about all three axes at the same time.

LISTING-31.3

Rotating an Object about all Three Axes

```
REM ** Set display mode ***
SET DISPLAY MODE 1280,1024,32

REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200

REM *** Rotate cube 1 degree at a time ***
REM *** around all three axes ***
FOR angle = 1 TO 360
   XROTATE OBJECT 1, angle
   YROTATE OBJECT 1, angle
   ZROTATE OBJECT 1, angle
   WAIT 10
NEXT angle

REM *** End program ***
WAIT KEY
END
```

Activity 31.8

Type in and test the program given above (*object3D05.dbpro*).

The ROTATE OBJECT Statement

Rather than use three separate statements to rotate an object about all three axes, the same effect can be achieved using the ROTATE OBJECT statement. This statement takes three values specifying, for each axis, the degree of rotation. The format of the statement is shown in FIG-31.26.

FIG-31.26

The ROTATE OBJECT Statement

In the diagram:

objno	is the integer value previously assigned to the object.
xangle	is a real number giving the angle (in degrees) to which the object is to be rotated about its x-axis.
yangle	is a real number giving the angle (in degrees) to which the object is to be rotated about its y-axis.
zangle	is a real number giving the angle (in degrees) to which the object is to be rotated about its z-axis.

Activity 31.9

Rewrite the program you created in the previous Activity, replacing the XROTATE, YROTATE and ZROTATE statements with a single ROTATE OBJECT statement, producing the same effect as before.

All statements in the previous section rotate an object to a specific angle,

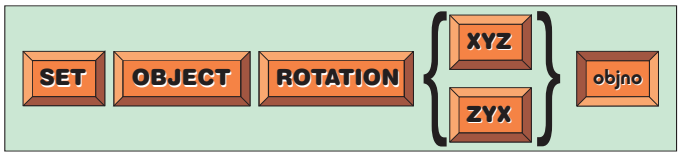
irrespective of that object's current inclination. For example, if we use the YROTATE OBJECT statement to turn a cube to 60°, the initial angle of the cube before the statement is executed is irrelevant since the end result will be that the cube will end up at the specified angle of 60°. This type of rotation is known as **absolute rotation**.

The SET OBJECT ROTATION Statement

When an object is rotated about all three axes at the same time, the action is normally implemented by first rotating the object about the x-axis, then the y-axis and finally the z-axis. Of course, it's all done so quickly that the operation will appear to be instantaneous. However, should we want to reverse the order in which the rotations take place (i.e. z-axis first, x-axis last) then we can use the SET OBJECT ROTATION statement. Once set, the order in which the axes are handled will remain on this new setting unless you revert to normal using a second option of the SET OBJECT ROTATION statement, which has the format shown in FIG-31.27.

FIG-31.27

The SET OBJECT ROTATION ZYX Statement



In the diagram:

XYZ	Use this option to return the order of rotations to the default x-axis, y-axis, z-axis order.
ZYX	Use this option to set the order of rotations to the z-axis, y-axis, x-axis order.
objno	is an integer value specifying the object whose order of rotation is to be modified.

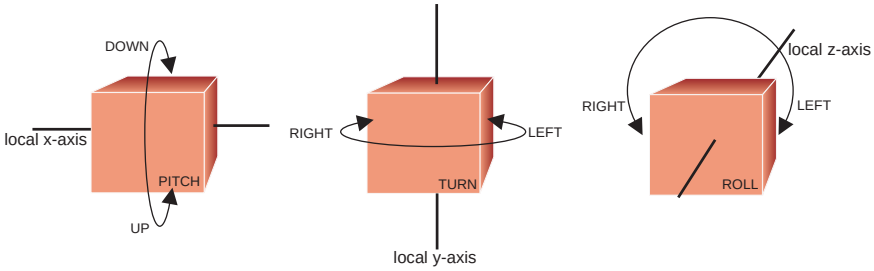
Rotating Objects - Relative Rotation

It is also possible to make an object rotate by a specific angle from its current setting. For example, if a cube has already been rotated 45° about the local y-axis, we can command it to be rotated by a further 60° giving a final rotation position of 105°. This type of rotation - where the angle specified is added to the initial tilt - is known as **relative rotation**.

When using relative rotation, different terms are used for rotation about each axis. Hence, we use the term PITCH for rotation about the x-axis, TURN for rotation about the y-axis and ROLL for rotation about the z-axis (see FIG-31.28).

FIG-31.28

Relative Rotation Terms

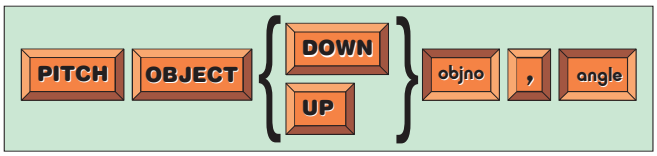


The PITCH OBJECT Statement

We can tilt an object upwards (i.e. rotate it in a positive direction about the x-axis) using the PITCH OBJECT UP statement which has the format shown in FIG-31.29.

FIG-31.29

The PITCH OBJECT Statement



In the diagram:

DOWN, UP	Choose DOWN to make the object rotate clockwise (as viewed from the positive side of the y-axis); choose UP to make the object rotate anticlockwise.
objno	is the integer value previously assigned to the object.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated relative to its current position. The angle can be a positive or negative value.

The program in LISTING-31.4 performs the same function as the one you created in Activity 31.7 where you used the XROTATE statement to rotate a cube through 360^{o}. However, this time the XROTATE statement has been replaced by a PITCH OBJECT UP command.

LISTING-31.4

Using Relative Rotation

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200

REM *** Revolve the cube ***
FOR c = 1 TO 360
   PITCH OBJECT UP 1, 1.0
   WAIT 10
NEXT c
REM *** End program ***
WAIT KEY
END
```

Activity 31.10

Type in the program given above (*object3D06.dbpro*) and make sure it is equivalent to the earlier program in Activity 31.7.

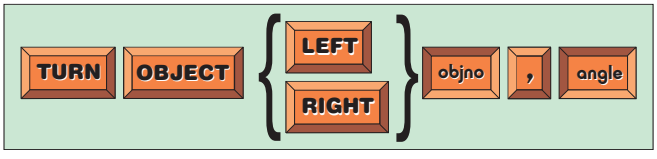
Modify the program so that the cube rotates in the opposite direction.

The TURN OBJECT Statement

This statement allows relative rotation about the y-axis and has the format shown in FIG-31.30.

FIG-31.30

The TURN OBJECT Statement



In the diagram:

LEFT, RIGHT	Choose RIGHT to make the object rotate to the right about the y-axis; choose LEFT to make the object rotate to the left.
objno	is an integer value giving the ID of the object to be rotated.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated relative to its current position. The angle can be a positive or negative value.

Activity 31.11

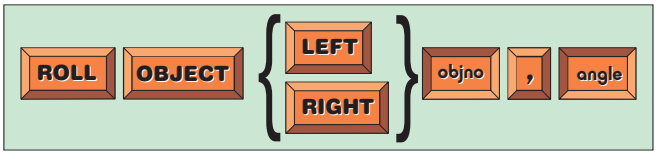
Modify your previous program so that the cube rotates to the right about the y-axis.

The ROLL OBJECT Statement

Relative rotation about the z-axis is achieved using the ROLL OBJECT statement which has the format shown in FIG-31.31.

FIG-31.31

The ROLL OBJECT Statement



In the diagram:

LEFT, RIGHT	Choose RIGHT to make the object rotate to the right about the z-axis; choose LEFT to make the object rotate to the left.
objno	is an integer value giving the ID of the object to be rotated.
angle	is a real number giving the angle (in degrees) to which the object is to be rotated relative to its current position. The angle can be a positive or negative value.

Activity 31.12

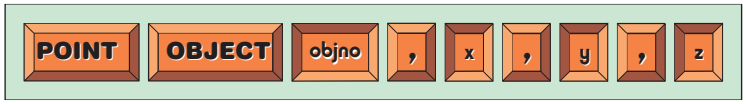
Modify your previous program so that the cube rotates to the left about the z-axis.

The POINT OBJECT Statement

The main polygon of a 3D object is directed towards the player's viewpoint when it is created. This polygon can be rotated to face any point in space using the POINT OBJECT statement. This statement has the format shown in FIG-31.32.

FIG-31.32

The POINT OBJECT Statement



In the diagram:

objno	is an integer value giving the ID of the object to be affected.
x,y,z	are the coordinates of the point in space at which the main polygon of the 3D object is to face.

In the program shown in LISTING-31.5 a cube is made to face the point (45,45,0) using the statement:

```
POINT OBJECT 1,45,45,0
```

LISTING-31.5

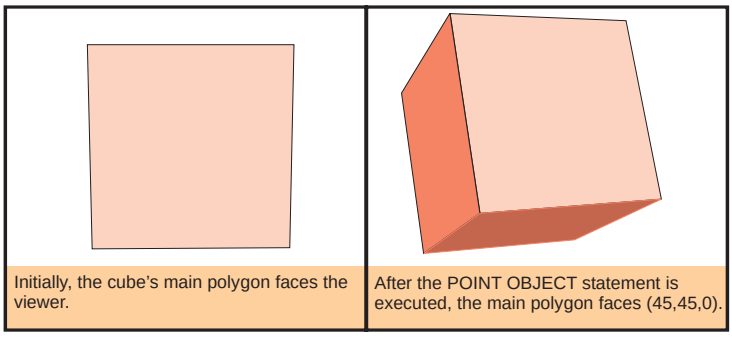
Using the POINT OBJECT Statement

```
REM *** Set display resolution and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,40
REM *** Move cube to (0,0,100) after key press ***
WAIT KEY
POSITION OBJECT 1,0,0,100
REM *** point cube at (45,45,0)***
WAIT KEY
POINT OBJECT 1,45,45,0
REM *** End program ***
WAIT KEY
END
```

The result is shown in FIG-31.33.

FIG-31.33

Turning an Object to Face a Specified Point



Initially, the cube's main polygon faces the viewer.

After the POINT OBJECT statement is executed, the main polygon faces (45,45,0).

Activity 31.13

Type in and test the program given in LISTING-31.5 (*object3D07.dbpro*).

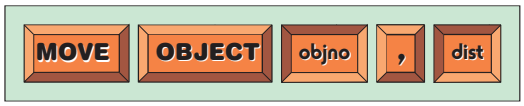
Modify the program to make the cube face the point (-20,17,-10).

The MOVE OBJECT *distance* Statement

We've already encountered a MOVE OBJECT statement which allows an object to be moved up, down, left, or right, but a second version of MOVE OBJECT exists which will move an object in the direction its main polygon is facing. This statement has the format shown in FIG-31.34.

FIG-31.34

The MOVE OBJECT *distance* Statement



In the diagram:

objno	is an integer value specifying the object to be moved.
dist	is a real value specifying the distance to be moved.

Activity 31.14

In your previous program, immediately after the POINT OBJECT statement, add the following lines:

```
REM *** Move cube ***
WAIT KEY
MOVE OBJECT 1, 20
```

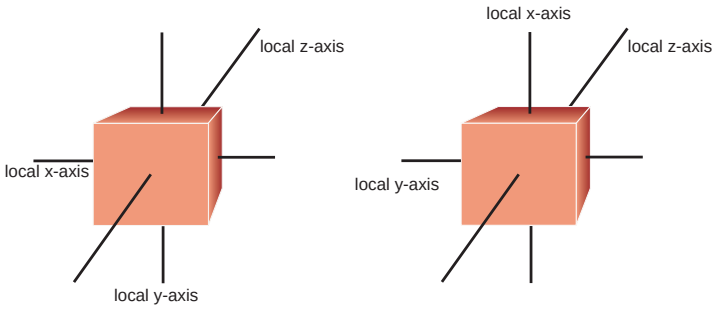
Run the updated program.

The FIX OBJECT PIVOT Statement

When an object rotates, its local axes rotate with it. In FIG-31.35 we see a cube and its local axes before and after it has been rotated to -90^{o} about its local z-axis.

FIG-31.35

How the Local Axes are Affected When an Object is Rotated



If we now rotate the cube about its own x-axis, it will turn left-to-right rather than up-and-over, because its x-axis has shifted position. This is demonstrated in LISTING-31.6 where the cube is rotated a full 360^{o} about its x-axis, rotated by -90^{o} about its z-axis and then rotated a full 360^{o} about its x-axis for a second time.

LISTING-31.6

Local Axes Move with the Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cube ***
MAKE OBJECT CUBE 1,40
POSITION OBJECT 1,0,0,100

REM *** Rotate cube 360 about x-axis ***
FOR degree = 0 TO 360
   XROTATE OBJECT 1, degree
   WAIT 1
NEXT degree
```

continued on next page

LISTING-31.6
(continued)

Local Axes Move with the Object

```
REM *** Rotate cube to -90 about z-axis
FOR degree = 0 TO -90 STEP -1
   ZROTATE OBJECT 1, degree
   WAIT 10
NEXT degree

REM *** Rotate cube 360 about x-axis ***
FOR degree = 0 TO 360
   XROTATE OBJECT 1, degree
   WAIT 1
NEXT degree

REM *** End program ***
WAIT KEY
END
```

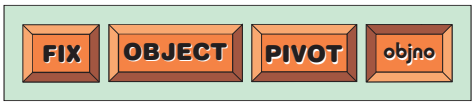
Activity 31.15

Type in and test the program given in LISTING-31.6 (*object3D08.dbpro*).

When an object has been rotated, it is possible to reset the local axes so that they are parallel to the main axes. This is done using the FIX OBJECT PIVOT statement whose format is shown in FIG-31.36.

FIG-31.36

The FIX OBJECT PIVOT Statement



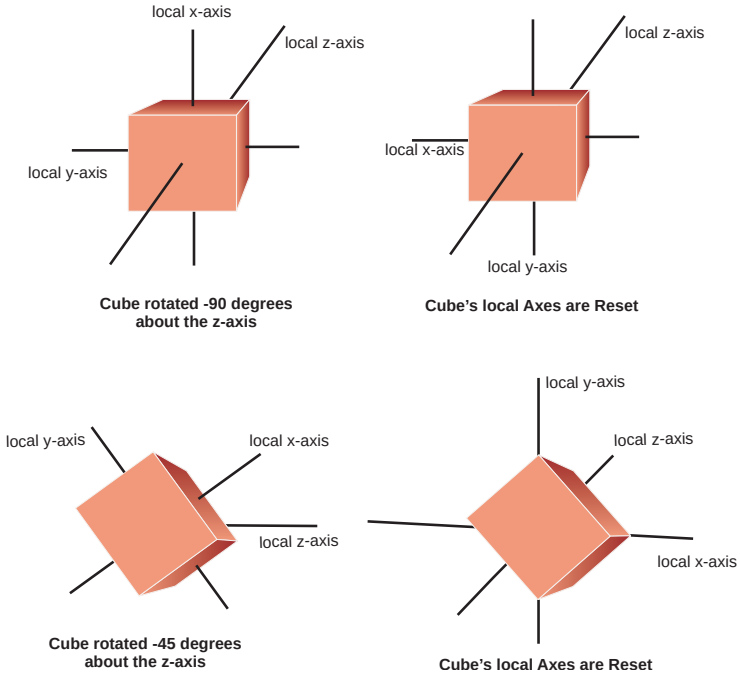
In the diagram:

objno	is an integer value specifying the object whose local axes are to be reset.

When this statement is executed, the object in question has its local axes reset so that the x-axis lies left-to-right, the y-axis top-to-bottom, and the z-axis in-to-out (see FIG-31.37).

FIG-31.37

The Effect of Using FIX OBJECT PIVOT



Activity 31.16

In your previous program (*object3D08.dbpro*), add the line

```
FIX OBJECT PIVOT 1
```

before the final FOR loop structure.

How does this affect the rotation of the cube?

Modify the program again so that the cube is only rotated to -45^{o} in the second FOR loop.

How is the cube's rotation affected this time?

Resizing Objects

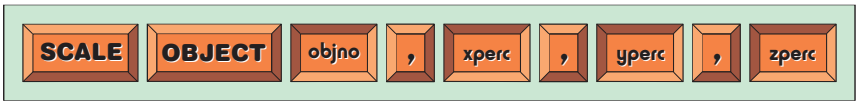
It is possible to change the size of an object after it has been created. You have the option to resize one, two, or all three of the object's dimensions. This allows you to make an object uniformly larger or smaller, or to distort the original shape by changing each dimension by differing amounts.

The SCALE OBJECT Statement

Resizing an existing 3D object is achieved using the SCALE OBJECT statement which has the format shown in FIG-31.38.

FIG-31.38

The SCALE OBJECT Statement



In the diagram:

objno	is the integer value previously assigned to the 3D object.
xperc	is a real number giving the new size of the object's x dimension as a percentage of its original size in that dimension. For example, a value of 100.0 will retain the current size, while 200.0 would double the object's length in the x dimension, and 50.0 would halve it.
yperc	is a real number giving the new size of the object's y dimension as a percentage of its original size in that dimension.
zperc	is a real number giving the new size of the object's z dimension as a percentage of its original size in that dimension.

The program in LISTING-31.7 creates a sphere with a radius of 20 units. The sphere is then resized so that the x dimension is doubled and the z dimension reduced to 10 units. The new shape is then rotated about the y-axis. The user must press ESC to terminate the program.

LISTING-31.7

Resizing an Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Make and position the sphere **
MAKE OBJECT SPHERE 1, 20
POSITION OBJECT 1, 0,0,200

REM *** Resize sphere ***
SCALE OBJECT 1, 200.0,100.0,50.0

REM *** Rotate shape ***
DO
   TURN OBJECT RIGHT 1, 1.0
LOOP

REM *** End program ***
WAIT KEY
END
```

Activity 31.17

Type in and test the program in LISTING-31.6 (*object3D09.dbpro*).

Modify the program so that a cone is used in place of the sphere.

Showing and Hiding Objects

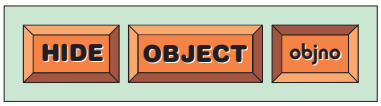
Any 3D object is immediately visible from the moment it is created (assuming it's within view), but it is possible to hide an object using the HIDE OBJECT statement, making it reappear later using the SHOW OBJECT command.

The HIDE OBJECT Statement

An object can be made invisible using the HIDE OBJECT statement which has the format shown in FIG-31.39.

FIG-31.39

The HIDE OBJECT Statement



In the diagram:

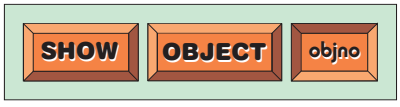
objno	is the integer value previously assigned to the 3D object which is to be hidden.

The SHOW OBJECT Statement

An object which has been previously hidden can be made to reappear using the SHOW OBJECT statement which has the format shown in FIG-31.40.

FIG-31.40

The SHOW OBJECT Statement



In the diagram:

objno	is the integer value previously assigned to the hidden 3D object which is to reappear.

The program in LISTING-31.8 rotates a cube continually, hiding the cube when '*h*' is pressed and showing it again when '*s*' is pressed.

LISTING-31.8

Hiding and Showing Objects

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Make  and position cube ***
MAKE OBJECT CUBE 1, 20
POSITION OBJECT 1, 0,0,200

REM *** Rotate object ***
DO
   PITCH OBJECT UP 1, 1.0
   REM *** Read key ***
   ch$ = INKEY$()
   REM *** IF its s - show cube ***
   IF ch$ = "s"
      SHOW OBJECT 1
   ENDIF
   REM *** IF its h - hide cube ***
   IF ch$ = "h"
      HIDE OBJECT 1
   ENDIF
LOOP
REM *** End program ***
END
```

Activity 31.18

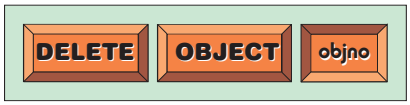
Type in and test the program given above (*object3D10.dbpro*).

The DELETE OBJECT Statement

When a 3D object is no longer required, its RAM space can be released using the DELETE OBJECT statement which has the format shown in FIG-31.41.

FIG-31.41

The DELETE OBJECT Statement



In the diagram:

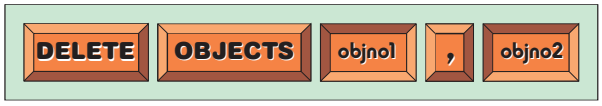
objno	is an integer value specifying the ID of the 3D object to be deleted.

The DELETE OBJECTS Statement

If we need to delete several objects at one time, then the most efficient way to do this is to use the DELETE OBJECTS statement which has the format shown in FIG-31.42.

FIG-31.42

The DELETE OBJECTS Statement



In the diagram:

objno1	is an integer value specifying the lowest ID of the 3D objects to be deleted.
objno2	is an integer value specifying the highest ID of the 3D objects to be deleted.

For example, if we needed to delete 10 3D objects with ID values ranging from 8 to 17, then we would use the statement:

```
DELETE OBJECTS 8,17
```

Copying a 3D Object

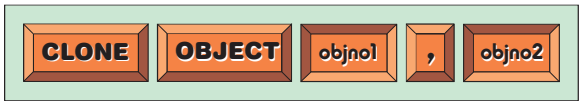
We can create a copy of an existing 3D object in one of two ways, as described below.

The CLONE OBJECT Statement

The CLONE OBJECT statement creates an independent copy of an existing 3D object. The statement has the format shown in FIG-31.43.

FIG-31.43

The CLONE OBJECT Statement



In the diagram:

objno1	is an integer value specifying the ID to be assigned to the object being created.
objno2	is an integer value specifying the ID of the existing object to be copied.

The program in LISTING-31.9 uses the CLONE OBJECT statement to create a duplicate cube.

LISTING-31.9

Creating a Copy of a 3D Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cube ***
MAKE OBJECT CUBE 1,40
POSITION OBJECT 1,-40,0,100
REM *** Rotate cube ***
WAIT KEY
XROTATE OBJECT 1 , -45
REM *** Copy cube ***
WAIT KEY
CLONE OBJECT 2,1
REM *** Position copy ***
POSITION OBJECT 2, 40,0,50
REM *** Delete original cube ***
WAIT KEY
DELETE OBJECT 1
REM *** End program ***
WAIT KEY
END
```

Activity 31.19

Type in and test the program given in LISTING-31.9 (*object3D11.dbpro*).

Is the copied cube rotated to the same angle as the original?

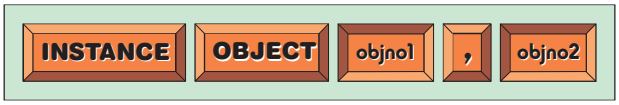
Is the copied cube deleted when the original cube is removed?

The INSTANCE OBJECT Statement

A second way to duplicate an existing object is to use the INSTANCE OBJECT statement. This statement has the format shown in FIG-31.44.

FIG-31.44

The INSTANCE OBJECT Statement



In the diagram:

objno1	is an integer value specifying the ID to be assigned to the object being created.
objno2	is an integer value specifying the ID of the existing object to be copied.

Although this may seem to have the same affect as the CLONE OBJECT statement, in fact the two statements differ in how data about the copied object is held. When CLONE OBJECT is used, the new object has its own independent data area; with INSTANCE OBJECT the two objects share parts of the same data area. The consequence of this is that objects created using INSTANCE OBJECT will disappear if the original object from which they were created is deleted.

Activity 31.20

Modify your last program, replacing the CLONE OBJECT statement with a INSTANCE OBJECT statement.

How does this change affect the operation of the program?

Change the DELETE OBJECT statement so that object *2*, rather than object *1*, is deleted. How does this affect the program?

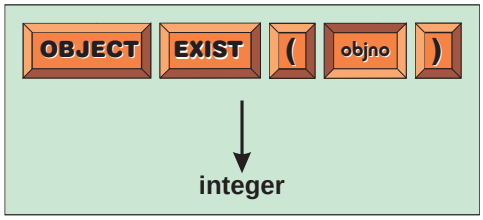
Retrieving Data on 3D Objects

The OBJECT EXIST Statement

We can check that an object of a specified ID actually exists using the OBJECT EXIST statement which has the format shown in FIG-31.45.

FIG-31.45

The OBJECT EXIST Statement



In the diagram:

objno	is an integer specifying the ID of the object to be checked.

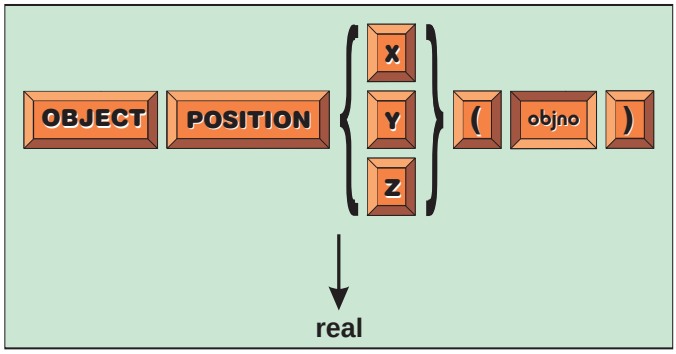
If the object exists, 1 is returned, otherwise zero is returned.

The OBJECT POSITION Statement

The exact position of a 3D object's centre can be determined using the OBJECT POSITION statement. Three variations of the statement exist, with each variation returning one of the object's ordinates. The statement has the format shown in FIG-31.46.

FIG-31.46

The OBJECT POSITION Statement



In the diagram:

X,Y,Z	One of these options should be chosen. Choose X if the x-ordinate of the specified object is required, Y for the y-ordinate, and Z for the z-ordinate.
objno	is an integer value specifying the object whose ordinate is to be returned.

For example, we could determine the position in space of object 1's centre using the lines:

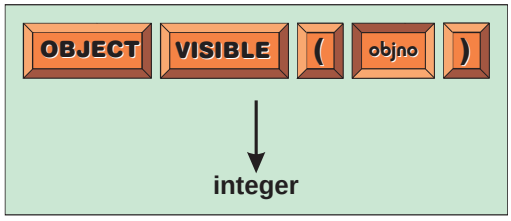
```
x = OBJECT POSITION X(1)
y = OBJECT POSITION Y(1)
z = OBJECT POSITION Z(1)
PRINT "Object 1 has its centre at (",x,",",y,",",z,")"
```

The OBJECT VISIBLE Statement

The OBJECT VISIBLE statement returns 1 if a specified 3D object is currently, visible; if the object is hidden, the value zero is returned. The statement has the format shown in FIG-31.47.

FIG-31.47

The OBJECT VISIBLE Statement



In the diagram:

objno	is an integer value specifying the ID of the object to be checked.

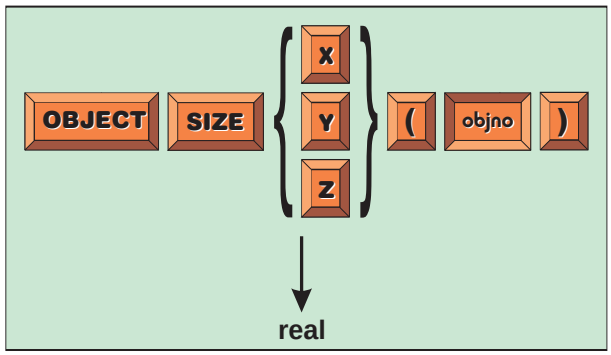
If the object is currently showing, 1 is returned, otherwise zero is returned.

The OBJECT SIZE Statement

The dimensions of a specified object can be determined using the OBJECT SIZE statement. Like OBJECT POSITION, there are three variations available in this statement, each returning one dimension of the object in question. The format for this statement is given in FIG-31.48.

FIG-31.48

The OBJECT SIZE Statement



In the diagram:

X,Y,Z	One of these options should be chosen. Choose X if the width of the specified object is required, Y for the height, and Z for the depth. All three options can be omitted and the statement will return an overall value for the size of the 3D object.
objno	is an integer value specifying the object whose dimension is to be returned.

The value returned by the statement is real and, because of rounding errors, this may be slightly out. For example, if we create a cube (object 1) 40 units in all directions, then the statement

```
PRINT "Width ", OBJECT SIZE X(1)
```

will display the value 39.9999961853.

Also, the OBJECT SIZE (1) statement - with no reference to any specific dimension - gives an overall size based on all three dimensions.

Activity 31.21

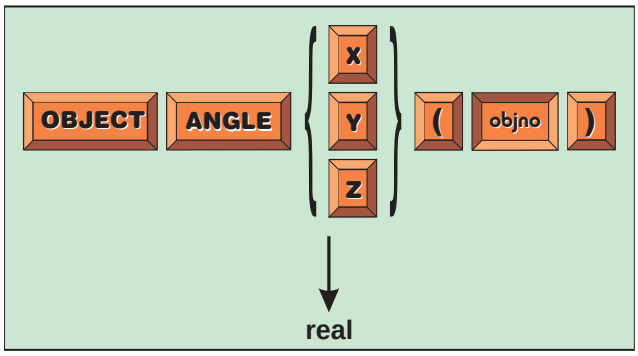
Write a program (*object3D12.dbpro*) which creates a box of random size (using limits 5 to 50) and then displays the box's width, height and depth.

The OBJECT ANGLE Statement

The angle to which an object has been rotated about any of its local axes can be determined using the OBJECT ANGLE statement which has the format shown in FIG-31.49.

FIG-31.49

The OBJECT ANGLE Statement



In the diagram:

X,Y,Z	One of these options should be chosen. Choose X if the rotation about the local x axis is required, Y for rotation about the y-axis, and Z for the rotation about the z-axis.
objno	is an integer value specifying the object whose rotation angle is to be returned.

Activity 31.22

Modify your previous program so that the box object is rotated by a random number of degrees about all three axes. Display the amount of rotation in each case.

Controlling an Object's Rotation Using the Mouse

In the example that follows, we're going to make a cube face towards the mouse pointer. As the user moves the mouse pointer about the screen, so the cube will continually re-orientate itself to face the pointer.

Before looking at the code, we have one main obstacle to overcome. The mouse pointer commands (MOUSE X() and MOUSE Y()) use a 2D coordinate system with the origin at the top left corner of the screen; 3D objects use a coordinate system in which the origin is (initially, at least) at the centre of the screen. To convert the mouse's x ordinate readings to 3D space we need to use the line:

```
x3D = MOUSE X() - SCREEN WIDTH()/2
```

The y ordinate also needs to have it's sign changed, since for the mouse the positive section of the y-axis is down, while in 3D space the positive section of the y-axis is up! We can solve this with the line:

```
y3D = -(MOUSE Y() - SCREEN HEIGHT()/2)
```

Of course, there is no third dimension as far as the mouse pointer is concerned, so we'll keep that set to zero.

We're now ready to describe the logic required by the program:

```
Create cube
Move cube backwards to reduce its apparent size
DO
```

```
        Get mouse coordinates and convert to 3D space
        Make cube point at these coordinates
    LOOP
```

The code for the program is given in LISTING-31.10.

LISTING-31.10

Making a 3D Object Face the Mouse Pointer

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,0,0,100
DO
   REM *** Convert mouse 2D coords to 3D ***
   x3D = MOUSEX() - SCREEN WIDTH()/2
   y3D = -(MOUSEY() - SCREEN HEIGHT()/2)
   REM *** Re-orient cube ***
   POINT OBJECT 1,x3D,y3D,0
LOOP
REM *** End program ***
END
```

Activity 31.23

Type in and test the code given above (*object3D13.dbpro*).

Modify the program to use two spheres, set side-by-side, both of which should face towards the mouse pointer.

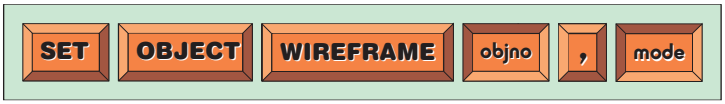
Wireframe and Culling

The SET OBJECT WIREFRAME Statement

It is possible to show a 3D object in wireframe mode (which show only the edges of the polygons that make up a shape) using the SET OBJECT WIREFRAME statement which has the format shown in FIG-31.50.

FIG-31.50

The SET OBJECT WIREFRAME Statement



In the diagram:

objno	is an integer value identifying the object which is to have its display mode altered.
mode	is 0 or 1. 0 - solid mode 1 - wireframe mode

LISTING-31.11 demonstrates the use of this statement, switching between solid and wireframe mode every time a key is pressed.

LISTING-31.11

Switching Between Normal and Wireframe Mode

```
REM *** Set screen mode ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100
REM *** Start in solid mode ***
wire = 0
```

continued on next page

LISTING-31.11
(continued)

Switching Between Normal and Wireframe Mode

```
   REM *** Rotate cube ***
DO
   REM *** IF key pressed, switch mode ***
   IF INKEY$() <> ""
      wire = 1 - wire
      SET OBJECT WIREFRAME 1, wire
   ENDIF
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 31.24

Type in and test the program in LISTING-31.11 (*object3D14.dbpro*).

The SET OBJECT CULL Statement

Under normal circumstances, it is impossible to see every part of a 3D object at the same time. The polygons that make up the hidden parts of an object are not drawn by the computer. Obviously, this saves processing time and creates a more realistic effect. If we take a second look at the previous program when running in wireframe mode, we'll see that the polygons at the back of the cube are not drawn. This elimination of hidden polygons is known as **culling**. Culling can be toggled on or off using the SET OBJECT CULL statement which has the format shown in FIG-31.51.

FIG-31.51

The SET OBJECT CULL Statement

In the diagram:

objno	is an integer value identifying the object which is to have its cull mode altered.
mode	is 0 or 1. 0 - culling off 1 - culling on

Activity 31.25

Modify your last program by adding the line

```
   SET OBJECT CULL 1,0
```

immediately before the DO..LOOP structure. *Notice that the hidden polygons are now being drawn.*

Modify the program again so that pressing the *w* key toggles between wireframe and solid mode and that pressing *c* toggles between culling on and culling off.

There's a slight problem when it comes to displaying cylinders and cones, as we can see from the output produced by LISTING-31.12.

LISTING-31.12

A Problem with Cones and Cylinders

```
REM *** Set screen mode ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cone and cylinder ***
MAKE OBJECT CONE 1, 5
POSITION OBJECT 1,-6,-5,0
XROTATE OBJECT 1, 45
MAKE OBJECT CYLINDER 2,5
POSITION OBJECT 2, 6,-5,0
REM *** End program ***
WAIT KEY
END
```

Activity 31.26

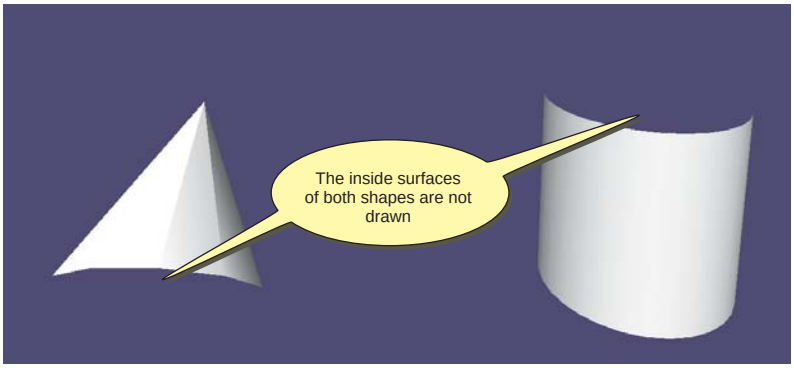
Type in and test the program in LISTING-31.12 (*object3D15.dbpro*).

What problem occurs with both shapes?

The inside surfaces of both shapes have not been drawn (see FIG-31.52).

FIG-31.52

Hidden Surfaces on Cones and Cylinders



But we can solve this problem by switching off culling, so that the hidden polygons are drawn.

Activity 31.27

Modify your previous program so that culling is switched off for both the cone and the cylinder.

Storage Methods

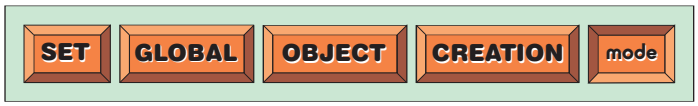
When a 3D object is shown on screen, the coordinates of its vertices are stored in memory in an area known as a **vertex buffer**. Normally, each object will have its own vertex buffer. However, some video cards allow different objects to share vertex buffers, other video cards don't. As a default, 3D objects in DARKBASIC Pro do not share vertex buffers - this ensures compatibility with the maximum number of video cards. However, it is possible to force vertex buffer sharing and, if your video card can handle this, improve the performance of your program.

The SET GLOBAL OBJECT CREATION Statement

To force vertex buffer sharing, we use the SET GLOBAL OBJECT CREATION statement which has the format shown in FIG-31.53.

FIG-31.53

The SET GLOBAL OBJECT CREATION Statement



In the diagram:

mode	0 or 1. 0 - no vertex buffer sharing (this is the default setting); 1 - vertex buffer sharing allowed.

Even if your own video card does allow vertex buffer sharing, your customer's may not, so it's probably best to ignore this option.

Summary

- A point in 3D space is specified using x, y, and z coordinates.
- There are three main planes in 3D space - XY, YZ, and XZ.
- The computer screen uses a positive-up, negative-down y-axis when operating in 3D mode. This is the opposite from the 2D settings.
- The positive z-axis travels away from the viewer "into" the screen.
- A point in space is known as a vertex.
- A set of vertices, when joined, form a polygon.
- The join between two vertices is known as an edge.
- Basic 3D shapes are known as primitives.
- When first created a 3D object has it centre position at the origin.
- The x and y axes intersect at the centre of the screen at start-up.
- Use the MAKE OBJECT CUBE to create a cube.
- Use MAKE OBJECT BOX to create a cuboid.
- Use MAKE OBJECT SPHERE to create a sphere.
- The number of polygons used can be specified when creating a sphere.
- Use MAKE OBJECT CYLINDER to create a cylinder.
- Use MAKE OBJECT CONE to create a cone.
- Use MAKE OBJECT PLAIN to create a plane.
- A plane is initially oriented as an XY plane.
- Use MAKE OBJECT TRIANGLE to create a triangle.
- The initial position of a triangle is determined by the vertices given.
- Use POSITION OBJECT to place the centre of an object at a new location.

- Use MOVE OBJECT to move an object along the x or y axis.
- Every 3D object has its own local axes with the origin at the centre of the object.
- Use the XROTATE, YROTATE or ZROTATE OBJECT statements to rotate an object to a specific angle about one of its local axes.
- Use ROTATE OBJECT to rotate an object to specific angles about all three local axes at the same time.
- Use PITCH, TURN or ROLL OBJECT statements to rotate an object by a number of degrees around a given axis.
- Use POINT OBJECT to make an object face towards a specified point.
- Use MOVE OBJECT *distance* to move the object a specified number of units in the direction in which an object is pointing.
- Use FIX OBJECT PIVOT to reset an object's local axes to be in line with the global axes.
- Use SCALE OBJECT to change the dimensions of an object.
- Use HIDE OBJECT to make an object invisible.
- Use SHOW OBJECT to make an invisible object reappear.
- Use DELETE OBJECT to erase an object from RAM.
- Use DELETE OBJECTS to erase a group og objects from RAM.
- Use CLONE OBJECT to make an independent copy of an existing object.
- Use INSTANCE OBJECT to create a dependent copy of an existing object.
- Use OBJECT EXIST to check if a specified object exists.
- Use OBJECT VISIBLE to check if a specified object is visible.
- Use OBJECT POSITION to determine the position in space of an object's centre.
- Use OBJECT SIZE to determine the dimensions of a specified object.
- Use OBJECT ANGLE to determine the current angle of rotation a specified object has about its local axes.
- Use SET OBJECT WIREFRAME to display a 3D object in wireframe or normal mode.
- Use SET OBJECT CULL to toggle culling for a specified 3D object.
- Use SET GLOBAL OBJECT CREATION to enable/disable vertex buffer sharing.

Merging Primitives

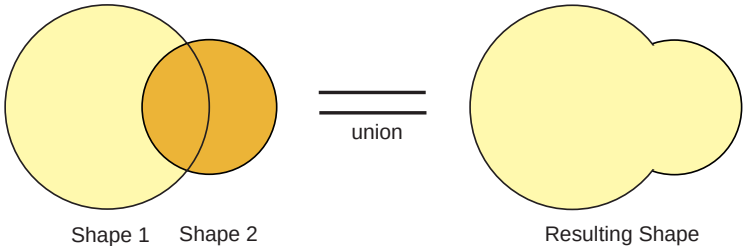
Introduction

DarkBASIC Pro version 1.058 introduced statements which allow us to create new shapes by merging two primitives. There are three basic options available:

Create the new shape from the combination of the two original shapes - known as **union** (see FIG-31.54).

FIG-31.54

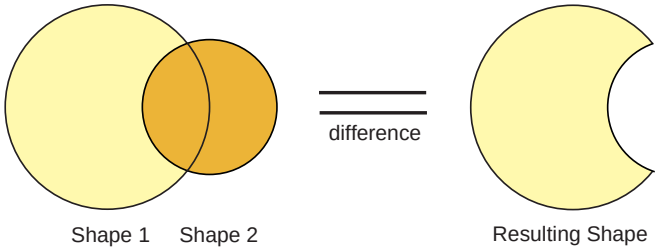
Shape Union



Create the new shape by removing the overlapping section of shape 2 from shape 1 - known as **difference** (see FIG-31.55).

FIG-31.55

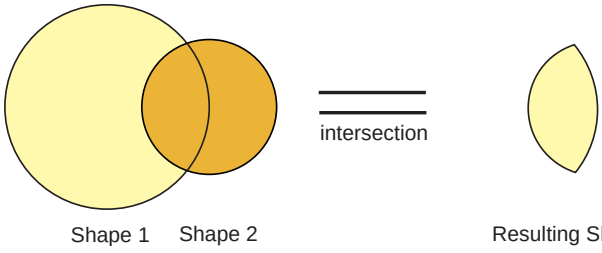
Shape Difference



Create the new shape from the overlapping area between shape 1 and shape 2 - known as **intersection** (see FIG-56).

FIG-31.56

Shape Intersection



The Statements

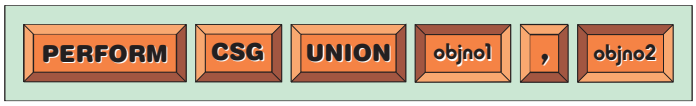
When we join two shapes using the merge statements, the resulting shape is stored in a format knows as **Constructive Solid Geometry** (CSG). We need not concern ourselves with the details of this format, and it is only mentioned here so that you know the meaning of the initials used in the statements.

The PERFORM CSG UNION Statement

We can create a new shape from the union of two existing shapes using the PERFORM CSG UNION statement which has the format shown in FIG-31.57.

FIG-31.57

The PERFORM CSG UNION Statement



In the diagram:

objno1	is an integer value specifying the ID of the first 3D object to be used in the union.
objno2	is an integer value specifying the ID of the second 3D object to be used in the union.

The statement modifies the shape of *objno1* without affecting *objno2*. Normally, the programmer would delete the second object once the union is completed.

The program in LISTING-31.13 demonstrates the union of a cube and a box. After the box has been deleted, the resulting shape is then rotated about its local x and y axes.

LISTING-31.13

Creating a New 3D Shape using Union

This program positions the camera which is responsible for the picture we see on the screen. Full details of camera usage are covered in Chapter 33.

```
REM *** Set screen resolution and background ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON

REM *** Position camera ***
AUTOCAM OFF
POSITION CAMERA 0,0,-100

REM *** Create two shapes used ***
MAKE OBJECT CUBE 1,40
MAKE OBJECT BOX 2,10,30,10
POSITION OBJECT 2,0,15,0

REM *** Let viewer see position of shapes ***
WAIT KEY

REM *** Union shapes ***
PERFORM CSG UNION 1,2

REM *** Remove object 2 ***
DELETE OBJECT 2

REM *** Rotate new shape ***
DO
   TURN OBJECT LEFT 1,1.0
   PITCH OBJECT UP 1,1.0
LOOP

REM *** End program ***
END
```

Activity 31.28

Type in and test the program given in LISTING-31.13 (*object3D16.dbpro*).

Try changing the second object to a sphere of diameter 10 and change the POSITION OBJECT statement to read

```
POSITION OBJECT 2, 0, 25, 0
```

As you've just discovered, the UNION statement only works predictably with cubes and boxes. Other shapes give unpredictable results (although the cone is close).

The PERFORM CSG DIFFERENCE Statement

The PERFORM CSG DIFFERENCE statement removes from object 1 the volume it shares in common with object 2. The statement has the format shown in FIG-31.58.

FIG-31.58

The PERFORM CSG DIFFERENCE Statement

In the diagram:

objno1	is an integer value specifying the ID of the first 3D object to be used in the difference operation.
objno2	is an integer value specifying the ID of the second 3D object to be used in the difference operation.

As before, the second object is unaffected by the operation and would normally be deleted. Also, we are again restricted to cubes and boxes if we are to obtain consistent results.

Activity 31.29

Restore your last project to its original code (as shown in LISTING-31.13).

Change the union operation to a difference operation and observe the new shape created.

Modify the width and depth of the box to be 35.

The PERFORM CSG INTERSECTION Statement

The PERFORM CSG INTERSECTION statement does not perform intersection as defined at the beginning of this chapter, but it does create a different shape from that produced by the PERFORM CSG DIFFERENCE statement and so is worth looking at. The statement has the format shown in FIG-31.59.

FIG-31.59

The PERFORM CSG INTERSECTION Statement

In the diagram:

objno1	is an integer value specifying the ID of the first 3D object to be used in the intersection operation.
objno2	is an integer value specifying the ID of the second 3D object to be used in the intersection operation.

Again, the second object would normally be deleted and we are restricted to cubes and boxes if we are to obtain predictable results.

Activity 31.30

Modify your previous program and determine how the shape created by the PERFORM CSG INTERSECTION differs from that produced by PERFORM CSG DIFFERENCE.

Summary

- Cubes and boxes can be merged to create new shapes.
- Use PERFORM CSG UNION to modify a shape so that it becomes the amalgamation of the original two shapes.
- Use PERFORM CSG DIFFERENCE to modify a shape so that the volume it shares with a second shape is removed.
- Use PERFORM CSG INTERSECTION to modify a shape to remove polygons which touch the second shape.

Solutions

Activity 31.1

No solution required.

Activity 31.2

```
1.
REM *** Set display and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the sphere ***
MAKE OBJECT SPHERE 1,10
REM *** End program ***
WAIT KEY
END

2.
REM *** Set display and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the sphere ***
MAKE OBJECT SPHERE 1,10,40,40
REM *** End program ***
WAIT KEY
END
```

Activity 31.3

```
1.
REM *** Set display and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the cylinder ***
MAKE OBJECT CYLINDER 1,5
REM *** End program ***
WAIT KEY
END

2.
REM *** Set display and backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the cone ***
MAKE OBJECT CONE 1,5
REM *** End program ***
WAIT KEY
END
```

Activity 31.4

```
REM *** Set display & backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the cube ***
MAKE OBJECT CUBE 1, 10
REM *** Cube to (50,0,0) after key
press ***
WAIT KEY
POSITION OBJECT 1,9,0,0
REM *** Move the cube backwards ***
WAIT KEY
POSITION OBJECT 1, 9,0,30
REM *** End program ***
WAIT KEY
END
```

Activity 31.5

```
REM *** Set display & backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,4
MAKE OBJECT BOX 2,10,15,5
MAKE OBJECT SPHERE 3,7
MAKE OBJECT CYLINDER 4,12
MAKE OBJECT CONE 5,12
REM *** Cube to (9,0,0) after key
press ***
WAIT KEY
POSITION OBJECT 1,9,0,0
REM *** Box to (-60,0,100) ***
WAIT KEY
POSITION OBJECT 2,-60,0,100
REM *** Sphere to (-30,-40,50) ***
WAIT KEY
POSITION OBJECT 3,-30,-40,50
REM *** Cylinder to (25,0,120) ***
WAIT KEY
POSITION OBJECT 4,25,0,120
REM *** Cone to (0,25,100) ***
WAIT KEY
POSITION OBJECT 5,0,25,100
REM *** End program ***
WAIT KEY
END
```

Activity 31.6

```
REM *** Set display & backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,10
REM *** Cube to(9,0,100) after key
press ***
WAIT KEY
POSITION OBJECT 1,9,0,100
REM *** Cube 31.3 units to the right
***
WAIT KEY
MOVE OBJECT RIGHT 1, 31.3
REM *** End program ***
WAIT KEY
END
```

Activity 31.7

```
Version 1
REM *** Set display & backdrop
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,40
REM ***Cube to(0,0,100) after key
press ***
POSITION OBJECT 1,0,0,100
REM *** Rotate cube ***
FOR degree = 1 TO 360
   XROTATE OBJECT 1, degree
   WAIT 1
NEXT degree
REM *** End program ***
WAIT KEY
END
```

```
Version 2
REM *** Set display & backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,40
REM ***Cube to(0,0,100) after key press ***
POSITION OBJECT 1,0,0,100
REM *** Rotate cube ***
FOR degree = 359 TO 0 STEP -1
   XROTATE OBJECT 1, degree
   WAIT 1
NEXT degree
REM *** End program ***
WAIT KEY
END
```

Activity 31.8

No solution required.

Activity 31.9

```
REM ** Set display mode ***
SET DISPLAY MODE 1280,1024,32
REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200
REM *** Rotate cube 1 degree at a time ***
REM *** around all three axes ***
FOR angle = 1 TO 360
   ROTATE OBJECT 1, angle,angle,angle
   WAIT 10
NEXT angle
REM *** End program ***
WAIT KEY
END
```

Activity 31.10

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200
REM *** Revolve the cube ***
FOR c = 1 TO 360
   PITCH OBJECT UP 1, -1.0
   WAIT 10
NEXT c
REM *** End program ***
WAIT KEY
END
```

Activity 31.11

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200
REM *** Revolve the cube ***
FOR c = 1 TO 360
   TURN OBJECT RIGHT 1, 1.0
   WAIT 10
NEXT c
REM *** End program ***
WAIT KEY
END
```

Activity 31.12

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Create and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1, 0, 0, 200
REM *** Revolve the cube ***
FOR c = 1 TO 360
   ROLL OBJECT LEFT 1, 1.0
   WAIT 10
NEXT c
REM *** End program ***
WAIT KEY
END
```

Activity 31.13

```
REM *** Set display & backdrop ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Make the set of objects ***
MAKE OBJECT CUBE 1,40
REM ***Cube to(0,0,100) after key press ***
WAIT KEY
POSITION OBJECT 1,0,0,100
REM *** point cube at (-20,17,-10)***
WAIT KEY
POINT OBJECT 1,-20,17,-10
REM *** End program ***
WAIT KEY
END
```

Activity 31.14

The changes should cause the cube should move in the direction it is pointing.

Activity 31.15

No solution required.

Activity 31.16

The cube rotates in the same direction on both occasions.

Activity 31.17

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position the cone ***
MAKE OBJECT CONE 1, 20
POSITION OBJECT 1, 0,0,200
REM *** Resize cone ***
SCALE OBJECT 1, 200.0,100.0,50.0
REM *** Rotate shape ***
DO
   TURN OBJECT RIGHT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 31.18

No solution required.

Activity 31.19

The second cube is created with the same rotation.
The copied cube does not disappear when the original is deleted.

Activity 31.20

The duplicated cube is not rotated, but it is removed when the original cube is deleted.

When the second cube is deleted, the first cube is unaffected.

Activity 31.21

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Seed random number generator ***
RANDOMIZE TIMER()
REM *** Make and position the box ***
MAKE OBJECT BOX 1, RND(45)+5,RND(45)+5,
↳RND(45)+5
POSITION OBJECT 1, 0,0,0
REM *** Get dimensions of box ***
width# = OBJECT SIZE X(1)
height# = OBJECT SIZE Y(1)
depth# = OBJECT SIZE Z(1)
REM *** Display dimensions ***
DO
   SET CURSOR 10,20
   PRINT "Width : ",width#, "   Height :",
   ↳height#, "   Depth : ",depth#
LOOP
REM *** End program ***
WAIT KEY
END
```

Activity 31.22

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Seed random number generator ***
RANDOMIZE TIMER()
REM *** Make and position the box ***
MAKE OBJECT BOX 1, RND(45)+5,RND(45)+5,
ÄRND(45)+5
POSITION OBJECT 1, 0,0,0
REM *** Rotate box at random ***
ROTATE OBJECT 1,RND(359),RND(359),RND(359)
REM *** Get dimensions of box ***
width# = OBJECT SIZE X(1)
height# = OBJECT SIZE Y(1)
depth# = OBJECT SIZE Z(1)
REM *** Get rotations of box ***
x_axis_rotation = OBJECT ANGLE X(1)
y_axis_rotation = OBJECT ANGLE Y(1)
z_axis_rotation = OBJECT ANGLE Z(1)
REM *** Display details ***
DO
   SET CURSOR 10,20
   PRINT "Width : ",width#, "   Height : "
   ↳, height#, "   Depth : ",depth#
   SET CURSOR 10,40
   PRINT "X-axis : ",x_axis_rotation,
   ↳"  y-axis : ",y_axis_rotation,
   ↳"  z-axis : ", z_axis_rotation
LOOP
REM *** End program ***
WAIT KEY
END
```

Activity 31.23

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position spheres ***
MAKE OBJECT SPHERE 1, 40
POSITION OBJECT 1,25,0,100
MAKE OBJECT SPHERE 2, 40
POSITION OBJECT 2, -25,0,100
DO
   x3D = MOUSEX() - SCREEN WIDTH()/2
   y3D = -(MOUSEY() - SCREEN HEIGHT()/2)
   POINT OBJECT 1, x3D,y3D,0
   POINT OBJECT 2, x3D,y3D,0
LOOP
REM *** End program ***
END
```

Activity 31.24

No solution required.

Activity 31.25

```
REM *** Set screen mode ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100
REM *** Start in solid and culling on ***
wire = 0
cull = 1
REM *** Rotate cube ***
DO
   REM *** IF key pressed, switch mode ***
   IF INKEY$() ="w"
      wire = 1 - wire
      SET OBJECT WIREFRAME 1, wire
   ENDIF
   IF INKEY$()="c"
      cull = 1 - cull
      SET OBJECT CULL 1,cull
   ENDIF
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 31.26

The inside surfaces of both shapes are not drawn.

Activity 31.27

```
REM *** Set screen mode ***
SET DISPLAY MODE 1280,1024,32

REM *** Make & position cone and cylinder ***
MAKE OBJECT CONE 1, 5
POSITION OBJECT 1,-6,-5,0
XROTATE OBJECT 1, 45
MAKE OBJECT CYLINDER 2,5
POSITION OBJECT 2, 6,-5,0

REM ** Culling off for both objects ***
SET OBJECT CULL 1,0
SET OBJECT CULL 2,0

REM *** End program ***
WAIT KEY
END
```

Activity 31.28

In fact, the UNION operation only works predictably with cubes and boxes.

Activity 31.29

```
REM *** Set screen resolution and background ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Position camera ***
AUTOCAM OFF
POSITION CAMERA 0,0,-100
REM *** Create two shapes used ***
MAKE OBJECT CUBE 1,40
MAKE OBJECT BOX 2,10,30,10
POSITION OBJECT 2,0,15,0
REM *** Let viewer see position of shapes ***
WAIT KEY
REM *** Difference shapes ***
PERFORM CSG DIFFERENCE 1,2
REM *** Remove object 2 ***
DELETE OBJECT 2
REM *** Rotate new shape ***
DO
   TURN OBJECT LEFT 1,1.0
   PITCH OBJECT UP 1,1.0
LOOP
REM *** End program ***
END
```

Activity 31.30

```
REM *** Set screen resolution and background ***
SET DISPLAY MODE 1280,1024,32
COLOR BACKDROP 0
BACKDROP ON
REM *** Position camera ***
AUTOCAM OFF
POSITION CAMERA 0,0,-100
REM *** Create two shapes used ***
MAKE OBJECT CUBE 1,40
MAKE OBJECT BOX 2,10,30,10
POSITION OBJECT 2,0,15,0
REM *** Let viewer see position of shapes ***
WAIT KEY
REM *** Intersection shapes ***
PERFORM CSG INTERSETION 1,2
REM *** Remove object 2 ***
DELETE OBJECT 2
REM *** Rotate new shape ***
DO
   TURN OBJECT LEFT 1,1.0
   PITCH OBJECT UP 1,1.0
LOOP
REM *** End program ***
END
```

32

Texturing

Adding Texture

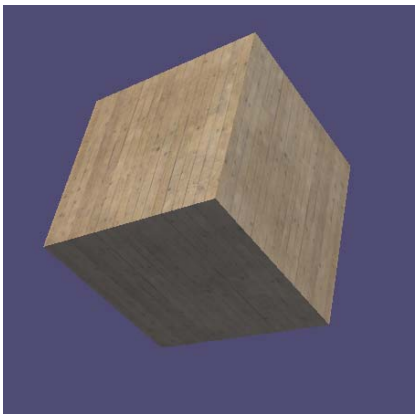
Introduction

The 3D shapes we have created so far look rather bland in white and shades of grey. To make things more interesting we can wrap an image around a 3D shape and thereby enhance its visual impact.

This is known as adding **texture** to the 3D object. An example of a textured cube is shown in FIG-32.1.

FIG-32.1

A Textured Cube



Loading a Texture Image

We need to start by loading the picture we intend to use to texture the 3D object into an image object with a statement such as:

```
LOAD IMAGE "texture01.bmp",1
```

After this has been done, we can transfer the image to the surface of one or more 3D objects.

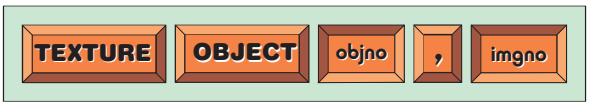
Using the Image as a Texture

The TEXTURE OBJECT Statement

A loaded image can become the texture of a 3D object by executing the TEXTURE OBJECT statement which has the format shown in FIG-32.2.

FIG-32.2

The TEXTURE OBJECT Statement



In the diagram:

objno	is an integer value specifying the object to which the texture is to be applied.
imgno	is an integer value specifying the image to be used as the texture.

For example, we could apply image 1 to object 2 using the line

```
TEXTURE OBJECT 2,1
```

The program in LISTING-32.1 demonstrates how a cube can be textured with a wood image to create the impression of a wooden crate.

LISTING-32.1

Adding Texture to an Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load texture image ***
LOAD IMAGE "textureWood.jpg",1

REM *** Create cube ***
MAKE OBJECT CUBE 1,40

REM *** Add texture to cube ***
TEXTURE OBJECT 1,1

REM *** Position cube ***
POSITION OBJECT 1,25,0,100

REM *** Rotate cube continuously ***
DO
   TURN OBJECT LEFT 1, 1.0
LOOP

REM *** End program ***
END
```

Activity 32.1

Type in and test the program in LISTING-32.1 (*texture01.dbpro*).

Change the texture image to *eyecol.bmp*.

We can see quite clearly from the results of the last Activity that the image is applied separately to each face of the cube. For other shapes, the image may be applied differently.

Activity 32.2

Modify your last program so that *eyecol.bmp* is applied as a texture to a box, cylinder, cone and sphere (any dimensions will do). Create a separate program for each shape.

How often is the image repeated on each of the shapes?

Mipmaps

Texturing a 3D object can be quite time consuming. It may be easy enough to map a 300 by 300 pixel image onto a flat surface which occupies exactly 300 by 300 pixels on the screen, but if the 3D object moves off into the distance, the computer has to work much harder to map the same 300 by 300 image onto an object which now occupies just 23 by 23 pixels on the screen.

To help with this problem DarkBASIC Pro creates more than one copy of any image that is loaded, with each copy being exactly half the size of the last (see FIG-32.3).

FIG-32.3

An Image with Added Mipmaps

As a textured 3D object becomes smaller on the screen, the version of the image used to texture that object changes from the the largest to the smallest.

The program in LISTING-32.2 demonstrates this effect.

LISTING-32.2

Mipmaps in Action

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load texture image ***
LOAD IMAGE "eyecol.bmp",1

REM *** Make cube ***
MAKE OBJECT CUBE 1,40
TEXTURE OBJECT 1,1
POSITION OBJECT 1,25,0,100

REM *** Move cube away from viewer ***
DO
   POSITION OBJECT 1,25,0,OBJECT POSITION Z(1)+10
LOOP

REM *** End program ***
END
```

Activity 32.3

Type in and test the program in LISTING-32.2 (*texture02.dbpro*).

The effect is a fairly subtle one. Look closely at the image on the cube as it moves away from your viewpoint. You should see it become less distinct as it gets smaller.

The LOAD IMAGE Statement Again

We met the LOAD IMAGE statement back in Chapter 20 when we created image objects which were then loaded into sprite objects. But the LOAD IMAGE statement has an expanded form which allows us to dictate whether mipmaps are to be created or not. This version of LOAD IMAGE has the format shown in FIG-32.4.

FIG-32.4

The LOAD IMAGE Statement

In the diagram:

filename	is a string specifying the name of the file to be loaded.

imgno is an integer specifying the ID to be allocated to the image object being created.

tflag is an integer value specifying how the image is to be stored.

- 0 - mipmaps are created
- 1 - no mipmaps are created
- 2 - loads the image in as a cubemap texture (see the chapter on shaders)

When an image is loaded without mipmaps, any object using that image as a texture must continue to use the original image even when the 3D object is greatly reduced in size on the screen.

Activity 32.4

Modify your last program so that no mipmaps are used. To do this change the LOAD IMAGE line to read:

```
LOAD IMAGE "eyecol.bmp",1,1
```

How does the texture on the cube differ in this program from the earlier version?

Tiling

In the next example we'll create the floor of a dungeon by texturing a plane using a cobblestone image.

The program uses the following logic:

```
Load cobblestone image
Create large plane
Rotate plane to be horizontal
Texture plane using the cobblestone image
```

The program itself is given in LISTING-32.3.

LISTING-32.3

Creating a Floor Texture

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture image ***
LOAD IMAGE "stonetile3.jpg",1
REM *** Create large plane ***
MAKE OBJECT PLAIN 1,400,400
REM *** Rotate plane to horizontal ***
XROTATE OBJECT 1,90
REM *** Texture plane ***
TEXTURE OBJECT 1,1

REM *** End program ***
WAIT KEY
END
```

Activity 32.5

Type in and test the program in LISTING-32.3 (*texture03.dbpro*).

The screen dump in FIG-32.5 highlights the problem with the floor - the image has stretched over the whole plane giving a floor that contains only a few unrealistically large blocks rather than hundreds of smaller ones.

FIG-32.5

Floor Texturing

One way to solve the problem would be to use an image which actually shows the hundreds of blocks that we need to create a realistic floor. However, this may not be possible and the image would certainly have to be large if the visuals are to look convincing as a character moves over the floor.

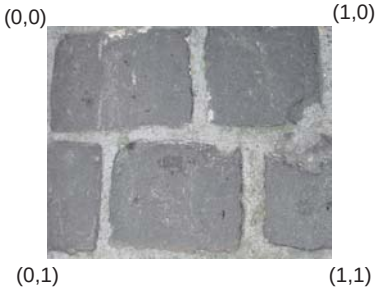
A second option is to make the texture image repeat itself several times over the surface of the plane. This, for rather obvious reasons, is known as **tiling**.

The SCALE OBJECT TEXTURE Statement

Any image employed as a texture uses a UV coordinate system with the top left being point (0,0) and the bottom right (1,1) no matter what the actual size of the image is (see FIG-32.6).

FIG-32.6

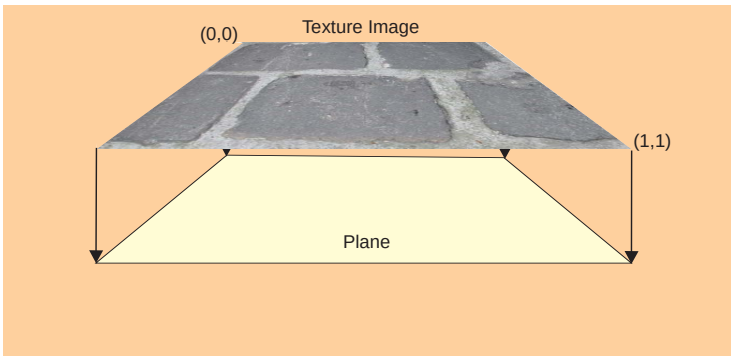
The UV Coordinate System used by a Texture Image



When mapped to a flat plane, the image spreads itself over the object with point (0,0) of the image mapping to the top-left corner of the plane and point (1,1) to the bottom-right corner (see FIG-32.7).

FIG-32.7

The Default Mapping of an Image to a Plane



Using the SCALE OBJECT TEXTURE statement, we can adjust this mapping making only a part of the image stretch over the whole object, or have the image duplicate itself several times creating a tiled effect as shown in FIG-32.8.

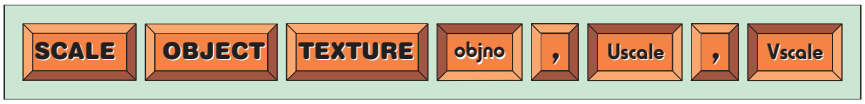
FIG-32.8

The Effects of Using SCALE OBJECT TEXTURE

The SCALE OBJECT TEXTURE statement has the format shown in FIG-32.9.

FIG-32.9

The SCALE OBJECT TEXTURE Statement



In the diagram:

objno	is an integer value specifying the object to which the texture scaling is to be applied.
Uscale	is a real number specifying the multiplication factor along the U axis. Values less than 1 will result in only part of the image being used. Values greater than 1 will result in duplication of the image over the 3D object.
Vscale	is a real number specifying the multiplication factor along the V axis. Values less than 1 will result in only part of the image being used. Values greater than 1 will result in duplication of the image over the 3D object.

The first example shown in FIG-32.8 was created using the line:

```
SCALE OBJECT TEXTURE 1, 0.5, 0.5
```

while the second example was produced using:

```
SCALE OBJECT TEXTURE 1, 5.0, 5.0
```

The program in LISTING-32.4 demonstrates the effect of texture scaling by applying cobblestones texture repeated 10 times in each direction to the plane.

LISTING-32.4

Changing the Texture's Scaling

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture image ***
LOAD IMAGE "stonetile3.jpg",1
REM *** Create and position plain ***
MAKE OBJECT PLAIN 1, 400,400
XROTATE OBJECT 1, -90
REM *** Texture plain ***
TEXTURE OBJECT 1,1
REM *** Scale texture ***
SCALE OBJECT TEXTURE 1,10.0,10.0
REM *** End program ***
WAIT KEY
END
```

Activity 32.6

Type in and test the program given in LISTING-32.4 (*texture04.dbpro*).

Modify the scaling factors to each of the following settings and observe the results:

Uscale	Vscale
5.0	5.0
2.0	2.0
0.5	0.5
5.0	1.0
1.0	5.0

Change the 3D object used in your program from a plane to a sphere and retry each of the settings given above.

Scaling a texture image in this way affects the image itself, so there is no way to return to the original image settings within a program.

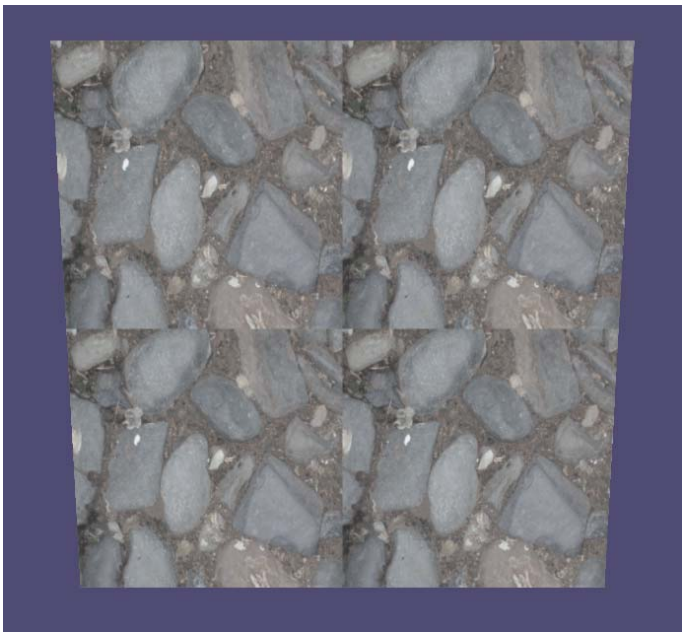
Seamless Tiling

For tiling to be convincing, the ends of the repeating image must butt together without too obvious a join.

If we start with a simple picture and use it as a tiled texture (as shown in FIG-32.10) we get a disappointing effect in which the edge of each image tile is very obvious.

FIG-32.10

A Visible Join Between Tiles

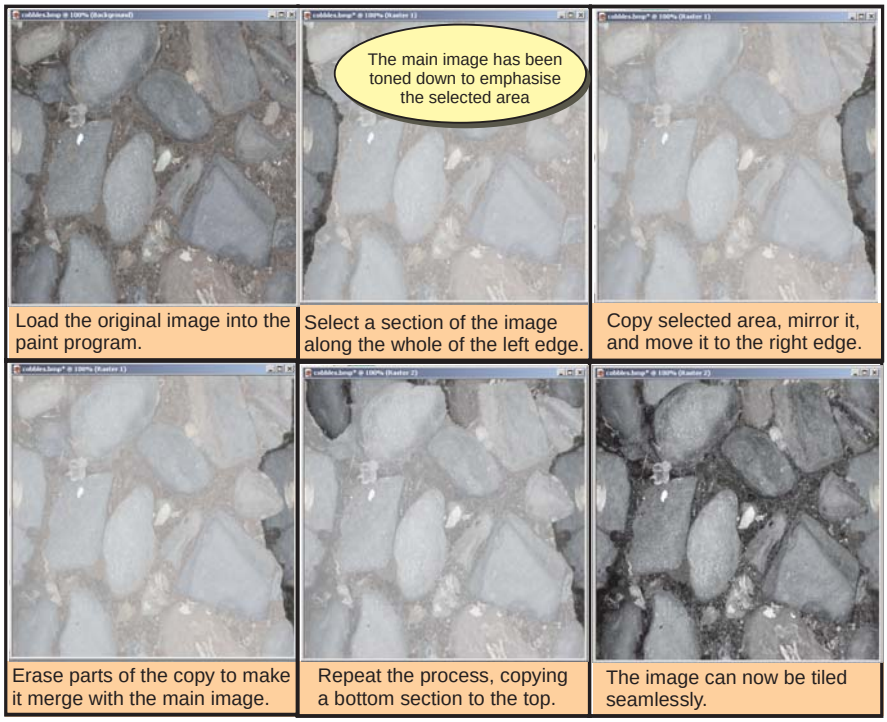


To avoid this, we need to modify the image using a paint package such as Paint Shop Pro or Photoshop.

The stages involved are shown in FIG-32.11.

FIG-32.11

Creating an Image for Seamless Tiling



Of course, it is equally possible to copy the right edge area to the left side and the top to the bottom.

You should choose whatever combinations suit the image in question.

It takes a bit of practice to achieve good results when creating a texture image, but the results can be worth it.

Even when an image is not tiled, we can still have a problem with seams. For example, when the image *eyecol.bmp* is applied as a texture to a sphere, the join between the left and right edges of the image is quite apparent at the back of the sphere, while the top and bottom edges are squeezed into single points at the two "poles".

Activity 32.7

Attempt to modify *eyecol.bmp* (creating a new file named *seamlesseye.bmp*) to give a seamless effect when textured onto a sphere.

Write a short program (*act3207.dbpro*) which applies the new file to a sphere and then rotates the sphere continuously about its local y-axis.

Video Texture

It is even possible to use a video clip as a surface texture. To do this we need to start by loading up a video with an instruction such as:

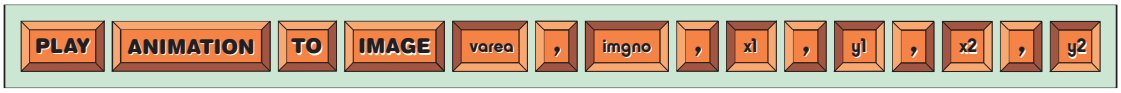
```
LOAD ANIMATION "mv1.mpg",1
```

The PLAY ANIMATION TO IMAGE Statement

Now we need to transfer the video to an image object and this is done using the PLAY ANIMATION TO IMAGE statement which has the format shown in FIG-32.12.

FIG-32.12

The PLAY ANIMATION TO IMAGE Statement



In the diagram:

varea	is an integer value specifying the video that is to be copied to an image area.
imgno	is an integer specifying the image area to which the video is to be copied.
x1,y1,x2,y2	are the coordinates for the top-left and bottom right space which the video is to occupy.

The size of the play area (as set by *x1*, *y*1, *x2*, *y2*) affects the quality of the image when it appears on the 3D object; use too small a set of values and the video will be heavily pixellated; use too large a set of values and displaying the video will put too great a load on the processor/video card and slow the whole thing down.

Once the video has been transferred to the image object, we can then use the image to texture a 3D object in the usual manner. LISTING-32.5 demonstrates the effect by placing a video on a rotating cube.

LISTING-32.5

Using a Video as a Texture

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load video ***
LOAD ANIMATION "lion.mpg",1

REM *** Transfer video to image ***
PLAY ANIMATION TO IMAGE 1,1,0,0,200,200

REM *** Create cube and texture with video ***
MAKE OBJECT CUBE 1,40
TEXTURE OBJECT 1,1

REM *** Move cube away from viewer ***
POSITION OBJECT 1,0,0,100

REM *** Rotate cube ***
DO
   TURN OBJECT LEFT 1,1.0
   PITCH OBJECT DOWN 1,1.0
LOOP

REM *** End program ***
END
```

Activity 32.8

Type in and test the program given in LISTING-32.5 (*texture05.dbpro*).

Modify the program to use very low values for the bottom right corner of the video (i.e. 10,10) and very high values (i.e. 1000,1000). What affect do these changes have on the final result?

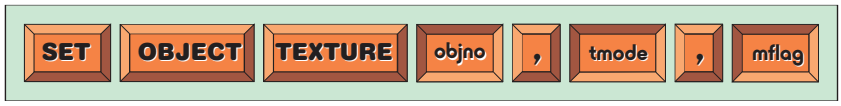
Other Texture Effects

The SET OBJECT TEXTURE Statement

An alternative way to achieve a seamless tiled texture is to use the SET OBJECT TEXTURE statement that adjusts the way in which each copy of the basic image is tiled onto the surface of a 3D object. The statement has the format shown in FIG-32.13.

FIG-32.13

The SET OBJECT TEXTURE Statement



In the diagram:

objno — is an integer value specifying the object whose texture is to be modified.

tmode — is an integer value (1,2,3,4) which directly affects how the tiled texture is applied to the object.

- 1 - normal tiling.
- 2 - images are mirrored/flipped so that identical edges meet.
- 3 - The last pixel along each edge is extended over the remainder of the surface.
- 4 - The image appears only once. The remainder of the surface is black.

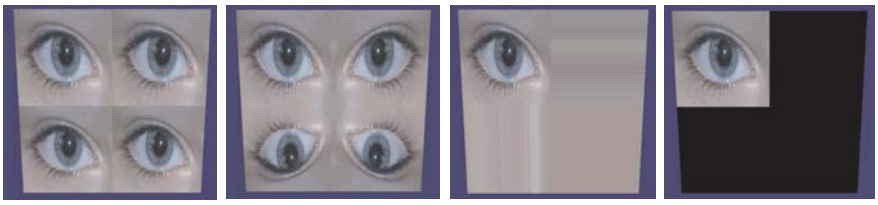
mflag — Determines if mipmapping is to be used.

- 0 - mipmapping used
- 1 - no mipmapping

In FIG-32.14 we see the effects of each possible value for *tmode* when applying *eyecol.bmp* as a tiled (2 by 2) texture on a cube.

FIG-32.14

The Effects of Using Different *tmode* Settings



set object texture 1,1,0 | set object texture 1,2,0 | set object texture 1,3,0 | set object texture 1,4,0

The program in LISTING-32.6 shows a tile textured cube with the *tmode* value of the SET OBJECT TEXTURE statement set to 2.

LISTING-32.6

Using the SET OBJECT TEXTURE Statement

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image ***
LOAD IMAGE "eyecol.bmp",1

REM *** Create a plain ***
MAKE OBJECT PLAIN 1,200,200
```

continued on next page

LISTING-32.6
(continued)

Using the SET OBJECT TEXTURE Statement

```
REM *** Texture object ***
TEXTURE OBJECT 1,1
SCALE OBJECT TEXTURE 1,2,2

REM *** Modify tile mapping ***
SET OBJECT TEXTURE 1,4,0

REM *** End program ***
WAIT KEY
END
```

Activity 32.9

Type in and test the program in LISTING-32.6 (*texture06.dbpro*).

Try other settings for *tmode* and check the effects produced.

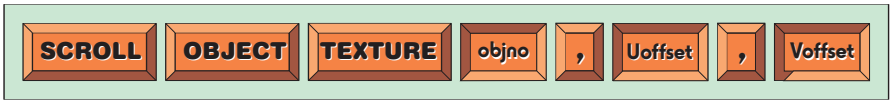
The SCROLL OBJECT TEXTURE Statement

The texture image can be mapped onto an object with a varying degree of offset along either the U or V axes. The overall effect is to modify which part of the texture image is placed at the top left corner of the object.

The effect is created using the SCROLL OBJECT TEXTURE statement which has the format shown in FIG-32.15.

FIG-32.15

The SCROLL OBJECT TEXTURE Statement



In the diagram:

objno	is an integer value specifying the object whose texture is to be scrolled.
Uoffset,Voffset	are a pair of real values representing the coordinates of the image that are to be the top-left corner of the texture when placed on a 3D object.

Examples of this statement in use are shown in FIG-32.16 where *eyecol.bmp* is mapped to a plane object with various *Uoffset, Voffset* values.

FIG-32.16

The Effects of the SCROLL OBJECT TEXTURE Statement

scroll object texture 1,0.1,0.0 | scroll object texture 1,0.0,0.1 | scroll object texture 1,0.1,0.1

A complete program demonstrating the effect is given in LISTING-32.7.

LISTING 32.7

Using the SCROLL OBJECT TEXTURE Statement

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image ***
LOAD IMAGE "eyecol.bmp",1

REM *** Create and texture plane object ***
MAKE OBJECT PLAIN 1,200,200
TEXTURE OBJECT 1,1

REM *** Offset the texture placed on the image***
SCROLL OBJECT TEXTURE 1,0.1,0.0

REM *** End program ***
WAIT KEY
END
```

Activity 32.10

Type in and test the program in LISTING-32.7 (*texture07.dbpro*).

The effect makes a permanent change to the texture for that image, so repeating the same statement creates a further offset.

Activity 32.11

In your last program, add another SCROLL OBJECT TEXTURE statement immediately after the first using the same values.

How does this affect the texture?

By placing the SCROLL OBJECT TEXTURE statement in a loop, the texture can scroll over the surface of the 3D object.

Activity 32.12

Remove the second SCROLL OBJECT TEXTURE statement from your last program and insert the remaining SCROLL OBJECT TEXTURE statement in a DO..LOOP structure.

What affect does this have on the 3D object's texture?

Modify the SCROLL OBJECT TEXTURE statement so that the texture scrolls horizontally rather than vertically.

Activity 32.13

In Chapter 31 you created a program in which two spheres followed the movement of the mouse pointer (*object3D11.dbpro*).

Modify that program so that the spheres are textured using *seamlesseye.bmp*.

Add the appropriate SCROLL OBJECT TEXTURE statements so that the pupils of the eyes face the mouse pointer.

The SET OBJECT TRANSPARENCY Statement

A colour other than black can become the transparent colour using the SET COLOR KEY statement.

When an image containing black is mapped to a sprite, any black areas in the image are automatically transparent when the sprite appears on the screen. However, this is not the case with 3D objects.

To demonstrate this, the next program (see LISTING-32.8) uses the image shown in FIG-32.17 as the texture on a rotating cube.

FIG-32.17

Image used to Texture a Cube

LISTING-32.8

Black Textured Areas are not Transparent on a 3D Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load texture image ***
LOAD IMAGE "DoNot.bmp",2

REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100

REM *** Texture cube with image ***
TEXTURE OBJECT 1,2

REM *** Rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP

REM *** End program ***
END
```

Activity 32.14

Type in and test the program given in LISTING-32.8 (*texture08.dbpro*).

However, we can force a 3D object to make black (or whatever other colour as been set as the background colour using the SET COLOR KEY statement) transparent using the SET OBJECT TRANSPARENCY statement which has the format shown in FIG-32.18.

FIG-32.18

The SET OBJECT TRANSPARENCY

In the diagram:

objno	is an integer value specifying the object whose background texture colour is to be made transparent.

transflag is 0 or 1.
0 - background colour not transparent.
1 - background colour transparent.

Activity 32.15

Modify your last program by adding the line

```
SET OBJECT TRANSPARENCY 1,1
```

immediately before the DO..LOOP structure.

How does this affect the appearance of the cube.

The SET DETAIL MAPPING ON Statement

A second image can be combined with the basic texture image of an object to create a new texture consisting of both images.

For example, if we take the images shown in FIG-32.19 with image 1 being the basic texture and image 2 the overlaid texture, then we achieve the effect shown in FIG-32.20 when these are applied to a cube.

FIG-32.19

The Images Used To Texture a 3D Object

Image 1 Image 2

FIG-32.20

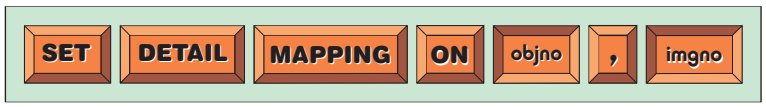
The Two Images Applied to a Cube

Notice that any black areas in image 2 are automatically transparent.

A second image is applied to the texture of an object using the SET DETAIL MAPPING ON statement which has the format shown in FIG-32.21.

FIG-32.21

The SET DETAIL MAPPING ON Statement



In the diagram:

objno is an integer value specifying the object to which a second texture image is to be added.

imgno	is an integer value specifying the image object containing the picture to be used as a second texture.

LISTING-32.9 creates the rotating cube shown above. The main lines of code are

```
LOAD IMAGE "DoNot.bmp",2
```

which loads the secondary image being used and

```
SET DETAIL MAPPING ON 1,2
```

which applies this image as an overlaid texture.

LISTING-32.9

Using a Secondary Texture on a 3D Object

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNot.bmp",2
REM *** Create and texture cube ***
MAKE OBJECT CUBE 1, 40
TEXTURE OBJECT 1,1

REM *** Add secondary texture ***
SET DETAIL MAPPING ON 1,2

REM *** Position cube ***
POSITION OBJECT 1,25,0,100

REM *** Rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 32.16

Type in and test the program given in LISTING-32.9 (*texture09.dbpro*).

Replace *DoNot.bmp* with *DoNotMag.bmp* which contains black text on a magenta background. Set magenta as the transparent colour using the SET IMAGE COLORKEY statement.

We are limited to a single image when overlaying an object's texture with detail. So, attempting to add a second detail image will simply remove the first from the object.

Activity 32.17

Add *FlagMag.bmp* as a second detail image to the cube object in your last program.

What effect does this create?

If an object's texture has been tiled using the SCALE OBJECT TEXTURE statement, any additional image added using SET DETAIL MAPPING ON will also be tiled to the same extent as the original texture.

Activity 32.18

In your last program, remove all references to the *flagmag.bmp* file. Create a tiled effect on the cube by using the SCALE OBJECT TEXTURE statement with the *Uscale* and *Vscale* parameters both set to 2.

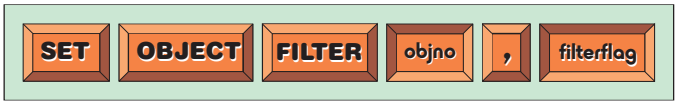
How is the DETAIL MAPPING image on the cube affected by the tiling?

The SET OBJECT FILTER Statement

Different methods of texturing can be specified using the SET OBJECT FILTER statement. The differences achieved have little obvious effect on the visible appearance of the textured object itself, but modify the algorithm used to create that texture. The statement has the format shown in FIG-31-22.

FIG-32.22

The SET OBJECT FILTER Statement



In the diagram:

objno	is an integer value specifying the object whose texture is to be filtered.
filterflag	is 0, 1, or 2 0 - always uses the original image to texture (it never uses the smaller images created by mipmapping). 1 - no smoothing is used. 2 - uses linear filtering.

The program in LISTING-32.10 creates 3 spheres, each textured using one of the filter options. You may see a slight difference in the appearance of the spheres as they move off into the background and reduce in size.

LISTING-32.10

Using the SET OBJECT FILTER Statement

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load image used as texture ***
LOAD IMAGE "grid8by8.bmp",1,1
REM *** Create three spheres ***
MAKE OBJECT SPHERE 1 ,40
MAKE OBJECT SPHERE 2 ,40
MAKE OBJECT SPHERE 3 ,40

REM *** Texture each sphere ***
TEXTURE OBJECT 1,1
TEXTURE OBJECT 2,1
TEXTURE OBJECT 3,1

REM *** Set different filter for each sphere ***
SET OBJECT FILTER 1, 0
SET OBJECT FILTER 2, 1
SET OBJECT FILTER 3, 2

REM *** Position spheres ***
POSITION OBJECT 1,-42,0,0
POSITION OBJECT 2,0,0,0
POSITION OBJECT 3,42,0,0
```

continued on next page

LISTING-32.10
(continued)

Using the SET OBJECT FILTER Statement

```
REM *** Moves spheres ***
FOR z = 1 TO 4000
   MOVE OBJECT 1,1
   MOVE OBJECT 2,1
   MOVE OBJECT 3,1
   WAIT 10
NEXT z

REM *** End program ***
WAIT KEY
END
```

Activity 32.19

Type in and test the program given above (*texture10.dbpro*).

Summary

- Texturing involves mapping an image onto the surface of a 3D object.
- Use LOAD IMAGE to load any image which is to be used to texture a 3D object.
- Use TEXTURE OBJECT to map an image to an object.
- Images stretch automatically to fit the surface of an object.
- On a cube or box the image is repeated on each face.
- On other 3D objects, the image appears only once.
- Mipmaps are smaller versions of the original image which are used to speed up mapping when a textured object becomes much smaller than the original image.
- Tiling is the application of an image multiple times to the same surface.
- Use SCALE OBJECT TEXTURE to create a tiled texture.
- To create a seamless tile, make sure the opposite edges are complementary.
- Use PLAY ANIMATION TO IMAGE and TEXTURE OBJECT to display a video on the surface of an object.
- Use SET OBJECT TEXTURE to specify how an image is mapped to a surface.
- Use SCROLL OBJECT TEXTURE to create an offset mapping of the image on a 3D surface.
- Use SET OBJECT TRANSPARENCY to make black areas of a 3D object disappear.
- Use SET DETAIL MAPPING ON to apply a second image to an already textured object.
- Use SET OBJECT FILTER to modify how an image is filtered when being mapped onto an object.

Other Visual Effects

Introduction

Although adding texture to a 3D object is the commonest way of changing an object's appearance, it is by no means the only option available. In this section we'll see some other options that are available to us in DarkBASIC Pro.

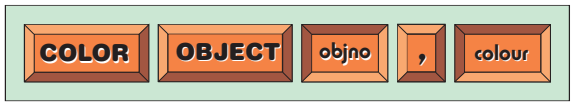
Changing Colour and Transparency

The COLOR OBJECT Statement

Rather than add a texture to a 3D object, we can give it a surface of a specific colour using the COLOR OBJECT statement which has the format shown in FIG-32.23.

FIG-32.23

The COLOR OBJECT Statement



In the diagram:

objno	is an integer value specifying the object which is to be coloured.
colour	is an integer value specifying the colour to be used on the object's surface.

For example, we could give object 1 a red surface using the line:

```
COLOR OBJECT 1, RGB(255,0,0)
```

An example of this statement in operation is given in LISTING-32.11 which colours a rotating cube in red, changing to green when a random event occurs.

LISTING-32.11

Using COLOR OBJECT

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Seed random number generator ***
RANDOMIZE TIMER()
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100
REM *** Colour cube red ***
COLOR OBJECT 1, RGB(255,0,0)
REM *** Rotate Cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
   REM *** One chance in 1000 of changing to green ***
   IF RND(1000) = 500
      COLOR OBJECT 1, RGB(0,255,0)
   ENDIF
LOOP
REM *** End program ***
END
```

Activity 32.20

Type in and test the program given in LISTING-32.11 (*texture11.dbpro*).

A coloured surface cannot be used in conjunction with a main texture, but secondary textures (created using SET DETAIL MAPPING ON) may still be used.

Activity 32.21

Modify your last program so that *DoNot.bmp* is used as a secondary texture on the surface of the cube.

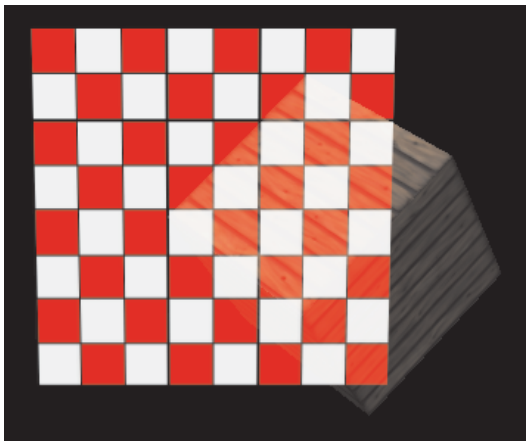
The GHOST OBJECT ON Statement

In FIG-32.24 we can see a cube which is semi-transparent with the grided plane in the background showing through the cube.

FIG-32.24

A Transparent Cube

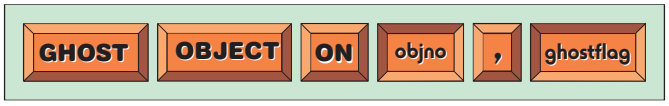
You'll have to look closely to see the cube!



We can create this effect using the GHOST OBJECT ON statement which has the format shown in FIG-32.25.

FIG-32.25

The GHOST OBJECT ON Statement



In the diagram:

objno	is an integer value specifying the 3D object to be made semi-transparent.
ghostflag	is 0 to 5. 0 - object is semi-transparent 1 - object uses negative transparency 2 - object is semi-transparent but lighter 3 - uses the image's alpha channel 4 - similar to 1 but lighter 5 - object is opaque

The program in LISTING-32.12 demonstrates the effect of this instruction by rotating the cube with a textured plane in the background.

LISTING-32.12

Using the GHOST OBJECT ON Statement

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load texture images ***
LOAD IMAGE "grid8by8.bmp",1
LOAD IMAGE "DoNot.bmp",2

REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100

REM *** Texture cube ***
TEXTURE OBJECT 1,1

REM *** Create background plane ***
MAKE OBJECT PLAIN 2,100,100
TEXTURE OBJECT 2, 2
POSITION OBJECT 2,0,0,200

REM *** Create semi-transparent cube ***
GHOST OBJECT ON 1,0

REM *** Rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP

REM *** End program ***
END
```

Activity 32.22

Type in and test the program given in LISTING-32.12 (*texture12.dbpro*).

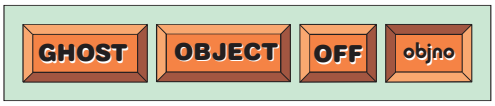
Modify the *transflag* value in the GHOST OBJECT ON statement and observe the effects of the various settings.

The GHOST OBJECT OFF Statement

The semi-transparency effect created by GHOST OBJECT ON can be disabled using the GHOST OBJECT OFF statement which has the format shown in FIG-32.26.

FIG-32.26

The GHOST OBJECT OFF Statement



In the diagram:

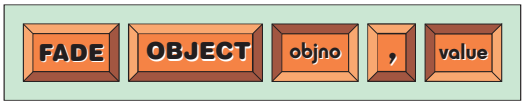
objno	is an integer value specifying the 3D object in which the semi-transparency mode is switched off.

The FADE OBJECT Statement

The amount of light reflected from the surface of a 3D object can be modified using the FADE OBJECT statement. This allows settings varying between no light reflected and twice normal reflection. The statement has the format shown in FIG-32.27.

FIG-32.27

The FADE OBJECT Statement



In the diagram:

objno	is an integer value specifying the 3D object whose reflective index is to be changed.
value	is an integer value between 0 and 200. A value of zero means that the object will reflect no light; a value of 100 creates normal amount of reflected light; a value of 200 gives twice the normal amount of reflected light.

The program in LISTING-32.13 demonstrates the effect of this statement by gradually reducing the reflective value from 200 to zero.

LISTING-32.13

Using FADE OBJECT to make a Transparent Object Disappear

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNot.bmp",2

REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100

REM *** Texture cube ***
TEXTURE OBJECT 1,1

REM *** Create background plain ***
MAKE OBJECT PLAIN 2,100,100
TEXTURE OBJECT 2, 2
POSITION OBJECT 2,0,0,200

REM *** Start reflective value at 200 ***
reflectivity = 200

REM *** rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
   FADE OBJECT 1,reflectivity
   REM *** Decrement reflective value until it reaches zero ***
   IF reflectivity > 0
      DEC reflectivity
   ENDIF
   WAIT 10
LOOP

REM *** End program ***
END
```

Activity 32.23

Type in and test the program given in LISTING-32.13 (*texture13.dbpro*).

When used in combination with the GHOST OBJECT ON statement, the FADE OBJECT statement can make a semi-transparent object disappear completely.

Activity 32.24

In your last program, add the line

```
GHOST OBJECT ON 1,0
```

immediately before the DO..LOOP structure.

How does this affect the cube?

There's still more to be said about texturing objects but we'll leave that to a later chapter after we've covered other basic concepts such as cameras and lighting.

Summary

- Use COLOR OBJECT to tint the surface of a 3D object.
- Use GHOST OBJECT ON to make an object transparent.
- Use GHOST OBJECT OFF to make a transparent object opaque.
- Use FADE OBJECT to reduce the amount of light reflected by an object.
- When used on a transparent object, FADE OBJECT can make that object invisible.
- When used on an opaque object, FADE OBJECT can make that object completely black.

Images with an Alpha Channel

Introduction

As we saw back in Chapter 30, some picture files contain an alpha channel which can affect the visibilty of the main image. This type of picture file, when used as a texture, affects the overall result obtained.

Using Images with an Alpha Channel

The image *windmillshaped.tga* contains an image and an alpha channel as shown in FIG-32.28.

FIG-32.28

An Image and its Alpha Channel

Image Alpha Channel

We've already seen that the SET OBJECT TRANSPARENCY statement affects the black area of an texture image, but the statement also controls how an alpha channel within an image affects the final texture.

The program in LISTING 32-14 textures as cube using *windmillshaped.tga*. With the default settings, the alpha channel within the image has no effect on the texturing of the cube, but after a SET OBJECT TRANSPARENCY statement is used to modify the texturing, the alpha channel changes the final look of the cube.

LISTING-32.14

Texturing with Alpha-Channel Images

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280, 1024,32

REM *** Create cube ***
MAKE OBJECT CUBE 1,10

REM *** Texture cube ***
LOAD IMAGE "windmillshaped.tga",1
TEXTURE OBJECT 1,1

REM *** Rotate cube ***
DO
   REM *** IF key pressed, use alpha channel ***
   IF INKEY$() <> ""
      SET OBJECT TRANSPARENCY 1,1
   ENDIF
   TURN OBJECT LEFT 1,1
LOOP
REM *** End program ***
END
```

Activity 32.25

Type in and test the program in LISTING-32.14 (*texture14.dbpro*).

Notice that the other parts of the image do not disappear completely when the alpha channel is activated. This is because the darkened areas of the alpha channel are grey and not black. If black had been used, all other parts of the image would have become invisible.

An image with an alpha channel also produces an effect when option 3 is used with the GHOST OBJECT ON statement.

Activity 32.26

In your last program, change the line

```
SET OBJECT TRANSPARENCY 1,1
```

to

```
GHOST OBJECT ON 1,3
```

Observe how this affect the program's display.

Summary

- Use SET OBJECT TRANSPARENCY with an alpha channel image to create transparent or semitransparent texturing effects.
- Use GHOST OBJECT ON with option 3 to make use of the alpha channel information in creating the final ghosting effect.

Creating a Complex 3D Shape

Introduction

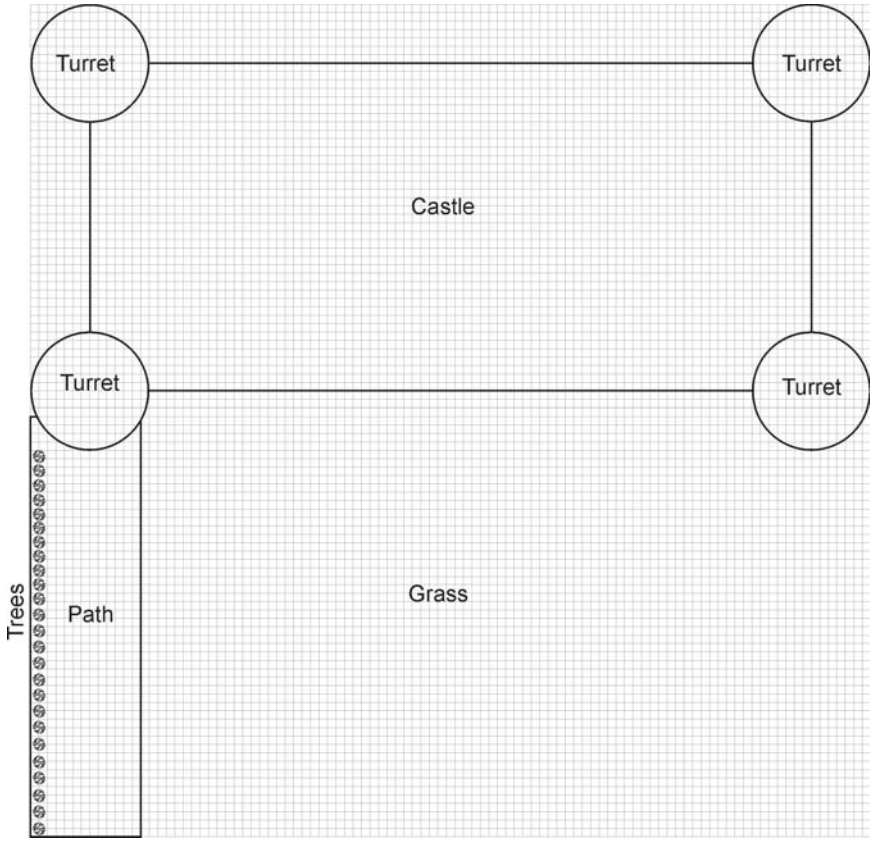
By combining the basic 3D shapes available to us we can create almost any complex shape. However, this may take a considerable amount of work and require a great deal of code. In reality we would probably create objects such as these in a separate 3D drawing package and then import the resulting file into our DarkBASIC Pro program (we'll see how to do this in Chapter 36). However, just to give us some practice, we'll try creating a simple castle using only DarkBASIC Pro.

Designing the Castle

We are going to save ourselves a great deal of time later if we start by doing a grided plan of the castle to give ourselves the basic layout and sizes. FIG-32.29 shows a plan of the castle.

FIG-32.29

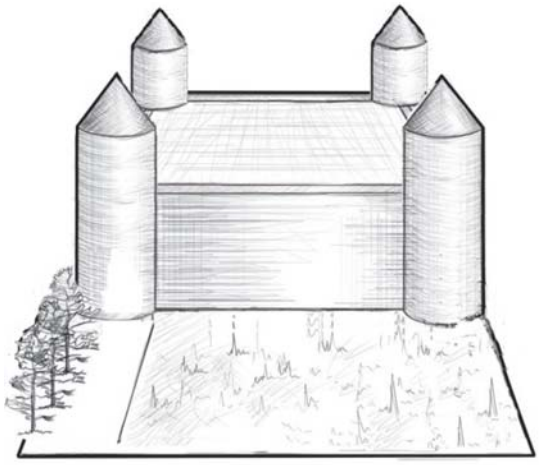
A Rough Sketch of the Object Required



Next, we need to create a more traditional drawing showing the characteristics of the castle (see FIG-32.30). We might also draw specific parts in more detail.

FIG-32.30

A More Detailed
Design of the Castle



Gathering the Components

The actual texture files being used need to be obtained or created. If you're not much of an artist, then you'll find plenty of texture files on the Internet, but if you're intending to create a commercial product, remember that almost everything you see on the Internet will be owned by someone and they expect to be paid if you are going to use their material. Even material that is advertised as free may still need to be paid for when used in a commercial product.

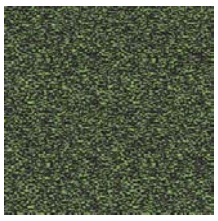
The textures used on the castle are shown in FIG-32.31.

FIG-32.31

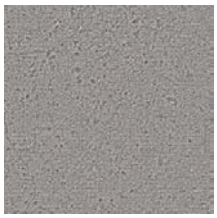
Textures Used

Filename : CobbleStones.jpg
Object : path

Filename : stone.jpg
Object : castle walls and turrets



Filename : grass1.jpg
Object : lawn



Filename : ceil_U3_01.jpg
Object : castle ceiling

Filename : rock2.jpg
Object : castle floor

Filename : tiles.jpg
Object : castle and turret roofs

Filename : tree.jpg
Object : trees lining path

Creating the Code

The coding (see LISTING-32.15) is long but fairly straight forward. The complete castle is created by a function, but this calls other functions which draw the various parts of the castle.

Splitting the code in this way will help us keep the structure as understandable as possible.

The Code

LISTING-32.15

Drawing the Castle

```
REM *** Building Components ***
REM *** Texture Images ***
#CONSTANT grass       1
#CONSTANT road        2
#CONSTANT tree        3
#CONSTANT flooring    4
#CONSTANT wall        5
#CONSTANT roofing     6
#CONSTANT cover       8
#CONSTANT trellis     9
#CONSTANT transport   10
REM *** 3D Objects ***
#CONSTANT lawn        1
#CONSTANT approach    2
#CONSTANT floor1      3
#CONSTANT ceiling     5
#CONSTANT tree1       7
#CONSTANT frontwall   51
#CONSTANT backwall    52
#CONSTANT leftwall    53
#CONSTANT rightwall   54
#CONSTANT innerwall   55
#CONSTANT turret1     56
#CONSTANT turretroof1 57
#CONSTANT turret2     58
#CONSTANT turretroof2 59
#CONSTANT turret3     60
#CONSTANT turretroof3 61
#CONSTANT turret4     62
#CONSTANT turretroof4 63
#CONSTANT roof1       64
#CONSTANT column      70


REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
DrawCastle()
WAIT KEY
END



FUNCTION DrawCastle()
   LoadImages()
   DrawGrounds()
   DrawExternalWalls()
   DrawRoofandCeiling()
   DrawTurrets()
   DrawInternalColumns()
ENDFUNCTION


FUNCTION LoadImages()
   REM *** Load texture images ***
   LOAD IMAGE "grass1.jpg",lawn
   LOAD IMAGE "CobbleStones.jpg",road
   LOAD IMAGE "tree.jpg",tree
   LOAD IMAGE "rock2.jpg",flooring
   LOAD IMAGE "stone.jpg",wall
   LOAD IMAGE "tiles.jpg",roofing
   LOAD IMAGE "ceil_U3_01.jpg",cover
ENDFUNCTION
```

continued on next page

LISTING-32.15
(continued)

Drawing the Castle

```
FUNCTION DrawGrounds()
   REM *** Create lawn ***
   MAKE OBJECT PLAIN lawn, 500,700
   TEXTURE OBJECT lawn, grass
   SCALE OBJECT TEXTURE lawn, 100,100
   XROTATE OBJECT lawn,-90
   POSITION OBJECT lawn,250,0,350
   SET DETAIL MAPPING ON lawn,transport
   REM *** Create approach road ***
   MAKE OBJECT PLAIN approach, 50,700
   TEXTURE OBJECT approach,road
   SCALE OBJECT TEXTURE approach,8,50
   XROTATE OBJECT approach,-90
   POSITION OBJECT approach, -25,0,350
   REM *** Create castle floor ***
   MAKE OBJECT PLAIN floor1, 550,300
   TEXTURE OBJECT floor1, flooring
   SCALE OBJECT TEXTURE floor1,100,100
   XROTATE OBJECT floor1,-90
   POSITION OBJECT floor1,225,0,850
   DrawTrees()
ENDFUNCTION

FUNCTION DrawExternalWalls()
   REM *** Create front wall ***
   MAKE OBJECT PLAIN frontwall,550,100
   TEXTURE OBJECT frontwall,wall
   SCALE OBJECT TEXTURE frontwall,30,6
   POSITION OBJECT frontwall, 225,50,700

   REM *** Create back wall ***
   MAKE OBJECT PLAIN backwall,550,100
   TEXTURE OBJECT backwall,wall
   SCALE OBJECT TEXTURE backwall,50,10
   POSITION OBJECT backwall, 225,50,1000

   REM *** Create left wall ***
   MAKE OBJECT PLAIN leftwall,300,100
   TEXTURE OBJECT leftwall,wall
   SCALE OBJECT TEXTURE leftwall,30,10
   YROTATE OBJECT leftwall, -90
   POSITION OBJECT leftwall,-50,50,850

   REM *** Create right wall ***
   MAKE OBJECT PLAIN rightwall,300,100
   TEXTURE OBJECT rightwall,wall
   SCALE OBJECT TEXTURE rightwall,30,10
   YROTATE OBJECT rightwall,90
   POSITION OBJECT rightwall,500,50,850
ENDFUNCTION

FUNCTION DrawTurrets()
   REM *** Create first turret ***
   MAKE OBJECT CYLINDER turret1,200
   SCALE OBJECT turret1,40,100,40
   MAKE OBJECT CONE turretroof1,81
   REM *** Texture turret ***
   TEXTURE OBJECT turret1,wall
   SCALE OBJECT TEXTURE turret1,10,10
   TEXTURE OBJECT turretroof1,roofing
   SCALE OBJECT TEXTURE turretroof1,5,10
   REM *** Position turret ***
   POSITION OBJECT turret1, -25,100,970
   POSITION OBJECT turretroof1,-25,240,970
```

continued on next page

LISTING-32.15
(continued)

```
   REM *** Second turret ***
   MAKE OBJECT CYLINDER turret2,200
   SCALE OBJECT turret2,40,100,40
   MAKE OBJECT CONE turretroof2,81
   REM *** Texture turret ***
   TEXTURE OBJECT turret2,wall
   SCALE OBJECT TEXTURE turret2,10,10
   TEXTURE OBJECT turretroof2,roofing
   SCALE OBJECT TEXTURE turretroof2,5,10
   REM *** Position turret ***
   POSITION OBJECT turret2, 475,100,970
   POSITION OBJECT turretroof2,475,240,970
   REM *** Third turret ***
   MAKE OBJECT CYLINDER turret3,200
   SCALE OBJECT turret3,40,100,40
   MAKE OBJECT CONE turretroof3,81
   REM *** Texture turret ***
   TEXTURE OBJECT turret3,wall
   SCALE OBJECT TEXTURE turret3,10,10
   TEXTURE OBJECT turretroof3,roofing
   SCALE OBJECT TEXTURE turretroof3,5,10
   REM *** Position turret ***
   POSITION OBJECT turret3, -25,100,725
   POSITION OBJECT turretroof3,-25,240,725
   REM *** Fourth turret ***
   MAKE OBJECT CYLINDER turret4,200
   SCALE OBJECT turret4,40,100,40
   MAKE OBJECT CONE turretroof4,81
   REM *** Texture turret ***
   TEXTURE OBJECT turret4,wall
   SCALE OBJECT TEXTURE turret4,10,10
   TEXTURE OBJECT turretroof4,roofing
   SCALE OBJECT TEXTURE turretroof4,5,10
   REM *** Position turret ***
   POSITION OBJECT turret4, 475,100,725
   POSITION OBJECT turretroof4,475,240,725
ENDFUNCTION

FUNCTION DrawRoofAndCeiling()
   REM *** Create main roof ***
   MAKE OBJECT PLAIN roof1,550,300
   REM *** Texture main roof ***
   TEXTURE OBJECT roof1,roofing
   SCALE OBJECT TEXTURE roof1,50,10
   REM *** Position roof ***
   XROTATE OBJECT roof1, -90
   POSITION OBJECT roof1,225,100,850
   REM *** Create ceiling ***
   MAKE OBJECT PLAIN ceiling,550,300
   REM *** Texture ceiling ***
   TEXTURE OBJECT ceiling,cover
   SCALE OBJECT TEXTURE ceiling,5,2
   REM *** Position ceiling ***
   XROTATE OBJECT ceiling, -90
   POSITION OBJECT ceiling,225,99,850
ENDFUNCTION

FUNCTION DrawInternalColumns()
   RANDOMIZE TIMER()
   FOR col = column TO column + 80
      MAKE OBJECT BOX col,20,99.8,20
      TEXTURE OBJECT col,wall
      SCALE OBJECT TEXTURE col,5,30
      POSITION OBJECT col,RND(515)-20,49.95, RND(270)+710
   NEXT col
ENDFUNCTION
```

continued on next page

LISTING-32.15
(continued)

```
FUNCTION DrawTrees()
   REM *** Create trees ***
   FOR c = tree1 TO tree1 + 30
      MAKE OBJECT PLAIN c,25,35
      TEXTURE OBJECT c, tree
      SET OBJECT TRANSPARENCY c,1
      POSITION OBJECT c,-45,17,(c-6) * 17.5
   NEXT c
ENDFUNCTION
```

The function *DrawInternalColumns()* adds randomly placed columns within the castle. This will allow our player to have obstacles to navigate without having to go to a great deal of trouble designing an exact layout for the castle's interior.

The *DrawTrees()* function draws a set of trees by texturing a set of planes with a tree image.

Activity 32.27

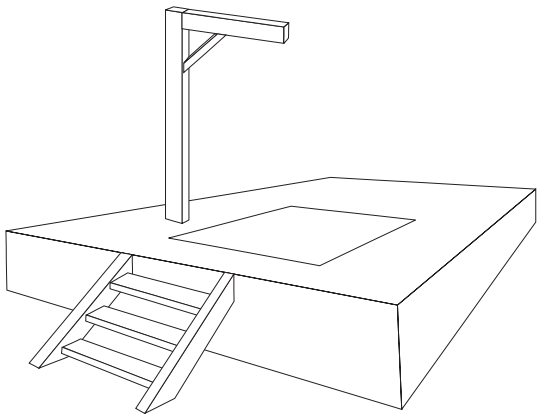
Type in and test the program given above (*castle01.dbpro*).

Activity 32.28

Remove the main section from the program, leaving only the constants and functions. Save this as *castle.dbpro*.

Activity 32.29

Create a program (*gallows.dbpro*) containing a function, *DrawGallows()*, that produces a 3D gallows similar to that in the sketch below. Use any appropriate textures.



The gallows platform should be centred on (-125,7.5,0) and be 15 units high, and 50 units in width and depth. Place the gallows on a 300 by 300 cobbled plane.

In the main section of the program include the lines

```
POSITION CAMERA 0,8,-100
POINT CAMERA -150,10,0
```

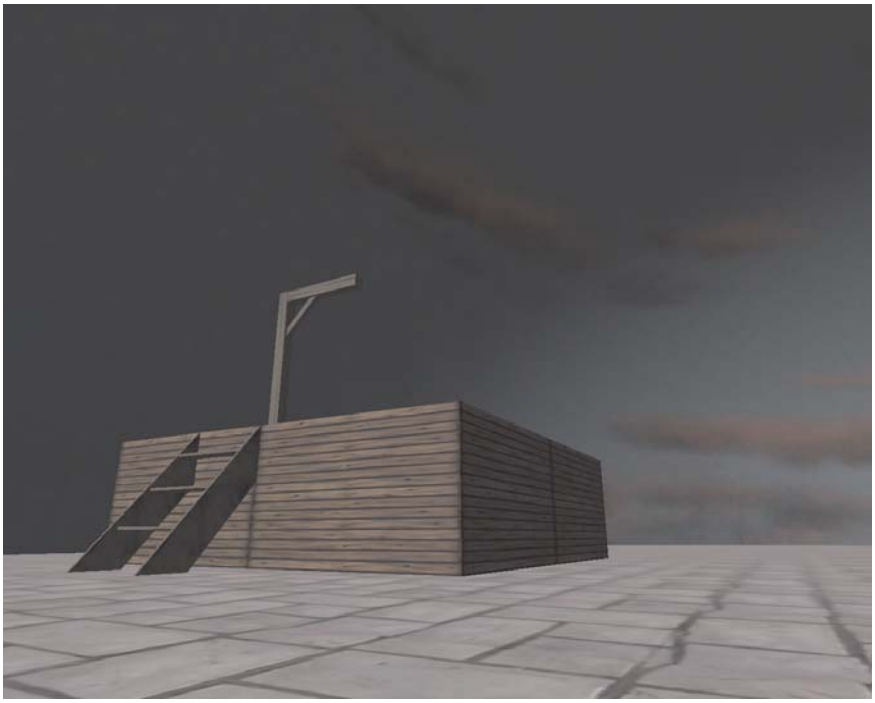
after the call to the *DrawGallows()* function. This will ensure that the camera is pointing at the gallows.

Sky Spheres

Both the castle and the gallows look a bit out of place with the blue background. One way to create a more natural environment is to place a large sphere round the whole thing and texture that sphere with an image of the sky. This is known as a **sky sphere**. FIG-32.32 shows the results obtained by adding a sky sphere to the gallows program.

FIG-32.32

Using a Sky Sphere



To implement a sky sphere in our gallows program, we'll start by making the ground plane set up in *DrawGallows()* a bit larger

```
MAKE OBJECT PLAIN GroundObj,2000,2000
```

and then increase the tiling so the cobbles don't get too large:

```
SCALE OBJECT TEXTURE GroundObj,150,150
```

These are the only changes required in the *DrawGallows()* function. Now we can add a few lines to the main section. First we need the image to be used to texture the sphere with a sky effect:

```
LOAD IMAGE "sky.jpg",1
```

Next we can create the sphere with the same diameter as the plane:

```
MAKE OBJECT SPHERE 1,-2000,50,50
```

Notice that the sphere has been created with extra polygons. This helps smooth out the sky background.

Apparently all we need to do now is texture the sphere:

```
TEXTURE OBJECT 1,1
```

Activity 32.30

Modify your *gallows.dbpro* program using the lines given above. The main section should be coded as:

```
REM *** Set up screen ***
SET DISPLAY MODE 1280,1024,32

DrawGallows()

REM *** Create the sky sphere ***
LOAD IMAGE "sky.jpg",1
MAKE OBJECT SPHERE 1,2000,50,50
TEXTURE OBJECT 1,1

REM *** Set up camera ***
POSITION CAMERA 0,8,-100
POINT CAMERA -150,10,0

REM *** End program ***
WAIT KEY
END
```

Don't worry if you don't see any sky!

We can't see the texture on the sphere because we're on the inside of the sphere and DarkBASIC Pro has culled the polygons that make up the sphere.

There are two ways to solve this problem. The first is to switch off culling on the sphere. This can be done using the line:

```
SET OBJECT CULL 1,0
```

Activity 32.31

Add the above line to the main section of your code. Is the sky now visible?

An alternative way of displaying the sphere's texture is to turn the sphere inside out! This is done by specifying a negative size for the sphere when it is being created.

Activity 32.32

Change the line

```
MAKE OBJECT SPHERE 1,2000,50,50
```

to have a negative size value:

```
MAKE OBJECT SPHERE 1,-2000,50,50
```

and remove the SET OBJECT CULL statement.

How does this affect the sphere's texture?

To cure the problem of an inverted mirror image (which isn't actually a problem in this case) we need to save the original image as an inverted mirror image in the first place and this will then be reversed when the image is used as a texture.

Activity 32.33

If you have an appropriate paint package, invert and mirror the image *sky.jpg* (save the resulting image as *skyIM.jpg*) and use the new image as a texture for the sky sphere.

If you don't have an appropriate package, the inverted mirror image is supplied with the images for this chapter.

Summary

- A sky sphere allows us to create a sky affect around our 3D world.
- A sky sphere is a large sphere textured with an image of the sky.
- To make the sphere's texture visible from within the sphere, switch off the sphere's culling or create and inverted mirror image of the sky and use it to texture a sphere with a negative diameter value.

Solutions

Activity 32.1

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture image ***
LOAD IMAGE "eyecol.bmp",1
REM *** Create cube ***
MAKE OBJECT CUBE 1, 40
REM *** Add texture to cube ***
TEXTURE OBJECT 1,1
REM *** Position cube ***
POSITION OBJECT 1,25,0,100
REM *** Rotate cube continuously ***
DO
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 32.2

To change shape requires the line

```
MAKE OBJECT CUBE 1,40
```

to be replaced by each of the following in turn:

```
MAKE OBJECT BOX 1, 10,20,30
MAKE OBJECT CYLINDER 1, 15
MAKE OBJECT CONE 1, 15
MAKE OBJECT SPHERE 1, 10
```

The cube and box repeat the texture image on each side; other shapes show the image only once.

Activity 32.3

No solution required.

Activity 32.4

The texture seems a little more blurred when using mipmaps, but the texturing process seems to be carried out at a faster frame rate.

Activity 32.5

No solution required.

Activity 32.6

No solution required.

Activity 32.7

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture image ***
LOAD IMAGE "seamlesseye.bmp",1
REM *** Create and texture sphere ***
MAKE OBJECT SPHERE 1, 40,40,40
TEXTURE OBJECT 1,1
DO
   TURN OBJECT LEFT 1, 1.0
LOOP
END
```

Activity 32.8

By changing the coordinates of the bottom right corner of the video to 10,10, the video itself is only 11 pixels by 11 pixels, and, when expanded to cover the surface of the cube becomes indistinct and blocky.

By changing the corner values from 10,10 to 1000,1000 we make the video 1001 pixels by 1001 pixels (actually larger than the original recording). Since we cannot add any detail which was not in the original recording, this size does not achieve any better results than a lower resolution (say 640 by 640), but does increase the load on the video hardware and slows down the whole process.

Activity 32.9

No solution required.

Activity 32.10

No solution required.

Activity 32.11

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load image ***
LOAD IMAGE "eyecol.bmp",1
REM *** Create and texture plain object ***
MAKE OBJECT PLAIN 1,200,200
TEXTURE OBJECT 1,1
REM *** Offset texture on image***
SCROLL OBJECT TEXTURE 1,0.1,0.0
WAIT KEY
SCROLL OBJECT TEXTURE 1,0.1,0.0
REM *** End program ***
WAIT KEY
END
```

We can see from the results produced by the program that the scroll effect is cumulative, with the image moving another step to the left when the second SCROLL OBJECT TEXTURE statement is applied.

Activity 32.12

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load image ***
LOAD IMAGE "eyecol.bmp",1
REM *** Create and texture plain object ***
MAKE OBJECT PLAIN 1,200,200
TEXTURE OBJECT 1,1
REM *** Offset texture placed on image***
DO
   SCROLL OBJECT TEXTURE 1,0.1,0.0
LOOP
REM *** End program ***
WAIT KEY
END
```

The image scrolls vertically.

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load image ***
LOAD IMAGE "eyecol.bmp",1
```

```
REM *** Create and texture plain object ***
MAKE OBJECT PLAIN 1,200,200
TEXTURE OBJECT 1,1
REM *** Offset texture placed on image***
DO
   SCROLL OBJECT TEXTURE 1,0.0,0.1
LOOP
REM *** End program ***
WAIT KEY
END
```

Activity 32.13

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make and position spheres ***
LOAD IMAGE "eyecol.bmp",1
MAKE OBJECT SPHERE 1, 40
POSITION OBJECT 1,25,-20,100
MAKE OBJECT SPHERE 2, 40
POSITION OBJECT 2, -25,-20,100
REM *** Texture spheres ***
TEXTURE OBJECT 1,1
TEXTURE OBJECT 2,1
REM *** Offset so eyes are at front ***
SCROLL OBJECT TEXTURE 1, 0.51,0.0
SCROLL OBJECT TEXTURE 2,0.51,0.0
REM *** Make eyes follow mouse ***
DO
   x3D = MOUSEX() - SCREEN WIDTH()/2
   y3D = -(MOUSEY() - SCREEN HEIGHT()/2)
   POINT OBJECT 1, x3D,y3D,-300
   POINT OBJECT 2, x3D,y3D,-300
LOOP
REM *** End program ***
WAIT KEY
END
```

Activity 32.14

No solution required.

Activity 32.15

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture image ***
LOAD IMAGE "DoNot.bmp",2
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100
REM *** Texture cube with image ***
TEXTURE OBJECT 1,2
REM *** Make black areas of texture ***
REM *** transparent ***
SET OBJECT TRANSPARENCY 1,1
REM *** Rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Any part of the cube which is textured with black disappears.

Activity 32.16

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Make magenta transparent ***
SET IMAGE COLORKEY 255,0,255
REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNotMag.bmp",2
REM *** Create and texture cube ***

MAKE OBJECT CUBE 1, 40
TEXTURE OBJECT 1,1
REM *** Add secondary texture ***
SET DETAIL MAPPING ON 1,2
REM *** Position cube ***
POSITION OBJECT 1,25,0,100
REM *** Rotate cube ***
DO PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Activity 32.17

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Set magenta as transparent ***
SET IMAGE COLORKEY 255,0,255
REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNotMag.bmp",2
LOAD IMAGE "FlagMag.bmp",3
REM *** Create and texture cube ***
MAKE OBJECT CUBE 1, 40
TEXTURE OBJECT 1,1
REM *** Add text as secondary texture ***
SET DETAIL MAPPING ON 1,2
REM *** Try using another texture ***
SET DETAIL MAPPING ON 1,3
REM *** Position cube ***
POSITION OBJECT 1,25,0,100
REM *** Rotate cube ***
DO PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

Only the wood and flag textures show; the text *"DO NOT OPEN"* is missing.

Activity 32.18

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Set magenta as transparent ***
SET IMAGE COLORKEY 255,0,255
REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNotMag.bmp",2
REM *** Create and texture cube ***
MAKE OBJECT CUBE 1, 40
TEXTURE OBJECT 1,1
REM *** Tile cube's texture ***
SCALE OBJECT TEXTURE 1,2,2
REM *** Add secondary texture ***
SET DETAIL MAPPING ON 1,2
REM *** Position cube ***
POSITION OBJECT 1,25,0,100
REM *** Rotate cube ***
DO PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
LOOP
REM *** End program ***
END
```

The DETAIL MAPPING image is also tiled.

Activity 32.19

No solution required.

Activity 32.20

No solution required.

Activity 32.21

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Seed random number generator ***
RANDOMIZE TIMER()
REM *** Load image ***
LOAD IMAGE "DoNot.bmp",1
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
REM *** Create detail mapping ***
SET DETAIL MAPPING ON 1,1
POSITION OBJECT 1,25,0,100
REM *** Colour cube red ***
COLOR OBJECT 1, RGB(255,0,0)
REM *** Rotate Cube ***
DO
   PITCH OBJECT DOWN 1,1.0
   TURN OBJECT LEFT 1, 1.0
   REM *** 1 in 1000 of going green ***
   IF RND(1000) = 500
      COLOR OBJECT 1, RGB(0,255,0)
   ENDIF
LOOP
REM *** End program ***
END
```

Activity 32.22

No solution required.

Activity 32.23

No solution required.

Activity 32.24

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load texture images ***
LOAD IMAGE "textureWood.jpg",1
LOAD IMAGE "DoNot.bmp",2
REM *** Make and position cube ***
MAKE OBJECT CUBE 1, 40
POSITION OBJECT 1,25,0,100
REM *** Texture cube ***
TEXTURE OBJECT 1,1
REM *** Create background plain ***
MAKE OBJECT PLAIN 2,100,100
TEXTURE OBJECT 2, 2
POSITION OBJECT 2,0,0,200
REM *** Start reflective value at 200 ***
reflectivity = 200
REM *** Make cube transparent ***
GHOST OBJECT ON 1,0
REM *** rotate cube ***
DO
   PITCH OBJECT DOWN 1, 1.0
   TURN OBJECT LEFT 1, 1.0
   FADE OBJECT 1,reflectivity
   REM *** Reduce reflectivity to zero ***
   IF reflectivity > 0
      DEC reflectivity
   ENDIF
   WAIT 10
LOOP
REM *** End program ***
END
```

The cube fades until it is completely invisible.

Activity 32.25

No solution required.

Activity 32.26

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280, 1024,32
REM *** Create cube ***
MAKE OBJECT CUBE 1,10
REM *** Texture cube ***
LOAD IMAGE "windmillshaped.tga",1
TEXTURE OBJECT 1,1
REM *** Rotate cube ***
DO
  REM *** IF key pressed, ghost ***
   IF INKEY$() <> ""
      GHOST OBJECT ON 1,3
   ENDIF
   TURN OBJECT LEFT 1,1
LOOP
REM *** End program ***
END
```

Activity 32.27

No solution required.

Activity 32.28

No solution required.

Activity 32.29

```
REM *** Set display resolution ***
SET DISPLAY MODE 1280,1024,32
DrawGallows()
POINT CAMERA -150,10,0
WAIT KEY
END



FUNCTION DrawGallows()
   REM *** Set up names ***
   REM *** Object names ***
   #CONSTANT   GroundObj         901
   #CONSTANT   PlatformObj       902
   #CONSTANT   VerticalPostObj   903
   #CONSTANT   HorizontalPostObj 904
   #CONSTANT   DiagonalPostObj   905
   #CONSTANT   TopStepObj        906
   #CONSTANT   MiddleStepObj     907
   #CONSTANT   BottomStepObj     908
   #CONSTANT   StepEdgeRightObj  909
   #CONSTANT   StepEdgeLeftObj   910
   REM *** Image names ***
   #CONSTANT   CobbleImg         901
   #CONSTANT   PlanksImg         902
   #CONSTANT   WoodImg           903
   REM *** Load texture images ***
   LOAD IMAGE "TextureWood.jpg",PlanksImg
   LOAD IMAGE "CobbleStones.jpg",CobbleImg
   LOAD IMAGE "Wood.jpg",WoodImg
   REM *** Create cobbled square ***
   MAKE OBJECT PLAIN GroundObj,300,300
```

```
    TEXTURE OBJECT GroundObj,CobbleImg
    SCALE OBJECT TEXTURE GroundObj,30,30
    XROTATE OBJECT GroundObj,-90
    REM *** Create platform ***
    MAKE OBJECT BOX PlatformObj,50,15,50
    TEXTURE OBJECT PlatformObj,PlanksImg
    SCALE OBJECT TEXTURE PlatformObj,2,2
    POSITION OBJECT PlatformObj,-125,7.5,0
    REM *** Create vertical post ***
    MAKE OBJECT BOX VerticalPostObj,2,30,2
    TEXTURE OBJECT VerticalPostObj,WoodImg
    POSITION OBJECT VerticalPostObj,
    ↳-147.5,30,0
    REM *** Create horizontal post ***
    MAKE OBJECT BOX HorizontalPostObj,
    ↳2,15,2
    TEXTURE OBJECT HorizontalPostObj,
    ↳WoodImg
    ZROTATE OBJECT HorizontalPostObj,90
    POSITION OBJECT HorizontalPostObj,
    ↳-139,44,0
    REM *** Create diagonal post ***
    MAKE OBJECT BOX DiagonalPostObj,1,10,1
    TEXTURE OBJECT  DiagonalPostObj,WoodImg
    ZROTATE OBJECT  DiagonalPostObj,-45
    POSITION OBJECT DiagonalPostObj,
    ↳-144,40,0
    REM *** Make top step ***
    MAKE OBJECT BOX TopStepObj,10,0.3,3
    TEXTURE OBJECT  TopStepObj,WoodImg
    POSITION OBJECT TopStepObj,-130,12,-26
    REM *** Make middle step ***
    CLONE OBJECT  MiddleStepObj,TopStepObj
    POSITION OBJECT MiddleStepObj,-130,8,-29
    REM *** Make bottom step ***
    CLONE OBJECT BottomStepObj,TopStepObj
    POSITION OBJECT BottomStepObj,-130,4,-32
    REM *** Make right step edge***
    MAKE OBJECT BOX StepEdgeRightObj,0.3,4,20
    TEXTURE OBJECT StepEdgeRightObj,WoodImg
    XROTATE OBJECT StepEdgeRightObj,-50
    POSITION OBJECT StepEdgeRightObj,
    ↳-125,6,-30
    REM *** Make left step edge ***
    CLONE OBJECT StepEdgeLeftObj,
    ↳StepEdgeRightObj
    POSITION OBJECT StepEdgeLeftObj,
    ↳-135,6,-30
ENDFUNCTION
```

Activity 32.30

The changes to the cobbled square in the *DrawGallows()* function are shown in bold below:

```
REM *** Create cobbled square ***
MAKE OBJECT PLAIN GroundObj,2000,2000
TEXTURE OBJECT GroundObj,CobbleImg
SCALE OBJECT TEXTURE GroundObj,150,150
```

Activity 32.31

The sky should now be visible.

Activity 32.32

The sphere's texture is upside down and mirrored.

Activity 32.33

No solution required.