BRENDER

x86 Installation Guide

DOS/Windows Platforms

Author: Robbie McQuaid
Technical Direction: Crosbie Fitch, Sam Littlewood, Dan Piponi, Philip Pratt, Simon Everett, Vinay Gupta, John Gay, Neela Dass, Tony Roberts
Project Managers: Paul Ayscough, Stefano Zammattio

Argonaut Technologies Limited
Capitol House, Capitol Way
Colindale, London NW9 0DZ
United Kingdom

Contents

Welcome To BRender

Thank you for purchasing BRender, Argonaut Technologies' real-time 3D Renderer. BRender's functions and data structures are designed to facilitate rapid and intuitive 3D development on the following platforms:

- MS-DOS/Windows (3.1/3.11/NT/95)
- Macintosh/PowerPC
- Sony PlayStation(PSX)
- Sega Saturn
- Silicon Graphics

BRender for MS-DOS/WINDOWS

This guide is your introduction to BRender x86. It describes how to install BRender on an MS-DOS/Windows platform, and how to configure your C compiler to work with BRender libraries. BRender supports the following C compilers:

- Watcom C V10 and V10.5
- Borland C V4.5 with PowerPack or PharLap TNT
- MS Visual C V2.2 and V4.0 with PharLap TNT

Your BRender Pack

In addition to this *Installation Guide*, your BRender pack contains the following items:

- Installation Disks
- Tutorial Programs
- Kasan 3DMAX Support Programs
- Licence Agreement
- Technical Reference Manual
- Tutorial Guide
- BRender User Registration Form
- Last Minute Additions Sheets

Please contact Argonaut Technologies' Sales Department on +44 (0)181 358 2993 if any items are missing.

BRender Technical Support

The BRender Technical Support Team will endeavour to answer all technical queries within 24 hours. Telephone lines are open Monday to Friday, from 10.00hrs to 20.00hrs UK Standard Time.

EMAIL brender@argonaut.com

FAX	0181 358 2980	(From UK)
	(+44)181 358 2980	(Other Locations)

PHONE	+44 (0)181 358 2992
	+44 (0)181 358 2946
	+44 (0)181 358 2924

Installation 1

1.1 Hardware Requirements

BRender x86 requires the following minimum hardware configuration :

- IBM PC compatible
- An 80386 or higher processor
- A hard drive with 15MB of free disk space

In addition to the above hardware requirements, you will need to be running either MS-DOS V3.0 or later or Windows 3.1/3.11/NT/95.

1.2 Installing BRender

From DOS :

1. Insert Disk 1 in drive A and type :

   a:install

2. Follow the instructions on the screen

From Windows :

1. Insert Disk 1 in drive A
2. Choose Run from the File menu in Program Manager
3. Enter the following command line

   a:install

4. Follow the instructions on the screen

WATCOM C

Arithmetic/Calling Convention	BRender Libraries
Fixed Point/Stack Calling	brfwwxrs.lib, brzbwxrs.lib, brstwxrs.lib, brfmwxrs.lib
Fixed Point/Register Calling	brfwwxrr.lib, brzbwxrr.lib, brstwxrr.lib, brfmwxrr.lib
Floating Point/Stack Calling	brfwwfrs.lib, brzbwfrs.lib, brstwfrs.lib, brfmwfrs.lib
Floating Point/Register Calling	brfwwfrr.lib, brzbwfrr.lib, brstwfrr.lib, brfmwfrr.lib
DOS Extender/Calling Convention	**DOS I/O LIBRARY**
DOS4GW/Stack Calling	dio4gwrs.lib
DOS4GW/Register Calling	dio4gwrr.lib
PHARLAP/Stack Calling	dioplwrs.lib
PHARLAP/Register Calling	dioplwrr.lib
X-32VM/Stack Calling	diox3wrs.lib
X-32VM/Register Calling	diox3wrr.lib

BORLAND C

Arithmetic	BRender Libraries
Fixed Point	brfwbxr.lib, brzbbxr.lib, brstbxr.lib, brfmbxr.lib
Floating Point	brfwbfr.lib, brzbbfr.lib, brstbfr.lib, brfmbfr.lib
DOS Extender	**DOS I/O LIBRARY**
POWERPACK	dioppbr.lib
PHARLAP	dioplmr.lib

MICROSOFT C

Arithmetic	BRender Libraries
Fixed Point	brfwmxr.lib, brzbmxr.lib, brstmxr.lib, brfmmxr.lib
Floating Point	brfwmfr.lib, brzbmfr.lib, brstmfr.lib, brfmmfr.lib
DOS Extender	**DOS I/O LIBRARY**
PHARLAP	dioplmr.lib

Table 1 BRender and Dos I/O Libraries

2 Configuring Your

To set up your compiler to work with BRender, the following steps need to be performed.

- Install/select the appropriate target platform
- Include relevant BRender libraries in your project
- Tell the compiler where to find BRender header files
- Set data alignment to 4-byte/double word

2.1 Compiling DOS Programs

The following sections guide you through compiling and running the sample BRender program `minimal.c`. For Watcom C and Borland C, it describes how to compile applications from within the IDE, as well as from the command line. For Visual C, only the command line option is available for DOS programs. The program `minimal.c` is included on the Tutorial Programs disk shipped with your BRender pack. Before proceeding, it is recommended that you create a directory, `C:\BRENDER\TUTORIAL`, and copy all files from the Tutorial Programs disk to it.

Watcom C V10/10.5

It is assumed that you have installed BRender and copied the contents of the Tutorial Programs disk to the directory `C:\BRENDER\TUTORIAL`.

From Within the IDE

1. Start the IDE and create a new project (pull down the *File* menu and select *New Project*) called tutorial(.wpj), in the directory `C:\BRENDER\TUTORIAL`.

2. Select DOS - 32-bit as the target environment. The image type is DOS4GW Executable[.exe].

3. Add the following source files (pull down the *Sources* menu and select *New Source*) to the project:

minimal.c
brfwwxrr.lib, brzbwxrr.lib, brstwxrr.lib, brfmwxrr.lib
dio4gwrr.lib

`minimal.c`, the file containing the C source code, can be found in the `C:\BRENDER\TUTORIAL` directory.

The `.lib` files can be found in `C:\BRENDER\LIB`. Note that the Fixed Point, Register Calling BRender libraries have been selected in this example (see Table 1). Select one of the other supported options if you would prefer.

You also need to add the appropriate DOS I/O library. For Watcom, Register Calling (DOS4GW) this is `dio4gwrr.lib`.

4. Select 4-byte alignment:

Options
C Compiler Switches
File Option Switches
Source Switches
[4 byte alignment]

(pull down the *Options* menu and select *C Compiler Switches,* pull down *the File Option Switches* sub-menu and select *Source Switches*, select *4 byte alignment*).

5. To tell Watcom where to find BRender Header Files, add `c:\brender\inc` and `c:\brender\dosio` to the *Include directories:* option under *File Option Switches.*

Options
C Compiler Switches
File Option Switches
Include directories:
[.....; c:\brender\inc; c:\brender\dosio]

6. Make sure the target processor matches your machine and the calling convention (register or stack) is consistent with the selected BRender libraries.

Options
C Compiler Switches
File Option Switches
Memory Model and Processor Switches
Target Processor
(e.g. 80486 Register based calling)

7. Compile and Run the sample program. It displays a revolving grey cube.

From the Command Line

Make `C:\BRENDER\TUTORIAL` the current directory, then enter the following command from the DOS prompt:

```
wmake -f makefile.wat
```

The makefile '`makefile.wat`', which can be found in `C:\BRENDER\TUTORIAL`, specifies relevant libraries and header files and selects appropriate compiler options.

```
# Copyright (c) 1993 Argonaut Software Ltd. All rights reserved.
#
#
# Makefile for Watcom C using WMAKE
#
TARGET=minimal.exe

BASE_DIR=c:\brender

# Watcom, Fixed, Release, Register
#
RENDERER=zb
LIB_TYPE=wxrr
BASED=FIXED

CC=wcc386
CFLAGS=-omaxnet -zp4 -5r -fp3&
 -DBASED_$(BASED)=1&
 -DRENDER_$(RENDERER)=1&
 -I$(BASE_DIR)\inc&
 -I$(BASE_DIR)\dosio

OBJS=&
     minimal.obj&

LIBS=&
     $(BASE_DIR)/lib/dio4gwrr.lib&
     $(BASE_DIR)/lib/brst$(LIB_TYPE).lib&
     $(BASE_DIR)/lib/brfw$(LIB_TYPE).lib&
     $(BASE_DIR)/lib/brfm$(LIB_TYPE).lib&
     $(BASE_DIR)/lib/br$(RENDERER)$(LIB_TYPE).lib&

# Default rules
#
# Cope with long command lines
#
.c.obj:
     set _ARGS=$(CFLAGS) $*.c
     $(CC) @_ARGS

$(TARGET): $(OBJS) wlink.rsp
     wlink @wlink.rsp

# Link response file
#
wlink.rsp: makefile.wat
     echo name $(TARGET) >wlink.rsp
     echo system dos4g >>wlink.rsp
     echo option dosseg,caseexact,quiet,stack=64k >>wlink.rsp
     for %i in ($(OBJS)) do echo file %i >>wlink.rsp
     for %i in ($(LIBS)) do echo library %i >>wlink.rsp
```

Set 4-byte alignment

Header Files

Include BRender libraries

32-bit DOS extender

makefile.wat

Borland C V4.5

Note that a 32-bit DOS extender must be installed on your system for BRender to work with Borland C. BRender supports Borland PowerPack and PharLap TNT.

It is assumed that you have installed BRender and copied the contents of the Tutorial Programs disk to the directory `C:\BRENDER\TUTORIAL`. For the purposes of this tutorial, it is also assumed that you have installed Borland PowerPack 32-bit DOS extender.

From Within the IDE

1. Create a new project called tutorial(.ide), in the directory `C:\BRENDER\TUTORIAL`.

Project
New Project
Project Path and Name:
c:\brender\tutorial\tutorial.ide

2. In the same window, select DOS 32-bit as the target. Accept the default target type (Application) and other default settings.

Platform:
DOS (32-bit DPMI)

3. Delete default node tutorial [.cpp] if present (highlight node, then press [Delete]). Add the following source files (press [Insert] then select files):

`minimal.c`
`brfwbxr.lib, brzbbxr.lib, brstbxr.lib, brfmbxr.lib`
`dioppbr.lib`

`minimal.c`, the file containing the C source code, can be found in the `C:\BRENDER\TUTORIAL` directory.

The `.lib` files can be found in `C:\BRENDER\LIB`. Note that the Fixed Point BRender libraries have been selected (see Table 1). Fixed point arithmetic is faster, though less accurate, than floating point arithmetic.

You also need to add the appropriate DOS I/O library. For Borland C using POWERPACK, this is `dioppbr.lib`.

4. Select 4-byte alignment:

Options
Project
32-bit Compiler
Processor
Data alignment
Double word (4-byte)

5. To tell Borland where to find BRender Header Files, add c:\brender\inc and c:\brender\dosio to the *Include Source Directories* option under *Directories*.

Options
Project
Directories
Source Directories
Include: **.....;** c:\brender\inc; c:\brender\dosio

6. Compile and Run the sample program. It displays a revolving grey cube.

:

From the Command Line

Make `C:\BRENDER\TUTORIAL` the current directory, then enter the following command from the DOS prompt:

```
make -f makefile.bcc
```

The makefile '`makefile.bcc`', which can be found in `C:\BRENDER\TUTORIAL`, specifies relevant libraries and header files and selects appropriate compiler options.

```
# Copyright (c) 1993 Argonaut Software Ltd. All rights reserved.
#
# Makefile for Borland C 4.5 + Powerpack
#
TARGET=minimal.exe

BASE_DIR=C:\BRENDER

# Borland, Fixed, Release
#
RENDERER=zb
LIB_TYPE=bxr
BASED=FIXED

CC=bcc32
CFLAGS=-c -a4  -DBASED_$(BASED)=1 -DRENDER_$(RENDERER)=1\
-I$(BASE_DIR)\inc\
-I$(BASE_DIR)\dosio\

LD=tlink32
LDFLAGS=-Tpe -ax -v

OBJS=\
     minimal.obj\

LIBS=\
     $(BASE_DIR)\lib\dioppbr.lib\
     $(BASE_DIR)\lib\brst$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\brfw$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\brfm$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\br$(RENDERER)$(LIB_TYPE).lib\

# Default rules
#
# Cope with long command lines
#
.c.obj:
     $(CC) @&&!
$(CFLAGS)
$<
!

$(TARGET): $(OBJS)
     $(LD) $(LDFLAGS) @&&!
c0x32.obj $(OBJS)
$(TARGET)
$*
dpmi32.lib cw32.lib $(LIBS)
!
```

Set 4-byte Alignment

Header Files

Include BRender Libraries

32-bit DOS Extender

makefile.bcc

MS VISUAL C V2.2/4.0

Note that PharLap TNT 32-bit DOS extender must be installed on your system for BRender to work with Visual C V2.0.

It is assumed that you have installed BRender and copied the contents of the Tutorial Programs disk to the directory `C:\BRENDER\TUTORIAL`.

From the Command Line

Make `C:\BRENDER\TUTORIAL` the current directory, then enter the following command from the DOS prompt:

```
nmake -f makefile.msc
```

The makefile '`makefile.msc`', which can be found in `C:\BRENDER\TUTORIAL`, specifies relevant libraries and header files and selects appropriate compiler options.

```
# Copyright (c) 1993 Argonaut Software Ltd. All rights reserved.
#
#
# NMAKE makefile with Microsoft Visual C 2.0 + Pharlap TNT
#
TARGET=minimal.exe

BASE_DIR=C:\brender

# Microsoft, Fixed, Release
#
RENDERER=zb
LIB_TYPE=mxr
BASED=FIXED

CC=cl
CFLAGS=-c -Zp4 -Gz -DBASED_$(BASED)=1 -DRENDER_$(RENDERER)=1 -
D__PHARLAP386__=1\
-I$(BASE_DIR)\inc\
-I$(BASE_DIR)\dosio\

LD=386link
LDFLAGS=@msvc32.dos -stack 65536

OBJS=\
     minimal.obj\

LIBS=\
     $(BASE_DIR)\lib\dioplmr.lib\
     $(BASE_DIR)\lib\brst$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\brfw$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\brfm$(LIB_TYPE).lib\
     $(BASE_DIR)\lib\br$(RENDERER)$(LIB_TYPE).lib\

# Default rules
#
# Cope with long command lines
#
.c.obj:
     $(CC) @<<cl.rsp
$(CFLAGS)
$<
<<

$(TARGET): $(OBJS)
     $(LD) $(LDFLAGS) -exe $(TARGET) @<<link.rsp
$(OBJS)
-lib $(LIBS)
<<
```

Set 4-byte alignment

32-bit DOS extender

Header Files

Include BRender Libraries

makefile.msc

2.2 Compiling Windows Programs

The following sections guide you through compiling and running the sample BRender program `simpview` from within the Watcom, Borland and Visual C IDE's. It is assumed that you have installed BRender in the default directory. Simpview files can be found in the directory `C:\BRENDER\SAMPLES\WIN\SIMPVIEW`.

Watcom C V10/10.5

1. Start the IDE and create a new project (pull down the *File* menu and select *New Project*) called simpview(.wpj), in the directory `C:\BRENDER\SAMPLES\WIN\SIMPVIEW`.

2. Select Win 32 as the target environment. The image type is Windowed Executable [.exe].

3. Add the following source files (pull down the *Sources* menu and select *New Source*) to the project:

`app.c, buffer.c, dispatch.c, main.c, mattub.c, world.c, app.rc`
`brfwwxrr.lib, brzbwxrr.lib, brstwxrr.lib, brfmwxrr.lib`

The `.c` and `.rc` files can be found in the `\SIMPVIEW` directory.

The `.lib` files can be found in `C:\BRENDER\LIB`. Note that the Fixed Point, Register Calling BRender libraries have been selected (see Table 1). Select one of the other supported options if you would prefer.

4. Select 4-byte alignment:

Options
C Compiler Switches
File Option Switches
Source Switches
4 byte alignment

(pull down the *Options* menu and select *C Compiler Switches*, pull down *the File Option Switches* sub-menu and select *Source Switches*, select *4 byte alignment*).

5. To tell Watcom where to find BRender Header Files, add `c:\brender\inc` to the *Include directories: option* under *File Option Switches*.

Options
C Compiler Switches
File Option Switches
Include directories:
.....; c:\brender\inc

6. Make sure the target processor matches your machine and the calling convention (register or stack) is consistent with the selected BRender libraries.

Options
C Compiler Switches
File Option Switches
Memory Model and Processor Switches
Target Processor
(e.g. 80486 Register based calling)

7. Compile and Run the sample program.

Borland C V4.5

1. Create a new project called simpview(.ide), in the directory `C:\BRENDER\SAMPLES\WIN\SIMPVIEW`.

Project
New Project
Project Path and Name
c:\brender\samples\win\simpview\simpview.ide

2. In the same window, select WIN32 as the target. Accept the default target type (Application) and other default settings.

Platform
WIN32

3. Delete default nodes **simpview.cpp**, **simpview.def** and **simpview.rc** if present (highlight nodes, then press [Delete]). Add the following source files (press [Insert] then select files):

app.c, buffer.c, dispatch.c, main.c, mattub.c, world.c, app.rc
brfwbxr.lib, brzbbxr.lib, brstbxr.lib, brfmbxr.lib

The *.c* and .rc files can be found in the \SIMPVIEW directory.

The .lib files can be found in C:\BRENDER\LIB. Note that the Fixed Point BRender libraries have been selected (see Table 1). Fixed point arithmetic is faster, though less accurate, than floating point arithmetic.

4. Select 4-byte alignment:

Options
Project
32-bit Compiler
Processor
Data alignment
Double word (4-byte)

5. To tell Borland where to find BRender Header Files, add c:\brender\inc to the *Include Source Directories* option under *Directories*.

Options
Project
Directories
Source Directories
Include: .....; c:\brender\inc

6. Compile and Run the sample program.

VISUAL C V2.2/4.0

Note that you must be running Windows NT or Windows 95 to compile BRender programs using VISUAL C.

1. Start the IDE and create a new project called simpview(.mak), in the directory `C:\BRENDER\SAMPLES\WIN\SIMPVIEW`.

File
New
c:\brender\samples\win\simpview.mak

2. Add the following source files:

`app.c, buffer.c, dispatch.c, main.c, mattub.c, world.c, app.rc, mlibcmnt.c`

The above files can be found in the `\SIMPVIEW` directory.

3. To tell VISUAL C where to find BRender Header and Library Files, add `c:\brender\inc` and `c:\brender\lib` to the *Include Files* and *Library Files* options, respectively:

Tools
Options
Directories
Include Files
c:\brender\inc

Library Files
c:\brender\lib

4. Select Win32 Fixed (Release) as the Target:

Target
Win32 Fixed (Release)

5. Compile and Run the sample program.

3 Platform Issues

3.1 Video Modes

BRender x86 Version 1.1.2 supports the following video modes:

- 8-bit indexed colour
- 15- and 24-bit true colour

3.2 BRender Data Types

BRender data types are defined in the header file `inc/brender.h`

3.3 Texture Mapping in True Colour

Note that:

- Lighting calculations are not performed for texture mapped models rendered in 15- or 24-bit true colour.
- Smooth shading is not supported for texture mapped models in 15- or 24-bit true colour and should be turned off.

4 Platform-Specific Functions

A number of functions are provided to simplify hardware initialisation and I/O operations. Note that the supplied functions are intended as generic examples of how these operations may be implemented. You may wish to optimize this code for your application, or to substitute your own functions.

4.1 Graphics Functions

DOSGfx is a small set of screen handling functions intended for DOS based BRender applications. The functions simplify initialisation of colour buffers, palettes and video display hardware.

The `BRENDER_DOS_GFX` environment variable can be used to set the default graphics mode and resolution. It has the following format:

```
VESA|MCGA,[W:<width>],[H:<height>],[B:<bits/pixel>],
[M:<mode number>]
```

If the environment variable specifies an unavailable mode or resolution, `DOSGfx` will fail to initialise and return an error message.

DOSGfxBegin()

Description: Create a pixelmap which represents the graphics hardware screen. Subsequently, `BrPixelmapMatch()` can be used to create a second screen, and `BrPixelmapDoubleBuffer()` can be used to swap between them.

Declaration: **`br_pixelmap* DOSGfxBegin(char* setup_string);`**

Arguments: **`char * setup_string`**

An options string, given in the following format:

```
VESA|MCGA,[W:<width>],[H:<height>],[B:<bits/pixel>],
[M:<modenumber>]
```

The default string `'MCGA,W:320,H:200,B:8'` is used if NULL is passed and the `BRENDER_DOS_GFX` environment variable has not been set. If the environment variable has been set, any options given here that differ will be used instead.

Result: **`br_pixelmap *`**

Returns a pointer to a pixelmap representing the graphics hardware screen.

DOSGfxEnd()

Description: Close down DOSGfx.

Declaration: **`void DOSGfxEnd(void);`**

DOSGfxPaletteSet()

Description: Copy the contents of a given pixelmap to the VGA/VESA hardware palette.

Declaration: **`void DOSGfxPaletteSet(br_pixelmap* pm);`**

Arguments: **`br_pixelmap * pm`**

A pointer to a 1x256 `BR_PMT_RGBX_888` pixelmap containing the palette.

DOSGfxPaletteSetEntry()

Description: Set a single colour in the `VGA/VESA` hardware palette.

Declaration: **`void DOSGfxPaletteSetEntry(int I, br_colour colour);`**

Arguments: **`int I`**

Colour number.

`br_colour colour`

New colour.

4.2 - Mouse Handling Functions

DOSMouseBegin()

Description: Initialise mouse hardware. Must be called before `DOSMouseRead()`.

Declaration: **`void DOSMouseBegin(void);`**

DOSMouseRead()

Description: Determine movement in the x and y directions, and the status of the mouse buttons (pressed or not pressed). Adds movement in x and y directions since last `DOSMousRead()` or `DOSMouseBegin()` to respective mouse parameters.

Declaration: **`void DOSMouseRead(br_int_32 *mouse_x,br_int_32 *mouse_y, br_uint_32 *mouse_buttons);`**

Arguments: **`br_int_32 *mouse_x`**

A pointer to a value used to track mouse movement in the x direction.

`br_int_32 *mouse_y`

A pointer to a value used to track mouse movement in the y direction.

`br_uint_32 *mouse_buttons`

A pointer to a variable used to monitor the status of the mouse buttons. Compare `mouse_buttons` with `BR_MSM_BUTTONSL`, `BR_MSM_BUTTONSR` and `BR_MSM_BUTTONSM` to determine mouse button status.

DOSMouseEnd()

Description: End mouse support.

Declaration: **`void DOSMouseEnd(void);`**

4.3 Reading the System Clock

DOSClockBegin()

Description: Begin support for DOSClock. Must be called before `DOSClockRead()`.

Declaration: **`void DOSClockBegin(void);`**

DOSClockRead()

Description: Read the system clock.

Declaration: **`br_uint_32 DOSClockRead(void);`**

Result: **`br_uint_32`**

Returns number of DOSClockRead ticks (as defined in `dosio.h`). Could be used to determine the frame rate :

```
start_time=DOSClockRead();
    |     |
    |     |
end_time=DOSClockRead();
frame_rate=BR_DOS_CLOCK_RATE/(end_time-start_time);
```

DOSClockEnd()

Description: End clock support.

Declaration: **`void DOSClockEnd(void);`**

4.4 Keyboard Handling

DOSKeyBegin()

Description: Initialise keyboard handling. Standard C keyboard handling functions will be disabled unless `DOSKeyEnableBIOS()` is called subsequently.

Declaration: **`void DOSKeyBegin(void);`**

DOSKeyTest()

Description: Scan the keyboard for a key selection.

Declaration: **`br_uint_8 DOSKeyTest(br_uint_8 scancode,br_uint_8 qualifiers, br_uint_8 repeats);`**

Arguments: **`br_uint_8 scancode`**

Keyboard scancode as detailed in `keyboard.h` (eg. SC_A, SC_B etc.).

`br_uint_8 qualifiers`

Shift qualifier, or qualifier combination, to select required SHIFT/CTRL/ALT key combination. See definitions below.

`br_uint_8 repeats`

Specifies auto repeat key setting.

Result: **`br_uint_8`**

Returns TRUE if specified key, or key combination, has been pressed.

Remarks: The following shift qualifiers are defined in `keyboard.h`:

QUAL_SHIFT 0x01
QUAL_CTRL 0x02
QUAL_SHIFT 0x01
QUAL_CTRL 0x02
QUAL_SHIFT 0x01
QUAL_CTRL 0x02

If 0 is specified, key will be detected whatever the status of the qualifier keys.

The following repeat control qualifiers are defined in `keyboard.h`:

REPT_FIRST_DOWN 0x10
REPT_AUTO_REPT 0x20

If 0 is specified, always returns TRUE while key is pressed.

DOSKeyEnableBIOS()

Description: Once `DOSKeyBegin()` has been called, standard C keyboard handling functions will be disabled unless this function is called.

Declaration: **`void DOSKeyEnableBIOS(br_uint_16 flag);`**

Arguments: **`br_uint_16 flag`**

A non-zero value enables standard C keyboard handling functions. Zero disables them.

DOSKeyEnd()

Description: End keyboard handling support.

Declaration: **`void DOSKeyEnd(void);`**

4.5 Divide Overflow Suppressor

DOSDivTrapBegin()

Description: Begin divide-by-zero and divide overflow exception handling.

Declaration: `int DOSDivTrapBegin(void);`

Result: `int`

Returns non-zero if handler installation fails.

DOSDivTrapCount()

Description: Returns the number of divide-by-zero or divide overflow occurrences.

Declaration: `int DOSDivTrapCount(int reset);`

Arguments: `int reset`

Any non-zero value will reset count.

Result: `int`

Returns the number of divide-by-zero or divide overflow occurrences.

DOSDivTrapEnd()

Description: End divide-by-zero and divide overflow exception handling.

Declaration: `void DOSDivTrapEnd(void);`

4.6 Event Queue for Mouse/Keyboard

DOSEventBegin()

Description: Initialise event queue handling for mouse and keyboard events. Events are queued until a `DOSEventWait()` command is encountered.

Declaration: **`void DOSEventBegin(void);`**

Remarks: Must be followed by `DOSMouseBegin` or `DOSKeyBegin`.

DOSEventWait()

Description: Monitors mouse and keyboard activity. Stores information describing mouse and keyboard activity in an event queue.

`br_uint_16 DOSEventWait(struct dosio_event *event, int block);`

Arguments: **`struct dosio_event * event`**

Structure for storing event information. Defined in `eventq.h`.

`int block`

If block is 0, return. If block is 1, wait until a mouse or keyboard event has occurred (always return TRUE).

Result: **`br_uint_32`**

Returns TRUE if an event has occurred, FALSE otherwise.

DOSEventEnd()

Description: End event queue handling.

Declaration: **`void DOSEventEnd(void);`**

Kasan 3DMAX 5 Support

A number of functions are provided to allow you to use Kasan 3DMAX glasses with BRender. These functions, documented below, can be found on the KASAN disk included with your BRender pack.

This disk contains 2 directories:

- API contains core calls used to drive the 3DMAX glasses from within BRender.
- DEMO contains a simple program that demonstrates the use of the API. Flat and smooth shading options are provided.

Batch files are supplied that pass sensible parameters to the demonstration program, `stereo.exe`. Run the relevant batch file, smooth.bat or `flat.bat`, to view the demo. Use the mouse to move around inside the demo. Use the left mouse button to zoom in, the right mouse button to zoom out, both mouse buttons together to quit the program.

Three functions, documented below, are provided to allow the 3DMAX glasses to be used with BRender programs. Function prototypes can be found in `stereo.h`, function definitions in `stereo.c`. `kasan.c` contains a low level wrapper that provides access to the KASAN drivers. The files `kasan.c` and `stereo.c` must be added to your project. Note that `kasan.c` contains calls specific to Watcom C and cannot be used with any other compiler. Complete program listing for `kasan.c`, `stereo.c` and `main.c` (the files included in the demo project) can be found at the back of the Programs Listings section of this guide.

Only a few lines of code need to be changed to convert a BRender program to support the 3DMAX glasses. A converted version of `BRTUTOR1.C` is listed on pages 64 and 65 of this guide. Required changes and additions are highlighted in bold.

Note that:

- Only Kasan 3DMax glasses are supported.
- There's no direct support for Kasan's video modes with high vertical resolution.
- There's no stereo equivalent to `BrSceneRenderBegin()`, `BrSceneRenderAdd()`, or `BrSceneRenderEnd`.
- Z-Sort rendering is not supported.
- WATCOM C is the only compiler currently supported.
- Windows programming is not supported.

KasanGfxBegin()

Description: Similar to `DOSGfxBegin()`. Sets up the stereo glasses and the screen. Each eye will see an image with a vertical resolution half that specified.

Declaration: **`br_pixelmap* KasanGfxBegin(char* pNew_setup_string);`**

Arguments: **`char * pNew_setup_string`**

An options string, given in the following format:

`VESA|MCGA,[W:<width>],[H:<height>],[B:<bits/pixel>],[M:<mode number>]`

Result: **`br_pixelmap *`**

Returns a pointer to a pixelmap representing the graphics hardware screen. Returns NULL if the glasses can't be initialised.

KasanGfxEnd()

Description: Closes down `StereoDOSGfx`.

Declaration: **`void KasanGfxEnd(void);`**

KasanZbSceneRender()

Description: Similar to BrZbSceneRender but renders the scene twice, from slightly different cameras, to produce the images to be seen from the left and right eyes. The first four arguments are the same.

`pInterocular_distance` is the distance between the 'eyes' in the camera's coordinate space. If the camera is scaled relative to the world then the interocular distance will be too. The left camera is half the interocular distance from `*pCamera`. The right camera is an equal distance the other way. The 'correct' value for the interocular distance depends on the units used in defining the scene.

`pCamera_yaw` is the angle between each of the left and right cameras and the Z-axis of `*pCamera`. The 'correct' value is `camera_yaw = BR_ATAN2(interocular_distance, 2 * monitor_distance)`. A camera yaw of 0 corresponds to an infinitely large monitor infinitely distant. So no matter how far away an object is it will appear to be in front of the monitor. Positive values of camera yaw allow distant objects to appear as though they are behind the screen. Values which are too large tend to result in images which are hard for users to 'fuse'. A value of 1 degree gives images which are fairly easy to view.

Declaration:

```
void KasanZbSceneRender(br_actor* pWorld, br_actor*
pCamera, br_pixelmap* pCol_buffer, br_pixelmap*
pDepth_buffer, br_scalar pInterocular_distance, br_angle
pCamera_yaw);
```

Arguments:

`br_actor * pWorld`

A non-NULL pointer to the root actor of a scene.

`br_actor * pCamera`

A non-NULL pointer to a camera actor that is a descendant of the root actor.

`br_pixelmap * pCol_buffer`

A non-NULL pointer to the pixelmap to render the scene into, whose type is `colour_type` as supplied to **`BrZsBegin()`**.

`br_pixelmap * pDepth_buffer`

A non-NULL pointer to the pixelmap to be used as a depth buffer whose type is `depth_type` as supplied to **`BrZsBegin()`**. It must have the same width and height as the colour buffer. See **`BrPixelmapMatch()`**.

`br_scalar pInterocular_distance`

The distance between the 'eyes' in the camera's coordinate space.

`br_angle pCamera_yaw`

The angle between each of the left and right cameras and the Z-axis of `*pCamera`.

Preconditions: Between **`BrBegin()`** & **`BrEnd()`**.Between **`BrZbBegin()`** & **`BrZbEnd()`**. Not currently rendering.

Appendix: Program Listings

BRTUTOR1.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Illuminated Cube.
 */
#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"
int main(int argc, char **argv)
{
   /*
    * Need screen and back buffers for double-buffering, a Z-Buffer to store
    * depth information, and a storage buffer for the currently loaded palette
    */
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   /*
    * The actors in the world: Need a root actor, a camera actor, a light actor,
    * and a model actor
    */
   br_actor *world, *observer, *light, *cube;
   int i;                                        /*counter*/

/************* Initialise BRender and Graphics Hardware *********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSetDOSGfxPaletteSet(palette);
   /*
    * Initialise z-buffer renderer
    */
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   /*
    * Allocate Back Buffer and Depth Buffer
    */
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Start with None actor at root of actor tree and call it 'world'
    */
   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   /*
    * Add a camera actor as a child of None actor 'world'
    */
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   /*
```

```
    * Add and enable the default light source
    */
   light = BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL));
   BrLightEnable(light);
   /*
    * Move camera 5 units along +z axis so model becomes visible
    */
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Add a model actor, the default cube
    */
   cube = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   /*
    * Rotate cube to enhance visibility
    */
   cube->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateY(&cube->t.t.mat,BR_ANGLE_DEG(30));

 /********************** Animation Loop *********************************/
   /*
    * Rotate cube around x-axis
    */
   for(i=0; i < 360; i++)  {
               /*
                * Initialise depth buffer and set background colour to black
                */
               BrPixelmapFill(back_buffer,0);
               BrPixelmapFill(depth_buffer,0xFFFFFFFF);
               /*
                * Render scene
                */
               BrZbSceneRender(world,observer,back_buffer,depth_buffer);
               BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
               /*
                * Rotate cube
                */
               BrMatrix34PostRotateX(&cube->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR1.C

BRTUTOR2.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to display a scene containing a Box, a Sphere and a Torus
 */
#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"
int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *light, *box, *sphere, *torus;
   int i;
   br_camera *camera_data;
/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Load root actor
    */

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   /*
    * Add and enable the default light source
    */
   light = BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL));
   BrLightEnable(light);
   /*
    * Load and position camera
    */
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                        BR_SCALAR(10.0));
   camera_data = (br_camera *)observer->type_data;
   camera_data->yon_z = BR_SCALAR(50);
   /*
    * Load and position Box model
    */
   box = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
```

```
    box->t.type = BR_TRANSFORM_MATRIX34;
    BrMatrix34RotateY(&box->t.t.mat,BR_ANGLE_DEG(30));
    BrMatrix34PostTranslate(&box->t.t.mat,BR_SCALAR(-2.5), BR_SCALAR(0.0),
                                                        BR_SCALAR(0.0));
    BrMatrix34PreScale(&box->t.t.mat,BR_SCALAR(2.0),BR_SCALAR(1.0),
                                                    BR_SCALAR(1.0));
    /*
     * Load and Position Sphere Model
     */
    sphere = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL);
    sphere->model = BrModelLoad("sph32.dat");
    BrModelAdd(sphere->model);
    sphere->t.type = BR_TRANSFORM_MATRIX34;
    BrMatrix34Translate(&sphere->t.t.mat,BR_SCALAR(2.0), BR_SCALAR(0.0),
                                                       BR_SCALAR(0.0));
    /*
     * Load and Position Torus Model
     */
    torus = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL);
    sphere->model = BrModelLoad("torus.dat");
    BrModelAdd(torus->model);
    torus->t.type = BR_TRANSFORM_MATRIX34;
    BrMatrix34Translate(&torus->t.t.mat,BR_SCALAR(0.0), BR_SCALAR(0.0),
                                                      BR_SCALAR(3.0));

  /********************** Animation Loop ********************************/
    for(i=0; i < 360; i++)  {
                BrPixelmapFill(back_buffer,0);
                BrPixelmapFill(depth_buffer,0xFFFFFFFF);
                BrZbSceneRender(world,observer,back_buffer,depth_buffer);
                BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
                BrMatrix34PostRotateX(&box->t.t.mat,BR_ANGLE_DEG(2.0));
                BrMatrix34PreRotateZ(&torus->t.t.mat,BR_ANGLE_DEG(4.0));
                BrMatrix34PreRotateY(&torus->t.t.mat,BR_ANGLE_DEG(-6.0));
                BrMatrix34PreRotateX(&torus->t.t.mat,BR_ANGLE_DEG(2.0));
                BrMatrix34PostRotateX(&torus->t.t.mat,BR_ANGLE_DEG(1.0));
                BrMatrix34PostRotateY(&sphere->t.t.mat,BR_ANGLE_DEG(0.8));
    }
    /* Close down */

    BrPixelmapFree(depth_buffer);
    BrPixelmapFree(back_buffer);
    BrZbEnd();
    DOSGfxEnd();
    BrEnd();
    return 0;
}
```

BRTUTOR2.C

BRTUTOR3.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to display a Planet, Moon, Satellite animation
 */
#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"
int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *universe, *observer, *light, *planet, *moon, *satellite;
   int i;
   br_camera *camera_data;

/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Load Root Actor
    */
   universe = BrActorAllocate(BR_ACTOR_NONE,NULL);
   /*
    * Load and Enable Default Light Source
    */
   light = BrActorAdd(universe,BrActorAllocate(BR_ACTOR_LIGHT,NULL));
   BrLightEnable(light);
   /*
    * Load and Position Camera
    */
   observer = BrActorAdd(universe,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                        BR_SCALAR(6.0));
   camera_data = (br_camera *)observer->type_data;
   camera_data->yon_z = BR_SCALAR(50);
   /*
    * Load Planet Model
    */
   planet = BrActorAdd(universe,BrActorAllocate(BR_ACTOR_MODEL,NULL));
```

```
    planet->model = BrModelLoad("sph16.dat");
    BrModelAdd(planet->model);
    /*
     * Load and Position Moon Model
     */
    moon = BrActorAdd(planet,BrActorAllocate(BR_ACTOR_MODEL,NULL);
    moon->model = BrModelLoad("sph8.dat");
    BrModelAdd(moon->model);
    moon->t.type = BR_TRANSFORM_MATRIX34;
    BrMatrix34Scale(&moon->t.t.mat,BR_SCALAR(0.5),BR_SCALAR(0.5),
                                                  BR_SCALAR(0.5));
    BrMatrix34PostTranslate(&moon->t.t.mat,BR_SCALAR(0.0), BR_SCALAR(0.0),
                                                        BR_SCALAR(2.0));
    /*
     * Load and Position Satellite Model
     */
    satellite = BrActorAdd(moon,BrActorAllocate(BR_ACTOR_MODEL,NULL);
    satellite->model = BrModelLoad("sph8.dat");
    BrModelAdd(satellite->model);
    satellite->t.type = BR_TRANSFORM_MATRIX34;
    BrMatrix34Scale(&satellite->t.t.mat,BR_SCALAR(0.25), BR_SCALAR(0.25),
                                                    BR_SCALAR(0.25));
    BrMatrix34PostTranslate(&satellite->t.t.mat,BR_SCALAR(1.5), BR_SCALAR(0.0),
                                                      BR_SCALAR(0.0));

  /********************** Animation Loop ********************************/
    for(i=0; i < 1000; i++)  {
                BrPixelmapFill(back_buffer,0);
                BrPixelmapFill(depth_buffer,0xFFFFFFFF);
                BrZbSceneRender(universe,observer,back_buffer,depth_buffer);
                BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
                BrMatrix34PreRotateY(&planet->t.t.mat,BR_ANGLE_DEG(1.0));
                BrMatrix34PreRotateY(&satellite->t.t.mat,BR_ANGLE_DEG(4.0));
                BrMatrix34PreRotateZ(&moon->t.t.mat,BR_ANGLE_DEG(1.5));
                BrMatrix34PostRotateZ(&satellite->t.t.mat,BR_ANGLE_DEG(-2.5));
                BrMatrix34PostRotateY(&moon->t.t.mat,BR_ANGLE_DEG(-2.0));
    }
    /* Close down */

    BrPixelmapFree(depth_buffer);
    BrPixelmapFree(back_buffer);
    BrZbEnd();
    DOSGfxEnd();
    BrEnd();
    return 0;
}
```

BRTUTOR3.C

BRTUTOR4.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to display a Planet-Satellite animation
 */
#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"
int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *cube, *planet, *sat, *wings1, *wings2;
   int i;
   br_camera *camera_data;

/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   /*
    * Load and Position Camera
    */
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                        BR_SCALAR(8.0));
   camera_data = (br_camera *)observer->type_data;
   camera_data->yon_z = BR_SCALAR(350);
   camera_data->hither_z = BR_SCALAR(0.5);
   /*
    * Load and Position Planet Actor
    */
   planet = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   planet->model = BrModelLoad("sph4096.dat");
   BrModelAdd(planet->model);
   planet->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&planet->t.t.mat,BR_SCALAR(14.0), BR_SCALAR(14.0),
                                                      BR_SCALAR(-40.0));
   /*
```

```
    * Load and Position Satellite
    */
   sat = BrActorAdd(planet,BrActorAllocate(BR_ACTOR_MODEL,NULL);
   sat->model = BrModelLoad("sph16.dat");
   BrModelAdd(sat->model);
   sat->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Scale(&sat->t.t.mat,BR_SCALAR(0.5), BR_SCALAR(0.5),
                                                 BR_SCALAR(0.5));
   BrMatrix34PostTranslate(&sat->t.t.mat,BR_SCALAR(2.0), BR_SCALAR(0.0),
                                                       BR_SCALAR(0.0));
   /* Add 'wings' to Satellite
    */
   wings1 = BrActorAdd(sat,BrActorAllocate(BR_ACTOR_MODEL,NULL);
   wings1->model = BrModelLoad("cylinder.dat");
   BrModelAdd(wings1->model);
   wings1->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Scale(&wings1->t.t.mat,BR_SCALAR(0.25), BR_SCALAR(0.25),
                                                 BR_SCALAR(2.0));
   /* Add more 'wings' to Satellite
    */
   wings2 = BrActorAdd(sat,BrActorAllocate(BR_ACTOR_MODEL,NULL);
   wings2->model = BrModelLoad("cylinder.dat");
   BrModelAdd(wings2->model);
   wings2->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Scale(&wings2->t.t.mat,BR_SCALAR(0.25), BR_SCALAR(0.25),
                                                 BR_SCALAR(2.0));
   BrMatrix34PostRotateY(&wings2->t.t.mat,BR_ANGLE_DEG(90.0));

 /********************** Animation Loop ********************************/
   for(i=0; i < 500; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(universe,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostTranslate(&planet->t.t.mat,BR_SCALAR(-0.033),
                                  BR_SCALAR(-0.032),BR_SCALAR(0.1));
              BrMatrix34PreRotateY(&planet->t.t.mat,BR_ANGLE_DEG(1.0));
              BrMatrix34PreRotateX(&sat->t.t.mat,BR_ANGLE_DEG(15.0));
              BrMatrix34PreRotateY(&sat->t.t.mat,BR_ANGLE_DEG(10.0));
              BrMatrix34PostRotateZ(&sat->t.t.mat,BR_ANGLE_DEG(1.0));
              BrMatrix34PostRotateY(&sat->t.t.mat,BR_ANGLE_DEG(3.0));
   }
   /* Close down */
   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR4.C

BRTUTOR5.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Illuminated Blue Cube.
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *cube;
   br_material *cube_material;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin("VESA,W:320,H:200,B:15");
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Load Root Actor. Load and Enable Deault Light Source
    */
   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   /*
    *Load and Position Camera
    */
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Position Cube Model
    */
   cube = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   cube->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateY(&cube->t.t.mat,BR_ANGLE_DEG(30));

   /*
```

```
    * Load and Apply Cube's Material
    */
   cube_material - BrFmtScriptMaterialLoad("cube.mat");
   BrMaterialAdd(cube_material);
   cube->material = BrMaterialFind("BLUE MATERIAL");

 /********************* Animation Loop ********************************/
   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateX(&cube->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR5.C

BRTUTR5B.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Illuminated Blue Cube (8-bit mode)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *cube;
   br_material *cube_material;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Load Root Actor. Load and Enable Deault Light Source
    */
   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   /*
    *Load and Position Camera
    */
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Position Cube Model
    */
   cube = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   cube->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateY(&cube->t.t.mat,BR_ANGLE_DEG(30));
```

```
    /*
     * Load and Apply Cube's Material
     */
    cube_material - BrFmtScriptMaterialLoad("cube8.mat");
    BrMaterialAdd(cube_material);
    cube->material = BrMaterialFind("BLUE MATERIAL");
  /********************** Animation Loop *********************************/
    for(i=0; i < 200; i++)  {
                BrPixelmapFill(back_buffer,0);
                BrPixelmapFill(depth_buffer,0xFFFFFFFF);
                BrZbSceneRender(world,observer,back_buffer,depth_buffer);
                BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
                BrMatrix34PostRotateX(&cube->t.t.mat,BR_ANGLE_DEG(2.0));
    }
    /* Close down */

    BrPixelmapFree(depth_buffer);
    BrPixelmapFree(back_buffer);
    BrZbEnd();
    DOSGfxEnd();
    BrEnd();
    return 0;
}
```

BRTUTR5B.C

BRTUTOR6.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Texture Mapped Sphere (15-bit colour)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *planet;
   br_material *planet_material;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin("VESA,W:320,H:200,B:15");
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Position Planet Actor
    */

   planet = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   planet->model = BrModelLoad("sph32.dat");
   BrModelAdd(planet->model);
   planet->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateX(&planet->t.t.mat,BR_ANGLE(90));

   /*
```

```
    * Load and Register 'earth' Texture
    */

   BrMapAdd(BrPixelmapLoad("earth15.pix"));

   /*
    * Load and Apply 'earth' Material
    */

   planet_material = BrFmtScriptMaterialLoad("earth.mat");
   BrMaterialAdd(planet_material);
   planet->material = BrMaterialFind("earth_map");

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateY(&planet->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR6.C

BRTUTR6B.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Texture Mapped Sphere (8-bit colour)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette, *shade;
   br_actor *world, *observer, *planet;
   br_material *planet_material;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

   /*
    * Load Shade Table
    */

   shade = BrPixelmapLoad("shade.tab");
   if (shade==NULL);
      BR_ERROR("Couldn't load shade.tab");
   BrTableAdd(shade);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));

   /*
```

```
    * Load and Position Planet Actor
    */

   planet = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   planet->model = BrModelLoad("sph32.dat");
   BrModelAdd(planet->model);
   planet->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateX(&planet->t.t.mat,BR_ANGLE(90));

   /*
    * Load and Register 'earth' Texture
    */

   BrMapAdd(BrPixelmapLoad("earth8.pix"));

   /*
    * Load and Apply 'earth' Material
    */

   planet_material - BrFmtScriptMaterialLoad("earth8.mat");
   BrMaterialAdd(planet_material);
   planet->material = BrMaterialFind("earth_.map");

 /********************** Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateY(&planet->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTR6B.C

BRTUTOR7.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Yellow Duck (15-bit)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *duck;
   br_material *mats[10]; /*for storing pointers to material descriptions*/
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin("VESA,W:320,H:200,B:15");
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Apply Duck Materials
    */

   i = BrFmtScriptMaterialLoadMany("duck.mat",mats,BR_ASIZE(mats));
   BrMaterialAddMany(mats,i);



   /*
```

```
    * Load and Position Duck Actor
    */

   duck = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   duck->model = BrModelLoad("duck.dat");
   BrModelAdd(duck->model);
   duck->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateY(&duck->t.t.mat,BR_ANGLE(30));

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateX(&duck->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR7.C

BRTUTR7B.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Yellow Duck (8-bit)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *duck;
   br_material *mats[10]; /*for storing pointers to material descriptions*/
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                        BR_SCALAR(5.0));
   /*
    * Load and Apply Duck Materials
    */

   i = BrFmtScriptMaterialLoadMAny("duck8.mat",mats,BR_ASIZE(mats));
   BrMaterialAddMany(mats,i);



   /*
```

```
    * Load and Position Duck Actor
    */

   duck = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   duck->model = BrModelLoad("duck.dat");
   BrModelAdd(duck->model);
   duck->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateY(&duck->t.t.mat,BR_ANGLE(30));

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);

              BrMatrix34PostRotateX(&duck->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTR7B.C

BRTUTOR8.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Texture Mapped Duck (15-bit)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *duck;
   br_pixelmap *gold_pm;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin("VESA,W:320,H:200,B:15");
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Register 'gold' Texture
    */
   gold_pm = BrPixelmapLoad("gold15.pix");
   if (gold_pm==NULL);
      BR_ERROR("Couldn't load gold15.pix");
   BrMapAdd(gold_pm);

   /*
    * Load and Apply 'gold' Material
```

```
    */

   BrMaterialAdd(BrFmtScriptMaterialLoad("gold15.mat"));

   /*
    * Load and Position Duck Actor
    */

   duck = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   duck->model = BrModelLoad("duck.dat");
   BrModelAdd(duck->model);
   BrModelApplyMap(duck->model,BR_APPLYMAP_PLANE,NULL);
   duck->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateX(&duck->t.t.mat,BR_ANGLE(30));
   duck->material = BrMaterialFind("gold15");

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateY(&duck->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR8.C

BRTUTR8B.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Texture Mapped Duck (8-bit)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette, *shade;
   br_actor *world, *observer, *duck;
   br_pixelmap *gold_pm;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

   /*
    * Load Shade Table
    */

   shade = BrPixelmapLoad("shade.tab");
   if (shade==NULL);
      BR_ERROR("Couldn't load shade.tab");
   BrTableAdd(shade);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
```

```
    * Load and Register 'gold' Texture
    */
   gold_pm = BrPixelmapLoad("gold8.pix");
   if (gold_pm==NULL);
      BR_ERROR("Couldn't load gold8.pix");
   BrMapAdd(gold_pm);

   /*
    * Load and Apply 'gold' Material
    */

   BrMaterialAdd(BrFmtScriptMaterialLoad("gold8.mat"));

   /*
    * Load and Position Duck Actor
    */

   duck = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   duck->model = BrModelLoad("duck.dat");
   BrModelAdd(duck->model);
   BrModelApplyMap(duck->model,BR_APPLYMAP_PLANE,NULL);
   duck->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateX(&duck->t.t.mat,BR_ANGLE(30));
   duck->material = BrMaterialFind("gold8");

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
               BrPixelmapFill(back_buffer,0);
               BrPixelmapFill(depth_buffer,0xFFFFFFFF);
               BrZbSceneRender(world,observer,back_buffer,depth_buffer);
               BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
               BrMatrix34PostRotateY(&duck->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTR8B.C

BRTUTOR9.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Chrome-Textured Fork(8-bit)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette, *shade;
   br_actor *world, *observer, *fork;
   br_pixelmap *chrome_pm;
   int i;

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

   /*
    * Load Shade Table
    */

   shade = BrPixelmapLoad("shade.tab");
   if (shade==NULL);
      BR_ERROR("Couldn't load shade.tab");
   BrTableAdd(shade);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
```

```
    * Load and Register 'chrome' Texture
    */
   chrome_pm = BrPixelmapLoad("refmap.pix");
   if (chrome_pm==NULL);
      BR_ERROR("Couldn't load refmap.pix");
   BrMapAdd(chrome_pm);

   /*
    * Load and Apply 'fork' Material
    */

   BrMaterialAdd(BrFmtScriptMaterialLoad("fork.mat"));

   /*
    * Load and Position Fork Actor
    */

   fork = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   fork->model = BrModelLoad("fork.dat");
   BrModelAdd(fork->model);
   BrModelApplyMap(fork->model,BR_APPLYMAP_PLANE,NULL);
   fork->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34RotateX(&fork->t.t.mat,BR_ANGLE(30));
   fork->material = BrMaterialFind("CHROME GIFMAP");

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
               BrPixelmapFill(back_buffer,0);
               BrPixelmapFill(depth_buffer,0xFFFFFFFF);
               BrZbSceneRender(world,observer,back_buffer,depth_buffer);
               BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
               BrMatrix34PostRotateX(&fork->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUTOR9.C

BRTUTR10.C

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Texture-Mapped Fork (15-bit mode)
 */

#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"

int main(int argc, char **argv)
{
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   br_actor *world, *observer, *fork;
   br_pixelmap *tile_pm;
   int i;
   br_material *mats[10];

/************* Initialise BRender and Graphics Hardware ********************/

   BrBegin();

   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */

   screen_buffer = DOSGfxBegin("VESA,W:320,H:200,B:15");
   palette = BrPixelmapLoad("std.pal");
   if(palette)
      DOSGfxPaletteSet(palette);
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/

   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   BrLightEnable(BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL)));
   observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
   observer->t.type = BR_TRANSFORM_MATRIX34;
   BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                          BR_SCALAR(5.0));
   /*
    * Load and Register TILE0011 Texture
    */
   tile_pm = BrPixelmapLoad("tile.pix");
   if (tile_pm==NULL);
      BR_ERROR("Couldn't load tile.pix");
   BrMapAdd(tile_pm);

   /*
```

```
    * Load and Apply fork Material
    */

   i = BrMaterialAdd(BrFmtScriptMaterialLoad("fork.mat",mats,BR_ASIZE(mats));
   BrMaterialAddMany(mats,i);
   /*
    * Load and Position fork Actor
    */

   fork = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
   fork->model = BrModelLoad("fork.dat");
   BrModelAdd(fork->model);
   BrModelApplyMap(fork->model,BR_APPLYMAP_PLANE,NULL);
   fork->t.type = BR_TRANSFORM_MATRIX34;
   /*
    * Assign fork material
    */
   fork->material = BrMaterialFind("GRIDMAP");

 /********************* Animation Loop *********************************/

   for(i=0; i < 200; i++)  {
              BrPixelmapFill(back_buffer,0);
              BrPixelmapFill(depth_buffer,0xFFFFFFFF);
              BrZbSceneRender(world,observer,back_buffer,depth_buffer);
              BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
              BrMatrix34PostRotateY(&fork->t.t.mat,BR_ANGLE_DEG(2.0));
   }
   /* Close down */

   BrPixelmapFree(depth_buffer);
   BrPixelmapFree(back_buffer);
   BrZbEnd();
   DOSGfxEnd();
   BrEnd();
   return 0;
}
```

BRTUR10.C

BRTUTOR1.C WITH 3DMAX SUPPORT

```
/*
 * Copyright (c) 1996 Argonaut Technologies Limited. All rights reserved.
 * Program to Display a Revolving Illuminated Cube.
 */
#include <stddef.h>
#include <stdio.h>
#include "brender.h"
#include "dosio.h"
#include "stereo.h"
#define kInterocular BR_SCALAR( 0.05 )
#define kYaw         BR_ANGLE_DEG( 1 )

int main(int argc, char **argv)
{
   /*
    * Need screen and back buffers for double-buffering, a Z-Buffer to store
    * depth information, and a storage buffer for the currently loaded palette
    */
   br_pixelmap *screen_buffer, *back_buffer, *depth_buffer, *palette;
   /*
    * The actors in the world: Need a root actor, a camera actor, a light actor,
    * and a model actor
    */
   br_actor *world, *observer, *light, *cube;
   int i;                                        /*counter*/
/************* Initialise BRender and Graphics Hardware ********************/
   BrBegin();
   /*
    * Initialise screen buffer and set up CLUT (ignored in true colour)
    */
   screen_buffer =KasanGfxBegin(NULL);
   palette = BrPixelmapLoad("std.pal");
   if(palette)
     DOSGfxPaletteSet(palette);
   /*
    * Initialise z-buffer renderer
    */
   BrZbBegin(screen_buffer->type, BR_PMT_DEPTH_16);
   /*
    * Allocate Back Buffer and Depth Buffer
    */
   back_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_OFFSCREEN);
   depth_buffer = BrPixelmapMatch(screen_buffer,BR_PMMATCH_DEPTH_16);

/*************** Build the World Database *********************************/
   /*
    * Start with None actor at root of actor tree and call it 'world'
    */
   world = BrActorAllocate(BR_ACTOR_NONE,NULL);
   /*
    * Add a camera actor as a child of None actor 'world */
```

```
  observer = BrActorAdd(world,BrActorAllocate(BR_ACTOR_CAMERA,NULL));
  /*
   * Add and enable the default light source
   */
  light = BrActorAdd(world,BrActorAllocate(BR_ACTOR_LIGHT,NULL));
  BrLightEnable(light);
  /*
   * Move camera 5 units along +z axis so model becomes visible
   */
  observer->t.type = BR_TRANSFORM_MATRIX34;
  BrMatrix34Translate(&observer->t.t.mat,BR_SCALAR(0.0),BR_SCALAR(0.0),
                                                        BR_SCALAR(5.0));
  /*
   * Add a model actor, the default cube
   */
  cube = BrActorAdd(world,BrActorAllocate(BR_ACTOR_MODEL,NULL));
  /*
   * Rotate cube to enhance visibility
   */
  cube->t.type = BR_TRANSFORM_MATRIX34;
  BrMatrix34RotateY(&cube->t.t.mat,BR_ANGLE_DEG(30));

/********************** Animation Loop ********************************/
  /*
   * Rotate cube around x-axis
   */
  for(i=0; i < 360; i++)  {
            /*
             * Initialise depth buffer and set background colour to black
             */
            BrPixelmapFill(back_buffer,0);
            BrPixelmapFill(depth_buffer,0xFFFFFFFF);
            /*
             * Render scene
             */
            KasanZbSceneRender(world, observer, back_buffer,
                               depth_buffer, kInterocular, kYaw);
            BrPixelmapDoubleBuffer(screen_buffer,back_buffer);
            /*
             * Rotate cube
             */
            BrMatrix34PostRotateX(&cube->t.t.mat,BR_ANGLE_DEG(2.0));
  }
  /* Close down */

  BrPixelmapFree(depth_buffer);
  BrPixelmapFree(back_buffer);
  BrZbEnd();
  KasanGfxEnd();
  BrEnd();
  return 0;
}
```

BRTUTOR1.C WITH 3DMAX SUPPORT

KASAN.C

```
/*
Wrappers around Kasan's 3D BIOS calls.
Copyright (c) 1995 Stainless Software Ltd. and Argonaut Technologies Ltd.
*/
#include <i86.h>
#include <kasan.h>

/* INT 0x10 / AX = kKasan_presence_detect should return kKasan_signature */
#define kKasan_presence_detect0x4ED0
#define kKasan_signature0x3344

/* Most 3D BIOS calls are INT 0x10 / AX = kKasan_BIOS / BX = one of these: */
#define kKasan_BIOS0x4FD0
#define kKasan_screen_off0x3D00
#define kKasan_screen_on0x3D01
#define kKasan_glasses_off0x3D02
#define kKasan_glasses_on0x3D03
#define kKasan_glasses_phase_toggle0x3D04
#define kKasan_BIOS_information0x3D10
#define kKasan_3D_information0x3D13
#define kKasan_modify_IRQ_parameter0x3D14

/* The next one goes in BH and a screen mode code goes in BL */
#define kKasan_mode_change0x3D
#define kKasan_320x400x80x31
#define kKasan_512x700x80x35
#define kKasan_1024x700x80x39
#define kKasan_512x700x160x36
#define kKasan_1024x700x160x3A

/* Return presence of 3DMax glasses */
int KasanPresent( void )
{
   union REGPACKregs;

   /* according to the Watcom manual the segment registers should always be
      valid and can be 0 */
   regs.w.ds = 0;
   regs.w.es = 0;
   regs.w.fs = 0;
   regs.w.gs = 0;

   regs.w.ax = kKasan_presence_detect;
   intr( 0x10, &regs );
   return (regs.w.dx == kKasan_signature);
}

void KasanScreenOn( void )
{
   union REGPACKregs;
```

```
   regs.w.ds = 0;
   regs.w.es = 0;
   regs.w.fs = 0;
   regs.w.gs = 0;
   regs.w.ax = kKasan_BIOS;
   regs.w.bx = kKasan_screen_on;
   intr( 0x10, &regs );
}

void KasanGlassesOn( void )
{
   union REGPACKregs;

   regs.w.ds = 0;
   regs.w.es = 0;
   regs.w.fs = 0;
   regs.w.gs = 0;
   regs.w.ax = kKasan_BIOS; /* required */
   regs.w.bx = kKasan_glasses_on;
   intr( 0x10, &regs );
}

void KasanGlassesOff( void )
{
   union REGPACKregs;

   regs.w.ds = 0;
   regs.w.es = 0;
   regs.w.fs = 0;
   regs.w.gs = 0;
   regs.w.ax = kKasan_BIOS;
   regs.w.bx = kKasan_glasses_off;
   intr( 0x10, &regs );
}

void KasanScreenOff( void )
{
   union REGPACKregs;

   regs.w.ds = 0;
   regs.w.es = 0;
   regs.w.fs = 0;
   regs.w.gs = 0;
   regs.w.ax = kKasan_BIOS;
   regs.w.bx = kKasan_screen_off;
   intr( 0x10, &regs );
}
```

KASAN.C

STEREO.C

```
/*
   Copyright (c) 1995 Stainless Software Ltd. and Argonaut Technologies Ltd.
*/
#include <stdlib.h>
#include <brender.h>
#include <dosio.h>
#include <kasan.h>
#include <stereo.h>

br_pixelmap * BR_PUBLIC_ENTRY KasanGfxBegin( char *pNew_setup_string )
{
   br_pixelmap *res;

   if (!KasanPresent())
      return (NULL);

   res = DOSGfxBegin( pNew_setup_string );
   KasanScreenOn();
   KasanGlassesOn();
   return (res);
}

void BR_PUBLIC_ENTRY KasanGfxEnd( void )
{
   KasanGlassesOff();
   KasanScreenOff();
   DOSGfxEnd();
}
/*
 * Matrix multiply for top 3x3 only
 */
#define A(x,y) A->m[x][y]
#define B(x,y) B->m[x][y]
#define C(x,y) C->m[x][y]

static void Matrix33Mul(br_matrix34 *A, br_matrix34 *B, br_matrix34 *C)
{
        A(0,0) = BR_MAC3(B(0,0),C(0,0), B(0,1),C(1,0), B(0,2),C(2,0));
        A(0,1) = BR_MAC3(B(0,0),C(0,1), B(0,1),C(1,1), B(0,2),C(2,1));
        A(0,2) = BR_MAC3(B(0,0),C(0,2), B(0,1),C(1,2), B(0,2),C(2,2));

        A(1,0) = BR_MAC3(B(1,0),C(0,0), B(1,1),C(1,0), B(1,2),C(2,0));
        A(1,1) = BR_MAC3(B(1,0),C(0,1), B(1,1),C(1,1), B(1,2),C(2,1));
        A(1,2) = BR_MAC3(B(1,0),C(0,2), B(1,1),C(1,2), B(1,2),C(2,2));

        A(2,0) = BR_MAC3(B(2,0),C(0,0), B(2,1),C(1,0), B(2,2),C(2,0));
        A(2,1) = BR_MAC3(B(2,0),C(0,1), B(2,1),C(1,1), B(2,2),C(2,1));
        A(2,2) = BR_MAC3(B(2,0),C(0,2), B(2,1),C(1,2), B(2,2),C(2,2));
}
```

```
#undef A
#undef B
#undef C

static void Rotate33YInPlace( br_matrix34 *m, br_angle a )
{
   br_matrix34temp,
            rot;
   int      i,
            j;

   BrMatrix34RotateY( &rot, a );
   Matrix33Mul( &temp, &rot, m);
   /* copy the 'top' of the matrix out but leave the bottom row alone */
   for (i=0; i<3; i++)
   {
      for (j=0; j<3; j++)
         (*m).m[i][j]=temp.m[i][j];
   }
}

void BR_PUBLIC_ENTRY KasanZbSceneRender(br_actor *pWorld,
                  br_actor *pCamera,
                  br_pixelmap *pCol_buffer,
                  br_pixelmap *pDepth_buffer,
                  br_scalar pInterocular_distance,
                  br_angle pCamera_yaw )
{
   br_matrix34saved_camera_mat;
   br_uint_16old_height,
               old_field_bytes,
               old_Z_bytes;

   /* take copies of pixelmap fields which we frob to restore later */
   old_height = pCol_buffer->height;
   old_field_bytes = pCol_buffer->row_bytes;
   old_Z_bytes = pDepth_buffer->row_bytes;

   pCol_buffer->height /= 2;
   pCol_buffer->row_bytes *= 2;
   pDepth_buffer->row_bytes *= 2;

   saved_camera_mat = pCamera->t.t.mat;
   /* render the left field */
   pCamera->t.t.mat.m[3][0] -= BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][0] );
   pCamera->t.t.mat.m[3][1] -= BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][1] );
   pCamera->t.t.mat.m[3][2] -= BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][2] );
   Rotate33YInPlace( &pCamera->t.t.mat, -pCamera_yaw );
   BrZbSceneRender(pWorld, pCamera, pCol_buffer, pDepth_buffer);
```

```
    pCamera->t.t.mat = saved_camera_mat;
    /* render the right field */
    pCol_buffer->pixels =(char*)pCol_buffer->pixels +
                  pCol_buffer->row_bytes / 2;
    pDepth_buffer->pixels =(char*)pDepth_buffer->pixels +
                  pDepth_buffer->row_bytes / 2;
    pCamera->t.t.mat.m[3][0] += BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][0] );
    pCamera->t.t.mat.m[3][1] += BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][1] );
    pCamera->t.t.mat.m[3][2] += BR_MUL(pInterocular_distance / 2,
                  pCamera->t.t.mat.m[0][2] );
    Rotate33YInPlace( &pCamera->t.t.mat, pCamera_yaw );
    BrZbSceneRender(pWorld, pCamera, pCol_buffer, pDepth_buffer);

    /* restore all we frobbed */
    pCol_buffer->pixels =(char*)pCol_buffer->pixels -
                  pCol_buffer->row_bytes / 2;
    pDepth_buffer->pixels =(char*)pDepth_buffer->pixels -
                  pDepth_buffer->row_bytes / 2;
    pCamera->t.t.mat = saved_camera_mat;
    pCol_buffer->height = old_height;
    pCol_buffer->row_bytes = old_field_bytes;
    pDepth_buffer->row_bytes = old_Z_bytes;
}
```

STEREO.C

MAIN.C

```
/*
   General BRender/Stereo glasses test code.
   Build a simple world and let the user fly around it.
   Bruce Mardle, Stainless Software, 1995.
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

#include <brender.h>
#include <dosio.h>
#include "stereo.h"

#if defined(__WATCOMC__)
#define vsnprintf(str, size, format, args) _vbprintf(str, size, format, args)
#elif !defined(__bsdi__)
#define vsnprintf(str, size, format, args) sprintf(str, format, args)
#endif

#define kInterocular    BR_SCALAR( 0.05 )
#define kYaw                    BR_ANGLE_DEG( 1 )

#define kSmoothSwitch "smooth"

/* Fatal error messages. These all have "Error:" plonked before them. */
#define kUsageMsg  \
  "usage: %s image-width image-height distance-to-monitor ["kSmoothSwitch"]\n"
#define kDOSGfxMsg "failed to access graphics.\n"
#define kMemoryMsg "not enough memory.\n"
#define kStdPalMsg "failed to load std.pal\n"
#define kMatScrMsg "failed to load mat.scr\n"
#define kLoadFailedMsg "failed to load %s\n"

/* Measurements in the same units as the program arguments, nominally m: */
#define k4hedronEdge 1
#define kCameraHither 0.1
#define kCameraYon 200
#define kInitCameraHeight 1.6
#define kInitCameraDistance 9
#define kGndSide 20
  /* if kGndSide is too big it's displayed incorrectly. Hmm */

//#define kGroundPixelmapName "256.pix"
#define kGroundPixelmapName "256xlogo.pix"
#define kGroundShadeTableName "std.tab"
#define kSpin 10
#define kGravity 0.01
#define kParticles 20
#define kGndN 10
```

```
  /* number of triangles on each side of the 'ground' */
#define kBallRadius 100
  /* 'mickeys'. bigger numbers give less sensitivity */

#define MAX_MATERIALS 10

/* GLOBALS */
int flag3d,numMaterials;
br_material *material[MAX_MATERIALS];

static void cleanUp(void)
{
  DOSClockEnd();
  DOSMouseEnd();
  if(flag3d){
    KasanGfxEnd();
  }
  else{
    DOSGfxEnd();
  }
}

static void bomb(char *format, ...)
{
  va_list args;
  char msg[160];
  cleanUp();
  va_start(args, format);
  vsnprintf(msg, sizeof msg, format, args);
  BR_ERROR(msg);
  BrEnd();
  exit(1);
}

static br_actor *allocateAndAddActor
  (br_uint_8 actor_type, void *type_data, br_actor *parent)
{
  br_actor *n;
  if ((n=BrActorAllocate(actor_type, type_data))==NULL
      || (n=BrActorAdd(parent, n))==NULL /* impossible? */)
    bomb(kMemoryMsg);
  return(n);
}

typedef struct
{
  br_actor *actor;
  br_vector3 initV;
  int birthFrame;
} particle_t;

static void initParticle(particle_t *p, int frame) {
  static char count=0;
```

```
  count++;
  count%=numMaterials;
  /* Generate random initial velocity. */
  BrVector3Set(&p->initV,
    BR_SCALAR( (double)(rand()-(int)RAND_MAX/2)/RAND_MAX/5),
    BR_SCALAR(((double)(rand()-(int)RAND_MAX/2)/RAND_MAX+1)/2),
    BR_SCALAR( (double)(rand()-(int)RAND_MAX/2)/RAND_MAX/5));
      /* for some reason Watcom define RAND_MAX as an unsigned. Grr */
  p->birthFrame=frame;
  BrMatrix34Translate(&p->actor->t.t.mat, 0, 0, 0);
  /* Colour those hedrons */
  p->actor->material=material[count];
}

static void updateParticle(particle_t *p, int frame) {
  br_scalar y, t=BR_SCALAR(frame-p->birthFrame);
  /* Set particle pos.n to initV*t+(0,-kGravity,0)*t^2 */
  BrMatrix34Translate(&p->actor->t.t.mat,
    BR_MUL(p->initV.v[0], t),
    (y=BR_MUL(BR_SUB(p->initV.v[1], BR_MUL(BR_SCALAR(kGravity), t)), t)),
    BR_MUL(p->initV.v[2], t));
  BrMatrix34PreRotateY(&p->actor->t.t.mat,
    BrDegreeToAngle(BR_MUL(t, BR_SCALAR(kSpin))));
  if (y<-BR_SCALAR(k4hedronEdge))
    initParticle(p, frame);
}

int main(int argc, char **argv)
{
  unsigned imageWidth, imageHeight; /* as seen on the monitor */
  unsigned monitorDistance;
  br_pixelmap *screenBuffer, *backBuffer, *ZBuffer, *palette;
  br_actor *world, *viewpoint, *camera, *tetra, *actor /* temp */;
  particle_t particle[kParticles];
  br_model *model /* temp */;
  br_int_32 oldMouseX, mouseX, oldMouseY, mouseY;
  br_uint_32 mouseButtons;
  int frame, i, j, smooth,scaleDirection=!NULL;
  br_material *groundMaterial;
  br_scalar scale=BR_SCALAR(0.5);

  /*
     Set up all the things which can cause chaos and confusion
     so we know which things to tidy up when if we bomb out.
  */
  BrBegin();
  if (argc!=4 && argc!=5
      || sscanf(argv[1], "%u", &imageWidth)!=1
      || sscanf(argv[2], "%u", &imageHeight)!=1
      || sscanf(argv[3], "%u", &monitorDistance)!=1
      || (smooth=argc==5) && strcmp(argv[4], kSmoothSwitch))
  {
    BR_ERROR1(kUsageMsg, argv[0]);
```

```
    exit(1);
  }
  if (!(flag3d=(int)(screenBuffer=KasanGfxBegin(NULL))))
    if ((screenBuffer=DOSGfxBegin(NULL))==NULL){
      BR_ERROR0(kDOSGfxMsg);
      exit(1);
    }
  DOSMouseBegin();
  DOSClockBegin();

  /* Set up other vital things. */
  if ((backBuffer=BrPixelmapMatch(screenBuffer, BR_PMMATCH_OFFSCREEN))==NULL)
    bomb(kMemoryMsg);
  if ((ZBuffer=BrPixelmapMatch(screenBuffer, BR_PMMATCH_DEPTH_16))==NULL)
    bomb(kMemoryMsg);
  BrZbBegin(screenBuffer->type, ZBuffer->type);
  if ((palette=BrPixelmapLoad("std.pal"))==NULL)
    bomb(kStdPalMsg);
  DOSGfxPaletteSet(palette);
  if
                (!(numMaterials=BrFmtScriptMaterialLoadMany(smooth?"smooth.sc
                r":"flat.scr",material,MAX_MATERIALS)))
    bomb(kMatScrMsg);
  BrMaterialAddMany(material,numMaterials);

  /* Create the world. */
  if ((world=BrActorAllocate(BR_ACTOR_NONE, NULL))==NULL)
    bomb(kMemoryMsg);
  viewpoint=allocateAndAddActor(BR_ACTOR_NONE, NULL, world);
  BrMatrix34Translate(&viewpoint->t.t.mat,
    0, BR_SCALAR(kInitCameraHeight), BR_SCALAR(kInitCameraDistance));
  camera=allocateAndAddActor(BR_ACTOR_CAMERA, NULL, viewpoint);
  ((br_camera*)camera->type_data)->aspect=
    BR_DIV(BR_SCALAR(imageWidth),BR_SCALAR(imageHeight));
  ((br_camera*)camera->type_data)->field_of_view=2*
    BR_ATAN2(BR_CONST_DIV(BR_SCALAR(imageWidth),2),
                BR_SCALAR(monitorDistance));
  ((br_camera*)camera->type_data)->hither_z=BR_SCALAR(kCameraHither);
  ((br_camera*)camera->type_data)->yon_z=BR_SCALAR(kCameraYon);
  BrLightEnable(actor=allocateAndAddActor(BR_ACTOR_LIGHT, NULL, viewpoint));
  ((br_light*)actor->type_data)->type=BR_LIGHT_POINT|BR_LIGHT_VIEW;
  ((br_light*)actor->type_data)->attenuation_c=BR_SCALAR(1.0);
  ((br_light*)actor->type_data)->attenuation_l=BR_SCALAR(0.0);
  ((br_light*)actor->type_data)->attenuation_q=BR_SCALAR(0.001);
  /* Parent of lots of tetrahedra: */
  tetra=allocateAndAddActor(BR_ACTOR_NONE, NULL, world);
  if ((model=BrModelAllocate("", 4, 4))==NULL)
    bomb(kMemoryMsg);
#define pa BR_SCALAR(0.353*k4hedronEdge)
#define pb BR_SCALAR(0.5*k4hedronEdge)
#define na (-pa)
#define nb (-pb)
  BrVector3Set(&model->vertices[0].p, pb,pa,0);
```

```
  BrVector3Set(&model->vertices[1].p, nb,pa,0);
  BrVector3Set(&model->vertices[2].p, 0,na,pb);
  BrVector3Set(&model->vertices[3].p, 0,na,nb);
  model->faces[0].vertices[0]=3;
  model->faces[0].vertices[1]=2;
  model->faces[0].vertices[2]=1;
  model->faces[1].vertices[0]=2;
  model->faces[1].vertices[1]=3;
  model->faces[1].vertices[2]=0;
  model->faces[2].vertices[0]=1;
  model->faces[2].vertices[1]=0;
  model->faces[2].vertices[2]=3;
  model->faces[3].vertices[0]=0;
  model->faces[3].vertices[1]=1;
  model->faces[3].vertices[2]=2;
  BrModelUpdate(model, BR_MODU_ALL);
  tetra->model=model;
#undef pa
#undef pb
#undef na
#undef nb
  /* The ground: a big triangle chopped into smaller ones: */
  actor=allocateAndAddActor(BR_ACTOR_MODEL, NULL, world);
  if ((model=BrModelAllocate("", (kGndN+1)*(kGndN+2)/2, kGndN*kGndN))==NULL)
    bomb(kMemoryMsg);
  for (i=0; i<=kGndN; i++)
    for (j=0; j<=i; j++)
      BrVector3Set(&model->vertices[i*(i+1)/2+j].p,
   BR_SCALAR((j*2-i)*0.5*kGndSide/kGndN),
   BR_SCALAR(0),
   BR_SCALAR((i*-1.732+2*kGndN/1.732)*0.5*kGndSide/kGndN));
  for (i=0; i<kGndN; i++)
  {
    for (j=0; j<=i; j++)
    {
      model->faces[i*i+2*j].vertices[0]=i*(i+1)/2+j;
      model->faces[i*i+2*j].vertices[1]=(i+1)*(i+2)/2+j+1;
      model->faces[i*i+2*j].vertices[2]=(i+1)*(i+2)/2+j;
      model->faces[i*i+2*j].smoothing=1;
    }
    for (j=0; j<i; j++)
    {
      model->faces[i*i+2*j+1].vertices[0]=i*(i+1)/2+j+1;
      model->faces[i*i+2*j+1].vertices[1]=(i+1)*(i+2)/2+j+1;
      model->faces[i*i+2*j+1].vertices[2]=i*(i+1)/2+j;
      model->faces[i*i+2*j+1].smoothing=1;
    }
  }
  BrModelUpdate(model, BR_MODU_ALL);
  actor->model=model;
  if ((actor->material=BrMaterialAllocate(""))==NULL)
    bomb(kMemoryMsg);
  groundMaterial=actor->material;
```

```
if (smooth){
  actor->material->flags|=BR_MATF_SMOOTH|BR_MATF_PERSPECTIVE;
}
else{
  actor->material->flags|=BR_MATF_PERSPECTIVE;
}
if ((actor->material->colour_map=BrPixelmapLoad(kGroundPixelmapName))==NULL)
  bomb(kLoadFailedMsg, kGroundPixelmapName);
if ((actor->material-
              >index_shade=BrPixelmapLoad(kGroundShadeTableName))==NULL)
  bomb(kLoadFailedMsg, kGroundShadeTableName);
{
  br_matrix34 mm;
  BrModelFitMap(model, BR_FITMAP_PLUS_X, BR_FITMAP_PLUS_Z, &mm);
  BrModelApplyMap(model, BR_APPLYMAP_PLANE, &mm);
}
BrMaterialUpdate(actor->material, BR_MATU_ALL);

/* Lots of little tetrahedra */
for (i=0; i<kParticles; i++)
{
  particle[i].actor=allocateAndAddActor(BR_ACTOR_MODEL, NULL, tetra);
  initParticle(particle+i, 0);
}

/* Display the scene */
DOSMouseRead(&oldMouseX, &oldMouseY, &mouseButtons);
for (frame=0;
     !(mouseButtons&BR_MSM_BUTTONR) || !(mouseButtons&BR_MSM_BUTTONL);
     frame++)
{
  br_matrix34 m;

  groundMaterial->map_transform.m[0][0]=BR_MUL(scale,
              BR_COS(BrDegreeToAngle(BR_SCALAR(frame*3))));
  groundMaterial->map_transform.m[0][1]=BR_MUL(scale,
              BR_SIN(BrDegreeToAngle(BR_SCALAR(frame*3))));
  groundMaterial->map_transform.m[1][0]=BR_MUL(BR_SCALAR(-1),BR_MUL(scale,
              BR_SIN(BrDegreeToAngle(BR_SCALAR(frame*3)))));
  groundMaterial->map_transform.m[1][1]=BR_MUL(scale,
              BR_COS(BrDegreeToAngle(BR_SCALAR(frame*3))));

  scale+=scaleDirection?BR_SCALAR(0.1):BR_SCALAR(-0.1);
  if (scale>BR_SCALAR(1.5)||scale<BR_SCALAR(0.5))
 scaleDirection=!scaleDirection;
  /* Process user input */
  DOSMouseRead(&mouseX, &mouseY, &mouseButtons);
  /* Limit rate of change: */
  if (mouseX-oldMouseX>10)
    mouseX=oldMouseX+10;
  if (mouseX-oldMouseX<-10)
    mouseX=oldMouseX-10;
  if (mouseY-oldMouseY>10)
```

```
      mouseY=oldMouseY+10;
    if (mouseY-oldMouseY<-10)
      mouseY=oldMouseY-10;
    BrMatrix34RollingBall(&m, (br_int_16)(oldMouseX-mouseX),
      (br_int_16)(mouseY-oldMouseY), kBallRadius);
    BrMatrix34Pre(&viewpoint->t.t.mat, &m);
    if (mouseButtons&BR_MSM_BUTTONL)
      BrMatrix34PostTranslate(&viewpoint->t.t.mat,
   BR_CONST_MUL(viewpoint->t.t.mat.m[2][0], -1),
   BR_CONST_MUL(viewpoint->t.t.mat.m[2][1], -1),
   BR_CONST_MUL(viewpoint->t.t.mat.m[2][2], -1));
    if (mouseButtons&BR_MSM_BUTTONR)
      BrMatrix34PostTranslate(&viewpoint->t.t.mat,
   viewpoint->t.t.mat.m[2][0],
   viewpoint->t.t.mat.m[2][1],
   viewpoint->t.t.mat.m[2][2]);
    /* Update particles */
    for (i=0; i<kParticles; i++)
      updateParticle(particle+i, frame);
    /* Update display */
    BrPixelmapFill(backBuffer, 0);
    BrPixelmapFill(ZBuffer, 0xFFFFFFFF);
    if(flag3d){
      KasanZbSceneRender(world, camera, backBuffer, ZBuffer, kInterocular,
                kYaw);
    }
    else{
      BrZbSceneRender(world, camera, backBuffer, ZBuffer);
    }
    backBuffer=BrPixelmapDoubleBuffer(screenBuffer, backBuffer);
    /* Where were we? */
    oldMouseX=mouseX;
    oldMouseY=mouseY;
  }

  cleanUp();
  BrEnd();
  return(0);
}
```

MAIN.C