利用 Ogre 設計遊戲程式

27

Come, Watson, come!The game is afoot.
—Sir Arthur Conan Doyle
For it's one, two, three strikes you're out at the old ball game.
—Jack Norworth
The game is up.
—William Shakespeare
If you wish to avoid foreign collision, you had better abandon the ocean.
—Henry Clay

學習目標

在本章中，你將學到：

- 遊戲程式設計的基礎。
- 使用 Ogre 設計遊戲。
- 執行碰撞偵測。
- 使用 Ogre 匯入並顯示圖像。
- 利用 OgreAL，將 OpenAL 音效函式庫整合到你的遊戲中。
- 讓 Ogre 接受鍵盤輸入。
- 使用 Ogre 和 OgreAL 來建立簡單的乒乓遊戲。
- 使用 Ogre 來控制遊戲的速度。

本章綱要

27.1 簡介

我們現在要來介紹使用 Ogre 3D 繪圖引擎進行遊戲程式設計與圖形處理。Ogre 於 2000 年由 Steve Streeting 所建立，目前爲一開放原始碼專案，由 `www.ogre3d.org` 的 Ogre 團隊來維護。首先，我們會簡單介紹遊戲程式設計的基本觀念。然後我們會介紹如何利用 Ogre 建立一個簡單的遊戲，它類似 Atari 在 1972 年開發的經典乒乓球電視遊戲。我們將示範如何建立一個彩色 3D 圖形的遊戲場景、流暢地繪製移動物件的動畫、用計時器控制動畫的速度、偵測物件之間的碰撞、加入音效、接收鍵盤輸入，以及顯示文字輸出。

27.2 安裝 Ogre、OgreAL 和 OpenAL

Ogre 具有功能強大、容易使用的優點，但是想要將它安裝好，卻牽涉到各種不同平台和編譯器的問題。你有兩種安裝選項：安裝 Ogre SDK，或是下載並編譯原始碼。在本書中，我們使用 Ogre 1.4 (Eihort)。我們必須安裝 OgreAL 和 OpenAL 音效函式庫。OpenAL 負責音效功能，讓你可以將 OpenAL 功能與 Ogre 程式碼整合在一起。我們會提供 Ogre

SDK 和 OgreAL 所有元件的詳細安裝步驟，也會解釋如何設定 Visual C++ 2008 專案以使用 Ogre 和 OgreAL。你可以在本書網站 `www.deitel.com/books/cpphtp7` 的「Additional Resources」小節中找到這些安裝步驟。

27.3　遊戲程式設計基礎

我們現在要介紹遊戲程式設計的一些基礎觀念。第 27.4 節會則會利用 Ogre 和 OgreAL 函式庫實作一個乒乓遊戲。

繪圖

繪圖也許是電動遊戲最重要的功能了。繪圖曾經是一種專業技術，然而，圖形程式設計已經變得越來越普及，甚至連程式設計初學者也能上手。市面上有許多種 **3D 繪圖引擎 (3D graphics engines)**，這些框架隱藏了沉悶複雜的繪圖 API 程式設計工作，讓你可以輕鬆地管理圖形。

Ogre (Object-Oriented Graphics Rendering Engine) 是目前主要的繪圖引擎之一，用在許多商業產品 (包括電視遊戲) 上。它替 3D 繪圖程式設計提供了物件導向的介面。它支援 Direct3D 和 OpenGL 繪圖 API，可以在 Windows、Linux 和 Mac 平台上執行。**Direct3D** 是微軟的 Windows 3D 繪圖 API。**OpenGL** 是一個由許多視訊卡廠商共同制定的繪圖規格，跨越許多主要的平台，包括 Windows 系統。

Ogre 是一個純粹的繪圖渲染引擎，它不直接支援音效、物理 (physics)、碰撞偵測、網路或是其他與遊戲有關的功能。Ogre 社群建立了許多附加元件，讓使用者可以將 Ogre 與其他函式庫整合在一起，以支援這些功能。

3D 模型

3D 模型 (3D model) 是一個物件的電腦表示法，可以在螢幕上繪製出來，這個過程稱爲**渲染 (rendering)**。**材質 (Materials)** 可用來決定物件的外觀，這是藉由設定光源屬性、顏色和紋理來完成的。**紋理 (texture)** 是用來包裝在模型外表的圖形。

顯示在 3D 繪圖中的大部分物件 (地形、角色、建築等等一切事物) 都是 3D 模型。許多模型是以 **3D 模型工具 (3D modeling tools)** 建立的。常用的 3D 模型工具包括 Maya (`usa.autodesk.com/adsk/servlet/index?siteID=123112&id=7635018`)、SoftImage XSI (`www.softimage.com/`) 和 Blender (`www.blender.org/`)。它們都適用於 Windows、Linux 和 Mac 平台。Blender 是免費的，而 Maya 也提供 free 版本。

SoftImage 有 30 天的免費試用期。Ogre 社群也發展了幾個工具，讓使用者從這些常用的模型工具中，**將 3D 模型「匯出」(export 3D models)** 到 Ogre。

材質、紋理與顏色

顏色 (Colors) 是由紅、綠和藍光的強度 (範圍從 0 到 1.0) 所決定的，因此顏色值 (1.0,0,0)會產生亮紅色，(0,1.0,0) 會產生亮綠色，而 (0,0,1.0) 會產生亮藍色。第一個值是紅色的強度，第二個是綠色強度，第三個是藍色強度。要產生白色，請將三個顏色值都調到最大強度 (1.0,1.0,1.0)。要產生黑色，也就是沒有任何一種顏色，請使用 (0,0,0)。顏色值有時候包括 **alpha 通道 (alpha channel)**，可用來表示透明度，其範圍亦爲 0 到 1.0，0 表示完全透明，1 表示完全不透明。圖 27.1 列出了常用的顏色及其紅、綠、藍強度值。你可以也在網路上找到顏色表，例如 `www.tayloredmktg.com/rgb/`。

顏色	紅色值	綠色值	藍色值
紅色	1.0	0.0	0.0
綠色	0.0	1.0	0.0
藍色	0.0	0.0	1.0
橘色	1.0	0.784	0.0
粉紅色	1.0	0.686	0.686
青綠色	0.0	1.0	1.0
洋紅色	1.0	0.0	1.0
黃色	1.0	1.0	0.0
黑色	0.0	0.0	0.0
白色	1.0	1.0	1.0
灰色	0.5	0.5	0.5
淡灰色	0.75	0.75	0.75
深灰色	0.25	0.25	0.25

圖 27.1 Ogre 中常用顏色的紅、綠、藍強度

在 3D 繪圖中，材質 (materials) 可以用來決定 3D 模型的顏色，決定模型如何反射不同種類的光線，並將紋理套用到模型上。我們可以依據模型距離觀看者的遠近，將材質設定爲具有不同的**細節層次 (levels of detail，LoD)**。當距離很近時，應該儘量精細

地繪製。當距離很遠時，就不需要浪費電腦的力氣來繪製看不到的細節。爲了增加效能，我們可以較粗略地繪製這些物件。

光源

在 3D 場景中，有四種不同的光源：環境光 (ambient)、散射光 (diffuse)、自發光 (emissive) 和反射光 (specular)。[1] **環境光 (Ambient light)** 是場景中的一般光線，它來自很多物體的表面，沒有固定的來源。**散射光 (Diffuse light)** 來自特定的方向，但是會平均地從它所照射到的表面上反射出來。**自發光 (Emissive light)** 來自場景中的某個物件。自發光不會影響周圍的物件，但它會讓發射光源的物件看起來較爲明亮。**反射光 (specular)** 來自特定的方向，反射光會依觀察者的角度而有所不同，它讓物件看起來具有光澤。

碰撞偵測和回應

碰撞偵測 (Collision detection) 是判斷遊戲中某兩個物件是否產生觸碰的過程。你必須知道要測試哪一個物件，並處理一些複雜的數學運算。想要知道一個正方形是否碰到另一個，是比較容易的，假如它們都和地面平行。想要檢測圓形和球體就困難得多，曲面的數學運算比較複雜。

當物件碰撞到其他物件時，應該要做出適當的回應。某些物件，例如牆壁，是固定的；某些物件則會四處移動。移動物件的物理模擬是十分複雜的。你可以利用碰撞偵測函式庫和物理模擬函式庫來處理這些複雜的問題。這些函式庫可以創造出具眞實感的遊戲體驗。

音效

對電動遊戲來說，音效是非常重要的感受。遊戲玩家會想要聽到雷射光束在戰艦上爆炸的聲音，或是賽車離開起跑線時的引擎聲。音效函式庫可以讓你利用聲音，讓遊戲增色不少。這些函式庫中有許多支援 **3D 音效 (3D sound)**。在 3D 場景中，物件可能是以不同的距離和方向朝使用者發出聲響。在播放音效時，音效函式庫會將這些因素列入考量。假如物件離聽者較近，就會比遠處物件所發出的聲音要來得大。同樣地，從聽者的某一方向傳來的聲音，也會和來自另一方的播放方式不同。

---

[1] D. Astel and K. Hawkins, *Beginning OpenGL:Game Programming*.Boston, MA:Thompson, 2004, pp. 104–110.

文字

遊戲通常會以顯示文字的方式來與使用者溝通。這些溝通方式範圍包括了給予使用者指引，或是單純地告訴使用者他到目前爲止取得多少分數。在很多遊戲中，文字是玩家之間的重要溝通管道。你可以在網站 `www.1001freefonts.com` 中找到免費的字形以供使用。

計時器

遊戲的執行速度可能會因系統的不同而有所變化，這是因爲處理器的速度差異所致。爲了解決這個問題，遊戲程式設計者使用**計時器 (timers)** 來控制動畫的速度。假如物件在每個**影格 (frame**，每一次的螢幕重繪) 所移動的距離相同，則它在不同電腦上的移動速率可能都不相同。它在以每秒 100 影格的速度執行的遊戲上，比以每秒 50 影格執行的同樣遊戲上，移動的速度快一倍。計時器可以幫助玩家調整遊戲的速度爲一致的。

使用者體驗

遊戲應該是有趣好玩的，而且應該以各種方法僅可能地吸引玩遊戲的人。我們剛才討論的基本概念，都可以讓使用者感覺遊戲更好玩。你可以藉由圖形和聲音，來吸引玩家的注意。遊戲動作通常要搭配聲音。許多網站都提供免費的音效，讓你可以運用在遊戲之中，我們列舉出一些受歡迎的音效網站供你參考，包括：Sound Hunter (`www.soundhunter.com`)、Absolute Sound Effects Archive (`www.grsites.com/sounds`) 和搜尋引擎 FindSounds (`www.findsounds.com`)。你也可以播放原聲帶做爲背景音樂，但是，假如你要將你的遊戲發布成爲產品，在使用版權之前，請記得取得許可。

玩家會跟遊戲產生互動。使用者的輸入裝置包括鍵盤、滑鼠、搖桿和遊戲控制器。請記得，儘量讓控制方式保持簡單：遊戲應該容易操作，而不容易獲勝。你可以使用文字與玩家溝通。

27.4 乒乓遊戲：實作程式碼

在接下來幾節中，我們提供完整的 C++ /Ogre 程式，實作一個簡單的遊戲。這個遊戲類似 Atari 在 1972 年開發的經典電動遊戲「Pong」 (請見圖 27.2 的投幣式電玩) 。我們會將程式碼從頭到尾看過一次，並依此解釋 Ogre 的功能。這是本書最大的範例程式之一。在閱讀接下來的程式碼討論之前，你應該先完整地測試這個程式。你會在本書範例的

ch22 資料夾中，找到屬於這個程式的全部檔案。將 PongResources 資料夾複製到 OgreSDK\media 資料夾中。假如找不到這些資源的其中一個，Ogre 會拋出執行時期例外。開啓 Pong 的 Visual C++ 專案。假如你有遵照指令，正確地安裝 Ogre 和 OgreAL，則此專案應該會正確建立。此專案的執行檔會複製到你的 OgreSDK\bin\debug 資料夾(假如你以 Release 模式建立，就是 OgreSDK\bin\releas 資料夾)。重要事項：記得將 OgreAL_d.dll (在 Release 模式下則是 OgreAL.dll) 和 alut.dll 檔案複製到 Pong 執行檔所在的資料夾。

圖 27.2　Atari 的投幣式 Pong 遊戲，它在 1972 年成為商業產品，一共售出了 38,000 台遊戲機。這個遊戲完全是由 TTL 電路設計而成的，沒有使用到 CPU 或是遊戲程式軟體。它的「程式碼」是由簡單的「閘極」晶片、計時器和計數器晶片實作而成的。這個簡單、有趣、讓人上癮的遊戲，奠定了未來電玩產業的基石。(PONG classic video game courtesy of Atari Interactive, Inc. 2007 Atari Interactive, Inc. All rights reserved.Used with permission.)

乒乓遊戲有四個主要的遊戲物件：一顆球、兩個球拍，以及一個方框 (圖 27.3)。乒乓球會在方框中，橫跨螢幕地彈跳，玩家則必須控制球拍，左右擊打乒乓球。假如乒乓球碰到方框的左邊或右邊，「攻擊」的那一方就會得到一分。分數會顯示在螢幕的上方。

試著玩一下遊戲，利用 A 和 Z 來控制左邊的球拍，up 和 down 來控制右邊的球拍。請注意物件的顏色、球與其他物件的互動，以及螢幕上顯示的分數。按下 Esc 鍵可以離開遊戲，關閉視窗無法停止遊戲。

27.4.1　Ogre 初始化

類別 Pong (圖 27.4–27.5) 代表我們的乒乓遊戲。圖 27.4 爲 `Pong` 類別的定義。第 6 行將標頭檔 `Ogre.h` 含括進來，這個檔案自動地含括了最常用的 Ogre 標頭檔。第 22 行是 `run` 函式的原型，這個函式會讓遊戲開始執行。第 25–26 行包含了處理使用者鍵盤輸入的函式原型。第 29–30 行定義的函式原型用來執行影格之間的遊戲邏輯。我們也宣告了指向重要 Ogre 物件 (第 38–42 行) 的指標，輸入處理的物件 (第 45–46)，以及遊戲中的物件 (第 49–51 行)。第 54–56 行定義一些變數，用來控制遊戲的行爲，我們會在稍後討論它們。

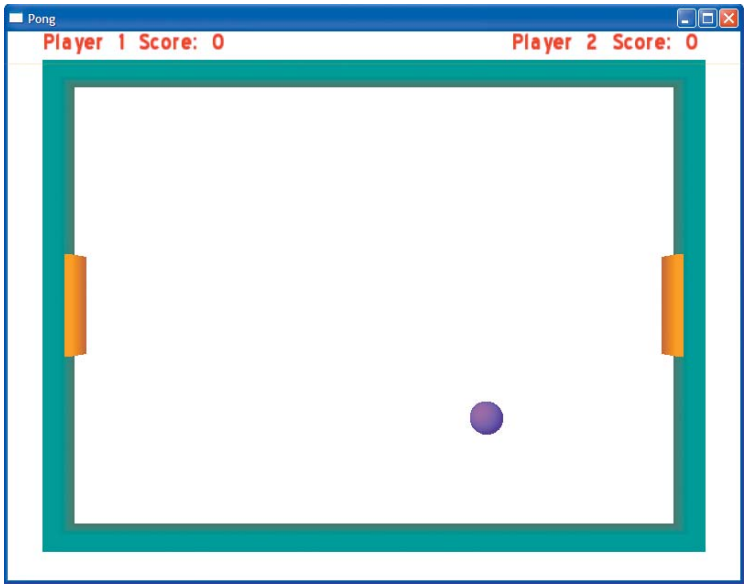

圖 27.3　乒乓遊戲中的物件

```
1  // Pong.h
2  // Pong class definition (represents a game of Pong).
3  #ifndef PONG_H
4  #define PONG_H
5
6  #include <Ogre.h> // Ogre class definitions
7  #include <OIS/OISEvents.h> // OISEvents class definition
8  #include <OIS/OISInputManager.h> // OISInputManager class definition
9  #include <OIS/OISKeyboard.h> // OISKeyboard class definition
10 using namespace Ogre;
11
```

圖 27.4　Pong 類別定義 (代表乒乓遊戲)

```
12  class Ball; // forward declaration of class Ball
13  class Paddle; // forward declaration of class Paddle
14
15  enum Players { PLAYER1, PLAYER2 };
16
17  class Pong : public FrameListener, public OIS::KeyListener
18  {
19  public:
20     Pong(); // constructor
21     ~Pong(); // destructor
22     void run(); // run a game of Pong
23
24     // handle keyPressed and keyReleased events
25     bool keyPressed( const OIS::KeyEvent &keyEventRef );
26     bool keyReleased( const OIS::KeyEvent &keyEventRef );
27
28     // move the game objects and control interactions between frames
29     virtual bool frameStarted( const FrameEvent &frameEvent );
30     virtual bool frameEnded( const FrameEvent &frameEvent );
31     static void updateScore( Players player ); // update the score
32
33  private:
34     void createScene(); // create the scene to be rendered
35     static void updateScoreText(); // update the score on the screen
36
37     // Ogre objects
38     Root *rootPtr; // pointer to Ogre's Root object
39     SceneManager *sceneManagerPtr; // pointer to the SceneManager
40     RenderWindow *windowPtr; // pointer to RenderWindow to render scene in
41     Viewport *viewportPtr; // pointer to Viewport, area that a camera sees
42     Camera *cameraPtr; // pointer to a Camera in the scene
43
44     // OIS input objects
45     OIS::InputManager *inputManagerPtr; // pointer to the InputManager
46     OIS::Keyboard *keyboardPtr; // pointer to the Keyboard
47
48     // game objects
49     Ball *ballPtr; // pointer to the Ball
50     Paddle *leftPaddlePtr; // pointer to player 1's Paddle
51     Paddle *rightPaddlePtr; // pointer to player 2's Paddle
52
53     // variables to control game states
54     bool quit, pause; // did user quit or pause the game?
55     Real time; // used to delay the motion of a new Ball
56     static bool wait; // should the Ball's movement be delayed?
57
58     static int player1Score; // player 1's score
59     static int player2Score; // player 2's score
60  }; // end class Pong
61
62  #endif // PONG_H
```

圖 27.4　Pong 類別定義 (代表乒乓遊戲) (續)

```
1  // Pong.cpp
2  // Pong class member-function definitions.
3  #include <sstream>
4  #include <stdexcept>
5  #include <Ogre.h> // Ogre class definitions
6  #include <OgreAL.h> // OgreAL class definitions
7  #include <OgreStringConverter.h> // OgreStringConverter class definition
8  #include <OIS/OISEvents.h> // OISEvents class definition
9  #include <OIS/OISInputManager.h> // OISInputManager class definition
10 #include <OIS/OISKeyboard.h> // OISKeyboard class definition
11 #include "Ball.h" // Ball class definition
12 #include "Paddle.h" // Paddle class definition
13 #include "Pong.h" // Pong class definition
14 using namespace std;
15 using namespace Ogre;
16
17 int Pong::player1Score = 0; // initialize player 1's score to 0
18 int Pong::player2Score = 0; // initialize player 2's score to 0
19 bool Pong::wait = false; // initialize wait to false
20
21 // directions to move the Paddles
22 const Vector3 PADDLE_DOWN = Vector3( 0, -15, 0 );
23 const Vector3 PADDLE_UP = Vector3( 0, 15, 0 );
24
25 // constructor
26 Pong::Pong()
27 {
28    rootPtr = new Root(); // initialize Root object
29
30    // use the Ogre Config Dialog Box to choose the settings
31    if ( !( rootPtr->showConfigDialog() ) ) // user canceled the dialog box
32       throw runtime_error( "User Canceled Ogre Setup Dialog Box." );
33
34    // get a pointer to the RenderWindow
35    windowPtr = rootPtr->initialise( true, "Pong" );
36
37    // create the SceneManager
38    sceneManagerPtr = rootPtr->createSceneManager( ST_GENERIC );
39
40    // create the Camera
41    cameraPtr = sceneManagerPtr->createCamera( "PongCamera" );
42    cameraPtr->setPosition( Vector3( 0, 0, 200 ) ); // set Camera position
43    cameraPtr->lookAt( Vector3( 0, 0, 0 ) ); // set where Camera looks
44    cameraPtr->setNearClipDistance( 5 ); // near distance Camera can see
45    cameraPtr->setFarClipDistance( 1000 ); // far distance Camera can see
46
47    // create the Viewport
48    viewportPtr = windowPtr->addViewport( cameraPtr );
49    viewportPtr->setBackgroundColour( ColourValue( 0, 0, 0 ) );
50
51    // set the Camera's aspect ratio
52    cameraPtr->setAspectRatio( Real( viewportPtr->getActualWidth() ) /
53       ( viewportPtr->getActualHeight() ) );
```

圖 27.5　Pong 類別的成員函式定義

```
54
55      // set the scene's ambient light
56      sceneManagerPtr->setAmbientLight( ColourValue( 0.75, 0.75, 0.75 )
57
58      // create the Light
59      Light *lightPtr = sceneManagerPtr->createLight( "Light" ); // a Light
60      lightPtr->setPosition( 0, 0, 50 ); // set the Light's position
61
62      unsigned long hWnd; // variable to hold the window handle
63      windowPtr->getCustomAttribute( "WINDOW", &hWnd ); // get window handle
64      OIS::ParamList paramList; // create an OIS ParamList
65
66      // add the window to the ParamList
67      paramList.insert( OIS::ParamList::value_type( "WINDOW",
68         Ogre::StringConverter::toString( hWnd ) ) );
69
70      // create the InputManager
71      inputManagerPtr = OIS::InputManager::createInputSystem( paramList );
72      keyboardPtr = static_cast< OIS::Keyboard* >( inputManagerPtr->
73         createInputObject( OIS::OISKeyboard, true ) ); // create a Keyboard
74      keyboardPtr->setEventCallback( this ); // add a KeyListener
75
76      rootPtr->addFrameListener( this ); // add this Pong as a FrameListener
77
78      // load resources for Pong
79      ResourceGroupManager::getSingleton().addResourceLocation(
80         "resources", "FileSystem", "Pong" );
81      ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
82
83      quit = pause = false; // player has not quit or paused the game
84      time = 0; // initialize the time since Ball was reset to 0
85   } // end Pong constructor
86
87   // Pong destructor erases objects contained in a Pong object
88   Pong::~Pong()
89   {
90      // free dynamically allocated memory for Keyboard
91      inputManagerPtr->destroyInputObject( keyboardPtr );
92      OIS::InputManager::destroyInputSystem( inputManagerPtr );
93
94      // free dynamically allocated memory for Root
95      delete rootPtr; // release memory pointer points to
96      rootPtr = 0; // point pointer at 0
97
98      // free dynamically allocated memory for Ball
99      delete ballPtr; // release memory pointer points to
100     ballPtr = 0; // point pointer at 0
101
102     // free dynamically allocated memory for Paddle
103     delete leftPaddlePtr; // release memory pointer points to
104     leftPaddlePtr = 0; // point pointer at 0
105
106     // free dynamically allocated memory for Paddle
```

圖 27.5　Pong 類別的成員函式定義 (續 1)

```
107    delete rightPaddlePtr; // release memory pointer points to
108    rightPaddlePtr = 0; // point pointer at 0
109 } // end Pong destructor
110
111 // create the scene to be displayed
112 void Pong::createScene()
113 {
114    // get a pointer to the Score Overlay
115    Overlay *scoreOverlayPtr =
116       OverlayManager::getSingleton().getByName( "Score" );
117    scoreOverlayPtr->show(); // show the Overlay
118
119    // make the game objects
120    ballPtr = new Ball( sceneManagerPtr ); // make the Ball
121    ballPtr->addToScene(); // add the Ball to the scene
122    rightPaddlePtr = new Paddle( sceneManagerPtr, "RightPaddle", 90 );
123    rightPaddlePtr->addToScene(); // add a Paddle to the scene
124    leftPaddlePtr = new Paddle( sceneManagerPtr, "LeftPaddle", -90 );
125    leftPaddlePtr->addToScene(); // add a Paddle to the scene
126
127    // create the walls
128    Entity *entityPtr = sceneManagerPtr->
129       createEntity( "WallLeft", "cube.mesh" ); // create the left wall
130    entityPtr->setMaterialName( "wall" ); // set material for left wall
131    entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
132
133    // create the SceneNode for the left wall
134    SceneNode *nodePtr = sceneManagerPtr->getRootSceneNode()->
135       createChildSceneNode( "WallLeft" );
136    nodePtr->attachObject( entityPtr ); // attach left wall to SceneNode
137    nodePtr->setPosition( -95, 0, 0 ); // set the left wall's position
138    nodePtr->setScale( .05, 1.45, .1 ); // set the left wall's size
139    entityPtr = sceneManagerPtr->createEntity( "WallRight", "cube.mesh" );
140    entityPtr->setMaterialName( "wall" ); // set material for right wall
141    entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
142
143    // create the SceneNode for the right wall
144    nodePtr = sceneManagerPtr->getRootSceneNode()->
145       createChildSceneNode( "WallRight" );
146    nodePtr->attachObject( entityPtr ); // attach right wall to SceneNode
147    nodePtr->setPosition( 95, 0, 0 ); // set the right wall's position
148    nodePtr->setScale( .05, 1.45, .1 ); // set the right wall's size
149    entityPtr = sceneManagerPtr->createEntity( "WallBottom", "cube.mesh" );
150    entityPtr->setMaterialName( "wall" ); // set material for bottom wall
151    entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
152
153    // create the SceneNode for the bottom wall
154    nodePtr = sceneManagerPtr->getRootSceneNode()->
155       createChildSceneNode( "WallBottom" );
156    nodePtr->attachObject( entityPtr ); // attach bottom wall to SceneNode
157    nodePtr->setPosition( 0, -70, 0 ); // set the bottom wall's position
158    nodePtr->setScale( 1.95, .05, .1 ); // set bottom wall's size
159    entityPtr = sceneManagerPtr->createEntity( "WallTop", "cube.mesh" );
```

圖 27.5　Pong 類別的成員函式定義 (續 2)

```
160     entityPtr->setMaterialName( "wall" ); // set the material for top wall
161     entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
162
163     // create the SceneNode for the top wall
164     nodePtr = sceneManagerPtr->getRootSceneNode()->
165        createChildSceneNode( "WallTop" );
166     nodePtr->attachObject( entityPtr ); // attach top wall to the SceneNode
167     nodePtr->setPosition( 0, 70, 0 ); // set the top wall's position
168     nodePtr->setScale( 1.95, .05, .1 ); // set the top wall's size
169  } // end function createScene
170
171  // start a game of Pong
172  void Pong::run()
173  {
174     createScene(); // create the scene
175     rootPtr->startRendering(); // start rendering frames
176  } // end function run
177
178  // update the score
179  void Pong::updateScore( Players player )
180  {
181     // increase the correct player's score
182     if ( player == PLAYER1 )
183        player1Score++;
184     else
185        player2Score++;
186
187     wait = true; // the game should wait to restart the Ball
188     updateScoreText(); // update the score text on the screen
189  } // end function updateScore
190
191  // update the score text
192  void Pong::updateScoreText()
193  {
194     ostringstream scoreText; // text to be displayed
195
196     scoreText << "Player 2 Score: " << player2Score; // make the text
197
198     // get the right player's TextArea
199     OverlayElement *textElementPtr =
200        OverlayManager::getSingletonPtr()->getOverlayElement( "right" );
201     textElementPtr->setCaption( scoreText.str() ); // set the text
202
203     scoreText.str( "" ); // reset the text in scoreText
204     scoreText << "Player 1 Score: " << player1Score; // make the text
205
206     // get the left player's TextArea
207     textElementPtr =
208        OverlayManager::getSingletonPtr()->getOverlayElement( "left" );
209     textElementPtr->setCaption( scoreText.str() ); // set the text
210  } // end function updateScoreText
211
212  // respond to user keyboard input
```

圖 27.5　Pong 類別的成員函式定義 (續 3)

```
213 bool Pong::keyPressed( const OIS::KeyEvent &keyEventRef )
214 {
215    // if the game is not paused
216    if ( !pause )
217    {
218       // process KeyEvents that apply when the game is not paused
219       switch ( keyEventRef.key )
220       {
221       case OIS::KC_ESCAPE: // escape key hit: quit
222          quit = true;
223          break;
224       case OIS::KC_UP: // up-arrow key hit: move the right Paddle up
225          rightPaddlePtr->movePaddle( PADDLE_UP );
226          break;
227       case OIS::KC_DOWN: //down-arrow key hit: move the right Paddle down
228          rightPaddlePtr->movePaddle( PADDLE_DOWN );
229          break;
230       case OIS::KC_A: // A key hit: move the left Paddle up
231          leftPaddlePtr->movePaddle( PADDLE_UP );
232          break;
233       case OIS::KC_Z: // Z key hit: move the left Paddle down
234          leftPaddlePtr->movePaddle( PADDLE_DOWN );
235          break;
236       case OIS::KC_P: // P key hit: pause the game
237          pause = true; // set pause to true when the user pauses the game
238          Overlay *pauseOverlayPtr =
239             OverlayManager::getSingleton().getByName( "PauseOverlay" );
240          pauseOverlayPtr->show(); // show the pause Overlay
241          break;
242       } // end switch
243    } // end if
244    else // game is paused
245    {
246       // user hit 'R' on the keyboard
247       if ( keyEventRef.key == OIS::KC_R )
248       {
249          pause = false; // set pause to false when user resumes the game
250          Overlay *pauseOverlayPtr =
251             OverlayManager::getSingleton().getByName( "PauseOverlay" );
252          pauseOverlayPtr->hide(); // hide the pause Overlay
253       } // end if
254    } // end else
255    return true;
256 } // end function keyPressed
257
258 // process key released events
259 bool Pong::keyReleased( const OIS::KeyEvent &keyEventRef )
260 {
261    return true;
262 } // end function keyReleased
263
264 // return true if the program should render the next frame
```

圖 27.5　Pong 類別的成員函式定義 (續 4)

```
265 bool Pong::frameEnded( const FrameEvent &frameEvent )
266 {
267    return !quit; // quit = false if the user hasn't quit yet
268 } // end function frameEnded
269
270 // process game logic, return true if the next frame should be rendered
271 bool Pong::frameStarted( const FrameEvent &frameEvent )
272 {
273    keyboardPtr->capture(); // get keyboard events
274    // the game is not paused and the Ball should move
275    if ( !wait && !pause )
276    {
277       // move the Ball
278       ballPtr->moveBall( frameEvent.timeSinceLastFrame );
279    } // end if
280    // don't move the Ball if wait is true
281    else if ( wait )
282    {
283       // increase time if it is less than 4 seconds
284       if ( time < 4 )
285          // add the seconds since the last frame
286          time += frameEvent.timeSinceLastFrame;
287       else
288       {
289          wait = false; // shouldn't wait to move the Ball any more
290          time = 0; // reset the control variable to 0
291       } // end else
292    } // end else
293
294    return !quit; // quit = false if the user hasn't quit yet
295 } // end function frameStarted
```

圖 27.5　Pong 類別的成員函式定義 (續 5)

在顯示任何圖形之前，我們必須先初始化 Ogre 引擎的設定，建立特定的 Ogre 物件。OGRE 引擎渲染設定對話框 (圖 27.6) 能讓使用者選擇渲染的設定，包括使用哪一個**渲染子系統** (**rendering subsystem**，像是 Direct3D 或是 OpenGL)、解析度、顏色深度、全螢幕模式以及其他超出本章範圍的選項。Direct3D 僅爲 Windows 所用。OpenGL 可以支援全部的主要平台。**解析度 (resolution)** 是由兩個值所定義的：寬度和高度，用來決定繪製場景的像素。渲染子系統的解析度選項，範圍從 640x400 到 1680x1050 或更高，依你的硬體而定。較高的解析度可以產生較細緻的圖形。假如你選擇關掉全螢幕模式，解析度就會決定遊戲顯示的視窗大小。我們以 800x600 的解析度執行遊戲。**顏色深度 (color depth)** 爲 n，表示有 2^n 種顏色可以顯示在螢幕上。顏色深度越低程式所需的記憶體就越少，但是它的視覺效果可能就不如顏色深度較高的畫面。Direct3D 和 OpenGL 都支援 16 位元和 32 位元的顏色深度。在 32 位元的顏色中，只有 24 個位元可以用在顏色上，剩下的 8 個位元要用來表示 alpha 值 (透明度)。

要顯示對話框，你必須建立一個 **Root 物件** (**Root object**，圖 27.5 第 28 行) － 用來啓動引擎的 Ogre 物件。唯一可以在 Root 之前建立的 Ogre 物件是 `LogManager`，但是它不在本書介紹範圍之內。接下來，我們會呼叫 Root 類別 (第 31 行) 的 `showConfigDialog` 函式，以顯示對話框。一旦你點選了 OK，Ogre 就會儲存這些設定，將它們做爲下一次開啓對話框時的預設設定。假如使用者點選了 **`Cancel`**，程式應該要終止，因爲設定可能沒有完成而導致錯誤。假如 `showConfigDialog` 回傳 `false` 值 (使用者選擇 **`Cancel`**)，則會拋出 `runtime_error`。

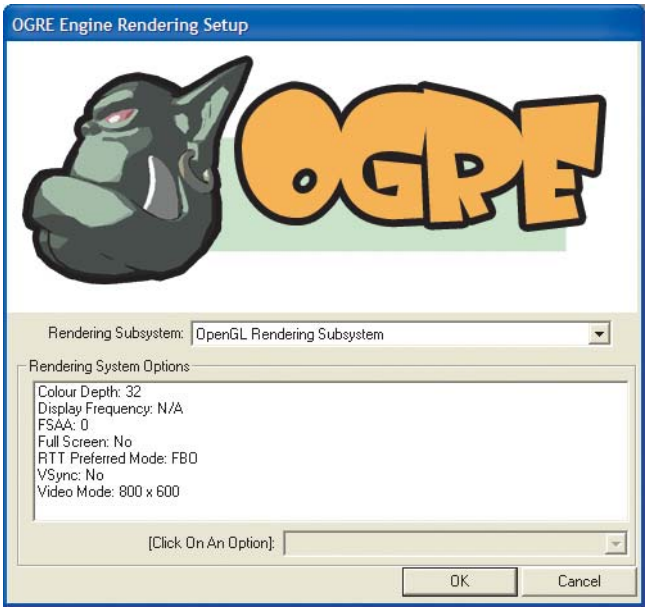

圖 27.6　　OGRE 引擎渲染設定對話框

一旦渲染子系統和視窗的設定都完成以後，我們就可以建立 **`RenderWindow`**，Ogre 會呼叫類別 `Root` 的 `initialise` 函式 (第 35 行) ，在這個視窗中繪製圖形。第一個參數是 `true`，告訴 Ogre 依據使用者在對話框中選擇的設定建立視窗。假如傳遞 `false` 給這個參數，會讓你在稍後手動地建立視窗。第二個參數 `"Pong"` 是引擎中的視窗名稱，會出現在視窗的標題列中 (非全螢幕時)。你可以發現，`initialise` 這個字和圖 27.6 的 **colour** 都是英式英語，因爲 Ogre 是來自英國。

27.4.2　建立場景

我們現在已經將 Ogre 設定完成，也建立了一個視窗可以用來繪製我們的圖形，接下來，我們會加入一些物件，來建立**場景 (scene)**，也就是顯示在螢幕上的一組影像。

SceneManager

我們使用 Ogre 的 **SceneManager** 物件 (第 38 行) 來控制場景。SceneManager 會管理**場景圖 (scene graph)**，這是一個包含場景內所有物件 (可見的和不可見的) 的資料結構。我們使用 SceneManager 建立物件，然後將它們加入場景圖之中，並決定要呈現出哪一個物件。將不在可見場景中的物件去除不呈現，又稱爲**「剔除 (culling)」**，可以減少渲染時間並增進效能，會自動執行。我們會維護一個指向 SceneManager 物件的指標，整個遊戲都會使用到它。

有好幾種 SceneManager 被設計用來處理不同的場景，像是室內場景或是寬闊的自然場景。Ogre 應用程式可以分別或同時使用一個以上的 SceneManager。本章只需要用到一個一般性的場景型別 (ST_GENERIC)，這個 SceneManager 並沒有爲了任何特殊的場景予以最佳化。

Camera

在建立 SceneManager 之後，我們就可以開始建構場景了。首先，我們要加入一個 Camera。在 Ogre 中，**Camera** 相當於是你的眼睛，用來檢視場景。3D 場景通常太大了，無法在單一視窗中顯示。Camera 會檢視場景，並告訴 Ogre，你眞正看到的是哪一部分。Camera 可以放在場景的任何位置，或是附加在 SceneNode 上 (我們稍後會討論它)。假如 Camera 附加在 SceneNode 上，當節點在場景內移動時，Camera 就會跟著它。Ogre 支援在單一場景中使用多個 Camera，但是我們只需要一個就好了。

我們使用 SceneManager 來建立 Camera (第 41 行)，然後設定位置、方向、拍攝距離以及 Viewport (第 42–49 行)。位置 (Position) 是 Camera 在場景中的位置。我們將 Camera 放在沿著 z 軸的正向，離原點 200 個單位，面向遊戲者的位置上。這個位置讓 Camera 與場景原點的距離夠遠，能夠位在遊戲的中心位置。我們可以將所有的 z 座標設爲 0，不需要更動它們，方向 (Camera) 則是 Camera 拍攝的方向。我們將遊戲配置在原點的四週，因此讓 Camera 朝向原點拍攝 (第 43 行)。拍攝距離 (clip distances) 則定義 Camera 能見到的遠近範圍。假如某個物體與 Camera 的距離小於 Camera 的最近拍攝距離，或是大於 Camera 的最遠拍攝距離，則 Camera 就不會看到這個物體。

Viewport 是用來顯示 Camera 所拍攝事物的螢幕範圍。我們將 Viewport 的背景設爲白色。Camera 可以擁有超過一個 Viewport，但是本遊戲中只使用到一個。Camera 擁有許多其他的功能和函式，但是我們目前只需要上述幾樣功能就足夠了。

光源

光源是 3D 場景中最重要的部份之一。Ogre 有三種**光源 (Light)**：點光源 (Point)、聚光燈 (Spot) 和平行光源 (Directional)。**點光源 (Point)** 會由固定的位置朝各個方向平均照射。**聚光燈 (Spot)** 跟點光源一樣具有固定的來源，但是光線只朝一個方向照射，光線強度會隨與光源的距離增加而衰退。**平行光源 (Directional)** 沒有固定的來源，它們僅朝一個方向平均地發射光線，無論你在場景的哪個位置，光線都來自同樣的方向。

我們的遊戲使用的是點光源。第 59–60 行使用 SceneManager 來建立 Light，並且設定它在場景中的位置。createLight 函式的引數是這個 Light 的名稱。我們指定光源的 x、y、z 座標，以設定光源的位置。現在，我們的場景已經完成，可供使用了。

27.4.3 將元素加入場景

我們先前曾經提過，乒乓遊戲有四個主要的遊戲物件：一顆球、兩個球拍，以及一個方框。我們必須將這些元素加入場景，才能將它們顯示出來。

將球加入場景

類別 Ball (圖 27.7–27.8) 代表在螢幕上彈跳的球。我們必須將 Ball 加入場景，才能將它顯示出來。成員函式 addToScene (圖 27.8, lines 26–51) 會建立一個代表 Ball 的 Entity，將它加入場景，然後建立一些與 Ball 有關的音效。我們稍後會討論有關音效的部份。在場景中，**Entity** 是一個 mesh 的實體。**mesh** 是一個檔案，內容爲 3D 模型的幾何資訊。程式中可以有多個 Entity 都以同一個 mesh 爲基礎，但是每個 Entity 都有不同的名稱。第 29-30 行建立一個 Entity，我們可以經由指標來參照這個 Entity。第一個引數是 Entity 的名稱。第二個引數 "sphere.mesh" 是一個 mesh 檔案，用來決定這個 Entity 的形狀。我們使用 Ogre SDK 提供的球狀 mesh，你可以在你的電腦的 OgreSDK\media\models 資料夾中找到這個 mesh。

```
1  // Ball.h
2  // Ball class definition (represents a bouncing ball).
3  #ifndef BALL_H
4  #define BALL_H
5
6  #include <Ogre.h> // Ogre class definitions
7  #include <OgreAL.h> // OgreAL class definitions
8  using namespace Ogre;
9
10 class Paddle; // forward declaration of class Paddle
11
12 const int RADIUS = 5; // the radius of the Ball
13
14 class Ball
15 {
16 public:
17    Ball( SceneManager *sceneManagerPtr ); // constructor
18    ~Ball(); // destructor
19    void addToScene(); // add the Ball to the scene
20    void moveBall( Real time ); // move the Ball across the screen
21
22 private:
23    SceneManager *sceneManagerPtr; // pointer to the SceneManager
24    SceneNode *nodePtr; // pointer to the SceneNode
25    OgreAL::SoundManager *soundManagerPtr; // pointer to the SoundManager
26    OgreAL::Sound *wallSoundPtr; // sound played when Ball hits a wall
27    OgreAL::Sound *paddleSoundPtr; // sound played when Ball hits a Paddle
28    OgreAL::Sound *scoreSoundPtr; // sound played when someone scores
29    int speed; // speed of the Ball
30    Vector3 direction; // direction of the Ball
31
32    // private utility functions
33    void reverseHorizontalDirection(); // change horizontal direction
34    void reverseVerticalDirection(); // change vertical direction
35    void hitPaddle(); // control the Ball hitting the Paddles
36 }; // end class Ball
37
38 #endif // BALL_H
```

圖 27.7　　Ball 類別定義 (代表一個跳動的球)

```
1  // Ball.cpp
2  // Ball class member-function definitions.
3  #include <Ogre.h> // Ogre class definitions
4  #include <OgreAL.h> // OgreAL class definitions
5  #include "Ball.h" // Ball class definition
6  #include "Paddle.h" // Paddle class definition
7  #include "Pong.h" // Pong class definition
8  using namespace Ogre;
9
10 // Ball constructor
11 Ball::Ball( SceneManager *ptr )
```

圖 27.8　　Ball 類別的成員函式定義

```
12 {
13    sceneManagerPtr = ptr; // set pointer to the SceneManager
14    soundManagerPtr = new OgreAL::SoundManager(); // create SoundManager
15    speed = 100; // speed of the Ball
16    direction = Vector3( 1, -1, 0 ); // direction of the Ball
17 } // end Ball constructor
18
19 // Ball destructor
20 Ball::~Ball()
21 {
22    // empty body
23 } // end Ball destructor
24
25 // add the Ball to the scene
26 void Ball::addToScene()
27 {
28    // create Entity and attach it to a node in the scene
29    Entity *entityPtr =
30       sceneManagerPtr->createEntity( "Ball", "sphere.mesh" );
31    entityPtr->setMaterialName( "ball" ); // set material for the Ball
32    entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
33    nodePtr = sceneManagerPtr->getRootSceneNode()->
34       createChildSceneNode( "Ball" ); // create a SceneNode
35    nodePtr->attachObject( entityPtr ); // attach the Entity to SceneNode
36    nodePtr->setScale( .05, .05, .05 ); // scale SceneNode
37
38    // attach sounds to Ball so they will play from where Ball is
39    wallSoundPtr = soundManagerPtr->
40       createSound( "wallSound", "wallSound.wav", false );
41    nodePtr->attachObject( wallSoundPtr ); // attach a sound to SceneNode
42    paddleSoundPtr = soundManagerPtr->
43       createSound( "paddleSound", "paddleSound.wav", false );
44    nodePtr->attachObject( paddleSoundPtr ); // attach sound to SceneNode
45    scoreSoundPtr = soundManagerPtr->
46       createSound( "cheer", "cheer.wav", false ); // create a Sound
47
48    // attach the score sound to its own node centered at ( 0, 0, 0 )
49    sceneManagerPtr->getRootSceneNode()->createChildSceneNode( "score" )->
50       attachObject( scoreSoundPtr );
51 } // end function addToScene
52
53 // move the Ball across the screen
54 void Ball::moveBall( Real time )
55 {
56    nodePtr->translate( ( direction * ( speed * time ) ) ); // move Ball
57    Vector3 position = nodePtr->getPosition(); // get Ball's new position
58
59    // get the positions of the four walls
60    Vector3 topPosition = sceneManagerPtr->
61       getSceneNode( "WallTop" )->getPosition();
62    Vector3 bottomPosition = sceneManagerPtr->
63       getSceneNode( "WallBottom" )->getPosition();
64    Vector3 leftPosition = sceneManagerPtr->
```

圖 27.8 Ball 類別的成員函式定義 (續 1)

```
65          getSceneNode( "WallLeft" )->getPosition();
66       Vector3 rightPosition = sceneManagerPtr->
67          getSceneNode( "WallRight" )->getPosition();
68
69       const int WALL_WIDTH = 5; // the width of the walls
70
71       // check if the Ball hit the left side
72       if ( ( position.x  - RADIUS ) <= leftPosition.x + ( WALL_WIDTH / 2 ) )
73       {
74          nodePtr->setPosition( 0, 0, 0 ); // place Ball in center of screen
75          Pong::updateScore( PLAYER2 ); // update the score
76          scoreSoundPtr->play(); // play a sound when player 2 scores
77       } // end if
78       // check if the Ball hit the right side
79       else if (
80          ( position.x + RADIUS ) >= rightPosition.x - ( WALL_WIDTH / 2 ) )
81       {
82          nodePtr->setPosition( 0, 0, 0 ); // place Ball in center of screen
83          Pong::updateScore( PLAYER1 ); // update the score
84          scoreSoundPtr->play(); // play a sound when player 1 scores
85       } // end else
86       // check if the Ball hit the bottom wall
87       else if (
88          ( position.y  - RADIUS ) <= bottomPosition.y + ( WALL_WIDTH / 2 ) &&
89          direction.y < 0 )
90       {
91          // place the Ball on the bottom wall
92          nodePtr->setPosition( position.x,
93             ( bottomPosition.y + ( WALL_WIDTH / 2 ) + RADIUS ), position.z );
94          reverseVerticalDirection(); // make the Ball start moving up
95       } // end else
96       // check if the Ball hit the top wall
97       else if (
98          ( position.y + RADIUS ) >= topPosition.y - ( WALL_WIDTH / 2 ) &&
99          direction.y > 0 )
100      {
101         // place the Ball on the top wall
102         nodePtr->setPosition( position.x,
103            ( topPosition.y - ( WALL_WIDTH / 2 ) - RADIUS ), position.z );
104         reverseVerticalDirection(); // make the Ball start moving down
105      } // end else
106
107      hitPaddle(); // check if the Ball hit a Paddle
108   } // end function moveBall
109
110   // reverse the Ball's horizontal direction
111   void Ball::reverseHorizontalDirection()
112   {
113      direction *= Vector3( -1, 1, 1 ); // reverse the horizontal direction
114      paddleSoundPtr->play(); // play the "paddleSound" sound effect
115   } // end function reverseHorizontalDirection
116
117   // reverse the Ball's vertical direction
```

圖 27.8　Ball 類別的成員函式定義 (續 2)

```
118 void Ball::reverseVerticalDirection()
119 {
120    direction *= Vector3( 1, -1, 1 ); // reverse the vertical direction
121    wallSoundPtr->play(); // play the "wallSound" sound effect
122 } // end function reverseVerticalDirection
123
124 // control the Ball colliding with the Paddle
125 void Ball::hitPaddle()
126 {
127    // get the position of the Paddles and the Ball
128    Vector3 leftPaddlePosition = sceneManagerPtr->
129       getSceneNode( "LeftPaddle" )->getPosition(); // left Paddle
130    Vector3 rightPaddlePosition = sceneManagerPtr->
131       getSceneNode( "RightPaddle" )->getPosition(); // right Paddle
132    Vector3 ballPosition = nodePtr->getPosition(); // the Ball's position
133
134    const int PADDLE_WIDTH = 2; // width of the Paddle
135    const int PADDLE_HEIGHT = 30; // height of the Paddle
136
137    // is the Ball in range of the left Paddle?
138    if ( ( ballPosition.x - RADIUS ) <
139       ( leftPaddlePosition.x + ( PADDLE_WIDTH / 2 ) ) )
140    {
141       // check for collision with left Paddle
142       if ( ( ballPosition.y - RADIUS ) <
143          ( leftPaddlePosition.y + ( PADDLE_HEIGHT / 2 ) ) &&
144          ( ballPosition.y + RADIUS ) >
145          ( leftPaddlePosition.y - ( PADDLE_HEIGHT / 2 ) ) )
146       {
147          reverseHorizontalDirection(); // reverse the Ball's direction
148
149          // place the Ball at the edge of the Paddle
150          nodePtr->setPosition(
151             ( leftPaddlePosition.x + ( PADDLE_WIDTH / 2 ) + RADIUS ),
152             ballPosition.y, ballPosition.z );
153       } // end if
154    } // end if
155    // is the Ball in range of the right Paddle?
156    else if ( ( ballPosition.x + RADIUS ) >
157       ( rightPaddlePosition.x - ( PADDLE_WIDTH / 2 ) ) )
158    {
159       // check for collision with right Paddle
160       if ( ( ballPosition.y - RADIUS ) <
161          ( rightPaddlePosition.y + ( PADDLE_HEIGHT / 2 ) ) &&
162          ( ballPosition.y + RADIUS ) >
163          ( rightPaddlePosition.y - ( PADDLE_HEIGHT / 2 ) ) )
164       {
165          reverseHorizontalDirection(); // reverse the Ball's direction
166
167          // place the Ball at the edge of the Paddle
168          nodePtr->setPosition(
169             ( rightPaddlePosition.x - ( PADDLE_WIDTH / 2 ) - RADIUS ),
```

圖 27.8 Ball 類別的成員函式定義 (續 3)

```
170              ballPosition.y, ballPosition.z );
171         } // end if
172      } // end else
173   } // end function hitPaddle
```

圖 27.8　　Ball 類別的成員函式定義 (續 4)

第 31 行設定材質，將 Entity 著色。引數 "ball" 是將 Ball 著色的材質名稱。Ogre 用腳本 (script) 來建立材質 (雖然我們也可以直接在程式中建立材質)。材質**腳本** (**script**，見圖 27.9) 是一個文字檔，Ogre 用它來建立材質，其副檔名爲 .material。

```
1   // ball material definition
2   material ball
3   {
4      // define one technique for rendering the Ball
5      technique
6      {
7         // render the Ball in one pass
8         pass
9         {
10           // color the Ball violet
11           ambient 0.58 0 0.827
12           diffuse 0.58 0 0.827
13           specular 0.58 0 0.827 120
14        }
15     }
16  }
```

圖 27.9　　ball 材質腳本

請注意，ball 材質的結構與 C++ 程式碼很類似，它們都用大括號將每一個段落框起來。第 2 行定義一個材質，名稱爲 ball。在材質內部，我們定義一個 technique －渲染物件的方法 (第 5-15 行)。你可以在材質中定義多個 technique，不過我們在本書中不多做這方面的討論。在每個 technique 中，會定義一個以上的 pass (第 8-14 行)。每個 pass 會定義材質渲染流程中的一個步驟。本書不討論如何使用多個 pass。我們在 pass 中定義顏色，以設定乒乓球在場景的不同光源下的顏色。第 11-13 行將 Ball 設定爲紫羅蘭色，顏色值爲 (0.58, 0, 0.827)。每一種型態的光源 (ambient、diffuse 和 specular) 之後的數字都代表顏色值。specular 的第四個數字 (120) 決定 Ball 的光澤度。它可以是任何大於零的值。數值越高，物件看起來越有光澤。

注意，"ball" 不是材質定義檔的名稱，而是檔案內部材質的名稱。一個材質檔案中可以定義多個材質。一個材質可供多個 Entity 物件使用，但是每個材質定義都必需有唯一的名稱。

現在我們已經替 Ball 建立了一個 Entity，但是它還不是場景的一部分。第 33–35 行 (圖 27.8) 將它加入場景中，讓它呈現在螢幕上。我們使用 SceneManager 來建立 **SceneNode**，它是場景圖中的節點，用來儲存場景中某個物件的資訊，包括此物件的位置、物件爲可見的或不可見的。SceneNode 可以有許多子節點，但是只能有一個父節點。在 Ogre 引擎中，引數"Ball"這個名稱將會用來參照 SceneNode。場景圖中的每個 SceneNode 都必需有唯一的名稱。呼叫 getRootSceneNode 可取得場景圖最頂層的節點。**根節點 (root node)** 是所有其他節點的父節點。當你建立根節點的子節點時，它的初始位置爲 (0, 0, 0)。第 35 行會將代表 Ball 的 Entity 附加到新增的 SceneNode 上。現在，Ball 就成爲場景的一部分，會被呈現出來。請注意，我們用來將 Ball 加入場景的所有函式，都是 SceneManager 的成員函式。這就是爲什麼 Ball 建構子要接收指向 SceneManager 的指標作爲參數，Ball 物件必須能夠存取 SceneManager，才能將它自己加入場景。

Ogre SDK 提供的球狀 mesh 半徑爲 100，我們不需要這麼大的球。第 36 行改變了附加在 SceneNodeg 上的 Entity 的大小，但是這不會影響到 Entity 建立所依據的基礎 mesh 之大小。我們可以提供縮放係數 (scaling factor) 給每個座標軸 (x，y 和 z)。我們將縮放係數 0.05 傳給這三個座標軸。由於三個軸的縮放係數都一樣，因此 mesh 會等比例地被縮放，維持原有的形狀。此函式會將球體在每個軸上的半徑乘以縮放係數，將半徑從 100 改爲 5。當你縮放 mesh 時，光源效果會有一點失眞。爲了修正這個問題，每次執行縮放時，我們會讓 Entity 計算 mesh 的新法線 (第 32 行)。[請注意：在 Ogre 1.6 (Shoggoth) 中，這個函式是不存在的，這個動作會自動執行。你必須把這一行刪掉或變成註解，這樣才能正確編譯。] 在這裡，**法線 (normal)** 指的是物件的每個小平面所面對的方向。假如小平面朝向光源，它就會比較亮。假如背對光源，就會比較暗。

函式 scale 也可以改變 Entity 的大小。差別在於 scale 是以目前的大小爲基礎來做變更，而 setScale 是以節點的原始大小爲基礎。這些函式也會以相同的係數將 SceneNode 所有的子節點做縮放。你也可以告訴父節點的每個子節點，在改變父節點的大小時，不需要改變子節點的大小，只要以 flase 值呼叫 setInheritScale 函式即可做到這點。

將球拍加入場景

Paddle 類別 (圖 27.10–27.11) 代表球拍。將球拍加入場景的方式與我們將球加入場景的方式雷同。成員函式 addToScene (圖 27.11，第 23–34 行) 使用的前五個 Ogre 函式都相同，只是引數不同。

```
1  // Paddle.h
2  // Paddle class definition (represents a paddle in the game).
3  #ifndef PADDLE_H
4  #define PADDLE_H
5
6  #include <Ogre.h> // Ogre class definitions
7  using namespace Ogre;
8
9  class Paddle
10 {
11 public:
12    // constructor
13    Paddle( SceneManager *sceneManagerPtr, String paddleName,
14       int positionX );
15    ~Paddle(); // destructor
16    void addToScene(); // add a Paddle to the scene
17    void movePaddle( const Vector3 &direction ); // move a Paddle
18
19 private:
20    SceneManager* sceneManagerPtr; // pointer to the SceneManager
21    SceneNode *nodePtr; // pointer to a SceneNode
22    String name; // name of the Paddle
23    int x; // x-coordinate of the Paddle
24 }; // end of class Paddle
25
26 #endif // PADDLE_H
```

圖 27.10　Paddle 類別定義 (代表遊戲中的球拍)

```
1  // Paddle.cpp
2  // Paddle class member-function definitions.
3  #include <Ogre.h> // Ogre class definitions
4  #include "Paddle.h" // Paddle class definition
5  using namespace Ogre;
6
7  // constructor
8  Paddle::Paddle( SceneManager *ptr, String paddleName,
9     int positionX )
10 {
11    sceneManagerPtr = ptr; // set the pointer to the SceneManager
12    name = paddleName; // set the Paddle's name
13    x = positionX; // set the Paddle's x-coordinate
14 } // end Paddle constructor
15
16 // destructor
17 Paddle::~Paddle()
18 {
19    // empty body
20 } // end Paddle default destructor
21
22 // add the Paddle to the scene
23 void Paddle::addToScene()
24 {
```

圖 27.11　Paddle 類別的成員函式定義

```
25       Entity *entityPtr = sceneManagerPtr->
26          createEntity( name, "cube.mesh" ); // create an Entity
27       entityPtr->setMaterialName( "paddle" ); // set the Paddle's material
28       entityPtr->setNormaliseNormals( true ); // fix the normals when scaled
29       nodePtr = sceneManagerPtr->getRootSceneNode()->
30          createChildSceneNode( name ); // create a SceneNode for the Paddle
31       nodePtr->attachObject( entityPtr ); // attach Paddle to the SceneNode
32       nodePtr->setScale( .02, .3, .1 ); // set the Paddle's size
33       nodePtr->setPosition( x, 0, 0 ); // set the Paddle's position
34    } // end function addToScene
35
36    // move the Paddle up and down the screen
37    void Paddle::movePaddle( const Vector3 &direction )
38    {
39       nodePtr->translate( direction ); // move the Paddle
40       if ( nodePtr->getPosition().y > 52.5 ) // top of the box
41          nodePtr->setPosition( x, 52.5, 0 ); // place Paddle at top of box
42       else if ( nodePtr->getPosition().y < -52.5 ) // bottom of the box
43
44          // place the Paddle at the bottom of the box
45          nodePtr->setPosition( x, -52.5, 0 );
46    } // end function movePaddle
```

圖 27.11 Paddle 類別的成員函式定義 (續)

我們先建立一個 Entity 來表示螢幕中的球拍 (第 25–26 行)。在這裡，我們將 name 提供給建構子做爲 Entity 的名稱。我們不能像使用 "Ball" 那樣，只使用 "Paddle"，這是因爲每個 Entity 都要有唯一的名稱，而遊戲中有兩個 Paddle。我們使用 Ogre SDK 提供的 mesh 作爲 Paddle 的模型。立方體的 mesh 位在 OgreSDK\media\models 資料夾中。兩個 Paddle 都用同樣的 material 繪成深橘色 (圖 27.12)。這個材質腳本幾乎跟之前 ball 使用的腳本是一樣的。唯一的不同點是材質的名稱 (第 2 行) 和顏色值 (第 11-13 行)。

接著我們會替 Paddle 在根節點下面建立一個 SceneNode 子節點 (圖 27.11，第 29–30 行)。我們使用提供給 Entity 建構子的名稱作爲 Node 的名稱。這是被允許的，因爲 Node 和 Entity 是兩個不同型別的物件，所以不會有名稱衝突的問題。

接下來，我們將 Entity 附加到節點上 (第 31 行) ，並且將節點縮放成適當的大小 (第 32 行) 。立方體 mesh 的大小是 100×100×100，我們將它縮放成 2×30×10，這是適合做爲球拍的大小。我們也讓 Entity 重新計算它的法線 (第 28 行)，就像我們替 Ball 做的一樣。我們使用的唯一一個新 Ogre 函式是 setPosition (第 33 行)。這個函式會將節點放在指定的座標位置 (與父節點的相對位置)。在 Ball 中，我們不需要使用這個函式，因爲我們想要將 Ball 的 SceneNode 放在(0, 0, 0)，這是每一個被附加到父節點的子節點的預設位置。我們希望 Paddle 被放在螢幕的邊緣，因此需要將它們移到那裡去。在第 33 行，x 是 Paddle 類別的資料成員，它定義 Paddle 的 x 座標。

```
1  // paddle material definition
2  material paddle
3  {
4     // define one technique for rendering a Paddle
5     technique
6     {
7        // render a Paddle in one pass
8        pass
9        {
10          // color the Paddle dark orange
11          ambient 1 0.549 0
12          diffuse 1 0.549 0
13          specular 1 0.549 0 120
14       }
15    }
16 }
```

圖 27.12　Paddle 材質腳本

將 Wall 加入場景

現在我們要將含有乒乓球和球拍的方框加入場景，我們在 Pong 類別的 createScene 函式 (圖 27.5，第 128–168 行) 中建立這個方框。我們使用 Ogre SDK 提供的同一個立方體 mesh，將方框的牆壁縮放成適合的大小，並替光源重新計算法線。我們用類似 Ball 和 Paddle 的方法將牆壁加入場景，我們利用立方體 mesh 所建立的 Entity，來代表每面牆壁。我們使用簡單的材質將每面牆壁著色爲青綠色 (圖 27.13)。這個材質腳本就跟之前的其他兩個一樣，只有名稱和顏色值不同。

```
1  // wall material definition
2  material wall
3  {
4     // define one technique for rendering a wall
5     technique
6     {
7        // render a wall in one pass
8        pass
9        {
10          // color the wall cyan
11          ambient 0 0.545 0.545
12          diffuse 0 0.545 0.545
13          specular 0 0.545 0.545 120
14       }
15    }
16 }
```

圖 27.13　Wall 材質腳本

現在，我們要設定牆壁的位置和大小。左右牆壁的 x 座標各離原點 95 個單位。上下方牆壁的 y 座標各離原點 70 個單位。然後將每個牆壁縮放到適合的大小。上方牆壁和下方牆壁的 y 軸位置相隔 140 個單位。我們讓牆壁的寬度爲 5 個單位。你可以更改這個數值，依你的喜好來決定遊戲的外觀。假如你改變這個值，你也必須改變碰撞偵測的程式碼，我們稍後會討論碰撞的邏輯。考慮到左右牆必須擺放在上下牆之間，因此它們的長度必須是 145 個單位 (140 加上每個牆壁寬度的一半)。如此一來，左右牆的 x 軸縮放係數應該是 1.45。左右牆的寬度也是 5 個單位，彼此之間的 x 座標距離爲 185 單位。因爲上下牆要擺在左右牆之間，因此它們的長度必須是 195 個單位，y 軸縮放係數爲 1.95。

將文字加入

我們利用 Ogre Overlay 以文字顯示出遊戲的分數。**`Overlay`** 指的是你想要呈現在場景 3D 元素的最上方的任何東西。在本書中，我們只使用 `Overlay` 來顯示文字。`Overlay` 的定義儲存在副檔名爲 `.overlay` 的腳本中。

`Overlay` 是由 `OverlayElement` 組成的。`Overlay` 中的第一個元素必須是 `OverlayContainer`。**`OverlayContainer`** 可以存放任何型別的 `OverlayElement`。**`TextAreaOverlayElement`** 可以存放文字，顯示在螢幕上。`Overlay` 中的每一個物件都有三個主要屬性：度量模式 (metrics mode)、位置和大小。位置 (position) 是由物件的左上角決定的，相對於物件的父 `OverlayContainer`。大小 (size) 則由寬度和高度來決定。**度量模式 (metrics mode)** 決定物件如何放置和度量大小。**像素模式 (Pixel mode)** 會以像素爲基礎來宣告寬度和高度。**相對模式 (Relative mode)** 會以相對於父 `OverlayContainer` (假如是最外圍的 `OverlayContainer`，就以視窗) 的位置與大小，來決定物件的位置和大小在相對模式中，大小和位置的數值範圍是從 0.0 到 1.0，你可以將它視爲父 `OverlayElement` 大小的某個百分比。假如你將某個元件定位在 (0.0, 0.0)，它會位在父元件的左上角，(0.5, 0.0) 則位在父元件頂端百分之 50 的位置上。

我們會建立一個 `Overlay` 以顯示分數 (圖 27.14) 。第 2 行建立 `Overlay` 的名稱 `Score`。一個 `overlay` 檔案可以儲存數個 `Overlay` 定義。Ogre 會以 `overlay` 的名稱而不是檔名來參照每個 `Overlay`。`Overlay` 的 **z-order** (第 5 行)是用來決定 `Overlay` 層疊呈現的順序。當我們使用多個 `Overlay` 時，z-order 較高的 `Overlay` 會出現在 z-order 較低的 `Overlay` 上方。第 8-58 行會建立一個 **`PanelOverlayElement`**，其中包含兩個 `TextAreaOverlayElement`。`OverlayContainer` 位在螢幕的左上角 (第 13-14 行)，寬度橫越整個視窗 (第 17 行)。這個容器佔視窗高度的 10% (第 18 行)。第一個 `TextAreaOverlayElement` (第 21-38 行) 位在容器頂端，距離左邊 5%的位置上，寬度爲容器的一半，高度與容器相同 (第 28-31 行)。另一個 `TextAreaOverlayElement`

(第 40-57 行) 位在容器頂端，由 69% 的位置一直延伸到最末端。`TextAreaOverlayElement` 也會宣告使用的字型 (定義在 `PongResources` 資料夾的 `.fontdef` 腳本檔案中)、字元高度、字型顏色 (再次注意到 `colour` 的英式拼音)，以及標題 (第 34-37 行以及第 53-56 行) 。

圖 27.15 是用來定義 `BlueBold` 字型的 `.fontdef` 檔案。第 2 行替字型指定 Ogre 所參照的名稱。第 5 行告訴 Ogre 字型的類型爲何。`True type` 是常用的字型檔案格式 (`.ttf` 檔)。`source` (第 8 行) 指定字型檔。我們將 `.ttf` 檔與 `.fontdef` 檔放在同一個資料夾。假如你把兩個檔案放在不同的位置，則必須在第 8 行指定 `.ttf` 的路徑，或是將包含 `.ttf` 檔的資料夾加入資源位置 (在第 27.4.8 節討論)。

```
1  // An Overlay to display the score
2  Score
3  {
4     // set a high z-order, displays on top of anything with lower z-order
5     zorder 500
6
7     // create a PanelOverlayElement container to hold the text areas
8     container Panel(ScorePanel)
9     {
10       // use relative metrics mode to position this container at the
11       // top-left corner of the screen
12       metrics_mode relative
13       left 0.0
14       top 0.0
15
16       // make it 1/10 the height and the full width of the screen
17       width 1.0
18       height .1
19
20       // create a TextAreaOverlayElement to display player 1's score
21       element TextArea(left)
22       {
23          // position and size the element relative to the container
24          metrics_mode relative
25
26          // position it at the top of the container 5% from the left and
27          // make it the same height and half as long as the container
28          left 0.05
29          top 0.0
30          width 0.5
31          height 1.0
32
33          // set font used for caption and set the size and color
34          font_name BlueBold
35          char_height .05
36          colour 1.0 0 0
37          caption Player 1 Score: 0
```

圖 27.14　用來顯示分數的 `Overlay` 腳本

```
38          }
39          // create a TextAreaOverlayElement to display player 2's score
40          element TextArea(right)
41          {
42             // position and size the element relative to the container
43             metrics_mode relative
44
45             // position it at the top of the container 69% from the left and
46             // make it the same height as the container, stretch to the end
47             left 0.69
48             top 0.0
49             width 0.5
50             height 1.0
51
52             // set font used for caption and set the size and color
53             font_name BlueBold
54             char_height 0.05
55             colour 1.0 0 0
56             caption Player 2 Score: 0
57          }
58       }
59    }
```

圖 27.14　用來顯示分數的 Overlay 腳本 (續)

Pong 類別的第 115–117 行會在螢幕上顯示分數 (圖 27.5)。我們使用 OverlayManager 類別的 static 成員函式 getSingleton 取得指向 OverlayManager 物件的指標。我們使用該物件取得指向分數 Overlay 的指標，然後呼叫 show 函式，將它顯示在螢幕上。當玩家得分時，我們會更新 Overlay 中的文字，以反應這個改變 (第 192–210 行)。首先，我們會建立新的文字。接著我們會從 OverlayManager 取得適當的 TextAreaOverlayElement 指標，並使用 TextAreaOverlayElement 的成員函式 setCaption 來替換文字。

```
1  // define the BlueBold font
2  BlueBold
3  {
4     // define the font type
5     type truetype
6
7     // set the source file for the font
8     source bluebold.ttf
9
10    // set the font size
11    size 16
12
13    // set the font resolution (96 is standard)
14    resolution 96
15 }
```

圖 27.15　BlueBold 字型定義腳本

27.4.4　動畫與計時器

既然我們已經知道要如何在螢幕上繪製一顆球，則將它變爲動畫，讓它們在螢幕四處移動，就是很簡單直接的事情了。`moveBall` 函式 (圖 27.8 第 54–108 行) 讓 `Ball` 在螢幕中四處移動。在大部分的乒乓遊戲中，球可能會用許多不同的角度來移動。然而，由於我們才剛開始學習 Ogre，我們希望事情能越簡單越好。因此，在我們的乒乓遊戲中，球只會有四種可能的移動方向。右下、右上、左下、左上——全部都與此程式的 x 軸及 y 軸成 45 度角。

第 56 行是實際讓 `Ball` 移動的程式碼，函式的其他部分是用來控制場景中各種物件的碰撞，我們會在稍後討論碰撞。`translate` 函式接收一個 `Vector3` 物件作爲引數，`Vector3` 是 Ogre 所定義的一個三維向量型別。這個向量代表 `Ball` 移動的方向和距離。我們將 `Ball` 的方向乘以它要移動的距離 (speed×time) 以決定最終的向量，並將它當作引數傳給 `translate` 函式。`speed` 參數是球每秒所能移動的單位數量。`time` 參數是 `Ball` 上一次移動到現在所經過的秒數。稍後我們會看到 `time` 是從哪裡得到的。根據預設，`SceneNode` 的移動是在**父空間 (parent space)** 中完成的。也就是說，節點的移動是與父節點的位置和方向有關的。我們也可以對 `translate` 函式加入參數 (TS_LOCAL 或是 TS_WORLD)，就可以在世界空間或本地空間執行節點的移動。在**世界空間 (world space)** 移動是以場景的原點 (0, 0, 0) 作爲基準。在**本地空間 (local space)** 移動則是以節點的原點 (無論節點的位置和它面對的方向爲何) 爲基準。[2]

我們要讓 `Ball` 持續地在螢幕上移動，因此每次新的影格呈現時，都要呼叫 `moveBall` 函式。圖 27.4 定義 Pong 類別，這是我們遊戲的主類別，繼承自 Ogre 的 `FrameListener` 類別。**`FrameListener`** 類別會處理 `Ogre::FrameEvent`。每當影格開始或結束時，都會產生一個 **`FrameEvent`**。每個 `FrameListener` 都有兩個函式：`frameStarted` 和 `frameEnded` (第 29–30 行)。這些函式都會回傳布林值。Ogre 會持續呈現影格，直到其中一個函式回傳 `false` 值。我們使用 `frameStarted` 函式 (圖 27.5 第 271–295 行) 來控制 `Ball` 的動畫，特別是第 278 行。Ogre 會在呈現每個影格之前呼叫這個函式。在每個影格呈現之前，`frameStarted` 函式會呼叫類別 `Ball` 的成員函式 `moveBall`，讓 `Ball` 持續在螢幕上移動。就像之前所討論的，控制動畫速度對動作的流暢是很重要的。在不同的機器上，影格速率 (場景重繪的速度) 可能會有很大的不同，

---

[2] Junker, Gregory, *Pro OGRE 3D Programming*.New York:Springer-Verlag, 2006, pp. 82—89.

因此 `Ball` 可能會用不同的速度移動。因此我們每隔一小段時間將 `FrameEvent` 的成員函式 `timeSinceLastFrame` 傳送給 `moveBall` 函式。我們將這個時間乘以 `Ball` 的速率，以決定 `Ball` 應該在螢幕上移動的距離。這是一個使用計時器控制動畫的範例。

27.4.5　使用者輸入

現在我們要討論如何使用 `Paddle` 類別的 `movePaddle` 函式來上下移動球拍 (圖 27.11，第 37–46 行)。我們再次利用 `SceneNode` 的 `translate` 函式 (第 39 行) 來移動 `Paddle`。這次我們不以時間爲基礎來移動 `Paddle`，我們依據使用者的鍵盤輸入來移動它。使用者會按下鍵盤按鍵，指示球拍往上或往下移動，然後 `Paddle` 會根據這些指示來移動。這些指示會以 `Vector3` 的形式傳遞給 `movePaddle`。

Ogre 不直接支援來自鍵盤、滑鼠或搖桿的使用者輸入。Ogre SDK 利用**物件導向輸入系統 (Object Oriented Input System，OIS)** 來處理使用者的輸入。你不一定要用 OIS 配合 Ogre 處理輸入，但是這不失爲一個好選擇。

我們必須建立一個 `InputManager`、一個 `Keyboard` 和一個 `KeyListener` 來處理使用者的輸入，並控制對 `movePaddle` 的呼叫。**`InputManager`** 是用來建立各種輸入裝置。我們在 `Pong` 類別的建構子 (圖 27.5 第 71 行) 中建立了 `InputManager`。我們必須提供一個視窗，才能建立 `InputManager` (第 62–68 行)。

我們建立一個 **`Keyboard`** 物件，用來代表實際的鍵盤。爲了收集 `KeyEvent`，我們必須呼叫 `Keyboard` 類別的 `capture` 函式。我們必須重複地呼叫這個函式，因此將它放在 `frameStarted` 函式中，每次影格開始時就會被呼叫它。`Pong` 類別繼承自 **`KeyListener`**，這是一個可以處理鍵盤輸入的 OIS 類別。我們用 `Keyboard` 物件將它註冊 (第 74 行)，用來接收 **`KeyEvent`**，表示使用者按下了鍵盤。`KeyListener` 定義兩個成員函式 (圖 27.4 第 25–26 行)，我們只使用到一個 (第 25 行)。然而，我們還是要實作另外一個，因爲它們都被宣告爲 `KeyListener` 類別的純粹 `virtual` 函式。

鍵盤處理常式的實作列在第 213–256 行。當偵測到鍵盤被按下時，`Keyboard` 會傳送 `KeyEvent` 到這個成員函式。OIS 替所有的案件都定義了列舉型別，我們可以用它來決定按下的是哪一個鍵。`switch` 敘述 (第 219–242) 只針對特定的按鍵予以回應。我們從 `KeyEvent` 中擷取按鍵的列舉型別，將它傳送到 `switch` 敘述 (第 219 行) 中。如果使用者按下 A 鍵或 Z 鍵，則位於左側的球拍分別會往上或往下移動。同樣的，如果使用者按下上鍵或下鍵，右側的球拍也應該依相應的方向移動。傳遞給 `movePaddle` 的方向會定義爲 `const Vector3` (第 22–23 行)。

我們讓使用者可以按下「P」鍵來暫停遊戲 (第 236–241 行)，將 Pong 的資料成員 pause 設定爲 true 值。當 pause 等於 true 時，if 敘述 (第 216 行) 會跳過控制球拍移動的 switch 敘述。當資料成員 pause 爲 true 時，也會停止 Ball 的移動 (第 275 行)。我們也使用一個 Overlay (圖 27.16) 在螢幕上顯示「Game Paused」的字樣。當玩家按下「R」鍵時，遊戲就會恢復執行。

```
1  // An Overlay to display "Game Paused" when the player pauses the game
2  PauseOverlay
3  {
4     // set a high z-order, displays on top of anything with lower z-order
5     zorder 500
6
7     // create a PanelOverlayElement container to hold the text area
8     container Panel(Pause)
9     {
10       // use relative metrics mode to position and size this container
11       metrics_mode relative
12
13       // place the container at the top-left corner of the window
14       left 0.0
15       top 0.0
16
17       // make the container the same size as the window
18       width 1.0
19       height 1.0
20
21       // create a TextAreaOverlayElement to display the text
22       element TextArea(pauseText)
23       {
24          // position and size the element relative to its container
25          metrics_mode relative
26
27          // center it vertically in the container
28          vert_align center
29
30          // put the left corner 4/10 from the left of the container and
31          // make it 2/10 the width of the container and 1/10 the height
32          left 0.4
33          width 0.2
34          height 0.1
35
36          // set the font used for caption and set the size and color
37          font_name BlueBold
38          char_height 0.05
39          colour 0 0 0
40          caption Game Paused
41       }
42    }
43 }
```

圖 27.16　在玩家暫停遊戲時，用來顯示「Game Paused」的 Overlay 腳本

假如使用者按下 Esc 鍵，程式會將 `quit` 資料成員設爲 `true` 以離開遊戲 (圖 27.5 第 221–223 行)。還記得 Ogre 會持續呈現影格，直到 `frameStarted` 或是 `frameEnded` 函式回傳 `false`。這兩個函式都回傳 `!quit`，因此假如我們將 `quit` 設定爲 `true`，則函式會回傳 `false`，並讓 Ogre 關閉程式。假如你沒有使用 Esc 鍵離開程式，這個程式就不會眞的關閉，它會在背景繼續執行。請記得使用 Esc 鍵。

27.4.6 碰撞偵測

當 `Ball` 在螢幕中彈跳時，會和許多物件產生碰撞。我們必須偵測這些碰撞，讓 `Ball` 正確地與環境互動。圖 27.8 的第 60-105 行會控制遊戲區域中 `Ball` 和牆壁之間的碰撞。呼叫 `SceneNode` 的成員函式 `getPosition` 會回傳 `Vector3`，代表此節點與其父節點的相對位置。因爲我們的節點都是根節點 (0,0,0) 的直接子節點，所以回傳的位置都是以原點爲基準的。成員函式 `_getDerivedPosition` 則會回傳節點與原點的相對位置，無論它的父節點位置爲何。

你可以將節點名稱傳遞給 `SceneManager` 的成員函式 `getSceneNode`，以取得場景圖中的任何節點。第 60–67 行取得四個牆的節點，並利用它們的位置來偵測 `Ball` 的碰撞。假如 `Ball` 的 `x` 座標 (減去半徑) 小於等於左牆的 `x` 座標 (加上半個牆的寬度)，就表示 `Ball` 碰撞到左邊的牆壁。一旦發生了碰撞，就應該採取適當的行動。`Ball` 會被放在螢幕的中央，開始下一回合的比賽 (第 74 行)。程式會呼叫 `Pong` 類別的成員函式 `updateScore`，讓 2 號玩家得到一分 (第 75 行)。我們將 `updateScore` 定義爲 `static` 的，因此我們可以從類別 `Ball` 呼叫它，不需要建立 `Pong` 類別的實體物件。最後，程式會播放一段音效，表示玩家贏得一分 (第 76 行)，我們會在 27.4.7 節解釋這個函式呼叫。我們用同樣的流程來判斷 `Ball` 是否擊中右邊的牆壁。我們會檢查 `Ball` 的 x 座標和右牆的 `x` 座標，假如 `Ball` 擊中右牆，則需要採取適當的行動。圖 27.17 顯示 1 號玩家贏得一分。`Ball` 並沒有眞的進入牆壁，這是由 3D 繪圖造成的幻覺。

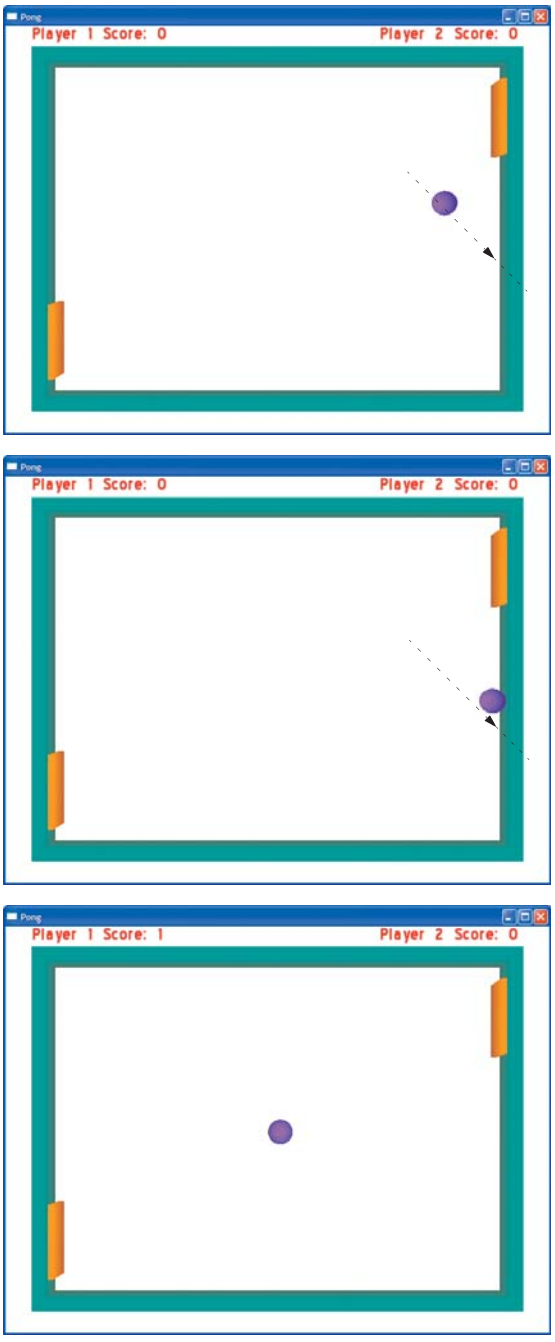

圖 27.17　1 號玩家贏得一分

接著我們要偵測上方和下方的牆壁。程式使用相同的碰撞邏輯。假如 Ball 的 y 座標 (加上或減去半徑，視它擊中哪一面牆而定) 在移動之後會穿過牆壁內層的 y 座標 (實際上是不可能的，它們都是堅硬的物體)，就表示乒乓球碰撞到上方或下方的牆。爲了要避免 Ball 與牆壁重疊，在碰撞之後，我們將球放到牆壁的邊緣 (第 92–93 行以及第 102–103 行)。技術上，改變 Ball 在固定時間間隔內的移動距離，是違反了 Ball 的物理特性。我們應該要計算球體在擊中牆壁之後的移動距離和方向，並將球繪製在那一個點，才是正確的。但是爲了保持程式碼的簡單易懂，我們並沒有處理這個問題。場景重繪的速度很快，因此 Ball 在每個影格移動的距離是很小的。對 Ball 移動的影響幾乎難以察覺。圖 27.18 顯示 Ball 從上方牆壁彈回的情景。

在 moveBall 的最後，我們呼叫函式 hitPaddle (第 125–173 行)，檢查 Ball 是否碰撞到球拍，以採取適當的動作。第 128–129 行取得左球拍的節點，並回傳節點的位置。第 130–131 行對右球拍執行同樣的動作。我們利用這些位置來偵測球拍和球之間是否發生碰撞，類似用來偵測牆壁碰撞的邏輯。我們首先測試 Ball 的 x 座標是否超過 Paddle 的 x 座標。也測試 Ball 是否位在 Paddle 的 y 座標內。圖 27.19 顯示 Ball 從 Paddle 上彈回的情景。

請注意改變乒乓球行進方向的那條線。第 113 行讓往右移的 Ball 改爲往左移動，讓往左移的 Ball 改爲往右移動。第 120 行讓往下移的 Ball 改爲往上移動，往上移的球改爲往下移動。爲什麼我們可以這樣做？Ball 的方向是由 Vector3 所決定的。每個值都代表 x、y、z 軸上的一段距離。正的 x 值表示 Ball 會沿著 x 軸往右移動，負值則代表 Ball 會往左移動。假如 Ball 往右移動，將它的 x 值乘以 -1，就會改變正副號，進而反轉方向。垂直方向也是一樣的道理。

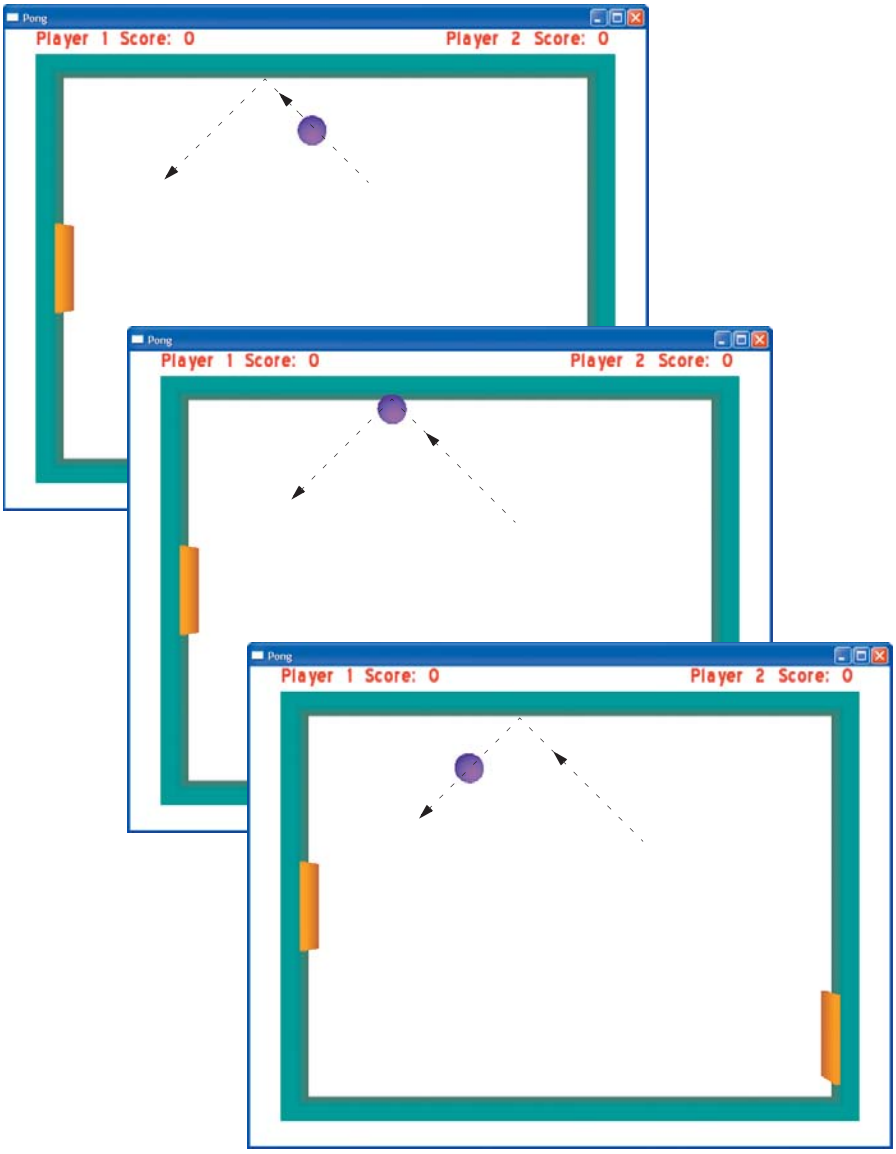

圖 27.18　`Ball` 從上方的牆壁彈回來

我們的遊戲所使用的碰撞是很簡單的範例，我們讓邏輯保持簡單。有一些函式庫是專門處理碰撞和物理模擬的，像是 Open Dynamics Engine (ODE, `www.ode.org/`)，Bullet (`www.continuousphysics.com/Bullet/`)，Newton Game

Dynamics (`www.newtondynamics.com/`) 以及 PhysX (`www.ageia.com/`)。你可以在 Ogre Community 的 Add-on 頁面找到這些函式庫的 Ogre 連結。

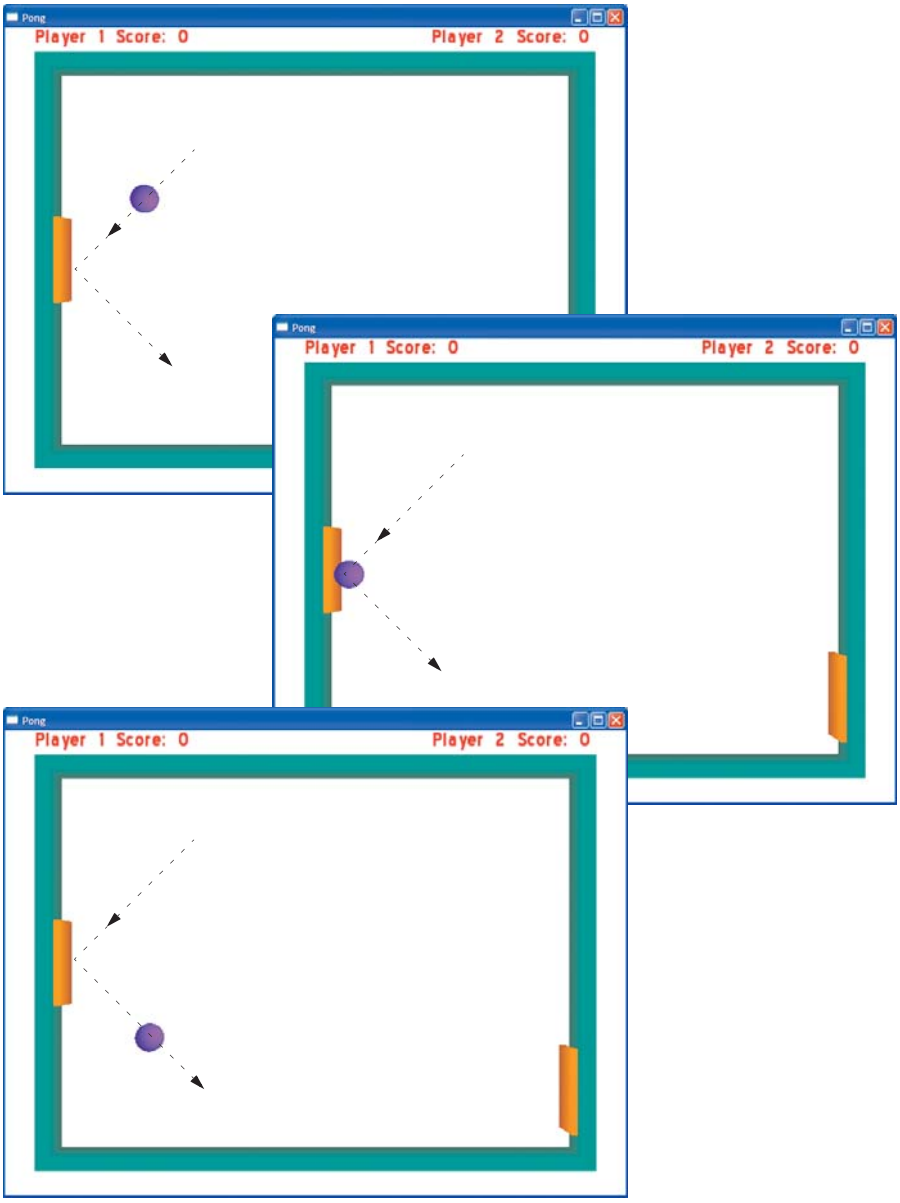

圖 27.19　`Ball` 從左 `Paddle` 彈回的情景

27.4.7　音效

接著我們要討論如何匯入音效，並在 Ogre 程式中播放音效檔，讓我們的乒乓遊戲更加豐富。當 `Ball` 碰到牆壁時，我們會播放「boing」的聲音，當 `Ball` 碰到球拍時，我們會播放另一個「boing」。當某個玩家贏得分數時，我們會播放歡呼聲。

我們利用 OgreAL 將音效加入遊戲之中。**OgreAL** 包裝了 OpenAL 音效函式庫，OgreAL 是由 Ogre 社群的成員之一，Casey Borders (`www.mooproductions.org`) 所開發的。OpenAL 函式庫是由 Creative Labs，`developer.creative.com` 所維護的。這個包裝器能讓我們將音效附加到場景節點上，藉此將音效功能整合到 Ogre 程式碼中。因爲我們播放的音效都跟 `Ball` 有關，所以我們將 OgreAL 程式碼放到 `Ball` 類別中。OgreAL 函式匯入以及播放音效的方式類似我們匯入和顯示 Ogre 模型的方式。

我們將 `OgreAL.h` 標頭檔含括進來。OgreAL 利用 **`SoundManager`** 類別以及 Ogre 的資源管理機制來管理音效。我們在 `Ball` 建構子中建立 `SoundManager` (圖 27.8，第 14 行)。程式中只能有一個 `SoundManager`。我們用 `SoundManager` 來建立 **Sound** 的實體物件，這是一個內含音效資料的 OgreAL 物件。我們建立了三個 `Sound`，並將它們附加到節點上 (第 39–50 行)。函式 `createSound` 會接收三個參數，第一個是 `Ogre::String`，這是 `Sound` 的名稱。第二個參數是 `Sound` 要播放的音效檔的名稱。第三個是布林值，用來判斷 `Sound` 是否應該循環播放。假如傳入的是 `false` 值，程式會將 `Sound` 播放一次，然後停止。假如傳入的是 `true` 值，程式會循環播放音效，直到我們將它停止。我們以將 `Entity` 附加到節點上的同樣方法，在 `attachObject` 函式中將 `Sound` 附加到節點上。

我們建立的第一個 `Sound` (第 39–40 行) 會在 `Ball` 碰到上方或下方的牆壁時播放音效。我們將它附加到 `Ball` 的節點上。OpenAL 支援 3D 音效，因此當 `Sound` 附加到 `Ball` 時，會根據 `Ball` 的位置播放音效。我們的場景比較小，所以你可能不會注意到 `Sound` 是以立體聲道播放的，但是如果你仔細聆聽，就會聽出一些不同。因爲我們將 `play` 函式 (第 121 行) 的呼叫放在將 `Ball` 朝垂直方向反轉的函式中，因此每當 `Ball` 的垂直方向被反轉時，就會發出「boing」的聲音，也就是在乒乓球打到上下牆的時候。

第二個聲音也是以同樣的方式來建立的，然後也附加到 `Ball` 的節點上。當 `Ball` 碰到 `Paddle` 時，就會播放這個聲音，我們在 `reverseHorizontalDirection` 中播放這個 `Sound` (第 114 行)，原因跟我們在 `reverseVerticalDirection` 播放的另一個音效相同。

第三個 Sound 會在贏得分數時播放。這個 Sound 沒有特定的播放地點，我們把它直接附加到場景的根節點，也就是原點。每當玩家贏得分數時，我們就在 moveBall 函式中播放這個聲音 (第 76 和 84 行)。

關於 OgreAL 的 Sound，有一些應該注意的事項。跟 Entity 物件和 Node 一樣，每一個 Sound 都應該有唯一的名稱。在重新播放之前，應該先把 Sound 的播放結束。

27.4.8 資源

如同之前提到的，Ogre 使用腳本來建置 Material、Overlay 和其他本書沒有提到的進階功能。Ogre 也使用 .mesh 檔來表示 3D 物件。OgreAL 則使用音效檔。在使用這些資源之前，應該先載入它們。假如你使用尚未載入的資源，Ogre 會拋出執行期例外。我們使用 **ResourceGroupManager** 來管理遊戲的資源。我們首先告訴 ResourceGroupManager 要去哪裡尋找資源，以將這些資源載入遊戲。addResourceLocation 函式 (圖 27.5 第 79–80 行) 會接收三個 Ogre::String 引數。第一個是資源的位置。我們將所有的資源都放在範例資料夾的 resources 資料夾中。一般來說，你應該將資源分門別類，放在不同的資料夾中，像是 material、model 和 overlay。但是爲了簡化程式，我們將所需的資源全部放在一個資料夾中。第二個引數是資源的檔案類型。第三個則是檔案所屬的資源群組。我們會將這些檔案放在 "Pong" 群組中。這樣就會載入剛才加入的位置中的資源 (第 81 行)。

27.4.9 測試乒乓遊戲

最後一個步驟是撰寫 main 函式 (圖 27.20)。Ogre 支援各種平台，因此你應該儘量避免撰寫屬於特定平台的程式碼。前置處理的 if else 包裝器 (第 8–17 行) 會判斷程式是否在 Windows 平台上執行。假如是，它會將 windows.h 含括進來，並定義 WinMain 函式。假如不是，則它會定義一般的 main 函式。這樣就可以在不變更程式碼的情況下，在各種不同的平台上執行這個遊戲。你可能沒有看過 Windows 特定的程式碼。前置處理器指令將 windows.h 標頭檔含括進來，讓程式能夠存取 Windows API 以執行程式。WIN32_LEAN_AND_MEAN 的定義 (第 9 行) 會排除 windows 標頭檔中很少使用的標頭。這樣可以增快我們程式的編譯時間。

```
1  // PongMain.cpp
2  // Driver program for the game of Pong
3  #include <iostream>
4  #include "Pong.h" // Pong class definition
5  using namespace std;
6
7  // If running on Windows, include windows.h and define WinMain function
8  #if OGRE_PLATFORM==PLATFORM_WIN32 || OGRE_PLATFORM==OGRE_PLATFORM_WIN32
9  #define WIN32_LEAN_AND_MEAN
10 #include "windows.h"
11
12 int WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine, INT )
13
14 // If not, define normal main function
15 #else
16 int main()
17 #endif
18 {
19    try
20    {
21       Pong game; // create a Pong object
22       game.run(); // start the Pong game
23    } // end try
24    catch ( runtime_error &error )
25    {
26 #if OGRE_PLATFORM==PLATFORM_WIN32 || OGRE_PLATFORM==OGRE_PLATFORM_WIN32
27       MessageBoxA( NULL, error.what(), "Exception Thrown!",
28          MB_OK | MB_ICONERROR | MB_TASKMODAL );
29 #else
30       cerr << "Exception Thrown: " << error.what() << endl;
31 #endif
32    } // end catch
33 } // end main
```

圖 27.20　乒乓遊戲的測試程式

`main` 函式會在 `try` 區塊中建立 `Pong` 物件 (第 19-23 行)。還記得，`Pong` 建構子會在使用者取消 Ogre 設定對話框時，拋出一個例外。假如使用者按下對話框中的 OK，程式就會建立 `Pong` 物件，然後呼叫 `run` 成員函式 (第 22 行)。`Pong` 類別的成員函式 `run` (圖 27.5 第 172– 176 行) 會先建立遊戲場景 (第 174 行)，接著呼叫 `Root` 類別的 `startRendering` 成員函式 (第 175 行)，持續地繪製場景，直到 `frameStarted` 或是 `frameEnded` 函式回傳 `false`。

27.5 總結

在本章中，你學到了使用 Ogre 開發電腦遊戲的基本技巧。我們討論了繪圖、模型、光源和顏色的基本概念。你學到了如何使用免費的繪圖引擎來建立 3D 遊戲。我們介紹了如何使用 `SceneManager` 來建立和管理場景。你學到了如何使用 `Camera` 檢視場景，也討論了如何使用 OIS 來回應使用者的鍵盤輸入。我們介紹了如何以固定速度移動物件，也介紹了碰撞偵測的基本概念，解釋它對遊戲設計的重要性。你學到如何使用 Overlay 在螢幕上顯示訊息。我們示範了 Ogre 如何使用腳本來管理材質和 Overlay，這樣你在每次改變它們時就不用重新編譯。我們也示範了如何使用 OpenAL 的 OgreAL 包裝器將音效加入遊戲中。

本章只是一個簡單的介紹。我們使用了簡單的範例 `Pong` 來做示範。將它當作一個基礎，練習你自己的版本。去找到你自己想用的音效，增加一些新的功能，並發掘 Ogre 的其他功能，創造一些酷炫的效果。用你自己的方式開發這個遊戲。遊戲程式設計是個有關創造力的產業。

27.6 Ogre 網路資源

`www.ogre3d.org/`

這是 Ogre 的首頁。你可以在這裡找到最新的 Ogre 資訊，下載 Ogre 或是 Ogre 相關工具，瀏覽說明文件或是尋找使用 Ogre 的專案。

`www.ogre3d.org/index.php?option=com_content&task=view&id=411&Itemid=131`

Prebuilt SDK 的下載頁面。你可以找到適用於 Code::Blocks + MingGW C++ Toolbox、Visual C++ .Net 2003 和 Visual C++ .Net 2005 (必需安裝 Service Pack 1) 的 SDK 下載點。

`www.ogre3d.org/index.php?option=com_content&task=view&id=412&Itemid=132`

Ogre 原始碼的下載頁面。你可以找到適用於 Windows、Linux 和 Mac OS X 的原始碼。請一併下載你的平台所需的第三方相依套件。此網站亦提供如何以原始檔建立 Ogre 的教學指南連結。

`www.ogre3d.org/index.php?option=com_content&task=view&id=415&Itemid=144`

說明如何從 CVS 目錄取得 Ogre 原始碼。

www.ogre3d.org/wiki/index.php/Installing_An_SDK

Ogre SDK 在 Windows with Visual C++、Code::Blocks + MinGW、Code::Blocks + MinGW + STLPort、Eclipse + MinGW + STLPort、GCC & Make/Any IDE 以及 Linux、Debian、Gentoo、Fedora、Ubuntu、Mac OS X 等平台上的安裝指南。

www.ogre3d.org/wiki/index.php/Building_From_Source

在 Windows 平台上使用 Visual C++、Visual C++ Toolkit 2003 & Code::Blocks 和 GCC、在 Linux 上使用 GCC & Make、Debian、Fedora、Gentoo、Ubuntu/Kubuntu，在 Mac OS X 使用 Xcode 編譯 Ogre 原始碼的指南。

www.ogre3d.org/wiki/index.php/BuildFAQ

編譯原始碼時常見問題的解決方法。其中包括無法找到檔案、無法解析的外部符號以及其他的型別錯誤。

www.ogre3d.org/wiki/index.php/SettingUpAnApplication

在 Visual C++、Code::Blocks、GCC、Autotools、Scons、Eclipse、Anjuta IDE、KDevelop IDEGuide 上設定 Ogre 程式專案的教學指南。

www.ogre3d.org/phpBB2addons/viewtopic.php?t=3293

OgreAL 下載和安裝指南。

developer.creative.com/landing.asp?cat=1&sbcat=31&top=38

OpenAL 下載和安裝連結。

www.openal.org/downloads.html

OpenAL 下載頁面。

www.wreckedgames.com/wiki/index.php/WreckedLibs:OIS

物件導向輸入系統 (Object Oriented Input System，OIS) 的維基頁面，包括 OIS 手冊和 API 參考文件的連結。

www.tayloredmktg.com/rgb/

顏色表，提供十六進位和十進位的 RGB 值。以一般的顏色範圍來做分類 (例如：灰色、藍色、綠色、橘色) 。

www.htmlcenter.com/tutorials/tutorials.cfm/89/General/

顏色表，提供 RGB 和十六進位顏色值。

教學文件

www.ogre3d.org/wiki/index.php/Ogre_Tutorials

Ogre 教學網頁。基礎和進階的教學文件，主題包括 Ogre 簡介、FrameListeners、使用多個 SceneManagers，以及內容建置等等。

`www.blender.org/tutorials-help/`

Blender 教學文件。

`en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro`

「Blender 3D:Noob to Pro」是一本引導 Blender 新手學習 3D 模型建置的維基書，教你如何運用模型、光源、渲染、動畫、粒子與柔體，以及 python scripting 和進階動畫的教學。

`www.cegui.org.uk/wiki/index.php/Tutorials`

Ogre 所支援的 Crazy Eddie's GUI System (CEGUI) 的教學文件。

工具

`www.ogre3d.org/index.php?option=com_content&task=view&id=413&Itemid=133`

下載 Blender、Maya、Softimage XSI 和 3DS Max 的模型匯出工具。

`usa.autodesk.com/adsk/servlet/index?siteID=123112&id=7639525`

Autodesk Maya 個人學習版網頁。這是 Maya 的免費版本。

`www.softimage.com/downloads/default.aspx`

SoftImage XSI 下載頁面。30 天免費試用版。

`www.blender.org/download/get-blender/`

Blender 下載頁面。

程式碼範例

`www.ogre3d.org/wiki/index.php/CodeSnippits#HOWTO`

Ogre Cookbook，包含各種與幾何、渲染、材質、紋理、動畫、輸入 GUI 和音效有關的程式碼範例與說明。

`www.ogre3d.org/phpBB2/viewtopic.php?t=27326`

Asteroid Wars。這是一個以 Ogre 撰寫的遊戲。可以取得原始碼。

`www.ogre3d.org/phpBB2/viewtopic.php?t=27806`

五個以 Ogre 撰寫的遊戲。都可以取得原始碼。

書籍

www.amazon.com/Pro-OGRE-3D-Programming/dp/1590597109/ref=pd_bbs_sr_1/1022583408-2260151?ie=UTF8&s=books&qid=1173888297&sr=1-1
Pro OGRE 3D Programming，作者爲 Gregory Junker。

論壇

www.ogre3d.org/phpBB2/viewtopic.php?t=5706
這個論壇的討論主題是如何在 Debian GNU/Linux 上安裝 Ogre。

www.ogre3d.org/phpBB2/viewforum.php?f=2
Ogre 的支援論壇。假如你在使用 ogre 時遭遇任何困難，可以到這裡來尋求 Ogre 使用者的協助。

www.ogre3d.org/phpBB2/
Ogre 站台上的許多論壇，包括問題支援、Ogre 使用實務、內容建置、程式設計基礎等等。

www.ogre3d.org/phpBB2addons/
Ogre 附加元件論壇。討論常用的 Ogre 附加元件，包括 OgreAL、OgreODE、NxOgre、PyOgre 等等。

www.ogre3d.org/phpBB2addons/viewforum.php?f=10
Ogre 站台上的 OgreAL 論壇，包括安裝和使用 OgreAL 的資訊，你可以在這裡得到很好的支援。

www.wreckedgames.com/forum/viewforum.php?f=6&sid=dc5f903554a80ac5194213329f5e46e4
OIS 論壇，幫助你使用 OIS。

摘要

27.3　遊戲程式設計基礎

- 3D 繪圖引擎隱藏了沉悶複雜的繪圖 API 程式設計工作。
- Ogre 支援 Direct3D 和 OpenGL 繪圖 API，可以在 Windows、Linux 和 Mac 平台上執行。
- Ogre 是一個純粹的繪圖引擎。Ogre 社群建立了許多附加元件，讓使用者可以將 Ogre 與其他函式庫整合在一起，以支援這些功能。

- 3D 模型是一個物件的電腦表示法，可以在螢幕上繪製出來。
- 材質 (Materials) 可用來決定物件的外觀，這是藉由設定光源屬性、顏色和紋理來完成的。
- 紋理 (texture) 是用來包裝在模型外表的圖形。
- 顏色 (Colors) 是由紅、綠和藍光的強度所決定的，alpha 通道 (alpha channel) 可用來表示透明度。數值範圍爲 0 到 1.0。
- 在 3D 場景中，有四種不同的光源：環境光 (ambient) 、散射光 (diffuse) 、自發光 (emissive) 和反射光 (specular) 。
- 碰撞偵測 (Collision detection) 是判斷遊戲中某兩個物件是否產生觸碰並適當回應的過程。
- 你可以利用碰撞偵測函式庫和物理模擬函式庫來處理這些複雜的問題。
- 音效函式庫可以讓你利用聲音，讓遊戲增色不少。這些函式庫中有許多支援 3D 音效。
- 遊戲通常會以顯示文字的方式來與使用者溝通。
- 計時器 (timers) 是用來控制動畫的速度，讓動畫看起來更自然。
- 使用者的輸入裝置包括鍵盤、滑鼠、搖桿和遊戲控制器。

27.4.1 Ogre 初始化

- `Root` 是用來啓動引擎的 Ogre 基本物件。我們必須先建立 `Root` 物件，才能呼叫 Ogre 的函式。
- 呼叫 `Root` 類別的 `showConfigDialog` 函式可以顯示對話框。`OGRE` 引擎渲染設定對話框能讓使用者選擇渲染的設定。
- 解析度 (resolution) 是由兩個值所定義的：寬度和高度，用來決定繪製場景的像素。較高的解析度可以產生較細緻的圖形。
- n 位元的顏色深度 (color depth)表示我們總共可以顯示 2^n 種可能的顏色。
- Ogre 可以在 `RenderWindow` 中繪製圖形。

27.4.2 建立場景

- 場景是一組圖案的集合。
- `SceneManager` 會管理場景圖 (scene graph)，這是一個包含場景內所有物件的資料結構。
- 我們使用 `SceneManager` 來建立物件，並決定應該要顯示哪一個物件。Ogre 應用程式可以使用一個以上的 `SceneManager`。
- 在 Ogre 中，`Camera` 相當於是你的眼睛，用來檢視場景。`Camera` 可以放在場景的任何位置，或是附加在 `SceneNode` 上。Ogre 支援在單一場景中使用多個 `Camera`。

- Viewport 是用來顯示 Camera 所拍攝事物的螢幕範圍。每一個 Camera 可以有一個以上的 Viewport。
- Ogre 有三種光源：點光源 (Point)、聚光燈 (Spot) 和平行光源 (Directional)。光源是經由 SceneManager 類別的 createLight 函式建立的。

27.4.3 將元素加入場景

- 在場景中，Entity 是一個 mesh 的實體。mesh 是一個檔案，內容爲 3D 模型的幾何資訊。程式中可以有多個 Entity 都以同一個 mesh 爲基礎，但是每個 Entity 都有不同的名稱。
- 我們用 SceneManager 建立 SceneNode，其中包含了物件及物件位置的資訊。
- 根節點 (root node) 是所有其他節點的父節點。當你建立根節點的子節點時，它的初始位置爲 (0, 0, 0)。
- 使用類別 SceneNode 的 attachObject 函式，將 Entity 物件附加到 SceneNode 上。
- scale 改變了附加在 SceneNodeg 上的 Entity 的大小，但是這不會影響到 Entity 建立所依據的基礎 mesh 之大小。setScale 是以節點的原始大小爲基礎來改變節點的大小。這些函式也會以相同的係數將 SceneNode 所有的子節點做縮放。如果你不想縮放子節點，可以用 false 值呼叫 setInheritScale。
- setPosition 函式會將節點放在指定的座標位置。
- Ogre 使用材質腳本來建立材質。其副檔名爲 .material。一個材質檔案中可以定義多個材質，但是每個材質定義都必需有唯一的名稱。
- Overlay 的定義儲存在副檔名爲.overlay 的腳本中。一個.overlay 檔案可以儲存數個 Overlay 定義。Overlay 中的每一個物件都有三個主要屬性：度量模式 (metrics mode)、位置和大小。
- Overlay 是由 OverlayElement 組成的。Overlay 中的第一個元素必須是 OverlayContainer。OverlayContainer 可以存放任何型別的 OverlayElement。TextAreaOverlayElement 可以存放文字。呼叫 show 函式，即可將 Overlay 顯現在螢幕上。
- 使用 TextAreaOverlayElement 來顯示文字。呼叫 setCaption 改變螢幕上的文字。
- 當我們使用多個 Overlay 時，z-order 較高的 Overlay 會出現在 z-order 較低的 Overlay 上方。
- 字型藉由腳本定義在.fontdef 檔案中。
- 我們使用 OverlayManager 類別的 static 成員函式 getSingleton 取得指向 OverlayManager 物件的指標。

27.4.4 動畫與計時器

- `translate` 函式可以移動 `SceneNode`。
- 根據預設，`SceneNode` 的移動是在父空間中完成的。父空間內的移動與父節點的原點有關。在世界空間 (world space) 移動是以場景的原點 (0, 0, 0) 作爲基準。在本地空間移動則是以節點的原點爲基準。
- `FrameListener` 類別會處理 `Ogre::FrameEvent`。每當影格開始或結束時，都會產生一個 `FrameEvent`。

27.4.5 使用者輸入

- Ogre 不直接支援來自鍵盤、滑鼠或搖桿的使用者輸入。
- Ogre SDK 利用物件導向輸入系統 (Object Oriented Input System，OIS) 來處理使用者的輸入。
- `InputManager` 是用來建立各種輸入裝置。我們必須提供一個視窗，才能建立 `InputManager`。
- `Keyboard` 物件收集 `KeyEvent`，將它們送到 `KeyListener`。
- OIS 替所有的案件都定義了列舉型別，我們可以用它來決定按下的是哪一個鍵。

27.4.6 碰撞偵測

- `getPosition` 會回傳 `Vector3`，代表此節點與其父節點的相對位置。成員函式 `_getDerivedPosition` 則會回傳節點與原點的相對位置
- `SceneManager` 可以參照節點建立時所取的名稱，以取得場景圖中的任何節點。
- `Ball` 的方向是由 `Vector3` 所決定的。正的 `x` 值表示 `Ball` 會沿著 `x` 軸往右移動，負值則代表 `Ball` 會往左移動。假如 `Ball` 往右移動，將它的 `x` 值乘以`-1`，就會改變正副號，進而反轉方向。
- 有一些函式庫是專門處理碰撞和物理模擬的。

27.4.7 音效

- OgreAL 包裝了 OpenAL 音效函式庫，這個包裝器能讓我們將音效附加到場景節點上，藉此將音效功能整合到 Ogre 程式碼中。
- 我們必須使用前置處理器指令，將 `OgreAL.h` 標頭檔含括進來。
- `Sound` 是一個內含音效資料的 OgreAL 物件。`SoundManager` 類別的 `createSound` 函式可以用來建立音效。程式中只能有一個 `SoundManager`。

- 函式 `createSound` 會接收三個參數，第一個是 `Ogre::String`，這是 OgreAL 系統中的 `Sound` 名稱。第二個參數是 `Sound` 要播放的音效檔的名稱。第三個是布林值，用來判斷 `Sound` 是否應該循環播放。假如傳入的是 `false` 值，程式會將 `Sound` 播放一次，然後停止。假如傳入的是 `true` 值，程式會循環播放音效，直到我們將它停止。
- 在 `attachObject` 函式中將 `Sound` 附加到節點上。
- 每一個 `Sound` 都應該有唯一的名稱。
- 在重新播放之前，應該先把 `Sound` 的播放結束。

27.4.8　資源

- 在使用資源之前，應該先載入它們。
- 我們使用 `ResourceGroupManager` 來管理遊戲的資源。
- `addResourceLocation` 函式會接收三個 `Ogre::String` 引數。第一個是資源的位置。第二個引數是資源的檔案類型。第三個則是檔案所屬的資源群組。

27.4.9　測試乒乓遊戲

- Ogre 支援各種平台，因此你應該儘量避免撰寫屬於特定平台的程式碼。

術語

3D 繪圖引擎 (3D graphics engine)
3D 模型 (3D model)
3D 模型工具 (3D modeling tools)
3D 音效 (3D sound)
`alpha` 通道 (alpha channel)
環境光 (ambient light)
`Camera` 類別 (Camera class)
碰撞偵測 (collision detection)
顏色 (color)
顏色深度 (color depth)
剔除 (culling)
散射光 (diffuse light)
Direct3D
平行光源 (Directional lights)
自發光 (Emissive light)
`Entity` 類別 (Entity class)
將 3D 模型匯出 (export 3D models)
影格 (frame)
`FrameEvent` 類別 (FrameEvent class)
`FrameListener` 類別 (FrameListener class)
`Keyboard` 類別 (Keyboard class)
`KeyEvent` 類別 (KeyEvent class)
`KeyListener` 類別 (KeyListener class)
細節層次 (levels of detail，LoD)
`Light` 類別 (Light class)
本地空間 (local space)
材質 (material)
`mesh`
度量模式 (metrics mode)
法線 (normal)
物件導向輸入系統 (Object Oriented Input System，OIS)

Ogre (Object-Oriented Graphics Rendering Engine)
OgreAL
OpenAL 音效函式庫 (OpenAL audio library)
OpenGL
`Overlay` 類別 (Overlay class)
`OverlayContainer` 類別 (OverlayContainer class)
`PanelOverlayElement` 類別 (PanelOverlayElement class)
父空間 (parent space)
像素模式 (Pixel mode)
點光源 (Point light)
相對模式 (relative mode)
渲染 (rendering)
渲染子系統 (rendering subsystem)
`RenderWindow` 類別 (RenderWindow class)
解析度 (resolution)
`ResourceGroupManager` 類別 (ResourceGroupManager class)
根節點 (root node)
場景 (scene)
場景圖 (scene graph)
`SceneManager` 類別 (SceneManager class)
`SceneNode` 類別 (SceneNode class)
腳本 (scripts)
`Sound` 類別 (Sound class)
`SoundManager` 類別 (SoundManager class)
反射光 (specular)
聚光燈 (Spot light)
TextAreaOverlayElement 類別 (TextAreaOverlayElement class)
紋理 (texture)
計時器 (timer)
`Viewport` 類別 (Viewport class)
世界空間 (world space)
z-order)

自我測驗

27.1 請在下列每題的空白欄處填入答案：

a) _____檔案自動地含括了最常用的 Ogre 標頭檔。
b) 我們必須在呼叫其他任何 Ogre 函式 (除了 logging 以外) 之前建立_____物件。
c) Ogre 所定義來指向音訊檔案資料的主要型態爲_____。
d) _____物件用於表示 Ogre 中的顏色。
e) _____檔案自動地含括了最常用的 OgreAL 頭檔。
f) _____是用來定義材質和 overlay 以供 Ogre 程式使用。
g) _____物件是用來載入資源以供 Ogre 程式使用。
h) Ogre 使用_____物件來管理場景。
i) 3D 模型定義在 Ogre 的_____檔案中。

27.2 說明下列何者爲對，何者爲錯。如果答案是錯，請解釋爲什麼。

a) 座標 (0, 0) 代表 `OverlayContainer` 的左下角。
b) 如果 Ogre 試圖載入不存在的外部檔案，便會產生執行時期錯誤。
c) Ogre 的顏色值範圍爲 0 到 255。

d) 傳入 `false` 值給 `createSound` 函式，會造成音效檔一直重複播放。

e) 要在螢幕上繪製文字的 `Overlay`，必須指定文字要使用何種字型繪製。

f) 每一個 `Entity` 都應該有唯一的名稱。

27.3 試撰寫能分別完成以下工作的敘述：

a) 將名稱爲 `entityPtr` 的 `Entity` 指標附加到名稱爲 `nodePtr` 的 `SceneNode`。

b) 將前一題的 `Entity` 縮放爲原來大小的一半。

c) 建立 `Sound sample`，重複播放 `sound.wav` 檔案。

d) 如果空白鍵有被按下，則將 `int number` 設定爲 0。

e) 設定一個 `Overlay Element`，以相對於父 `Container` 的大小來放置它的位置。

f) 在 OgreSDK 的 `media` 資料夾中加入一個名稱爲 `sounds` 的資料夾，作爲一般性的資源位置。

g) 將 `SceneNode` 往左移 15 個單位，往上移 4 個單位，朝向你自己移 8 個單位。

27.4 請找出以下程式碼的錯誤。

```
a) SceneNode node;
b) ColourValue( 0, 0, 255 );
c) Root *rootPtr = new Root();
   rootPtr->initialize( true, "Window" );
d) viewportPtr = sceneManagerPtr->addViewport( cameraPtr );
```

自我測驗解答

27.1 a) `Ogre.h`。b) `Root`。c) `Sound`。d) `ColourValue`。e) `OgreAL.h`。f) 腳本。g) `ResourceGrouphManager` 或是 `ResourceManager`。h) `SceneManager`。i) `.mesh`。

27.2

a) 錯。座標 (0, 0) 代表 `OverlayContainer` 的左上角。

b) 對。

c) 錯。Ogre 的顏色值範圍爲 0.0 到 1.0。

d) 錯。音效會播放一次，然後停止。

e) 對。

f) 對。

27.3 a)
```
nodePtr->attachObject( entityPtr );
```
b)
```
nodePtr->setScale( .5, .5, .5 );
```
c)
```
soundManagerPtr->createSound( "sample", "sound.wav", true );
```
d)
```
if ( keyEvent.key == OIS::KC_SPACE )
   number = 0;
```
e)
```
metrics_mode relative;
```
f)
```
ResourceGroupManager::getSingleton().addResourceLocation( "media/sounds",
   "FileSystem", "General" );
```
g)
```
sceneNodePtr->translate( -15, 4, 8 );
```

27.4 a) 變數 node 應該宣告爲指向 SceneNode 的指標。所有 Ogre 的 SceneNode 函式都是接收指標做爲參數，或是會傳回指標。

b) ColourValue 物件的參數只接受介於 0 與 1 之間的數值。

c) Ogre 使用英式拼字，所以此函式的正確寫法爲 initialise。此外，渲染設定應該在呼叫 initialise 之前完成。

d) addViewport 是 RenderWindow 類別的函式，而非 SceneManager 類別的函式。

習題

27.5 瀏覽我們的遊戲程式設計資源中心 www.deitel.com/computergames/gameprogramming/以及 C++ 遊戲程式設計資源中心 www.deitel.com/CplusplusGameProgramming/所列出的資源。

27.6 (乒乓遊戲的獲勝狀態) 更改乒乓球遊戲，當玩家得到 21 分時，比賽會自動結束，並顯示出哪一個玩家獲勝的訊息。

27.7 (加快球的速度) 在大多數的乒乓遊戲中，假如兩個玩家之間拉鋸了很長的一段時間，則球會開始加速以打破僵局。更改乒乓遊戲，每當玩家連續來回對打 10 次之後，將球加速。當某個玩家贏得分數時，讓球回到原先的速度。

27.8 (降低球拍速度) 某些乒乓球遊戲會更改一個或兩個玩家的球拍速度，讓遊戲保持平衡。修改乒乓球遊戲，當某個玩家領先五分以上時，讓他的球拍慢下來。玩家領先的分數越高，球拍移動的速度越慢。假如玩家的分數差距回到 5 分以下時，球拍就應該回到正常的速度。

27.9 (乒乓遊戲的選單) 修改乒乓遊戲，在遊戲開始之前，顯示一個選單，讓玩家選擇幾個不同的球速和球拍速度。

27.10 (轉動 Sphere) 撰寫一個程式，將 sphere.mesh 中的模型繪製在螢幕中央。當使用者按了上鍵或下鍵時，mesh 應該朝那個方向移動 10 個單位。

27.11 (轉動 Sphere 的加強版) 修改習題 27.10 的程式，當使用者按了上鍵或下鍵時，球體應該每秒移動一個單位。

27.12 (貪食蛇遊戲) 貪食蛇遊戲的目標是讓蛇在遊戲場中移動，吃掉一些食物。蛇是以一連串的圓球體來表示的，而遊戲區域是一個二維陣列。蛇可以往上下左右移動。假如蛇吃掉一塊食物 (以「F」表示)，它就會從尾端變長，多出一個球體 (圖 27.21)。假如蛇碰撞到遊戲區域的牆壁 (超出陣列)，則玩家就輸了 (圖 27.22)。假如蛇碰到它自己，玩家也輸了 (圖 27.23)。

27.13 (貪食蛇與障礙物) 請修改習題 27.12，將障礙物加入遊戲區域 (圖 27.24)。假如蛇碰到障礙物，玩家就輸了。

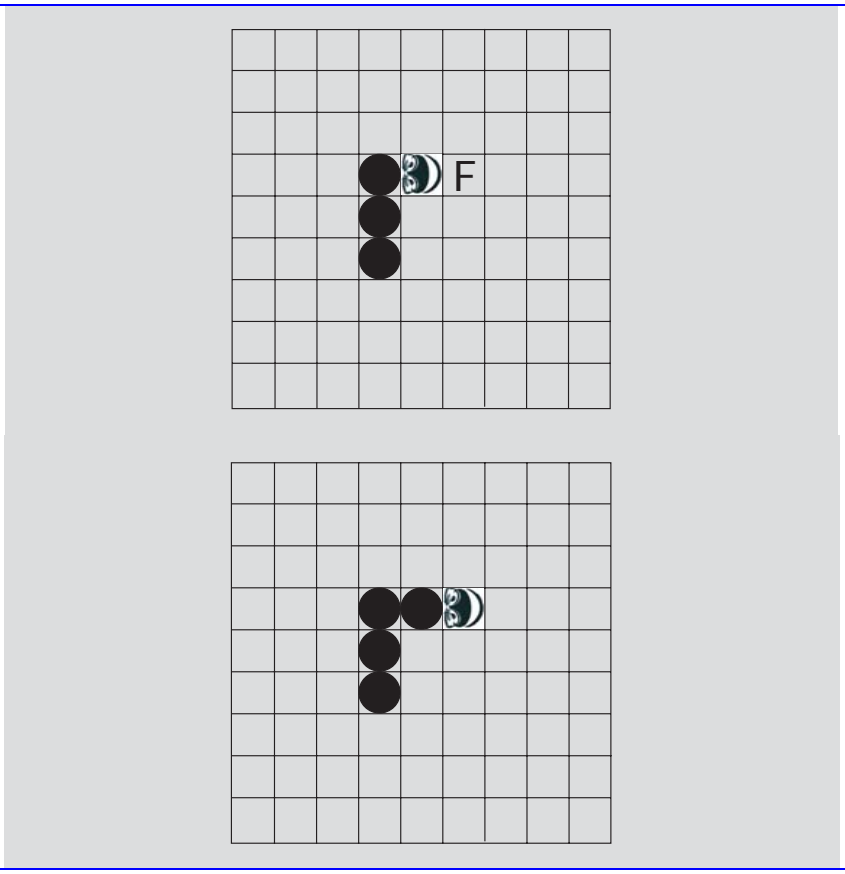

圖 27.21　蛇吃到食物時，就會長大

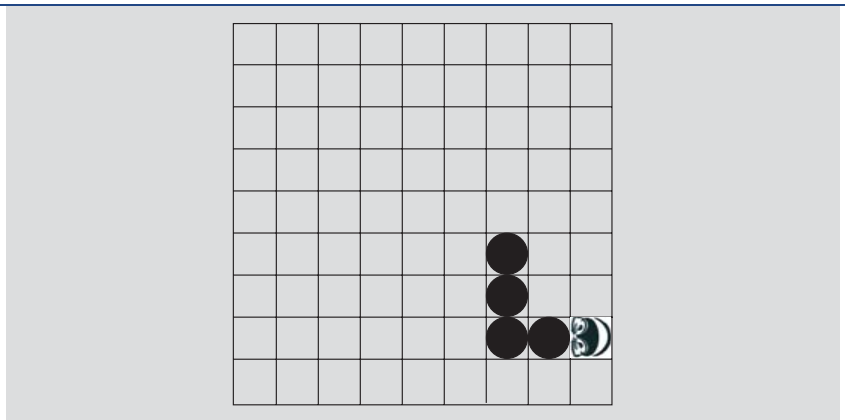

圖 27.22　當蛇碰到牆壁，玩家就輸了

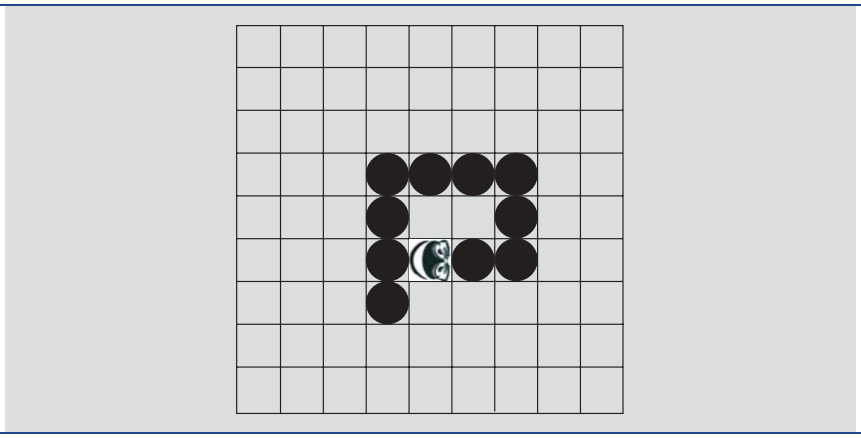

圖 27.23　　當蛇碰到自己，玩家就輸了

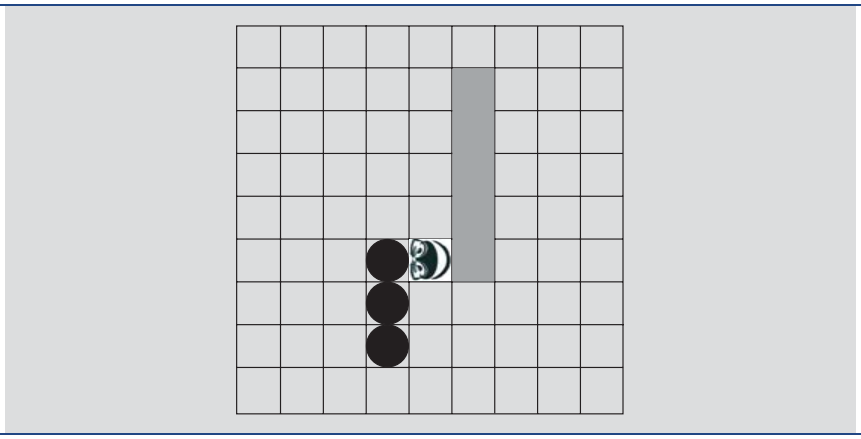

圖 27.24　　當蛇碰到障礙物，玩家就輸了