```
void Vertor3d :: Normalize () {
    float fLength = sqrt (x*x + y*y + z*z);
    #ifdef ASSERTBADDATA
    assert ( fLength > 0 );
    #endif
    if( fLength > 0) {
        x /= fLength;
        y /= fLength;
        z /= fLength;
    }
    // Return a unit vector along the x-axis if
    // this is a zero-length vector
    else {
        x = 1.0f; y = 0.0f; z = 0.0f;
    }
}

inline void Vertor3d :: NormalizeUnsafe () {
    float fLength = sqrt (x*x+y*y+z*z);
    assert ( fLength != 0 );
    x /= fLength;
    y /= fLength;
    z /= fLength;
}
```

16.4 結　论

本章的中心是关于如何避免游戏在运行时崩溃的技术。一旦发生崩溃，就会影响用户的体验（也影响我们的声誉）。第一部分介绍了如何在程序中高效地利用断言函数。了解了什么时候应该使用断言函数，以及什么时候应该使用其他方法。讨论了一旦游戏被发布，对断言函数的不同处理方法的优点，并介绍了一个在游戏中可以使用的非常基本的用户自定义断言函数。

第二部分解决了游戏运行一段时间后逐渐出现的一些问题。内存泄漏、内存碎片、时间误差的累计和在这些情况下所出现的一些问题。

最后给出了一些处理坏数据问题的解决方案。坏数据是指传递到函数中的没有意义的数据。一方面不想程序浪费时间，因此每个底层函数都能处理任何参数。另一方面希望确保游戏在某些情况下不会被终止。一个折衷的解决方案是每隔几个星期就转换到断言方式，来报告每个坏数据的实例，然后再换回忽略状态。通过这种方法有望使代码不断适应坏数据，但是仍然需要尽可能的在第一时间捕获到由于坏数据所产生的 BUG。

16.5　阅 读 建 议

下面是少数几本讨论如何使用断言函数调用以及为什么要使用它的书之一，似乎在其他的书中都将断言函数作为实现细节而将其忽略。

McConnell , Steve ,Code Complete ,Microsoft Press,1993.

下面这篇文章详细讨论了如何建立自己的断言函数。

Rabin, Steve, "Squeezing More Out of Assert ",Game Programming Gems ,Charles River Media,2000.

关于附带光盘

本书讲述了一些复杂的概念，本书所带光盘中包含了演示这些复杂概念的程序的源代码。所有的代码都尽可能地简单，以便让读者能够抓住问题的关键；但与此同时，它们又是一些非常实用的程序，可以反映不同方法之间的联系以及它们是怎样整合在一起的。

本书并没有将一页页大量繁琐的代码都放进来，各章节的内容中只包含一部分比较重要的代码段。其余的细节可以参考光盘中的源代码。如果喜欢读大量的源代码，可以打开源代码文件，一边学习本书，一边领悟代码。

使用源文件的最好方法是编译文件来验证它的运行结果，如果想检查某种操作的执行过程可以使用单步调试方式，另外也可以对它们进行修改来看看如何扩展和改进它们。

每个程序的源文件和工程文件都放在一个单独的文件夹中。

- 第 7 章：内存管理。一个完整的内存管理器的完整实现和一个小型的测试程序。
- 第 11 章：插件。一个使用插件的简单的 Win32 应用程序。这个应用程序展示了当前载入的插件，插件本身会在主程序中增加一些菜单项。插件可以被动态的加载和卸载。
- 第 12 章：运行期类型信息系统（RTTI）。一个自定义的运行期类型信息系统。它主要用于单继承。
- 第 12 章：多继承的 RTTI 系统。这是对前面自定义实时信息系统的改进。它通过添加一些附加信息来支持多继承。
- 第 13 章：对象工厂。这是一个模板化的游戏实体对象工厂。
- 第 14 章：序列化。一个非常简单的序列化程序，用于将完整的游戏实体树保存到磁盘中或从磁盘中载入。它使用一个自定义的流类和指针恢复方法。

所有的程序都使用 Visual Studio C++ 6.0 编译并在 Windows 2000 下进行运行。但插件程序实例是个例外，它是独立于特定平台和编译器的，所以，它们很容易在你所喜欢的环境下编译和运行。

提供了插件程序实例的可执行程序，不用经过编译就可以直接执行它。

关于附带光盘

本书讲述了一些复杂的概念，本书所带光盘中包含了演示这些复杂概念的程序的源代码。所有的代码都尽可能地简单，以便让读者能够抓住问题的关键；但与此同时，它们又是一些非常实用的程序，可以反映不同方法之间的联系以及它们是怎样整合在一起的。

本书并没有将一页页大量繁琐的代码都放进来，各章节的内容中只包含一部分比较重要的代码段。其余的细节可以参考光盘中的源代码。如果喜欢读大量的源代码，可以打开源代码文件，一边学习本书，一边领悟代码。

使用源文件的最好方法是编译文件来验证它的运行结果，如果想检查某种操作的执行过程可以使用单步调试方式，另外也可以对它们进行修改来看看如何扩展和改进它们。

每个程序的源文件和工程文件都放在一个单独的文件夹中。

- 第 7 章：内存管理。一个完整的内存管理器的完整实现和一个小型的测试程序。
- 第 11 章：插件。一个使用插件的简单的 Win32 应用程序。这个应用程序展示了当前载入的插件，插件本身会在主程序中增加一些菜单项。插件可以被动态的加载和卸载。
- 第 12 章：运行期类型信息系统（RTTI）。一个自定义的运行期类型信息系统。它主要用于单继承。
- 第 12 章：多继承的 RTTI 系统。这是对前面自定义实时信息系统的改进。它通过添加一些附加信息来支持多继承。
- 第 13 章：对象工厂。这是一个模板化的游戏实体对象工厂。
- 第 14 章：序列化。一个非常简单的序列化程序，用于将完整的游戏实体树保存到磁盘中或从磁盘中载入。它使用一个自定义的流类和指针恢复方法。

所有的程序都使用 Visual Studio C++ 6.0 编译并在 Windows 2000 下进行运行。但插件程序实例是个例外，它是独立于特定平台和编译器的，所以，它们很容易在你所喜欢的环境下编译和运行。

提供了插件程序实例的可执行程序，不用经过编译就可以直接执行它。