**南 开 大 学**

本 科 生 毕 业 论 文（设 计）

中文题目： 基于DirectX12搭建渲染系统

外文题目： Build rendering framework based on DirectX12

学 号： 1612895

姓 名： 王泽元

年 级： 2016 级

专 业： 软件工程

系 别： 软件工程

学 院： 软件学院

指导教师： 谢茂强

完成日期： 2020年4月30日

关于南开大学本科生毕业论文（设计）的声明

本人郑重声明：所呈交的学位论文，是本人在指导教师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或没有公开发表的作品内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年 月 日

本人声明：该学位论文是本人指导学生完成的研究成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：

年 月 日

摘 要

DirectX 12引入了最新的Direct3D（DirectX核心的3D图形API）版本。Direct3D 12对硬件的操作比之前任何版本都要更快和有效率，并且其可以实现更丰富的场景，操纵更多的对象，添加更复杂的效果。Direct3D 12为图形学的开发者提供了四个主要的好处：1. 大大减少了CPU的开销；2. 大大降低了功耗；3. GPU效率提高达20%；4. Windows 10设备（PC，平板电脑，控制台，移动设备）的跨平台开发。

但与此优势相对的，Direct3D 12专供高级图形程序员使用。它需要大量的图形专业知识和高水平的微调。Direct3D 12旨在充分利用多线程，谨慎的CPU/GPU同步以及资源从一种用途到另一种用途的过度和重用，这些技术需要大量的内存级编程技能。

本文旨在为研究计算机图形学的人提供易用的接口的同时享受Direct3D 12强大的处理能力。本论文介绍了DirectX 12设计思路，并深入分析了其框架模型和物理模型，以此说明其复杂度。将其与计算机图形学无关的部分（内存，硬件）进行了封装，为使用者暴露出渲染核心部分。介绍了该框架从设计到开发以及重构的详细内容。并且用此框架实现了数个样例以此证明该框架的易用性和健壮性。

关键词：DirectX 12；渲染框架；封装；渲染管线；

**Abstract**

DirectX 12 introduces the next version of Direct3D—the 3D graphics API at the heart of DirectX. Direct3D 12 is faster and more efficient than any previous version. Direct3D 12 enables richer scenes, more objects, more complex effects, and full utilization of modern GPU hardware. Direct3D 12 provides four main benefits to graphics developers (compared with Direct3D 11). 1. Vastly reduced CPU overhead; 2. Significantly reduced power consumption; 3. Up to (approximately) 20% improvement in GPU efficiency; 4. Cross-platform development for a Windows 10 device (PC, tablet, console, mobile).

However, with this benefit, Direct3D 12 is designed for advanced graphics programmers to use. It calls for significant graphics expertise, and a high level of fine-tuning. Direct3D 12 is designed to make full use of multi-threading, careful CPU/GPU synchronization, and the transition and re-use of resources from one purpose to another. These are techniques that require a considerable amount of memory-level programming skill.

This article aims to provide users of computer graphics with an easy-to-use interface while enjoying the powerful processing capabilities of Direct3D 12. This paper introduces the design ideas of DirectX 12 and analyzes its framework model and physical model in depth to illustrate its complexity. It encapsulates parts that are not related to computer graphics (memory, hardware), exposing the core parts of the rendering for users. The detailed content of the framework from design to development and refactoring is introduced. And use this framework to realize several examples to prove the ease of use and robustness of the framework.

**Keywords**: DirectX 12; rendering framework; packaging; rendering pipeline;

目 录

[第一章 绪论 1](#_Toc388302932)

[第一节 研究背景和意义 1](#_Toc388302933)

[第二节 国内外研究现状 1](#_Toc388302934)

[第三节 研究的关键问题 2](#_Toc388302935)

[第四节 论文的主要工作 3](#_Toc388302936)

[第五节 论文的内容组织 4](#_Toc388302937)

[第二章 文件系统碎片概述 6](#_Toc388302938)

[第一节 典型的文件系统布局 6](#_Toc388302939)

[第二节 文件系统碎片成因 8](#_Toc388302940)

[第三节 文件系统碎片对系统I/O性能影响 9](#_Toc388302941)

[第四节 本章小结 9](#_Toc388302942)

[第三章 文件系统碎片度模型 10](#_Toc388302943)

[第一节 文件系统碎片度定义 10](#_Toc388302944)

[第二节 影响文件系统碎片度的相关因素 10](#_Toc388302945)

[第三节 文件系统碎片度评估模型 11](#_Toc388302946)

[第四节 本章小结 12](#_Toc388302947)

[第四章 制造空闲块碎片整理算法 13](#_Toc388302948)

[第一节 制造空闲块算法原理 13](#_Toc388302949)

[第二节 制造空闲块算法描述 13](#_Toc388302950)

[第三节 制造空闲块算法测试与分析 14](#_Toc388302951)

[第四节 本章小结 16](#_Toc388302952)

[第五章 基于文件最近访问时间的碎片整理算法 17](#_Toc388302953)

[第一节 基于文件最近访问时间算法原理 17](#_Toc388302954)

[第二节 基于文件最近访问时间算法描述 17](#_Toc388302955)

[第三节 基于文件最近访问时间算法测试与分析 18](#_Toc388302956)

[第四节 本章小结 19](#_Toc388302957)

[第六章 基于文件类型的碎片整理算法 20](#_Toc388302958)

[第一节 基于文件类型算法原理 20](#_Toc388302959)

[第二节 基于文件类型算法描述 20](#_Toc388302960)

[第三节 基于文件类型算法测试与分析 21](#_Toc388302961)

[第四节 本章小结 23](#_Toc388302962)

[第七章 AsDefrag碎片整理系统设计 24](#_Toc388302963)

[第一节 总体设计 24](#_Toc388302964)

[第二节 模块划分 26](#_Toc388302965)

[第三节 详细设计 26](#_Toc388302966)

[第四节 原型实现 39](#_Toc388302967)

[第八章 总结与展望 44](#_Toc388302968)

[第一节 本文总结 44](#_Toc388302969)

[第二节 进一步展望 44](#_Toc388302970)

[参考文献 45](#_Toc388302971)

[致 谢 47](#_Toc388302972)

# 绪论

框架（Framework）是指为解决一个特定问题而设计的一组相互协作的类，其具有一定的约束性又具有一定的支撑性。框架规定了应用的体系结构，定义了程序的整体结构及其控制流，类和对象的分割及其各个部分的责任与合作。框架预定义了整体程序的设计模式，让开发人员或者科研人员可以集中精力在需求或者想法的实现，而不是在程序整体的结构上。

## 研究背景和意义

计算机图形学的目的是将二维或三维的数据结构通过一定的数学手段转化为计算机显示器像素点中的颜色数据。简单地说，计算机图形学研究的主要内容就是研究如何在计算机中将数据转化为肉眼可见的图像、利用计算机对图形进行的计算、处理和加工等的相关算法。计算机图形学研究的范围相对广泛且对一个人的综合素质的要求比较高，不同的研究方向要求的素养各不一样，总的来说有以下几点：良好的编程功底来实现自己的想法，很好的数学功底来推导和验证自己的想法，包括但不限于：几何、高等代数、微分几何、数学分析、概率论和统计学，物理学来提出模拟相关的想法，还有心理学和艺术修养让自己的想法得到其他人的认可。

编程功力是将自己的想法转化为结论的不可或缺的关键一步，但随着GPU的发展，以及对渲染流程的探索，我们可以操作GPU的物理内存，以此达到最佳的数据排布，让命中率大大提高，将常用数据放在高速缓存上，充分发挥GPU性能；我们摆脱了固定管线的束缚，渲染的过程不再是切换开关，而是以可编程管线的方式来实现我们的想法，我们有权利选择某些流程的省却，可以在CPU和GPU之间交互数据，即利用CPU进行逻辑控制，串行运算，又可以享受GPU的大量计算资源。但这带来了一个问题，那就是学习成本大幅提升，编写程序的代码量过大导致尝试想法的试错成本过高。对于某一方向的研究者其需要掌握渲染的全部流程才能看到最终的结果。本框架旨在将渲染核心暴露给用户并提供默认选项，将与渲染无关的设施进行屏蔽来简化API的记忆量和使用户可以集中精力于渲染本身的细节，而不需要关注平台、版本。

## 现代图形API

现代的图形接口的设计哲学是：通过更加显示的内存和状态管理、更贴近于硬件的接口，给开发者更大的自由度来实现自己的需求。从接口来看Vulkan彻底摒弃了OpenGL的接口重新设计，Metal是苹果公司为游戏开发者推出的新的平台技术，Direct3D 12的绝大部分接口也都被重新设计过，少量原来未被更名的接口的概念也与原来有着巨大的区别。

现代图形接口有以下几个特点：

1. 切换状态消耗的资源减少：使用管道状态（PipelineState）提前创建并绑定管道的流程，渲染时，通过加载不同的管道状态，驱动只需要少量的开销便可将预先创建的流程绑定到GPU中；
2. 预置的着色器资源布局和绑定：通过根签名（RootSignature）预先设置好所需要使用的资源布局信息，包括：恒定缓冲区视图（Constant Buffer View）, 着色器资源视图（Shader Resource View）, 无序访问试图（Unordered Accessed View）；
3. 资源状态管理显示声明：通过资源屏障（Resource Barrier）来要求程序明确控制资源的状态迁移
4. 渲染工作记录和提交和CPU/GPU同步控制显式声明；通过CommandList来记录渲染命令，再通过CommandQueue提交到GPU中执行；CPU/GPU的同步是通过Fence对象和WaitFence函数完成；
5. CPU/GPU堆内存管理显式声明：Map/Unmap将CPU和GPU的地址空间相连，CreateCommitResource函数在GPU上开辟资源存储的空间，UpdateSubresource函数可以将CPU的资源提交给GPU；
6. 资源生命周期控制显式声明：新的图形API明确要求程序确保渲染命令不能在使用资源时释放，释放的时机由应用层控制，提升了空间的利用效率；
7. 提交渲染工作并行处理：每个CommandList可以在不同线程上单独填充，即可以并行记录渲染指令，这个特点发挥了现代CPU的并行能力。可以创建多个CommandQueue来进行异步计算或上传数据，这个特点利用了GPU多核的并行机制，以此来实现渲染和计算、上传数据的并行。

## 第三节 研究的关键问题

经过以上说明，可以看到现代图形接口是通过更加显示内存和状态管理、和贴近于硬件接口的方式来提高实时渲染性能。但要面向现代接口写出优秀的程序并非易事，对于不熟悉底层的程序员甚至效果不及传统API。传统接口驱动层对复杂图形管线和内存的管理现在转移到开发者手中，本框架就是针对这些内容进行封装，以达到易用的目的。

### 1.3.1 对现代图形API的再次抽象

### 合并相互制约的数据结构，比如PipelineState中PrimitiveTopologyType与在渲染时要设置的PrimitiveTopology概念重合，在改变其中一个的时候另一个也需所知改变，故将其封装。类似于以往的API的接口不再合适，比如管线的状态，以前分为Vertex Layout、Sampler、DepthStencil、Blend、Rasterizer，但现在API已经将这些属性合并，因此需要抽象出PipelienState来进行设置。

### 1.3.2 堆内存管理

这其中包括了CPU和GPU内存。

首先是读写的限制问题，为了易用性的考虑，默认情况下其全部是可读可写的，以避免错误的设置导致Bug的出现；其次是对其问题，在CPU上内存是连续分配的，而在GPU上内存是离散的，每个存储单元至少占据256字节，因此本框架会在申请内存时自动对齐，不需要用户考虑内存分配问题。

### 1.3.3 描述符管理

在传统的图形API上着色器绑定资源只需要调用形如SetDepthStencil/ SetConstantBuffer/ SetTexture的接口就行，但在现代API中这些资源需要预先设置，然后通过对描述符间接寻址来寻址资源位置。对于描述符的声明基于使用的频率和描述符的类型不同，有静态寻址，一次间接寻址，两次间接寻址等方式。这增加了资源的可配置性但同时大大提升了资源调用的复杂度。

### 1.3.4 对象的生命周期

错误释放资源的时机会导致GPU设备的停用，比如：在并行过程中，Frame2释放了某个资源，但这个资源仍在Frame1中使用，就会导致程序的出错；错误转换对象状态的时机会导致指令抛出异常，比如：在CommandList执行力Close命令后便进入执行状态，若再向其中加入操作就会导致报错。

本框架旨在帮助不熟悉DirectX 12的图形学研究者利用此框架可以快速验证自己的想法，不断得到反馈最终取得成果。

## 论文的主要工作

本文主要研究了DirectX 12 API的设计思路和相关GPU/CPU运行技术问题。针对1.3节提出的问题，本文分别对现代API的再次抽象、堆栈内存管理、描述符管理、对象生命周期管理等几个方面进行了研究。由于贴近底层的API设计和为了面面俱到而激增的函数参数是造成学习曲线陡峭的主要原因。因此本框架不求面面俱到，而是仅将关于渲染的核心部分进行暴露。本文的主要工作如下：

1. 通过研究DirectX 12 API的设计思路，来初步了解现代图形API的工作流程，及其具有高复杂度的原因。了解国内外的渲染框架，并在此基础上根据DirectX 12自身特点构建渲染流程。
2. 实现几个流行算法。样例不仅可以展示该框架的易用性，更可以以直观的方式展示各个组件时如何结合的，为使用者提供更好的上手教程。

## 第五节 论文的内容组织

本文共分为八章，各章的内容如下：

第一章 绪论。介绍了本文的研究背景和意义，要解决的问题和本文的主要工作。

第二章 现代渲染流程介绍。对渲染的核心部分进行解析。介绍了点阵数据如何通过顶点着色器、几何着色器、外壳着色器、域着色器、几何着色器、光栅化、像素着色器生成完整图片。

第三章 DirectX 12渲染流程介绍。对DirectX 12的渲染流程进行解析，分割渲染核心与原本应该驱动完成的任务，在此基础上搭建渲染框架，说明模块的拆分和合作的关系。

第四章 渲染资源相关部件的实现。资源的加载是渲染中不可或缺的，无论是3D模型的加载和图片资源的加载。然而DirectX自身并没有实现相关的操作，因此为了简化使用者学习压力，本文介绍了图片资源的加载以及3D资源（.obj文件）的加载以供用户使用。

第五章 流行算法的实现。在实现算法时，先介绍此算法的原理，并且详细描述该算法的过程以及需要利用到的渲染流程，各组件的相互协作。然后实现该算法，对组件进行说明，并与原生API实现的同样算法的进行编写效率的对比，分析对比结果。

第六章 总结与展望。对本文进行的工作和工作中的不足进行了总结，并介绍了后续的研究方向。

# 第二章 文件系统碎片概述

## 第一节 典型的文件系统布局

对两种典型的文件系统，Windows平台的NTFS文件系统和Linux平台的Ext文件系统的布局进行分析。

### 2.1.1 NTFS文件系统结构分析

1. NTFS文件系统的物理结构特点

对于磁盘分区内的物理扇区，NTFS文件系统的组织和管理是以簇为基本单元的[15]。在NTFS文件系统中，簇是磁盘上若干个连续物理扇区的组合。簇的划分是从所属磁盘分区的第一个物理扇区开始(即DBR 扇区)，通常情况下簇的大小默认为4KB。NTFS分区上的所有簇由于在物理位置上前后相连，所以形成了一个连续不断的线性簇链[27]。NTFS文件系统按照簇在物理扇区的位置上排列的先后顺序，从0开始对其进行顺序编号，即逻辑簇号(Logical Cluster Number，LCN)，从而方便了簇的管理和定位，如图2.1所示。

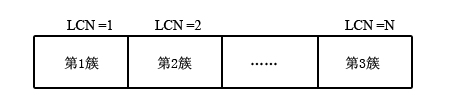


图2.1 磁盘簇的编号规则

1. NTFS文件系统的逻辑分区特点

从文件的角度上来看，NTFS分区内的文件大致可分为两类：元数据文件和用户数据文件[15]。其中，元数据文件是NTFS分区所必需的系统文件，它们是在磁盘分区格式化的过程中创建，用来组织、管理和架构NTFS分区的文件。这些文件统一以“﹩”开头命名。

在NTFS文件系统中，与磁盘分区的规划布局直接相关的元数据文件主要有以下3种：

1. ﹩Boot文件。﹩Boot文件的位置一般占用NTFS分区最开始的16个扇区，它由DBR扇区和操作系统的引导代码主要构成。
2. ﹩MFT文件。﹩MFT文件，即主文件目录表(Master File Table，MFT)，以文件目录的形式描述了NTFS磁盘分区内所有元数据文件和用户数据文件的相关的位置和信息，系统可以通过读取﹩MFT文件访问和操作NTFS分区上的任何文件。﹩MFT文件可由DBR扇区定位，是在﹩Boot文件之后的第一个被访问的NTFS元数据文件，是组织和管理NTFS文件系统的基础。
3. ﹩MFTMirr文件。﹩MFTMirr文件是﹩MFT文件的镜像文件，它备份了﹩MFT文件中的前几个关键的文件记录，它的作用是确保在﹩MFT出错的情况下，系统仍然能够正常访问由这些﹩MFT文件记录所指向的关键的元数据文件。

### Ext3文件系统结构分析

1. Ext3文件系统的物理结构特点

对于磁盘上的物理扇区，Ext3文件系统时以块为单位来进行管理的。文件也是以块为单位存储的，以块号来标识每一个块[4]。通常情况下，块的默认大小为1k。数据在磁盘上的存储如下：首先是引导块，其次是超级快、索引节点表（I-node），最后是数据块。

|  |  |  |  |
| --- | --- | --- | --- |
| 引导块 | 超级块 | 索引节点表 | 数据块 |

图2.2 Ext3磁盘数据分布

1. Ext3文件系统的逻辑结构特点

引导块用来存储磁盘的引导信息；超级块用来存储空闲索引节点表、空闲数据块表；索引节点表(I-node)用来存储文件相关信息和文件存储位置；数据块用来存储文件数据。

每一个文件对应一个节点Inode，该节点中包含文件的基本信息，如文件的所有者，用户读写权限，文件存取时间，文件数据地址表等等。对于文件的索引，分为直接指针，一级间接指针，二级间接指针和三级间接指针。直接指针可存储12k的数据，一级间接指针可存储268k，二级间接指针可存储65804k，三级间接指针可存储16G。所以对于数据块数小于12的文件，在节点中直接存储数据块的块号。若大于12，节点在块号12以后存储一个间接块的块号，在这个间接块中存储所对应的块号。更大的文件可以使用二级间接指针和三级间接指针。

## 第二节 文件系统碎片成因

磁盘碎片也被称作为文件碎片。就是指在磁盘分区中，文件无序分散存储在分区中的不连续的磁盘簇中，而没有有序的存储在连续的磁盘簇上。产生原因主要有一下两种情况：

一种情况是，在对文件进行访问的过程中，Windows操作系统使用虚拟内存来对使程序同步对文件进行访问，使得各个程序频繁对磁盘文件进行读写操作，进而导致产生磁盘碎片[23]。

另一种情况是，在一个磁盘分区中，位于磁盘中部的一部分数据被删除以后，在这个位置重新写入一个比较小的文件。于是在这个新创建的文件两端产生了一些空闲区域。这时候再写入这两段小的空闲空间的任意一个都不足以容纳的另一个新的文件，这时候这个新的文件就将分成多个数据块存储，文件碎片就产生了[23]。

1. 在图2.1的第一部分中，磁盘上保存了文件A、B、C，这三个文件分别占据3、2、4个扇区。这三个文件都连续的存储在磁盘上，没有文件碎片。磁盘还有空闲空间。
2. 在图2.1的第二部分中，对文件A进行修改，增加了A文件的数据大小，从图中可以看出文件A后的磁盘簇已经被别的文件使用，于是文件A新增的内容只能保存到文件C后的空闲区域里。如图2.3所示，文件A由于新数据插入存储在了多个分散的数据块中，形成了文件碎片。

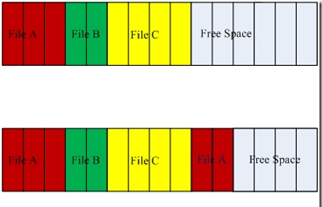


图2.3 磁盘碎片产生过程模拟

## 第三节 文件系统碎片对系统I/O性能影响

一般情况下操作系统运行一个程序的时候，首先磁盘驱动器的磁头要移动到这个程序运行时需要的的数据所在的位置，然后读取数据，最后读取数据存储到到磁盘高速缓存和内存中。在读取数据时，磁头首先要移动到数据所在的位置，这一操作所消耗的时间成为寻道时间。据统计，将磁头移到另一位置再读取相同大小数据所消耗时间是磁盘读取位于磁头所在位置的扇区内数据所消耗的时间的五倍。将应用程序运行所需的数据存储在磁盘上的连续磁盘簇中，将会有效的减少磁头的寻道时间[27]。

虽然在一般情况下磁盘碎片对正常工作影响并不太大，但是会很明显的降低磁盘访问速度。这主要是因为磁盘读取数据时需要在读取多个文件碎片，这些文件碎片都不连续，故需要磁头不断地移动，增加了等待硬盘盘片旋转到指定扇区所需要的时间和磁头移动到特定磁道所需要的寻道时间。

## 第四节 本章小结

本章首先介绍了常见的操作系统Windows平台和Linux平台上两种典型的文件系统——NTFS文件系统和EXT文件系统的结构。其次分析了文件系统碎片产生的原因，并就多个方面详细阐述了文件系统碎片产生的过程。最后分析了文件系统碎片对系统I/O性能的影响，详细介绍了文件碎片是如何影响系统I/O性能产生。

# 第三章 文件系统碎片度模型

## 第一节 文件系统碎片度定义

文件系统碎片化严重程度成为文件系统碎片度。文件系统碎片度用于描述一个文件系统的碎片化程度大小，包括文件碎片度和空闲空间碎片度。

## 第二节 影响文件系统碎片度的相关因素

提高计算机硬盘的存取速度大致可以从三个方面入手：一是设置适合的交错因子，使硬盘在较少的转动圈数下能够读取更多扇区中的数据；二是合理调整文件在磁盘上的物理存储位置，使文件在磁盘上能够在连续的磁盘簇中存放，进而达到 减少磁头读写时的寻道时间的目的；三是建立高速磁盘缓存，减少主机读写磁盘的次数和时间[13]。影响文件系统碎片化程度的因素有很多，主要有以下几种：

1. 文件碎片数量

文件系统中的碎片数是文件系统碎片化严重的标志。一般来讲文件系统中碎片数越多，则文件系统的碎片化程度越高。常见的磁盘碎片整理软件都以磁盘碎片数为碎片整理的重要指标，也是用户通常能直观了解的文件系统碎片化程度。

1. 文件大小

对于文件大小比较大的文件来说，在用户对其进行修改和操作的时候，该文件更容易超越本身所占有的磁盘簇，开辟出新的磁盘簇或删除原有的磁盘簇以保存被改动过的内容，从而在磁盘中产生了新的文件碎片。随着文件碎片数的增多，系统在读写文件过程中对I/O操作次数比较多，增大了用户的时间成本。并且在系统进行碎片整理时，大文件的搜索定位目标空间和移动都比较费时。

1. 文件碎片大小

对于单个文件碎片而言，碎片越小，文件的碎片化越严重。如一个大小为10M的文件，若有10个文件碎片，每个文件碎片的大小为1M，则在读取整个文件时，磁头需要跳转10次；若有2个文件碎片，每个文件碎片的大小为5M，则在读取整个文件时，磁头只需要跳转2次。因此单个文件碎片越小，则碎片化程度越严重。若整个磁盘上所有的碎片文件都是有破碎的小文件碎片组成，那么这个磁盘的碎片化程度高。

1. 文件使用频率

文件使用频率从用户使用角度影响磁盘的碎片化程度。用户总是关注于他们经常使用的文件，在系统实际的访问过程中，也更关心这些文件的访问效率。从某个角度而言，提高经常访问文件的性能则提高了系统I/O性能。对于不经常访问的文件，即使其碎片化程度严重，对于用户使用而言，也没有太大影响；但是对于系统而言，这些文件碎片依然存在。

1. 空闲空间碎片

空闲空间碎片，是指文件系统的空闲空间破碎存在。空闲空间碎片的存在，使得系统更容易产生文件碎片。如文件系统的空闲空间大小为10G，若这10G空间是零散存储为20块，每块大小约为500M，此时创建一个大小为1G的文件，由于空闲空间碎片存在，没有任何一个空闲空间大于1G，故该文件无法连续存储在磁盘上，导致该文件碎片化；若这10G空间是相对完整的存储为2块，每块大小为5G，那么在创建一个大小为1G的文件时，就可以使用其中任意一块空闲空间来完全存储该文件，不会产生文件碎片。因此空闲空间碎片化也是影响文件系统碎片化程度的因素之一。

## 第三节 文件系统碎片度评估模型

根据上述影响文件系统碎片度的因素，综合各个因素得出文件系统碎片度评估模型。文件系统碎片度可分为两个部分：文件碎片度和空闲空间碎片度。

### 3.3.1 文件碎片度

考率各个文件碎片度影响因素，文件碎片度，公式如下（其中n为文件系统文件数，为第i个文件的碎片数，为第i个文件使用频率，为第i个文件的第j个碎片的大小，以kb为单位）：





### 3.3.2 空闲空间碎片度

考虑空闲空间碎片度的影响因素，空闲空间碎片度，公式如下（其中n为空闲空间数，为第i个空闲空间的大小,以kb为单位）：





## 第四节 本章小结

本章定义了文件系统碎片度，并详细分析了影响文件系统碎片度的几个重要因素，阐述了各个因素是如何对文件系统碎片度产生影响。最后根据各个因素对文件系统碎片度的影响不同，综合几种重要因素，提出了文件系统碎片度评估模型。

# 第四章 制造空闲块碎片整理算法

## 第一节 制造空闲块算法原理

初始是指针指向磁盘头部，通过将文件系统中所有的文件移动到文件系统尾部，在指针指向的位置之后制造一块相对较大的空闲空间。按条件选出碎片文件移动到该空闲空间，使得该区域之前的文件全部连续。进而指针前移，移动到空闲空间末尾。重复上述过程，直到指针移动到磁盘尾部或无法再制造能容纳整个文件的空闲空间。

## 第二节 制造空闲块算法描述

1. 在第一次分析的时候，程序收集磁盘分区上所有文件的详细信息，包括是否为碎片文件、碎片文件的碎片列表，文件大小、文件全路径等。
2. 在优化的第一步，程序首先通过将所有的文件移到磁盘分区的末尾，尝试在磁盘分区的首部生成一块比较大的连续空闲区域。
3. 进行第二次磁盘分区分析，更新所有文件信息，空闲块信息。
4. 然后参照Defrag函数，对磁盘分区进行整理，最后得到的效果是，大碎片文件被移动到磁盘分区首部的连续空闲区域，剩下一些小的空闲块和碎片文件。
5. 优化的第二步，程序将所有剩下的文件（即尾部没有被整理的文件）全部移动到磁盘分区的首部。
6. 进行最后一次分析，更新磁盘分区信息。
7. 这个过程一直重复到SP（starting point）到达磁盘分区的末尾。SP在每一次重复这个过程之前向前移动跳过已经优化过的部分。

算法流程：

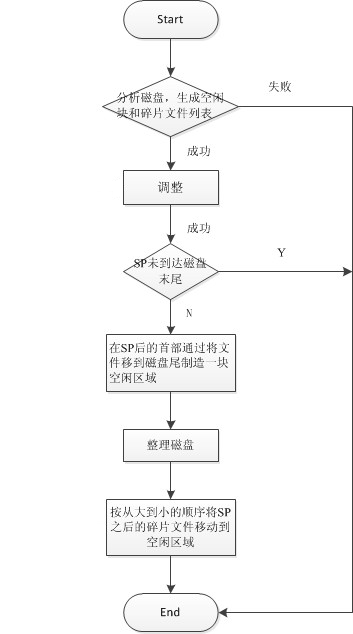


图4.1 算法流程图

## 第三节 制造空闲块算法测试与分析

### 4.3.1 测试环境

1. 软件环境
2. 虚拟机：Vmware7.0
3. 操作系统：Windows Server 2003
4. 磁盘分区文件系统：NTFS
5. 磁盘分区大小：20G
6. I/O测试工具：IOMeter
7. 硬件环境

表4.1 测试硬件环境

|  |  |  |
| --- | --- | --- |
|  | 虚拟机 | 宿主机 |
| 内存 | 1GB | 2GB |
| 处理器 | 1 | 2 |
| 硬盘 | SCSI | SCSI |
| CPU型号 | Intel 酷睿2双核 T6600 | Intel 酷睿2双核 T6600 |
| CPU主频 | 2.2GHz | 2.2GHz |

### 4.3.2 测试步骤

1. 在虚拟机中为添加一个硬盘，映射到宿主机中的磁盘文件1。将新添加的硬盘挂载到Windows Server 2003中，磁盘分区号为E。
2. 使用磁盘老化程序老化磁盘分区E，使得测试磁盘迅速碎片化。关闭虚拟机，保存文件1到宿主机的其他地方，命名为文件2，文件2记录了老化后的磁盘分区数据。
3. 使用文件2覆盖文件1，重启虚拟机，则此时Windows Server 2003中的分区E为老化后的状态，记录老化分区的原始数据。使用Windows系统工具中的“磁盘碎片整理程序”对磁盘分区E进行磁盘分析，记录初始的磁盘碎片信息，如碎片数、碎片文件数、碎片文件大小等。再使用I/O测试工具测试磁盘分区E的I/O速度即为碎片整理前老化分区的初始I/O速度。
4. 重启虚拟机，用文件2覆盖文件1。用制造空闲块碎片整理算法对磁盘分区E进行整理，整理结束以后，记录整理结果，如碎片数、碎片文件数、碎片文件大小等。再测试整理后的磁盘分区E的IO速度，记录测试结果。

### 4.3.3 测试结果及分析

1. 碎片整理磁盘分区E测试结果

表4.2 测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 磁盘碎片数 | 磁盘碎片文件数 | 碎片文件总大小 | 磁盘平均I/O速度 | 整理用时 |
| 整理前磁盘分区E | 583 | 87 | 3.3G | 13.41M/s |  |
| 制造空闲块算法整理后磁盘分区E | 56 | 28 | 1.5G | 8.47M/s | 10min |
| Windows自带整理程序整理后分区E | 216 | 52 | 2.2G | 8.20M/s | 20min |

续表4.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 磁盘碎片数 | 磁盘碎片文件数 | 碎片文件总大小 | 磁盘平均I/O速度 | 整理用时 |
| 整理前磁盘分区E | 583 | 87 | 3.3G | 13.41M/s |  |
| 制造空闲块算法整理后磁盘分区E | 56 | 28 | 1.5G | 8.47M/s | 10min |
| Windows自带整理程序整理后分区E | 216 | 52 | 2.2G | 8.20M/s | 20min |

（备注：此处只是为了显示“续表\*.\*”的用法，并非原文）

1. 测试结果分析

由表4.2可知，整理前的磁盘分E，磁盘文件碎片数和碎片文件数都很高，使用制造空闲块碎片整理算法整理后，磁盘文件碎片数和碎片文件数都有很大程度的减少。制造空闲块整理算法整理后磁盘的碎片数比Windows自带整理程序整理后磁盘碎片数要少，碎片文件数也要少。基于文件最近访问时间整理算法整理所用的时间比Windows自带整理程序用时短，但是就整理时间而言还是比较长的。

## 第四节 本章小结

本章介绍了一种文件系统碎片整理算法——制造空闲块碎片整理算法。首先介绍了该算法的原理，然后详细描述了该算法，包括使用的公式、数据结构等。接着实现该算法，并对其进行了算法整理效果测试，最后对测试结果进行了分析。

# 第五章 基于文件最近访问时间的碎片整理算法

## 第一节 基于文件最近访问时间算法原理

根据影响单个文件在碎片整理中的“值得”程度的因素，笔者针对单个碎片文件计算碎片文件在碎片整理中的“值得”程度。公式如下：



对每个碎片文件计算其在磁盘整理中的“值得”程度。按照其“值得”程度从大到小的顺序，对碎片文件进行整理。

## 第二节 基于文件最近访问时间算法描述

1. 根据整理价值公式计算单个碎片文件在碎片整理中的整理“值得”程度（不包括不可移动文件）
2. 根据计算出的各文件在碎片整理中的整理价值排序，这里举例根据文件“值得”程度排序后生成顺序链表为 A ->B-> C-> D-> E；
3. 执行指针指向顺序链表头部文件；
4. 位置指针指向磁盘头部；
5. 从 [位置指针所指部位] P0扫描磁盘，直到找出第一块可以容纳 [执行指针所指文件] F0的连续空闲块，若找到，执行f、g、h、j操作；若找不到，执行i、j操作；
6. 将F0移动到该空闲块；
7. 更新执行指针指向[链表的下一结点文件] F1；
8. 更新位置指针指向F0 尾部位置 P1；
9. 删除原F0 节点，把该节点的指向文件插入到F1节点后一节点。
10. 重复e过程；
11. 指针执行到链表尾部结束，得到整理价值最小的最优解；

## 第三节 基于文件最近访问时间算法测试与分析

### 5.3.1 测试环境

1. 软件环境
2. 虚拟机：Vmware7.0
3. 操作系统：Windows Server 2003
4. 磁盘分区文件系统：NTFS
5. 磁盘分区大小：20G
6. I/O测试工具：IOMeter
7. 硬件环境

表5.1 测试硬件环境

|  |  |  |
| --- | --- | --- |
|  | 虚拟机 | 宿主机 |
| 内存 | 1GB | 2GB |
| 处理器 | 1 | 2 |
| 硬盘 | SCSI | SCSI |
| CPU型号 | Intel 酷睿2双核 T6600 | Intel 酷睿2双核 T6600 |
| CPU主频 | 2.2GHz | 2.2GHz |

### 5.3.2 测试步骤

1. 在虚拟机中为添加一个硬盘，映射到宿主机中的磁盘文件1。将新添加的硬盘挂载到Windows Server 2003中，磁盘分区号为E。
2. 使用磁盘老化程序老化磁盘分区E，使得测试磁盘迅速碎片化。关闭虚拟机，保存文件1到宿主机的其他地方，命名为文件2，文件2记录了老化后的磁盘分区数据。
3. 使用文件2覆盖文件1，重启虚拟机，则此时Windows Server 2003中的分区E为老化后的状态，记录老化分区的原始数据。使用Windows系统工具中的“磁盘碎片整理程序”对磁盘分区E进行磁盘分析，记录初始的磁盘碎片信息，如碎片数、碎片文件数、碎片文件大小等。再使用I/O测试工具测试磁盘分区E的I/O速度即为碎片整理前老化分区的初始I/O速度。
4. 重启虚拟机，用文件2覆盖文件1。用基于文件最近访问时间碎片整理算法对磁盘分区E进行整理，整理结束以后，记录整理结果，如碎片数、碎片文件数、碎片文件大小等。再使用I/O测试工具测试整理后的磁盘分区E的IO速度，记录测试结果。

### 5.3.3 测试结果及分析

1. 碎片整理磁盘分区E测试结果

表5.2 测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 磁盘碎片数 | 磁盘碎片文件数 | 碎片文件总大小 | 磁盘平均I/O速度 | 整理用时 |
| 整理前磁盘分区E | 583 | 87 | 3.3G | 13.41M/s |  |
| 基于文件最近访问时间算法整理后磁盘分区E | 168 | 48 | 2.1G | 6.47M/s | 3min |
| Windows自带整理程序整理后分区E | 216 | 52 | 2.2G | 8.20M/s | 20min |

1. 测试结果分析

由表5.2可知，整理前的磁盘分E，磁盘文件碎片数和碎片文件数都很高，使用基于文件最近访问时间碎片整理算法整理后，磁盘文件碎片数和碎片文件数都有很大程度的减少。基于文件最近访问时间整理算法整理后磁盘的碎片数比Windows自带整理程序整理后磁盘碎片数要少，碎片文件数也要少。并且，基于文件最近访问时间整理算法整理所用的时间比Windows自带整理程序用时短。但是对于整理后整个磁盘的速度，基于文件最近访问时间整理后的整个磁盘速度比Windows自带整理程序整理程序整理后的慢，这是由于该算法主要是加快部分文件（最近访问文件）的I/O速度，而不关注整个磁盘的访问速度导致的。

## 第四节 本章小结

本章介绍了一种文件系统碎片整理算法——基于文件最近访问时间的碎片整理算法。首先介绍了该算法的原理，然后详细描述了该算法，包括使用的公式、数据结构等。接着实现该算法，并对其进行了算法整理效果测试，最后对测试结果进行了分析。

# 第六章 基于文件类型的碎片整理算法

## 第一节 基于文件类型算法原理

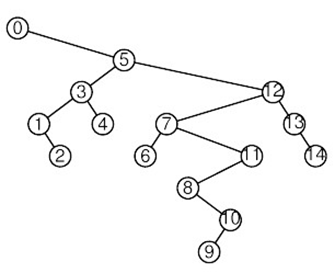
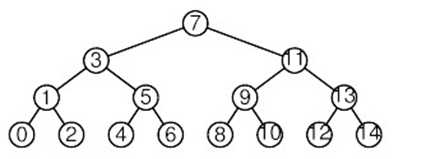
遍历磁盘分析模块生成的文件列表，对所有的碎片文件：

1. 首先根据其类型（目录文件、普通文件或者SpaceHog文件），优先在逻辑划分的区域内寻找能够容纳他的空闲块，返回找到的第一块能容纳该文件的空闲块；若在该区域内找不到能够容纳他的空闲块，则返回区域内的最大空闲块。
2. 若在该文件类型的区域里，找不到空闲块，则在整个磁盘范围内寻找空闲块。在整个磁盘区域内寻找空闲块的过程与在文件区域内一致。若找到容纳他的空闲块，则返回找到的第一块能容纳他的空闲块；找不到能容纳他的空闲块，则返回目前磁盘上最大的空闲块。
3. 若经过两次寻找都没有找到空闲块，则说明目前磁盘上已经没有空闲块。则不再继续整理。
4. 若找到的空闲块大小大于整个碎片文件大小，则将碎片文件移到该空闲块。
5. 若找到的空闲块大小小于整个碎片文件的大小，则遍历该碎片文件的每个碎片，跳过所有大小大于空闲块的文件碎片。找到第一个大小小于空闲块的碎片，将从该碎片LCN开始的大小为空闲块大小的数据块移动到空闲块。为碎片文件剩下的内容查找空闲块，直到整个碎片文件处理完毕。

## 第二节 基于文件类型算法描述

### 6.2.1 算法详细简述

1. 以二叉搜索树的形式组织文件。不论分区是NTFS文件系统或者是FAT文件系统；
2. 当获取到一个新的文件信息时往树中插入该文件信息。若树为空，则该文件作为根节点，Parent为空；若树不为空，则通过该文件的LCN（逻辑簇号）来尝试在树中找到其应该在位置，将其插入到叶子节点；
3. 插入以后更新插入的节点数，若新插入节点数大于1000，将该二叉搜索树转换为AVL树（平衡二叉搜索树），降低树的高度，增快查找速度。转换以后将新插入插入节点数更新为0；



（a）

（b）

图6.1 将二叉搜索树转换为AVL树

1. 计算各个区域的起始位置。理想状态，计算各个区域的起始位置，对磁盘进行分析，按四个区域，分别计算各种类型文件的总大小，包括可移动文件和不可移动文件。zone[0]的起始位置是磁盘首部，结尾位置是起始位置加上所有目录文件的大小；zone[1]的起始位置是zone[0]的结尾加上1%的磁盘大小（因为程序运行的时候会创建删除临时文件，若磁盘完全优化以后空闲区域将在所有文件之后，临时文件的访问较慢，故保留一点空闲区域给临时文件），结束位置是起始位置加上所有普通文件的大小。以此类推计算zone[2]和zone[3]的位置。对于MFT的保留区域，在以上计算完成以后，将其归在其LCN（Logical Cluster Number）所在的区域中；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 目录区域 | 1%空闲区域 | 普通文件 | 1%空闲区域 | SpaceHogs |

图6.2 磁盘逻辑划分

## 第三节 基于文件类型算法测试与分析

### 6.3.1 测试环境

1. 软件环境
2. 虚拟机：Vmware7.0
3. 操作系统：Windows Server 2003
4. 磁盘分区文件系统：NTFS
5. 磁盘分区大小：20G
6. I/O测试工具：IOMeter
7. 硬件环境

表6.1 测试硬件环境

|  |  |  |
| --- | --- | --- |
|  | 虚拟机 | 宿主机 |
| 内存 | 1GB | 2GB |
| 处理器 | 1 | 2 |
| 硬盘 | SCSI | SCSI |
| CPU型号 | Intel 酷睿2双核 T6600 | Intel 酷睿2双核 T6600 |
| CPU主频 | 2.2GHz | 2.2GHz |

### 6.3.2 测试步骤

1. 在虚拟机中为添加一个硬盘，映射到宿主机中的磁盘文件1。将新添加的硬盘挂载到Windows Server 2003中，磁盘分区号为E。
2. 使用磁盘老化程序老化磁盘分区E，使得测试磁盘迅速碎片化。关闭虚拟机，保存文件1到宿主机的其他地方，命名为文件2，文件2记录了老化后的磁盘分区数据。
3. 使用文件2覆盖文件1，重启虚拟机，则此时Windows Server 2003中的分区E为老化后的状态，记录老化分区的原始数据。使用Windows系统工具中的“磁盘碎片整理程序”对磁盘分区E进行磁盘分析，记录初始的磁盘碎片信息，如碎片数、碎片文件数、碎片文件大小等。再使用I/O测试工具测试磁盘分区E的I/O速度即为碎片整理前老化分区的初始I/O速度。
4. 重启虚拟机，用文件2覆盖文件1。用基于文件类型碎片整理算法对磁盘分区E进行整理，整理结束以后，记录整理结果，如碎片数、碎片文件数、碎片文件大小等。再使用I/O测试工具测试整理后的磁盘分区E的IO速度，记录测试结果。

### 6.3.3 测试结果及分析

1. 碎片整理磁盘分区E测试结果见表6.2。
2. 测试结果分析

由表6.2可知，整理前的磁盘分E，磁盘文件碎片数和碎片文件数都很高，使用基于文件类型碎片整理算法整理后，磁盘文件碎片数和碎片文件数都有很大程度的减少。基于文件类型碎片整理算法整理后磁盘的碎片数比Windows自带整理程序整理后磁盘碎片数要少，碎片文件数也要少。并且，基于文件类型碎片整理算法整理所用的时间比Windows自带整理程序用时短。

表6.2 测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 磁盘碎片数 | 磁盘碎片文件数 | 碎片文件总大小 | 磁盘平均I/O速度 | 整理用时 |
| 整理前磁盘分区E | 583 | 87 | 3.3G | 13.41M/s |  |
| 基于文件类型算法整理后磁盘分区E | 149 | 36 | 2.2G | 7.01M/s | 5min |
| Windows自带整理程序整理后分区E | 216 | 52 | 2.2G | 8.20M/s | 20min |

## 第四节 本章小结

本章介绍了一种文件系统碎片整理算法——基于文件类型的碎片整理算法。首先介绍了该算法的原理，然后详细描述了该算法，包括使用的公式、数据结构等。接着实现该算法，并对其进行了算法整理效果测试，最后对测试结果进行了分析。

# 第七章 AsDefrag碎片整理系统设计

## 第一节 总体设计

### 7.1.1 程序框架及说明

1. AsDefrag GUI为用户提供操作界面，负责响应用户请求。
2. AsDefragLib封装了磁盘分析、磁盘整理、磁盘优化等功能。
3. AgeVolume负责对磁盘进行老化，使得磁盘迅速碎片化。
4. ScanNtfs负责对NTFS文件系统的磁盘分区进行具体分析，获取MFT信息，生成文件列表和各个文件的碎片列表。
5. ScanFat负责对Fat文件系统的磁盘分区进行具体分析，扫描磁盘，获取文件信息，生成文件列表和各个文件的碎片列表。



图7.1 AsDefrag总体框架图

### 7.1.2 设计思路

程序使用多线程模型；用户线程用来支持用户界面，响应用户事件；工作线程用来处理用户命令。

使用Win32 API来实现程序功能，使得程序更具有通用性，增加程序的可移植性。

### 7.1.3 流程及说明

1. 用户线程响应用户事件，根据不同的用户事件，分类处理；
2. 若不为磁盘老化、磁盘分析、磁盘整理或者磁盘优化按钮被点击，则只在用户线程中响应事件。响应结束，程序返回；
3. 若为磁盘老化、磁盘分析、磁盘整理或者磁盘优化按钮被点击，则判断用户是否选中磁盘。若未选中磁盘，则弹出消息提示提示用户选择要操作的磁盘，程序返回；
4. 若用户选定磁盘，则根据被点击的按钮，创建相应的工作线程，传入线程所需的参数。若创建线程失败，程序返回；
5. 若创建线程成功，则调用在线程中相应的处理函数对选中磁盘进行处理；
6. 操作函数返回，线程结束。

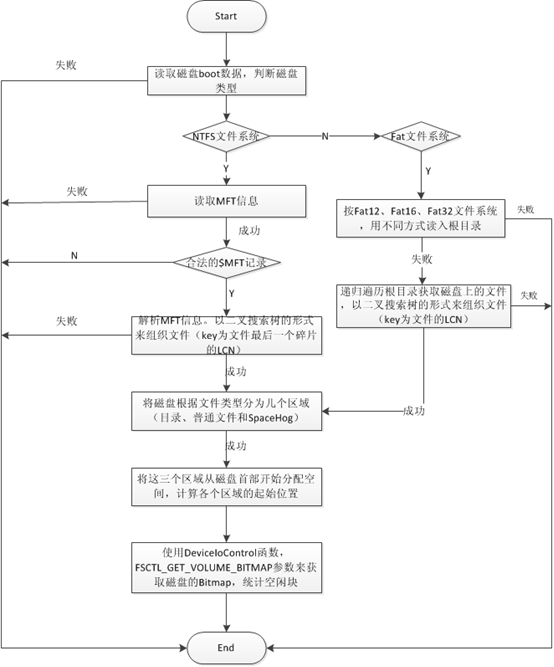


图7.2 磁盘分析模块流程图

## 第二节 模块划分

按功能将程序划分成以下几个模块：磁盘老化模块、磁盘整理模块、磁盘分析模块和磁盘优化模块。

### 7.2.1 磁盘老化模块

负责对选中的磁盘进行老化，通过随机创建文件、删除文件、修改已有文件等方式对磁盘进行频繁操作，使磁盘迅速碎片化。

### 7.2.2 磁盘分析模块

对选中的磁盘进行分析，获取分区的基本信息如分区大小、剩余大小、共有多少个簇、每个簇有多少个字节等。并通过扫描磁盘或者获取MFT信息（根据不同的文件系统选择）生成文件列表和各个文件的碎片列表。

### 7.2.3 磁盘整理模块

对磁盘上的碎片文件进行碎片整理，将碎片文件存储到连续的磁盘簇中，减少文件碎片数。

### 7.2.4 磁盘优化模块

用户可以根据自己的需要对磁盘进行优化。根据用户选择，将文件根据不同的文件类型在磁盘上分区域从区域首部到尾部有序排列。

## 第三节 详细设计

### 7.3.1 磁盘分析模块

1. 获取文件列表

首先读取磁盘引导信息，从而判断其文件系统类型。若为NTFS文件系统，则读取MFT信息，解析MFT信息，形成文件列表；若为FAT文件系统，则获得其根目录，对根目录进行遍历，扫描整个磁盘获得文件列表。以二叉搜索树的形式来组织磁盘文件，按文件的逻辑磁盘编号进行排序，以方便磁盘整理模块使用。

1. 获取空闲块列表

使用Win32API DeviceIoControl函数及其获取磁盘簇位图的控制参数来获取当前磁盘簇的位图，通过按位遍历该位图可以统计磁盘空闲块。

1. 计算各个区域的起始位置

将磁盘在逻辑上分为四个区域，从磁盘首部到尾部分别为目录区域、普通文件区域、SpaceHog区域（大小大于50M或者最近30天之内未访问的文件）和空闲区域。

排序依据：据统计，磁盘首部的IO速度比磁盘尾部快，有时甚至是磁盘尾部的两倍。目录文件是磁盘上最经常访问的文件故放在磁盘首部，普通文件其次，SpaceHog是不那么重要但占空间的文件故放在最后。

1. 算法描述

将普通二叉搜索树转换为平衡二叉搜索树分为两步，第一步将二叉搜索树（BST）转换为vine，第二步将vine转换为AVL树。

首先将一个二叉搜索树退化为类似于链表的数据结构（简称为vine）。Vine的每个节点最多有一个子节；

转换过程：

1. 我们使用指针p来遍历整棵树，从根节点开始，指针q记录p的根节点；
2. 若p没有右孩子，那么在这个部分已经是我们期望的形状，将q置为p，p置为q的右孩子，向下继续；
3. 若p有右孩子r，那么我们在p处进行翻转，r置为p的父节点，如图7.3中（c）所示。r变为新的p，若c是一棵空树，则我们继续向下走，否则p的右子树是一个比之前小的树，继续翻转；
4. 重复b)、c)直到p为叶子节点。

（c）

（a）

（b）

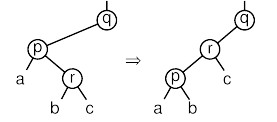
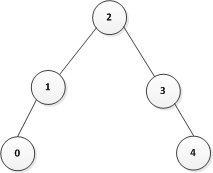
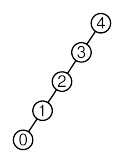


图7.3 将二叉搜索树（BST）转换为vine

部分代码：

q = tree->root;

p = tree->root;

while (p != NULL) {

if ( p->left == NULL) {//右节点为空，往左向下走

q = p;

p = p->left;

}

else{

/\*右节点不为空，在p处翻转，p为r的左孩子，

r的左孩子为p的右孩子。将r设置为p。\*/

r = p->right;

p->right = r->left;

r->left = p;

p = r;

q->left = r;

}

}

将vine转换为AVL树：设AVL的高度为h，则AVL的节点数 N 最多2^h – 1；若vine的节点数为m，且m不等于2^n– 1（n为非负数），则有m+1-2^n个节点需要单独作为一层。

算法简述：

1. 假设vine有m个节点，第一步首先将其转化为2^n-1个节点的情况。计算m-(2^n-1)即m+1-2^n的值。2^n的最大值为m除最左边的一位以外其他均置0的值。eg：m = 5 即0101，则2^n的值为0100即4，m = 96 即01100000 则2^n的值为01000000即64。
2. 将这m+1-2^n个节点压缩即从vine中删除作为其他节点的子节点，使vine的节点数为2^n-1。
3. 对vine进行压缩，从vine的根节点到尾部用黑色和红色交替进行着色，根节点为红色。对vine中除最尾部的节点以外的红色节点从vine中删除，将其及其子树作为前任黑色子节点的右子树。将黑色子节点的右子树，作为红色节点的左子树。
4. 将所有的红色节点置为白色。对vine剩下的节点重复c)，直到vine只有两个节点。

举例描述算法：

假设我们一个节点数为2^n-1的vine。如n=4，则vine有15个节点，我们可以通过三次压缩，将其转换为AVL树。

第一次：从根节点到叶子节点，为每一个节点着色，颜色在红和黑直接交替，根节点为红色。对除了叶子节点之外的所有红色节点，将其从vine中移除，将其及其子树作为前任黑色子节点的右子树。如图7.4所示。

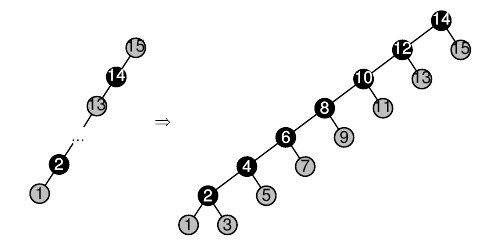


图7.4 第一次转换

第二次：将所有的红色节点置为白色。从根节点开始，间隔将黑色节点的颜色置为红色。和第一步一样，对vine除最后一个节点以外的所有红色节点，将其即其子树作为他前任黑色子节点的右子树。将黑色节点的右子树，作为红色节点的左子树。如图7.5所示。

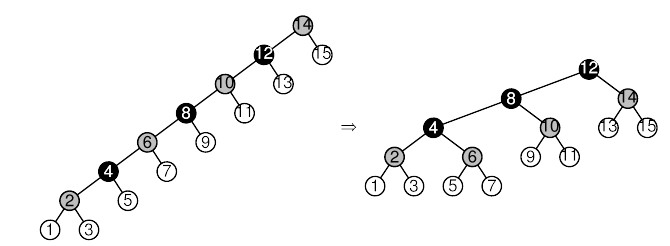


图7.5 第二次转换

第三次：第三步和第二步一样，将节点12和节点4置为红色，从vine中移出节点12，将其作为前任黑子数节点8及其孩子的右子树，节点8的右子树作为节点12的左子树。

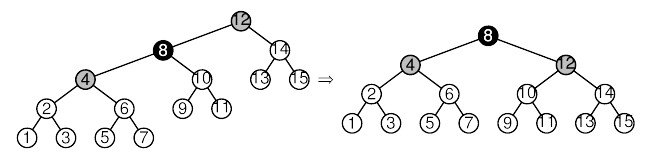


图7.6 第三次转换

部分代码：

unsigned long vine; /\* vine的节点数 \*/

unsigned long leaves; /\* m+1-2^n \*/

leaves = tree->bst\_count + 1;

for (;;){/\*获得m除了最左端一位位1，其余位为0的数，即2^n\*/

unsigned long next = leaves & (leaves - 1);

if (next == 0)

break;

leaves = next;

}

/\*获得m+1-2^n的值\*/

leaves = tree->bst\_count + 1 - leaves

/\*将m+1-2^n个节点压缩，使vine的节点数变为2^n-1\*/

compress (tree->bst\_root, leaves);

vine = tree->bst\_count - leaves; //2^n-1

while (vine > 1){

compress (tree->bst\_root,vine / 2);

vine /= 2;

}

### 7.3.2 磁盘整理模块

一、算法描述

采用第5章中基于文件类型的碎片整理算法。

二、模块流程图及说明

说明：

1. 首先对要进行整理的分区进行磁盘分析，获取该分区基本信息、文件列表信息及各文件的碎片列表；若磁盘分析失败，程序返回；
2. 将磁盘在逻辑上分为四个区域（分区域原则在磁盘分析模块中进行详细阐述）：目录区域、普通文件区域、SpaceHog区域和空闲区域。利用1中的文件列表计算这个四个区域分别的起始位置（具体计算过程在磁盘分析模块详细阐述）；
3. 对所有磁盘文件，按磁盘逻辑编号从小到大的顺序进行处理，以二叉搜索树的形式组织(1)中的文件列表，方便整理程序快速找到要处理的文件；
4. 对每一个磁盘文件，若该文件没有碎片（即文件内容存储在连续的磁盘簇中），则跳过该文件；否则，对碎片文件进行碎片整理；
5. 对每一个碎片文件，获取当前磁盘簇的位图，由于磁盘每个时刻都可能在使用状态，磁盘簇的位图随时在变化，故在每次为文件查找空闲块的时候都实时获取。
6. 根据文件类型，在其所属区域内寻找空闲块。若在所属区域内没有找到空闲块，则在整个磁盘内寻找空闲块。在所属区域内和在磁盘内寻找空闲块的方式一样：若有可以容纳整个碎片文件的空闲块，则返回第一个找到的可以容纳该空闲块的空闲块。若没有能容纳整个碎片文件的空闲块，则返回最大的空闲块；
7. 若没有找到空闲块，证明当前磁盘上已没有空闲块，程序返回；
8. 若找到的空闲块，且空闲块大小比正在处理的碎片文件大，则将整个碎片文件移动到该空闲块；
9. 若找到的空闲块，但空闲块大小比碎片文件小，则遍历该碎片文件的每个碎片，跳过所有比该空闲块大的文件碎片，找到第一个比空闲块小的文件碎片。若找不到比该空闲块小的文件碎片，则跳过该碎片文件。否则将从该文件碎片逻辑磁盘编号开始的大小为空闲块大小的文件数据块移动到该空闲块，更新碎片文件已被处理的数据大小。为文件剩下内容重复5-8，直到文件被处理完。

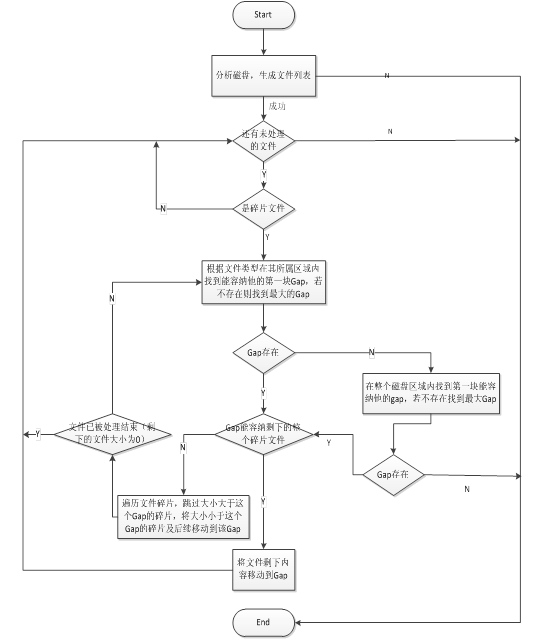


图7.7 磁盘整理模块流程图

三、数据结构及说明

用来记录NTFS格式磁盘分区基本信息的数据结构

struct NtfsDiskInfoStruct {

ULONG64 BytesPerSector; /\*每个扇区有多少个字节\*/

ULONG64 SectorsPerCluster; /\*每个簇有多少个扇区\*/

ULONG64 TotalSectors; /\*总共有多少个扇区\*/

ULONG64 MftStartLcn; /\*MFT区域起始LCN\*/

ULONG64 Mft2StartLcn; /\*MFT保留区域起始LCN\*/

ULONG64 BytesPerMftRecord; /\*MFT记录的字节数\*/

ULONG64 ClustersPerIndexRecord; /\*每个索引记录簇数\*/

};

用来记录单个文件碎片列表的数据结构

struct FragmentListStruct {

ULONG64 Lcn; /\* 当前碎片的LCN \*/

ULONG64 NextVcn; /\* 下一个碎片的LCN\*/

struct FragmentListStruct \*Next; /\*指向下一个碎片的指针\*/

};  /\*一个碎片文件的碎片列表\*/

用来记录磁盘上单个文件的数据结构

struct ItemStruct {

struct ItemStruct \*Parent; /\*父节点\*/

struct ItemStruct \*Smaller; /\*下一个小文件 \*/

struct ItemStruct \*Bigger; /\* 下一个大文件\*/

WCHAR \*LongFilename; /\*完整的文件名\*/

WCHAR \*LongPath; /\* 全路径\*/

WCHAR \*ShortFilename; /\* 缩略文件名(8.3 DOS). \*/

WCHAR \*ShortPath; /\* 全路径,缩略文件名 \*/

ULONG64 Bytes; /\* 总大小，以字节为单位 \*/

ULONG64 Clusters; /\* 总共的字节数\*/

ULONG64 CreationTime; /\*创建时间\*/

ULONG64 MftChangeTime;

ULONG64 LastAccessTime; /\*上次访问时间\*/

struct FragmentListStruct \*Fragments; /\*碎片列表. \*/

ULONG64 ParentInode; /\* 父目录的节点号\*/

struct ItemStruct \*ParentDirectory; / \*父目录\*/

BOOL Directory; /\* 是否为目录\*/

BOOL Unmovable; /\* 文件是否能被移动\*/

BOOL Exclude; /\* 是否被整理或优化\*/

BOOL SpaceHog; /\* 是否为大而不经常访问的文件\*/

};

### 7.3.3 磁盘优化模块

磁盘在使用过后，可能会存在一些小的空闲块，这些小的空闲块由于其太小而不能被使用。默认磁盘优化，会将这些小空闲块填充，使得文件尽量分布在磁盘前部，空出相对较大的空闲块，使得这些小的空闲块被利用起来。

根据用户需求不同，对磁盘使用侧重点不同，提供不同的磁盘优化模式。优化部分文件的访问速度。算法描述如下：

按某个关键字排序优化：

1. 对磁盘进行分析，获取磁盘文件列表及各个文件的碎片列表。以二叉搜索树的形式组织文件列表，以链表形式组织各个文件的碎片列表；
2. 逻辑将磁盘分为4个区域（分区域具体原则在磁盘分析模块中详细阐述）:目录区域、普通文件区域、SpaceHog区域、空闲区域。计算各个区域在磁盘上的逻辑起始位置（LCN）；
3. 对每一个区域，按照某关键字排序的顺序来处理类型与区域类型一直的文件。若该关键字排序时有相同的值,则一次按照文件大小、最近访问时间、最近修改时间和创建时间的顺序进行比较，其中任何一个值不相等都返回。若上述4个字段都相等，则比较两个文件逻辑磁盘编号，逻辑磁盘编号小的文件大。指针初始时指到区域的起始位置；
4. 将待处理文件移动到指针指向位置之后。若该位置之后有大于或者等于待处理文件大小的空闲块，则直接将待处理文件移到该空闲块，指针向前移动到文件尾部；
5. 若该位置之后没有大小大于或者等于待处理文件的空闲块，则通过将占据该位置文件向后移动，为待处理文件制造出能容纳他的空闲块。碰到不可移动文件则返回，将待处理文件部分移动到已经制造出的空闲块里。指针前移跳过不可移动文件。继续为待处理文件剩下内容制造空闲块，直到文件处理完毕，更新指针位置；

算法流程图如图7.8所示。

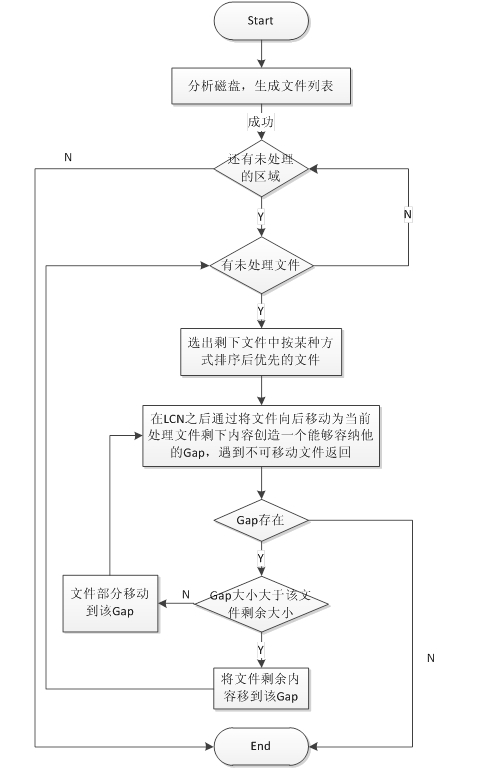


图7.8 按某个关键字优化流程图

默认方式优化：

1. 对磁盘进行分析，获取磁盘文件列表及各个文件的碎片列表。以二叉搜索树的形式组织文件列表，以链表形式组织各个文件的碎片列表；

具体获取文件碎片列表方法：调用DeviceIoControl函数，传入文件句柄，获取文件碎片列表的控制参数，存储碎片列表的缓冲区。通过遍历解析返回的数据可以得到各个文件的碎片列表。

部分代码：

STARTING\_VCN\_INPUT\_BUFFER RetrieveParam;

struct {

DWORD ExtentCount;

ULONG64 StartingVcn;

struct {

ULONG64 NextVcn;

ULONG64 Lcn;

Extents[1000];

} ExtentData;

RetrieveParam.StartingVcn.QuadPart = Vcn;

ErrorCode = DeviceIoControl(FileHandle,

FSCTL\_GET\_RETRIEVAL\_POINTERS,

&RetrieveParam,sizeof(RetrieveParam),

&ExtentData,sizeof(ExtentData),&w,NULL);

1. 逻辑对磁盘进行分为四个区域（分区域原则在磁盘分析模块中详细阐述）：目录区域、普通文件区域、SpaceHog区域、空闲区域。利用(1)中获得的文件列表，计算各个区域的起始位置；
2. 对每一个逻辑区域中的每一个空闲块，用大于空闲块逻辑磁盘编号，与逻辑分区类型一致的并且大小比该空闲块小的文件来填充该空闲块。优先查找大小等于该空闲块大小，最后一个碎片块的起始LCN比空闲块LCN大的与逻辑分区类型一致的文件；若找不到，则查找大小小于该空闲块，最后一个碎片块的起始LCN最大的与逻辑分区类型一致文件；
3. 若找到符合要求的文件，则将该文件移动到空闲块；
4. 若没找到符合要求的文件，则跳过该空闲块；

算法流程图：

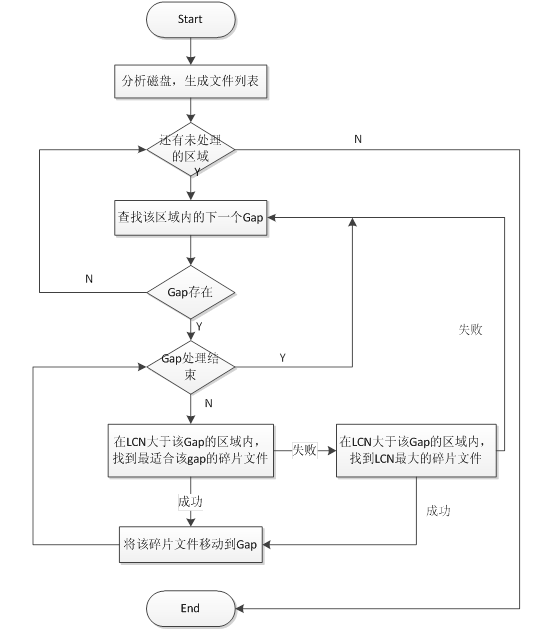


图7.9 磁盘优化默认方式流程图

### 7.3.4 磁盘老化模块

在指定分区，不断随机创建文件、删除文件、修改文件，使得磁盘碎片化。

磁盘为文件分配空间，是以磁盘簇为单位的，即一个磁盘簇只能属于一个文件，故每次写数据时，若数据大小不为磁盘簇大小的整数倍，更容易使文件产生碎片。算法详述如下：

1. 获取分区的基本信息，包括分区总大小、分区剩余大小、每个簇包含的字节数等；

具体方法：使用Win32 API DeviceIoControl函数来向设备发送命令。传入磁盘句柄，FSCTL\_GET\_NTFS\_VOLUME\_DATA参数为获取NTFS文件系统分区基本信息控制代码，和用于存储磁盘基本信息的缓冲区。

部分代码：

NTFS\_DATA NtfsData;

DWORD w;

ErrorCode = DeviceIoControl(

Data->Disk.VolumeHandle,

FSCTL\_GET\_NTFS\_VOLUME\_DATA,

NULL,0,&NtfsData,sizeof(NtfsData),

&w,NULL);

1. 程序中记录老化文件列表，包括文件全路径，文件大小等；
2. 生成随机数来判断是创建文件、删除文件还是修改文件；
3. 若为创建文件，生成随机数决定创建文件的数量，并随即决定写入的数据文件大小等；
4. 此处写入数据并不是真正写入数据，而是通过lseek到一个目前文件不存在的位置，然后在该位置写下一个字节的内容，为文件制造一个空洞，这种方法既可以改变文件大小，又大量的减少了读写数据的IO时间；

示例代码：

/\*以二进制写模式打开\*/

char end = EOF;

FILE \*fp = fopen(tempFile.filePath.c\_str(),"wb"); /\*从文件结尾便宜Size个字节\*/

fseek(fp,Size,SEEK\_END);

/\*在偏移位置，写下EOF（表示文件结尾）\*/

fwrite(&end,1,1,fp);

fclose(fp);

1. 若为删除文件，生成随机数决定删除文件的数量（删除文件的数量要小于所有已创建文件的一半），并随机决定删除哪些文件（删除的文件只能从老化程序写入的文件中取）等；
2. 若为修改文件，生成随机数判断是写入数据还是删除数据，写入或删除数据的大小等。修改方式与4中相似，使用lseek的方式，修改指针位置，再写入内容或者删除内容，减小或增大文件空洞，修改文件大小，减少IO时间；
3. 根据上述修改，更新老化文件列表；
4. 重复3-7，直到循环次数结束，或者老化文件大小大于磁盘剩余大小的80%，为磁盘留下一定的空闲空间，避免由于磁盘空闲空间太小导致磁盘整理程序整理效果不好。

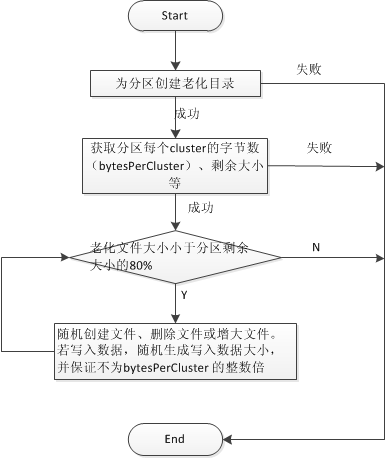


图7.10 磁盘老化模块流程图

## 第四节 原型实现

### 7.4.1 程序界面

程序界面主要包括三个部分：分区信息显示，程序输出，用户输入。

分区信息显示界面主要是向用户显示分区信息，包括磁盘上所有的分区名称，编号，分区总大小、分区剩余大小等。用户可以在该界面上选择想要操作的分区，对其进行相应操作。

程序输出界面，主要用于向用户输出程序信息。用户选择对一个磁盘进行分析时，该区域内就会显示与磁盘分析相关的信息，如操作分区的名称，文件系统类型，分区基本信息（磁盘簇个数，每个磁盘簇的字节数等）。并显示程序处理结果。

用户输入界面，用来接收用户输入，如用户选择操作分区，对选择的分区进行某种操作等。

程序主界面如图7.11所示：

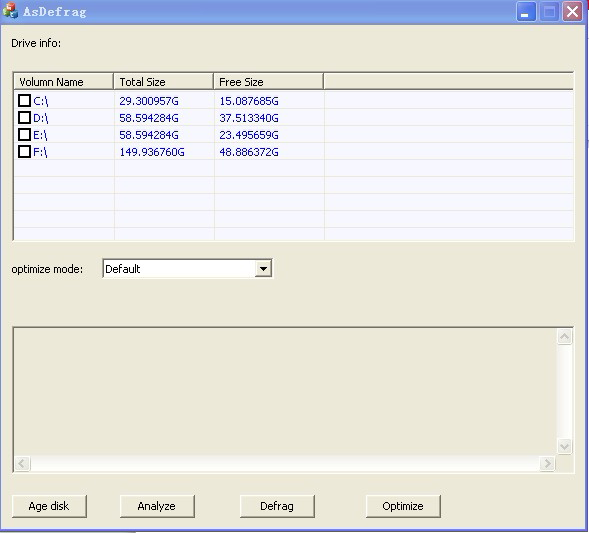


图7.11 程序主界面

### 7.4.2 分区信息

分区界面主要是为了用户更加直观的了解分区使用情况，选择操作分区。Volume Name表示分区名称，Total Size表示分区总大小，Free Size表示分区剩余大小。界面如图7.12所示。

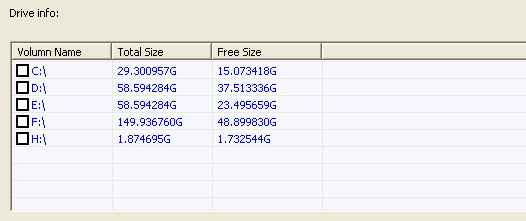


图7.12 磁盘分区信息显示

获取分区信息的局部代码如下：

TCHAR szDriverBuffer[512];

CString csDriver;

DWORD dwLength = GetLogicalDriveStrings(sizeof(szDriverBuffer)/sizeof(TCHAR),OUT szDriverBuffer); /\*获得分区表\*/

TCHAR \*lpDriverBuffer = szDriverBuffer;

while (\*lpDriverBuffer != NULL)

{

sDriver = lpDriverBuffer;

lpDriverBuffer = lpDriverBuffer + csDriver.GetLength() + 1;

/\*获得磁盘空间信息\*/

ULARGE\_INTEGER FreeAv,TotalBytes,FreeBytes;

/\*获得各个分区的空间信息\*/

GetDiskFreeSpaceEx(csDriver,&FreeAv,&TotalBytes,&FreeBytes);

}

### 7.4.3 程序输出

程序输出界面主要是向用户输出程序信息，让用户了解命令执行的情况，输出命令执行的结果。界面如下：

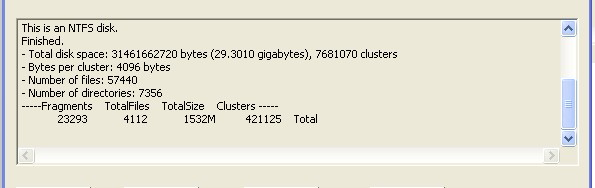


图7.13 程序输出显示

程序输出的局部代码如下：

/\*该函数可以传入多个参数，第一个参数是输出的格式，后续参数是输出的内容，这样的设置，任意调用该函数的函数，可以输出任意自定义格式的内容，且不限制参数个数\*/

void LogMessage(WCHAR \*Format, ...)

{

va\_list VarArgs;

/\*如果没有输出信息 \*/

if (Format == NULL) return;

CEdit\* m\_lblInfo = (CEdit\*)AfxGetApp()->m\_pMainWnd->GetDlgItem(lblInfo);

wchar\_t\* sour = new wchar\_t[1024];

CString s = "";

m\_lblInfo->GetWindowTextA(s);

/\*处理输出内容\*/

va\_start(VarArgs,Format);

\_vswprintf\_s\_l(sour,1024,Format,\_get\_current\_locale(),VarArgs);

va\_end(VarArgs);

s += sour;

/\*输出到用户界面\*/

m\_lblInfo->SetWindowText(s);

}

### 7.4.4 用户输入

用户输入界面主要是用于用户向程序输入命令，对要操作的分区发送操作命令。界面如图7.14所示。



图7.14 用户输入

接收用户命令创建工作线程执行命令，局部代码如下：

/\*创建工作线程\*/

defragThread = CreateThread(NULL,0,(LPTHREAD\_START\_ROUTINE)ProcJob,

(LPVOID)&defrag\_volumes,0,&defragThreadID);

/\*工作线程函数\*/

void ProcJob(LPVOID pParam)

{

vector<jobInfo>\* jobs = (vector<jobInfo>\*)pParam;

int i = 0,OptimizeMode,Speed;

double FreeSpace;

WCHAR \*\*Excludes;

WCHAR \*\*SpaceHogs;

wchar\_t volumeName[256];

/\* 设定程序默认值\*/

Debug = 1;

OptimizeMode = 0;

Speed = 100;

FreeSpace = 1;

Excludes = NULL;

SpaceHogs = NULL;

IamRunning = RUNNING;

while (i < jobs->size() && WaitForSingleObject(hExitEvent, 0) != WAIT\_TIMEOUT)

{

OptimizeMode = jobs->at(i).mode;

volumeName = jobs->at(i).volumeName.GetBuffer();

RunDefrag(volumeName,OptimizeMode,Speed,FreeSpace,Excludes,SpaceHogs,&Running,&RedrawScreen,ShowStatus,ShowMove,ShowAnalyze,ShowDebug,DrawCluster,ClearScreen,NULL);

i++;

}

}

# 第八章 总结与展望

## 第一节 本文总结

该项目主要分为三个阶段，初期主要是对Windows和Linux平台上的典型的文件系统结构进行分析。接下来对文件系统碎片产生的原因进行了深入分析，并提出了文件系统碎片度的概念。有了前期工作以后，对影响文件系统碎片度的主要因素进行了分析，提出文件系统碎片度评估模型，衡量文件系统碎片化的严重程度。其次，提出了两种文件系统碎片整理算法，优化系统I/O性能。接下来基于上述提出的碎片整理算法，开发了一款磁盘碎片整理软件——AsDefrag，并对提出的碎片整理算法进行测试。

到目前为止，取得了一定的成绩。第一，对主要操作系统平台上的典型文件系统结构进行了分析，深入研究了文件系统碎片产生的原因。第二，提出了文件系统碎片度的概念，并为评估文件系统碎片度及文件系统碎片化严重程度提供了评估模型。第三，提出了两种碎片整理算法，优化文件系统I/O性能。第四，开发了一款基于上述碎片整理算法的磁盘碎片整理软件，并对碎片整理算法效果进行了测试。

## 第二节 进一步展望

目前该研究还存在很多不足。第一，文件系统碎片度评估模型还不够完善，需要不断调整。第二，碎片整理算法的整理速度还是不够快捷，如何集快速和性能于一身是要解决的问题，对几块主流的碎片整理软件进行测试，有的软件整理效果很好但速度很慢，有的软件整理速度很快但效果差强人意。整理速度慢，主要是由于IO操作多导致的，需要减少IO次数；第三，碎片整理算法的效果比较依赖于磁盘的剩余空间大小，在剩余空间小的时候，碎片整理效果差强人意。

鉴于以上提出的几点，还需要不断对文件系统及文件系统碎片进行研究，形成比较通用的文件系统碎片度评估模型，不断改进碎片整理算法，兼顾整理效果和整理速度，在恶劣的磁盘环境下有效的进行碎片整理。

# 参考文献

1. William Boswell.Inside Windows Server 2003 [M].America:Addison—Wesley,2003:866.
2. Brian Knitte1.Windows XP under the hood [M].America:Que.Publishing,2002:649.
3. Mark E Russinovich, David A Solomon . Microsoft Windows Intemals [M].4med.America: Microsoft Press.2005:875—879.
4. Takashi Sato. ext4 online defragmentation. In: Proceedings of the Linux Symposium. Ottawa, Canada: Linux Symposium, 2007. 179~186
5. Bjmae Stroustrup. The C++ Programming Language. AddisonWelys,1996
6. 王侠,高胜哲. [Linux的文件系统](http://dlib.edu.cnki.net/kns50/detailref.aspx?filename=LSGZ200502016&dbname=CJFD2005&filetitle=Linux%e7%9a%84%e6%96%87%e4%bb%b6%e7%b3%bb%e7%bb%9f" \t "_top)[J]. [辽宁税务高等专科学校学报](http://dlib.edu.cnki.net/kns50/Navi/Bridge.aspx?DBCode=CJFD&LinkType=BaseLink&Field=BaseID&TableName=CJFDBASEINFO&NaviLink=%e8%be%bd%e5%ae%81%e7%a8%8e%e5%8a%a1%e9%ab%98%e7%ad%89%e4%b8%93%e7%a7%91%e5%ad%a6%e6%a0%a1%e5%ad%a6%e6%8a%a5&Value=LSGZ" \t "_blank), [2005](http://dlib.edu.cnki.net/kns50/Navi/Bridge.aspx?DBCode=CJFD&LinkType=IssueLink&Field=BaseID*year*issue&TableName=CJFDYEARINFO&Value=LSGZ*2005*02&NaviLink=%e8%be%bd%e5%ae%81%e7%a8%8e%e5%8a%a1%e9%ab%98%e7%ad%89%e4%b8%93%e7%a7%91%e5%ad%a6%e6%a0%a1%e5%ad%a6%e6%8a%a5" \t "_blank)
7. 汤自瀛,哲凤屏,汤小丹. 计算机操作系统.西安：西安电子科技大学出版社,2001.
8. 陈莉君,康华.Linux操作系统原理与应用[M].北京:清华大学出版社,2006.
9. 王艳平.程序设计.北京:人民邮电出版社，2005
10. 邹鹏，王广芳.操作系统原理.长沙：国防科技大学出版社，1995
11. 张新荣.计算机组成原理教程.北京：北京希望电子出版社，2002
12. 王澄, 董玮文, 杨宇航, 等. 影响VoD 服务器I/O 性能的关键因素[J] . 计算机工程, 2002, 28( 7) : 140-142.
13. SCOTT Mueller. Micro House PC Hardware Library: Vol 1 [M].Indianapolis Ind: QUC Pub,1998.
14. Smith K. File System Aging Increasing the Relevance of File System Benchmarks[C] //Proceedings of the 1997 SIGMETRICS Conference , Seattle, WA. 1997-06: 203-213.
15. DavidASolomon. Windows NT 技术内幕[M]. 第2 版.北京:清华大学出版社,1999.
16. Helen Custer. Inside the windows NT[M].Microsoft Press, 1993.
17. 王立柱.C/C++与数据结构.北京：清华大学出版社，2002
18. Bruce Eckel. Thinking in C++. Alan Apt,2002
19. Jim Beveridge.Win32多线程程序设计.候捷译.武昌：华中科技大学出版社，2002
20. Lippman. C++ Primer. Addison Welsy,1998
21. Abraham Silberschatz,Peter Baer Galvin,Greg Gagne.《操作系统概念》[M]，北京：高等教育出版社，2002.5
22. Giel de Nijs, Ard Biesheuvel, Ad Denissen et al. The Effects of Filesystem Fragmentation. In: Proceedings of the Linux Symposium. Ottawa, Canada: Linux Symposium, 2006. 193~208
23. 吕会红, 陈平仲. 磁盘碎片对计算机系统性能的影响. 现代计算机（专业版）, 2003,

(7): 99~100

1. Andrew Whitaker, Marianne Shaw, Steven Gribble. Scale and Performance in the Denali Isolation Kernel. ACM SIGOPS Operating Systems Review. USA: ACM Press, 2002, 36(SI): 195~209
2. Avantika Mathur, Mingming Cao, Suparna Bhattacharya. The new ext4 filesystem: current status and future plans. In: Proceedings of Linux Symposium, Ottawa, Canada. Linux Symposium, 2007. 21~33
3. Andrew Warfield, Russ Ross, Keir Fraser, et al. Parallax: Managing storage for a million machines. In: Proceedings of 10th Hot Topics in Operating Systems. Santa Fe, New Mexico, USA: USENIX Association, 2005. 1~11
4. Drew Robb. Defragmenting really speeds up WINDOWS NT machines. IEEE Spectrum, 2000, 37(9): 74~77
5. Shahram Ghandeharizadeh, Doug Ierardi. An Algorithm for Disk Space Management to Minimize Seeks. The Computer Journal, 1996, 57: 75~81
6. NSTL Company. System Performance and File Fragmentation on In Windows NT. NSTL White Paper, 1999
7. Nianlong Yin. Reducing application load time by rearranging disk data: [Master thesis]. USA: Brigham Young University, 1998
8. Jeffrey Katcher. PostMark: A New File System Benchmark. Technical Report TR3022. Network Appliance.1997

致 谢

值此论文顺利完成之际，我首先要向我的导师\*\*\*老师表达深深的敬意感谢。本论文是在\*老师的悉心指导之下完成的。在课题的研究过程中，\*老师每次对我的疑问都给予细心的解答并给出建议，使笔者顺利的解决了研究过程中遇到的各种困难。在撰写论文的过程中他还对我进行了细心的指导，认真阅读和修改了我的论文，使我顺利的完成了论文撰写。

同时，感谢和我一起完成研究和开发项目的团队成员姜申坪同学。我们在一起度过了很多辛苦但是快乐的研究时光。在她的辛勤付出下，项目的研究工作才得以更好的完成。

感谢所有在大学四年期间，帮助过我的老师们，是你们引导我一步一步向前走，让我不断成长。

感谢我所有的同学、朋友，感谢你们四年的陪伴和宽容，陪伴我走过四年的大学时光。

再次感谢你们，谢谢你们为我所做的一切。