学校代码： 10246
学　　号： 093053052

# 硕士学位论文

（专 业 学 位）

# J2ME 游戏的通用图形特效库实现及优化

院　　系： 软件学院

专　　业： 软件工程

姓　　名： 刘宇

指 导 教 师： 徐迎晓

完 成 日 期： 2010/08/15

# 目 录

# 摘 要

当前，随着信息技术的日新月异，手机游戏已经成为现代人尤其是现代年轻人一个重要的娱乐方式。全球手机游戏的市场已经达到一个很大的规模，开发手机游戏的个人和公司也不断的涌现，为广大手机用户提供了很多益智有趣的游戏。但是在手机游戏大量开发的同时，由于手机终端的性能巨大限制以及大部分手机游戏是基于 J2ME 开发所带来的速度限制，大量开发过程中实际问题需要解决。其中最让玩家和开发者诟病的是，手机游戏的图形特效难以达到令人满意的效果，同时在一定效果的基础上无法达到令人满意的速度，造成游戏的可玩性比较差，直接影响了手机游戏用户的游戏体验和手机游戏的市场拓展。

图形特效库是手机游戏开发过程中的应用技术积累，它一般针对某一类型的游戏入手，并且拓展，形成一套性能良好，效果良好的 2D 特效。在游戏开发中针对某一类型游戏开发特效库，是非常有必要的。特别是操控类游戏（赛车，滑雪，飞行，射击游戏等)，更有这样的迫切需求。不仅可以实现尽可能美观，实用的效果，在实现效果的基础上优化速度，达到一个让人满意的游戏体验，提高手机游戏的品质。本文正是从这个角度入手，进行探讨。

本文详细的分析了现有 J2ME 游戏开发中的不足以及特效方面的需求，在 J2ME 平台提供的图像精灵的基础上进行了大幅度的改进，使之更适合游戏开发的需求。在改进现有系统结构的基础上建立了一个完善的特效库，并且针对不同的游戏特性、粒子系统以及基于改进图像精灵的游戏特效进行了实现。并进行了性能的优化和测试比较，验证了特效库开发的成果。

**关键词** 手机游戏，J2ME，特效处理，性能优化

# ABSTRACT

By the rapid development of information technology, mobile phones are widely used and mobile games have become important entertainment to the modern people especially the youth. The mobile games has reached a very large market around the world and the amount of the game developers and companies grown up. Many good games have been published. But behind the large amount of games, the develop is facing great problems by the limit of the handset's performance and the J2ME platform. The worst part of develop is that the handsets is too slow to perform satisfying effects when drawing too much things. The low speed is destroying consumers' game experience.

The graphic effects library is a technique used when developing a mobile game, it is developed recording to certain kind of game and provides nice 2D graphics effects with acceptable painting speed. Developing a graphics effect library is quite necessary when developing a game, especially when developing a Control Game (like car racing、flying、shooting games etc).By this technique we can implement nice effects in game with acceptable speed after optimization. The graphic effects library can help the game developers to improve the quality of mobile game and provide a nice game experience to consumers. This article will focus on this point.

This thesis analyzed the shortcoming part of current mobile game developing and the needs in graphic effects, improved the image sprite system provided by J2ME, discussed and optimized current develop techniques and build a graphic effects library based on the improvement. This effects library implements effects on different kinds of games, particle systems and improved image sprite engine. At the end this thesis discussed the optimization design of the library and proves the achievement of this article by test and compare.

**Keywords** Mobile Games,J2ME,Special Effects,Optimization

# 第一章 绪 论

## 1.1 选题背景

在现代移动通信业务中，业务的重点已经慢慢的转移，运营商的收入增长点已经由传统的短信、电话业务转移到新的移动增值业务和互动娱乐业务上。大量不同种类新的增值业务和娱乐业务投入市场，包括大量 WEB2.0 企业开发的手机社区，手机上的社区客户端等，还包括手机视频应用，手机游戏应用等等。其中手机游戏在诸多的移动娱乐内容服务和互动娱乐服务中，最受到消费者的青睐，一款优秀的手机游戏可以让用户在短小的休息时间内随时随地进行娱乐。手机上的游戏应用以其娱乐性，移动性，方便性得到了手机用户的欢迎。手机游戏的市场规模在年年升高。调查显示，全球的手机游戏市场用户规模从 2005 年的 2.9 亿增至 2008 年的 10.3 亿，市场收入从 2007 年的 32 亿美元增加到 2008 年的 40 亿美元。[1]

反观中国，手机游戏经过 6 年的发展历史，用户增长速度十分惊人，到 2008 年底达到了 700 万人，到 2009 年底手机游戏的活跃用户达到了 1000 万人。尤其是中国发放 3G 拍照，大规模铺设 3G 网络以来，前期困扰中国手机游戏业务发展的一些因素，如产品质量、盗版、资费等问题正在得到逐步解决，整个手机游戏产业进入了快速发展的新阶段。根据艾瑞咨询的分析报告，2009 年，中国手机游戏产业的销售额将达到 18 亿元。[1]

无论是国内国外，手机游戏的销售额都在年年的大幅度的增长。同时行业的规模还在不断的增大，新的手机游戏公司越来越多，手机游戏的从业人员包含程序开发人员、美术设计、游戏策划等等人才都已经到达了供不应求的地步。同时各方面的开发技术也不断的成熟起来，各大手机厂商不停的投入人力、财力用来改善手机研发的技术、开拓新的手机游戏领域。手机网游、手机网页游戏客户端等手机游戏拓展类产品也在国内迅速发展。

同时，手机游戏产业的产品作为一种信息技术的产品，其主要传播载体为互联网下载以及通过运营商手机下载，相比传统的产业具有无需运输，直接面对互联网和大量手机用户的先天优势。作为中国的高科技企业，制作一款游戏不仅可以让国内消费者体验到游戏的乐趣，也很容易的就可以传播到国外。与国外运营商合作或者开放互联网的收费下载可以很轻易的进行市场拓展，在全世界收益。这不仅是企业的发展机遇，也符合了国家所倡导的高科技低能耗，低污染的经济

发展战略方向。

因此，在市场如此大好的情况下，对手机游戏开发技术的研究，并有重要的意义。

## 1.2 研究意义

在手机游戏中，给用户最直接最有效的体验无疑是一个美观的画面效果。一个游戏的画面、特效质量直接决定了一款游戏的质量。在 J2ME 游戏开发的过程中，由于性能的限制，如何快速有效的开发出画面特效优良的游戏已经成为许多厂商关注的热点问题。在传统的使用 MIDP 提供的图形函数的开发过程中，制作一个特效需要消耗大量的美术图片资源，由于手机的应用程序包大小和内存、运算速度的限制，这样做基本上是不可能的。因此，想要使用传统的开发框架在一款手机游戏中实现比较多、比较酷的特效比较困难。

针对上述情况，本文在传统的 MIDP 图形框架的基础上进行大规模的改进，优化了动画引擎，通过加入调色板、半透明等技术增强了图形引擎的功能，并在此基础上开发了一个包含粒子系统、资源特效系统等功能强大的特效库，不但使得手机游戏的特效开发更为简便高效，画面更为优美，也使得图形特效在手机存储空间的占用方面大大减少，资源消耗更少。不但提高了开发效率，还提高了手机游戏的质量。这无疑使得游戏的档次得到了提高，具有了更强的市场竞争力。

## 1.3 目前手机游戏特效开发中的主要问题

手机在图形图像开发过程中需要考虑手机的性能限制。美观的效果和内存、资源的消耗是一对矛盾。面临的主要问题有：

1. 运算速度的限制：手机由于电源功耗和体积的限制，决定了它的电路规模比较小，相应的手机的处理芯片也相对较慢。再加上 J2ME 手机上的应用运行于 JVM 之上，性能更受到影响。手机游戏开发的过程中，一旦同一屏幕绘制的图形增多，游戏中的动画帧数就会急剧下降。会给玩家的游戏体验带来糟糕的影响。
2. 包大小和内存大小的限制：手机的内存一般都比较小，在几百 K-5M 左右，而游戏的安装包最多也在几百 K。在这样的限制下，开发出的手机游戏的丰富程度受到比较大的限制，必须要做大量的优化设计才能做出让人满意的游戏来，对于很多比较小的公司来说，缺乏成熟的工具和配套优化技术，很难做出像大公司作品一样的高品质游戏来。

3. MIDP 提供的功能有限：作为一个底层函数库，MIDP 所提供的功能非常的简单，只能够满足简单开发的需求。想开发一款比较好的游戏，需要对图形部分进行拓展才能满足开发的需求。

## 1.4 国内外目前研究现状

目前国内外的手机游戏开发公司越来越多，各个公司都有一些图像优化、动画工具优化等相关的技术，在互联网上也可以查询到许多有关于手机开发的小窍门等等。整个开发领域可以说非常的红火，在欧美比较著名的有 Gameloft、EA Mobile、GLU 等著名的公司，国内也不乏数位红等优秀的公司，他们都有很多成熟的开发优化、工具。

但是在图形引擎上建立的特效库方面情况有所不同，整体还比较缺乏，尤其是针对 J2ME 游戏开发 2D 像素的特效库，基本上还没有成熟的工具。由于各种性能限制较大，开发比较困难，很多公司并没有在此领域进行开发，也有一些公司有了一些相关的技术由于市场竞争等因素无法公开，对行业相关技术的交流产生了不利的影响。

## 1.5 本文的研究内容

本文针对目前国内外手机游戏特效库研究的现状，对手机游戏图形开发架构进行改进并在此基础上设计、建立特效库。首先讨论目前手机游戏开发过程的现状和局限性，明确开发图形特效库的必要性。分析现有游戏开发架构，提出不足，对 MIDP 现有的简单图形引擎进行改进，提出一个特效库的设计方案，然后针对赛车游戏所需要实现的一些特效进行讨论，并由此引出游戏的一些通用特效需求，建立一个通用的特效库。然后针对建立好的特效库进行内存和处理速度的优化。具体工作如下：

1. 从不同游戏类型入手，讨论分析游戏的特殊特效需要。
2. 分析建立特效库需要解决的一些问题。
3. 分析并建立图形特效库。
4. 着重针对速度，内存占用等性能进行优化。
5. 测试特效库的效果和性能。

# 第二章 J2ME 手机开发概要及相关技术

J2ME 平台作为手机游戏开发中占有市场率最高的一个平台。虽然目前手机的平台越来越多，从开始的 BREW[2]、Symbian[3]到现在的 iPhone[4]、Andriod[5]平台，越来越丰富，但是 J2ME 仍然牢牢的占据着市场的最大份额，这与其方便的开发和较低的硬件支持要求是离不开的。本章讨论 J2ME 平台的基本特点和基本开发流程。

## 2.1 J2ME 及开发平台介绍

### 2.1.1 J2ME

J2ME 的全称为 Java 2 Micro Edition，即 JAVA2 袖珍版。作为 JAVA 的一个版本，同样具有“一次编写，到处运行”的特点，在不同的支持安装了 JVM 的手机上，同一个 J2ME 应用可以一样运行。J2ME 由 SUN 公司于 1996 年推出，的最初设计初衷是为了提供对各种消费类设备的支持，包括电视机顶盒、可视电话、导航系统、手机、智能电话等。

J2ME 的主要技术优势在于具有比较好的跨平台能力，保留了 JAVA 的良好特点，而且与 J2EE 可以无缝对接。当然，J2ME 是为了那些性能比较差，使用有限的能源、有限的网络链接以及有限的图形用户能力的设备开发的。在不同的支持 J2ME 的平台之间具有比较大的性能差异，高端和低端。依照各种设备的资源特点，J2ME 的技术架构分为简表（Profile）、配置（Configuration）和 JAVA 虚拟机（KVM）三层[6]。

简表：定义了特定设备上可用的应用程序接口，这是针对某一特定的配置上的实现，某一个应用应该针对简表支持的接口开发，此程序就可以移植到该简表支持的其他设备上。一个设备可以支持多个简表。

配置：其中为了满足不同的硬件的开发需求，J2ME 对硬件的数据处理能力、存储容量、网络链接能力等进行了统一规定，提出了配置（Configuration）的概念。配置由一个 Java 虚拟机和一组由 J2SE 缩减而来的核心类组成。在 J2ME 的规定中，运算能力有限、电力有限的嵌入式装置被定义在成为连接限制设备的配置中（CLDC），而另外一个则致力于高端消费类产品，具有高带宽的网络连接、电源稳定、设备性能受限，称为连接设备配置（CDC）。

KVM：一个专门为小型，资源受限设备设计的 JAVA 虚拟机，是 J2ME 里最小的虚拟机，适用于 CLDC 配置。

### 2.1.2 CLDC 与 MIDP

MIDP 和 CLDC 的标准化目标是建立一个具有高移植性、安全性、资源占用少的应用开发平台、使第三方可以为这些资源受限的互联网设备进行开发。CLDC 是一个通用的底层标准平台，定位于所有类型的资源受限设备，而 MIDP 则是建立于此上的特定用于无线双向通信设备的简表[6]。

CLDC 是为小型的、资源受限的、连接受限的设备上使用的标准 JAVA 平台[7]。它在很多方面进行了大量的优化，虚拟机很小，并且不支持 JAVA 的一些语言特性，这些特性有：

1. AWT，Swing 或其他图形库。
2. 用户定义类装载器。
3. 类实例的最终化。
4. RMI 和 Reflection（映射）。

MIDP(Mobile Information Device Profile)定义了针对移动信息处理设备的图形界面、输入和时间处理、持久性存储、网络连接消息处理、安全等 API[6]，提供了类似于 J2SE 中 Applet 的 MIDlet 应用程序框架。

它们的结构体系如下：

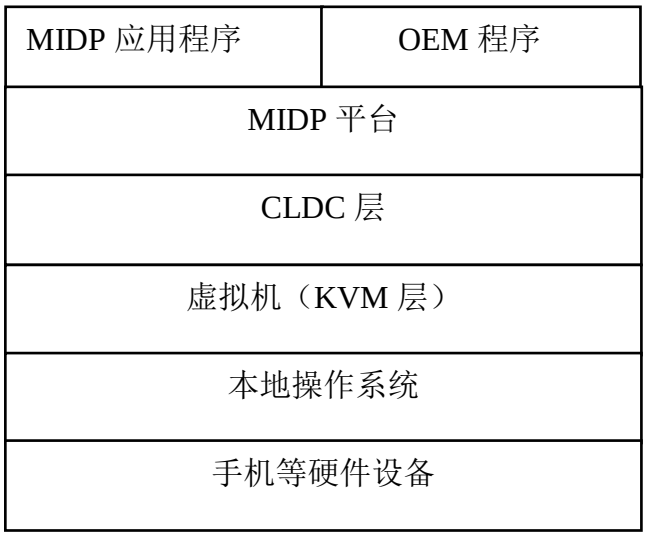


图 2.1 MIDP 体系结构

其中，CLDC 的类库大多是从 J2SE 中继承的类，还有国际化支持、系统属性和一些 CLDC 专用类。而 MIDP 建立在 CLDC 基础之上，除了 CLDC 拥有的类库还增加了一些新的类库，包含 java.mircoedtion.*的一些包，大致总结如下[8]：

java.mircoedtion.midlet,定义了 MIDlet 的生命周期和与运行环境的交互。

java.mircoedtion.lcdui 和 java.mircoedtion.lcdui.game，为应用程序提供的用户界面 API 和为游戏开发提供的高级 API。

java.mircoedtion.rms，为应用程序提供持久存储数据的支持。

java.mircoedtion.io，在 CLDC 这个包的基础上增加了*.HttpConnection 用于建立 HTTP 连接。

java.mircoedtion.media 和 java.mircoedtion.media.control 提供了录音，视频和播放器的控制功能。

以上介绍的 KVM/CLDC/MIDP 构成了无线应用程序的基本开发平台。

### 2.1.3 WTK

WTK 的全程是 J2ME Wireless Tool Kit，无线开发工具包。它是由 SUN 自己推出的官方版 MIDlet 开发工具，包含字节码预校验器、生成工具、模拟器、性能监视等工具。

## 2.2 J2ME 游戏开发流程简介

### 2.2.1 开发流程

开发一个 J2ME 游戏的流程如下：

首先，由策划人员对游戏的内容进行策划和设计，详细的设计出游戏内所有的要素、人物、场景、游戏模式等，并写成文档。

其次，由程序员对游戏中的代码部分进行实现，由美术人员对相关的美术资源需求进行绘画、由音乐制作人员为游戏创作音乐。

再次，程序员编写代码完成以及所有资源准备完成后，进行测试和调试。

最后，修改完毕后将所有资源打包成为一个完整的游戏包，进行发布。

与开发人员有关的过程有：

1. 编写程序代码、准备图片、音乐资源文件。
2. 编译代码到 CLASS 文件。
3. 预校验（Preverify）。
4. 制作描述文件（JAD）和清单文件（MANIFEST.MF）。
5. 打包成为 JAR 文件。
6. 进行模拟器的测试与调试。
7. 进行实际设备上的测试。

其中，编写、编译程序代码的过程一般在成熟的 IDE 上进行，成熟的 IDE 有 JBUILDER、ECLIPSE 等等。

预校验、制作描述和清单文件、打包 JAR 文件可以交给 WTK 来完成，也可以自己使用相关工具采用自动化流程来进行。

模拟器上的调试可以使用 WTK 提供的模拟器来进行调试，具有比较强的功能，包括监视程序系能、设置各种参数以模拟手机等。也可以使用很多手机厂商开发的模拟器，对于开发相关品牌设备上的应用有比较强大的调试功能，比如 Nokia 公司和 Sony Ericsson 公司都有各自开发的模拟器。当然，也有很多第三方的优秀模拟器平台，比如 KEmulator、国产的手机顽童等等。

### 2.2.2 开发调试平台

目前在手机游戏和应用开发领域,比较常见使用的 IDE 有 JBuilder、ECLIPSE 等。

JBuilder 是 Borland 公司针对 java 开发的开发工具，可以快速，有效的开发各类 java 应用，它使用经过了较多修改的与 sun 公司标准的 JDK 不同的 JDK，使得开发 JAVA 非常的方便。它的核心部分采用了 VCL 技术，使得程序的条理非常清晰，就算是初学者，也能完整的看懂代码，由于诸多原因，目前的 JBuilder 已经转向基于 Eclipse 框架开发。

Eclipse[9]是由 IBM 贡献给开源社区的一个 IDE 开发环境，它最初由 IBM 公司开发，是著名的跨平台自由集成开发环境。初期它主要用来进行 JAVA 语言开发，后来越来越多的开源开发者赋予了它更多的插件。作为一个框架平台，Eclipse 在得到了插件开发者的支持后具有很强的灵活性，许多其他的软件开发商以它作为框架开发自己的 IDE。使用 Eclipse 开发 J2ME 应用程序，可以安装 EclipseME 插件,它可以很方便的将各种模拟器紧密连接到 Eclipse 开发环境中，为开发者提供从编写代码到调试打包无缝的集成环境。

在 IDE 中可以导入 WTK 以及其他各厂商的模拟器,对编写好的代码进行模拟调试，十分的方便。

## 2.3 小结

本章先简要的叙述了 J2ME 的概念，MIDP 的架构以及 WTK 工具包。随后介绍了手机游戏开发的流程以及使用的开发调试平台。

# 第三章 特效库详细分析与设计

传统的游戏在特效上的表现是比较缺乏的，本章将根据目前开发中面临的具体问题，分析传统 J2ME 游戏特效开发中在特效开发方面，表现手段的不足以及主要存在的技术限制。并且针对这些限制，根据 J2ME 的性能特点，给出了一些解决方案，对现有的技术进行改进，在改进的同时加入一些新的实用技术，如调色板，方便的半透明实现等，形成一个新的特效开发底层框架。在此基础上针对操控类游戏的特效、粒子特效和通用特效等等做出设计。

## 3.1 游戏特效需求分析

目前市场的游戏分为以下几类：各种棋牌、趣味方块、迷宫、拼图等休闲类简单游戏；赛车、飞机、射击，体育等实时绘制量比较大的游戏；场景比较丰富的冒险类 RPG 游戏。


图 3.1 各种手机游戏截图

下面针对这些游戏逐一的进行分析。看看可以做哪些改进。

首先，棋牌类、拼图、方块、迷宫类游戏。见 3-1 图左下的图。游戏是比较常见的一种，在这类游戏中，对于其中比较优秀的游戏来说，玩家对于游戏的体验一般都集中在游戏的玩法上，包括各种牌的出牌规则、棋的走棋规则、程序设计的人工智能与玩家的对抗以及各种比赛胜利获得的奖励等等。在特效方面棋牌类游戏是比较欠缺的。整体方面，背景比较单调，玩久了玩家会对千篇一律的绿色、蓝色背景充满厌恶感。细节方面，由于手机像素的限制，很难表达出比较好的扑克和棋子的动作，扑克牌和棋子大多数操作都在玩家选择后“瞬移”了。如果能做出含有半透明效果的扑克和棋子，包括闪光动画效果的各种游戏内提示文字，层次丰富漂亮的棋盘与桌面，再结合原有的有趣的游戏内容，可以让玩家获得更好的游戏体验。

其次，赛车游戏，这是一种实时绘制量比较大的操控类游戏，也是本文讨论的重点。在这类游戏中，由于手机技能的限制，很难达到比较好的效果。如何能在画面里绘制比较丰富的游戏内容，同时又保证比较高的性能，是一个重大的难题。目前此类游戏业内大量的采取 2.5D 的方法来制作，采用 2D 和 3D 技术混合的方法来制作，可以达到比较酷的游戏效果。以图 3-1 最右下的赛车游戏为例，采用 2D 图片制作精细的赛车尾部视角图片，在主要背景中使用多层 2D 图片重叠绘制，用特殊方式处理的建筑物使得玩家感觉像在 3D 场景中进行游戏。此类的游戏制作流程极其复杂，且往往实现不了需要的效果，如果没有很高的制作水平，整体游戏感觉就是画面简陋且速度缓慢让人无法忍受。在这类游戏中，需要制作很多特效才能达到比较好的效果。比如场景的特效，需要采用尽量少的资源实现尽量多的场景。天气变化方面，需要一些下雨，下雪的特效。汽车本身的特效方面，需要速度线来表达速度感，需要一些对图片实时缩放的能力来实现汽车的远近，甚至需要粒子系统来表达一些诸如汽车尾气之类的效果。

第三，飞机射击游戏，这类游戏和前面的赛车游戏一样。也是操控类游戏中比较有趣的一种。现代的手机已经有性能可以做出类似于街机上《1942》等画面复杂，内容比较多的飞机射击游戏。这种游戏主要是操作一个飞行器，发射子弹，同时躲避屏幕上为数众多的地方飞行器，子弹等。在这种游戏中，各种怪物，子弹以及飞行器特别的多。如何实现比较多的同屏内容，同时又不至于单调是一个问题。需要炫酷的爆炸效果，还需要一些画面模糊的效果来改善游戏的画面。

第四，场景丰富的 RPG 游戏。见图 3-1 右上角的图。这类游戏主要是操作一个主角，在大量的场景里进行探索，杀敌等。需要使用大量的人物图形精灵的切割，场景的切割重用以及优化。这样的游戏需要从多方面着手来提高游戏的质量。物品的特效方面需要一些火焰，粒子等效果来改善画面的美观程度。对于场

景方面，对图形精灵进行拓展运算，扩展其可以半透明，发光等特性，提升整个场景的画面质量。

经过上面的分析，可以得出在特效方面，需要做哪些特效改进来改善的游戏质量。主要分为以下几类：

1. 需要对图片处理的特效。包括发光，缩放，变形，半透明。
2. 粒子系统、速度线、基于小规模图形资源实现的雨雪天气。
3. 对图形精灵的拓展优化得到的特效。

## 3.2 架构的改进

### 3.2.1 现状与存在的问题

常见的手机游戏开发的基本架构如图 3.2，存在的主要不足有：

1. 游戏中绘制方法受限，只能采用 MIDP 中的简单图像精灵，功能单一且浪费资源。
2. 图形的绘制比较繁琐，想实现一些简单的特效需要反复的写逻辑，这些不同的特效实现时没有统一的规划，所以一些通用功能没有单独实现，从而导致了开发的繁琐。由于开发的繁琐，在游戏中不可能应用大量特效，进一步导致了游戏的单调。
3. 游戏开发流程一般比较紧，没有大量时间消耗在特效的开发上。

### 3.2.1 对开发架构进行改进

基于以上现状，我们需要开展的工作有：

1. 在开发架构上对现有的手机开发架构进行优化，对原有的图形引擎和图形资源部分进行资源的重新整合和优化，扩展其功能。需要对图形资源进行处理，将图片资源进行整理整合，优化，方便操作。在代码部分进行准备，通过继承并大幅度的改进图形精灵类，来获得对图形的数据化处理。
2. 在原有图形引擎的基础上，将原本需要大量在游戏中去详细实现的一些常用特效实现在特效库中，同时大量利用原有的接口，简化开发流程。
3. 形成的应该是一个可扩展性比较强的特效库，确保今后有了新的需求可以快速的进行后续开发。
4. 特效库的开发要尽量利用现有的优秀资源，融入到现有的开发框架中

去，同时尽量不去动其他部分的结构。

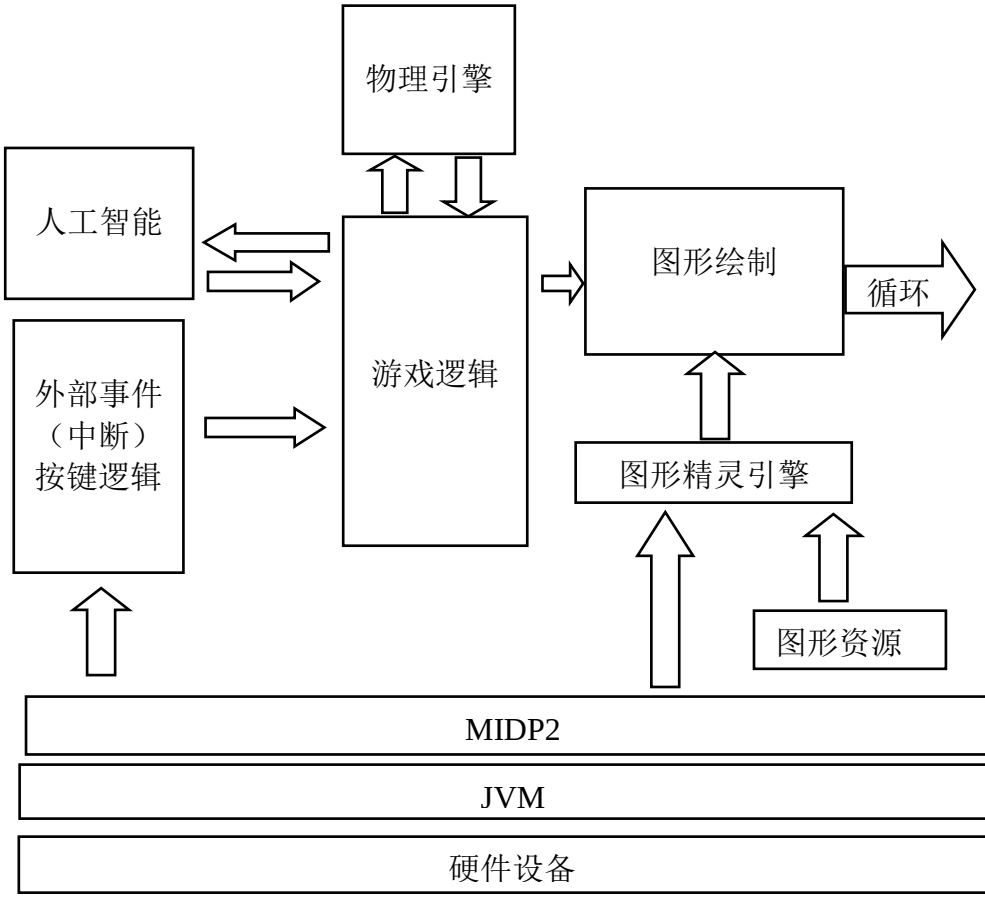


图 3.2 J2ME 游戏开发结构图

在开发的过程中，除了开发人员需要针对每个项目编写的人工智能和游戏逻辑代码外，与物理引擎和图形精灵引擎有关的逻辑代码也占了重要比例，其中图形引擎提供的功能直接决定了图形绘制过程代码的繁简，也决定了图形绘制部分的效果。

而图形的输出是整个游戏逻辑中最后一个环节，也是玩家直接接触的一个反馈。一个优秀的设计应该是在引擎部分提供强大的功能，在对图形资源进行处理的同时，提供相当部分的良好处理过的特效以及各种常用绘制的接口，这样在上层图形绘制的部分可以大大简化操作，加快整个项目开发的进度与效果。本文的内容正是对传统的图形引擎进行拓展并在此基础上提出将常用特效部分形成一个库，简化在开发图形绘制部分时的繁琐程度。将这一部分提出整理，并进行深度开发，形成一个库以后整合在整个大的引擎中，必然会大大有利于手机游戏的发开速度。 具体的改进见下图。

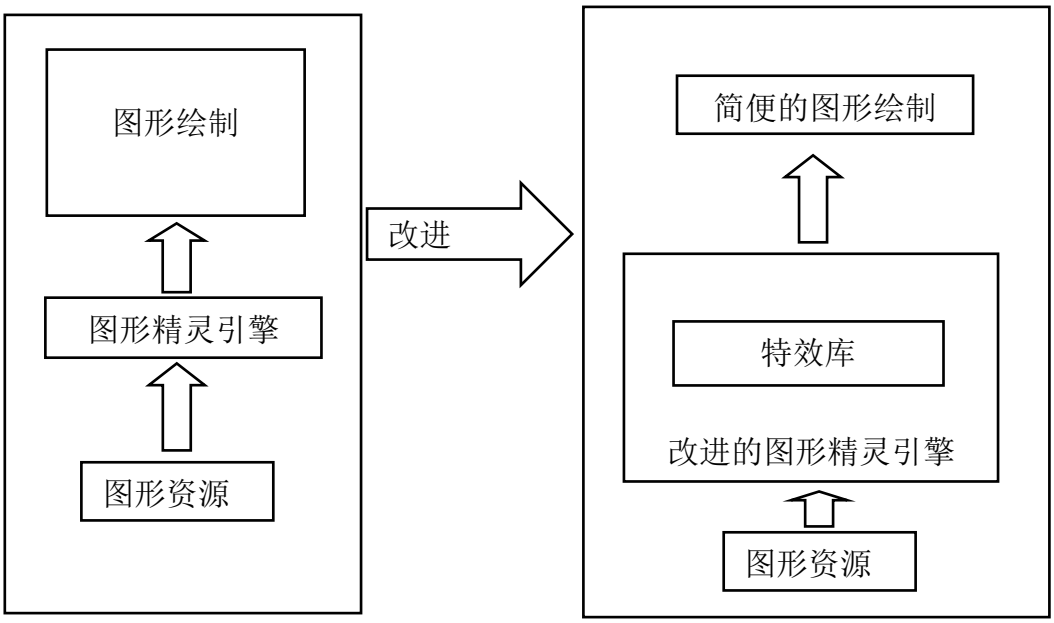


图 3.3 游戏图形部分改进图

## 3.3 改进现有图形精灵引擎

首先,看一下 MIDP 里提供的传统 Sprite 即图形精灵[10]的结构。它位于 MIDP2 的 javax.microedition.lcdui.game.Sprite 中。一般来说一个图形精灵由多张内容相似却有细致区别的帧（frame）构成，这些小图片为一张大图片上连续排列的图片按照等尺寸切割而成，为其加上编号。

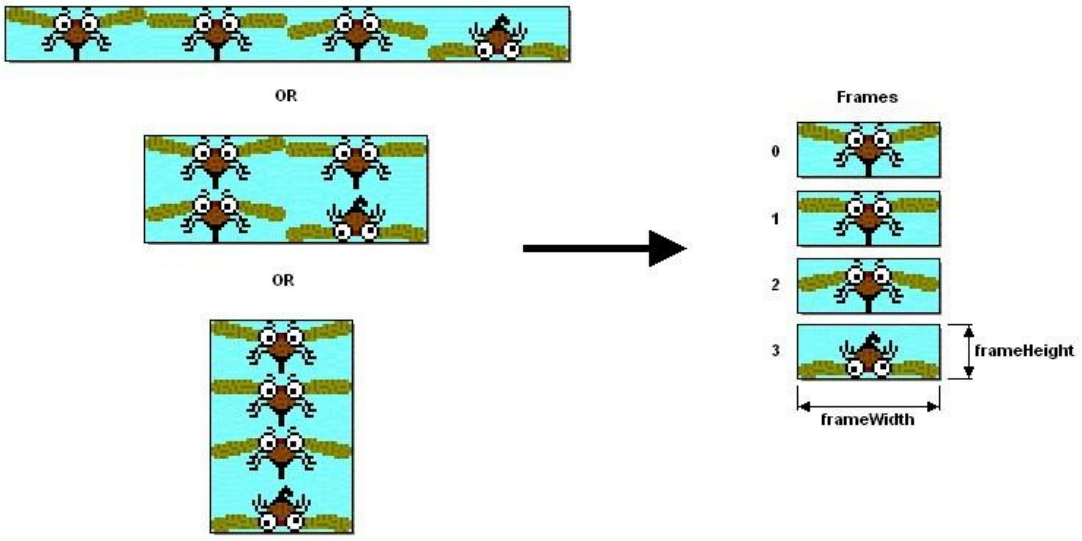


图 3.4 图形精灵

此时，图形精灵里已经有一个默认的顺序，读入的数据将按照{0,1,2,3}的顺序排列。将这些序号按照需要的顺序进行组合排列，在代码中每绘制一次调用一次 MIDP2 中的 nextFrame()即可得到想要的动画，即 Animation。

例如下面的一个序列组合起来即代表了蚊子飞行扇动翅膀和动鼻子的过程，具体动画效果可以到 WTK 文档中查询。

表 3.1 蚊子飞行顺序表

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 1 | 1 | 1 | 1 | 1 |

同时，图形精灵还用矩阵旋转的方法，提供了旋转 90 度整数倍的旋转功能，在绘制时加上相应参数即可。

### 3.3.1 图形精灵改进设计

不难发现提供的功能有明显的缺点，为很多类似的图形存储资源很浪费，为了改进这点，需要在图形精灵中加入碎片（Module）的概念，即将图片切割成很多小块，可以重用的部分只保存一份即可，随后再进行组合，形成不同的帧（frame），这样可以节省大量的资源，缺点是需要导出一份 XML 配置文件来说明每一帧的组成的碎片序列号以及位置。当然配置文件通过打包成为 2 进制文件导出，不需要耗费太多资源。优点是可以在 XML 里加入大量的额外信息，用于配置和图形有关的参数，比较灵活。详细的结构改进见下图：

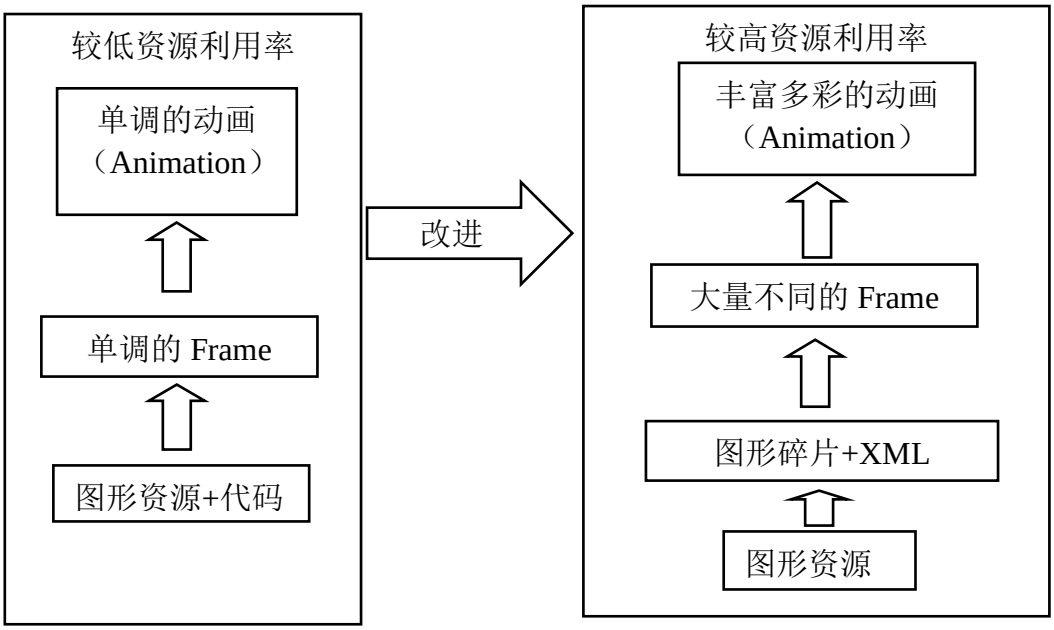


图 3.5 图形精灵改进图

以下图片即一个游戏人物的若干造型被切割成为很多碎片的图。

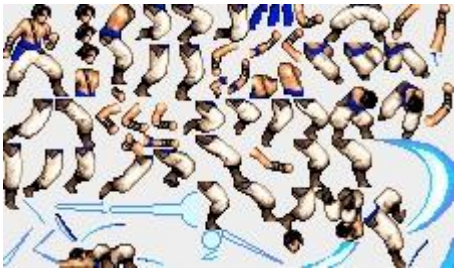

图 3.6 切割碎片图

处理后通过 XML 来记录，记录下碎片的起始位置、宽度、高度等信息。同时，对碎片进行组合成为帧，填入位置，偏移，碎片序号等参数，这样就由碎片形成了一个个完整的帧。对于动画来说，需要记录下每一不同画面所调用的帧的序号，再记下每一个画面播放的时间就可以了。

这样由碎片就形成了完整的动画，在每一块 XML 中都可以自定义很多的变量，方便开发的时候使用。

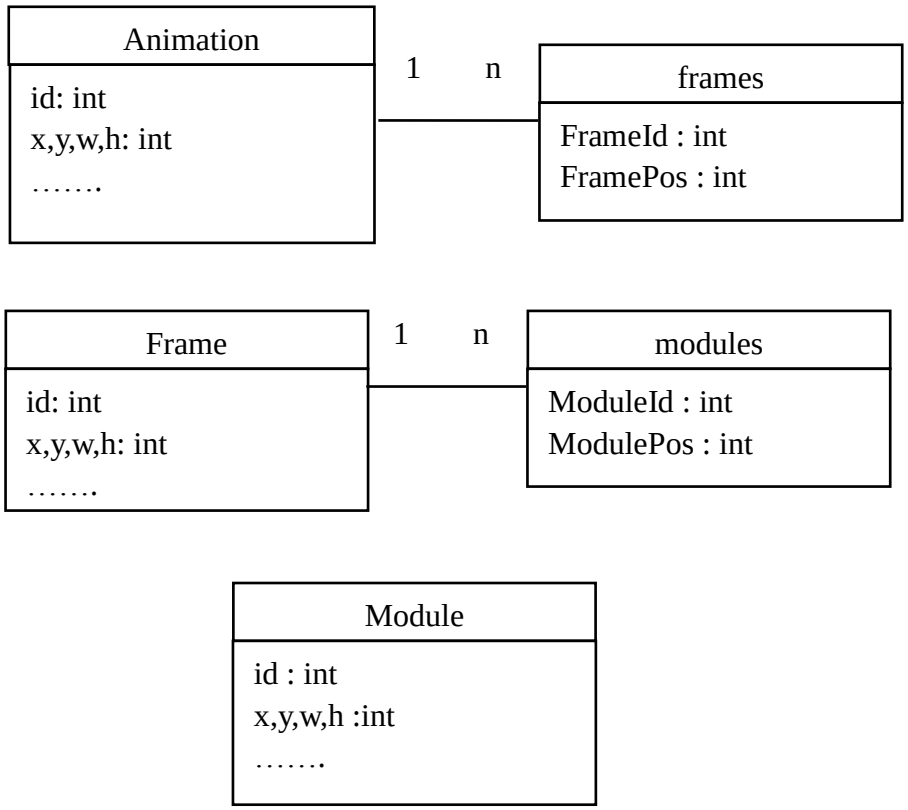


图 3.7 图形碎片 XML 结构图

### 3.3.2 图像精灵引入调色板概念

调色板[11]是 PC 游戏开发中常用技术，它针对索引色图片使用一组颜色值的信息，数量和原有图片颜色数量相同，在使用该调色板时将调色板的颜色一一对应，完全替换掉原有图片里的颜色，从而形成了相同外形不同颜色的丰富多彩的图片。在图片打包的时候进行处理，加入调色板信息，读入的时候可以设置调色板的序号，使用时也可以随时设置，这样的话可以让一张图片实现很多种不同的效果，是一种非常好的节省资源，丰富游戏画面的方法。然而在 J2ME 平台却没有提供调色板的支持，本文基于图像精灵的基础上加入调色板特性，更符合手机开发的需要。

在上一节的方法打包了图形的所有信息后，需要在代码里做好准备，以便读取配置好的 XML 信息，以下是扩展 SpritePlus 的类图。

注意到图中包含了关于调色板技术的内容，第一个_nPalette 是指调色板的数量，_colorData[][]存储着不同的调色板的信息，其他变量如_imageSize[]辅助调色板替换时的操作。

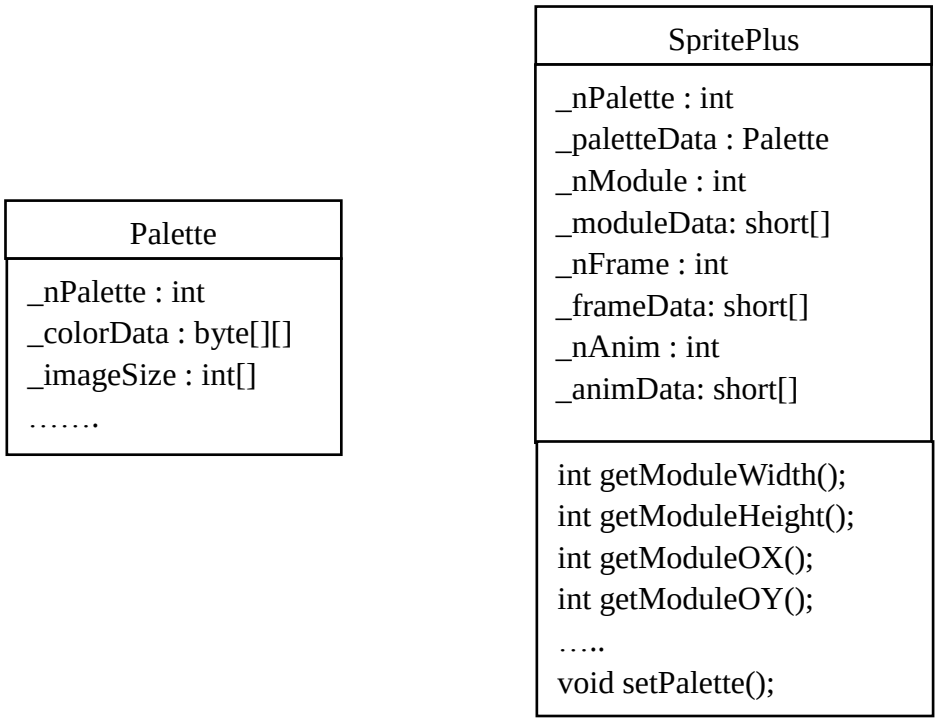


图 3.7 SpritePlus 设计类图

实现调色板后，可以用同一张图片替换不同的调色板来实现不同色彩风格的人物，建筑等。此时主要进行 4 个步骤的操作：

1. 读取原图的信息和需要替换的调色版信息，并保存；
2. 找出色彩值信息在原 PNG 图片中的位置，并用调色板替换；
3. 将替换完毕的结果保存到新的 Image 里去或者替换原来的图片；

4. 根据替换完毕的图形数据，更新 CRC 校验码信息。

主要需要的函数原型为（详细代码略）：

```
//寻找到 color 信息在图形数据中的偏移。
void getColorPosion(int [] data, int [] out );
//替换调色板信息到图片中
void replaceColor(int [] data, int []out, int oldColor, int palColor);
//更新 CRC 信息
int update_crc(byte[] buf, int off, int len);
//输出图片
Image getPalettedImage(byte[] data, int[] oldColors,int[] palColors);
```

调色板类的实例中包含了很多组调色板信息，利用这个方法，可以将不同的调色板信息与图片信息处理，形成很多张不同色调风格的图片。

扩展图像精灵类中加入调色板的信息，为后续大量相关的处理打下基础。

### 3.3.3 图像精灵加入整图像素级半透明效果

MIDP 是支持半透明的图片的，可是仅仅支持整张图片的半透明背景设置。如果用代码对每个像素的半透明级别进行设置，非常的麻烦。想要在制作图片资源的时候调节每个像素的半透明值也很困难，因此我们必须采用一些技巧实现。很明显，大多数时候需要的是对图片中每个像素有不同的半透明值，这样的图片才有很好的效果。rgbData 中的元素形式为 0xAARRGGBB。其中 AA 代表透明度，00 代表全透明 ，FF 代表完全模糊。采取下面的方法为图片里每一个像素完全的设置不同的透明度：

在绘图工具里制作一块同原图相同大小，形状完全相同的图片，他们的色彩不能做成彩色，而是做成灰度图片。从 0-100 的灰度值，可以完全由美术人员设置渐变等等效果。这样的一张灰度图片，在读取原图时同样读取它。在把原图保存成为 RGB 数组的时候保存为 ARGB 数组，每读一个点的色值，就读一个相同位置灰度图上的灰度值换算为透明度的值，写入到 ARGB 数组中。这样整个彩色图片就有了像素级的半透明效果，而且这样的方法十分简单，适合美术人员随时根据自己的要求调整。

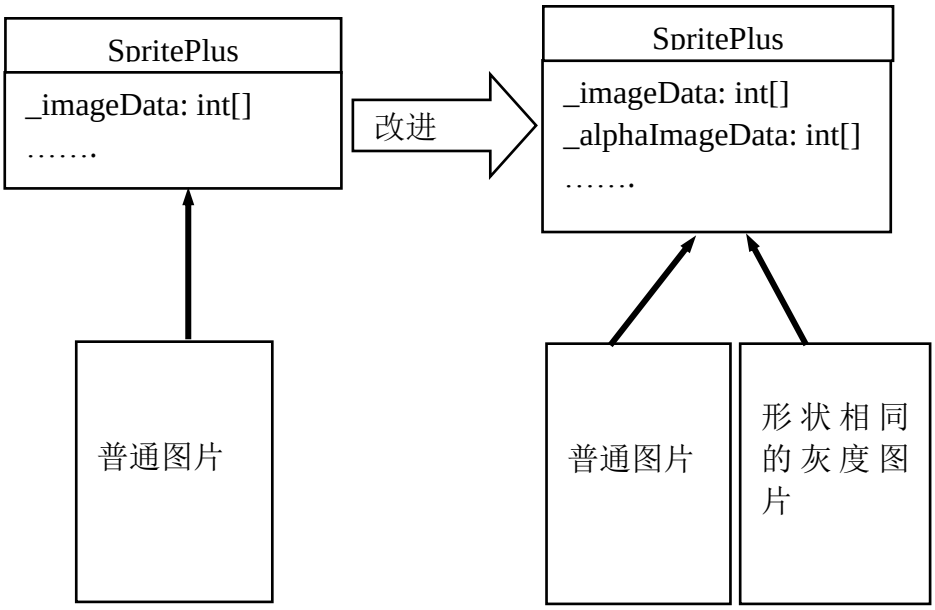


图 3.8 带 Alpha 通道的 SpritePlus

这样的话原有的图形精灵就由一张图片变成包含了 2 张图片：

核心步骤：

1. 原来 RGB 信息：0X00RRGGBB
2. 读入灰度信息：50
3. 按照比例计算透明度：AA = (FF-00)*50/100=7F;

计算最终 ARGB 信息：0X7FRRGGBB;

此方法在图片创建的时候就可以完成，比较节省开发的时间，同时在图片装载的时候就将半透明信息读入存储于 RGB 数组中，或者直接创建成 Image 图像，比较方便。

## 3.4 改进特效库底层缓冲逻辑

在图像精灵改进的基础上，对原有的图形缓冲逻辑也需要进行改进，首先在原有缓存的基础上要加入特效专用的缓存，虽然需要多使用内存，但是对特效的稳定性和性能来说是非常有利的，优化游戏逻辑中的缓存将使得特效库的底层更加稳定。

### 3.4.1 运行时逻辑的改进

在不使用特效库时，所有的图形图像都直接通过 MIDP 的绘图函数进行绘制，当引入特效库部分后，整个流程变得更加简单，清晰。

首先，在游戏装载的时候，载入图形图像资源，并且按照需求创建成为 RGB 数组或者 Image 对象，装载所有的调色板信息，创建并分配好操作用的 RGB 缓存的空间，创建并为参数的数组 m_parm[]分配好空间，创建好双缓冲（后面会详细介绍实现）。

其次，等候游戏循环通知绘制的消息（有可能是连续的消息），一旦接受到消息，立即对该特效指定的 RGB 信息读入进行处理。分为两类：第一，需要对图片处理的特效，包括发光，缩放，变形，半透明以及对图形精灵的拓展优化得到的特效，此类特效将根据不同的算法对图形信息数组进行处理并输出结果。第二，粒子系统、速度线、基于小规模图形资源实现的雨雪天气。此类特效主要接受很少的图形信息，或者不用图形信息，特定的算法生成图形信息到缓存数组中并输出结果。

再次，处理好图形信息后，将图形信息绘制到缓冲中。

然后，将离屏缓冲拷贝到屏幕上。

最后，继续等待消息，循环。

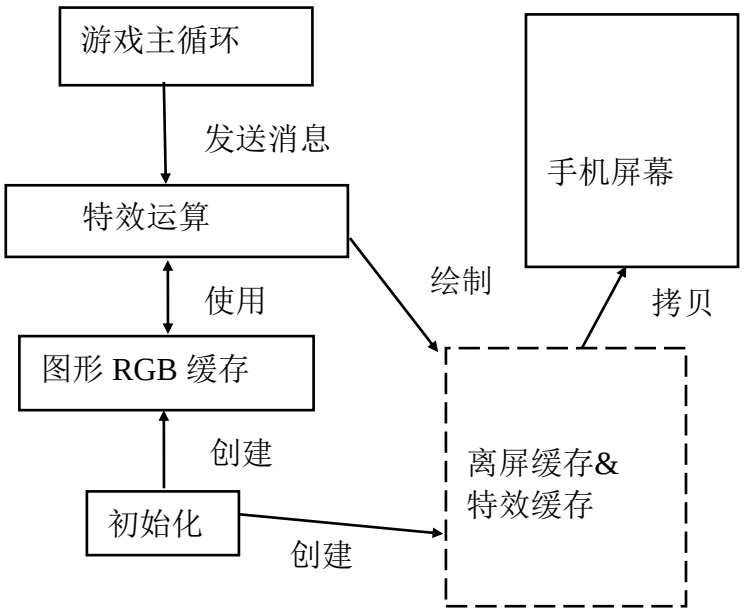


图 3.9 特效在游戏循环中的流程

### 3.4.2 缓存的改进

首先，开发时为了防止屏幕闪烁以及提高绘图效率一般都采用双缓冲[12]。原理是创建一块屏幕大小的 Image 对象在内存中，用来作为离屏缓冲，等待所有绘制动作完毕再往屏幕上拷贝。

其次，为了改善特效库的性能，在实现双缓冲的基础上，特效库还需要单独开辟一块特效缓存，作为特效计算时使用的缓存。缓存的大小由特效的实际需求而定。在游戏逻辑中可以预先在特效缓存中绘制好一些内容，提升游戏的速度。我们需要预先定义好特效缓存的相关底层逻辑：

```
//绘制部分
EffectLib.drawOnEffectBuffer();//向缓存上绘制内容
EffectLib.copyEffectBuffer(Image dstImage,int x,int y,int w, int h);//拷贝内容到其他缓存上
```

最后，固定的特效缓存之外，还可以临时创建其他的特效缓存，这又不同的特效的实际需求所决定。比如在大尺寸地图上进行操作的游戏，需要采用额外的缓存区域来实现屏幕上地图的平稳缓冲。[13]

## 3.5 特效库的详细功能设计

此部分详细的在现有工作的基础上，分析了特效库的结构以及各部分的设计特点。

### 3.5.1 特效类总体结构

将特效库命名为 EffectLib，那么它的继承关系以及结构图如图 3.10。

它继承了扩展的图形精灵引擎的接口，拥有了它的所有方法，包含基本的图形资源的载入、处理，资源的整合，动画的处理能力等等。拥有统一的输出接口，可以将上面进行处理过的图形信息统一以 RGB 信息或者其他格式进行输出，而上层的游戏中的图形绘制部分只要简单的遍历绘制一次即可。

Effect 接口称述了特效类所应具有的一些基本特征，特效库里的其他特效全部直接或者间接由它继承而来，需要实现它的接口。同时其他的子类也应该具有一些不同的参数和特点，可以通过重载函数来实现同一特效的多样性。

下面详细的描述一下需要实现的一些接口的含义以及功能。

Init()函数应该包含了这个特效的初始化操作，包括将需要处理的 sprite 或者 RGB 数据放入指定缓存，准备好进行特效处理所需的参数，并且开辟好需要

的相关内存空间，确保下面操作的进行。

applyEffect()函数应该有许多个重载版本，在不同参数和需求的基础上，对内存中的 RGB 信息数组进行处理，同时将结果输出到指定的位置。

draw()函数应该有许多个重载版本，它的参数应该包括 Image、Graphics 的实例，以确定它以怎么样的方式往游戏主循环的一些缓存上绘制。

性能监控部分应该包含内存的使用和绘制时间的监控上，方便对特效的性能进行分析和调试。

其他的函数应该针对不同特效的特点在继承的时候详细设计。同时对应这些函数都应该建立详细的异常处理机制，因为在大量的图形信息处理时，极容易产生各种异常，需要有效的处理并解决。

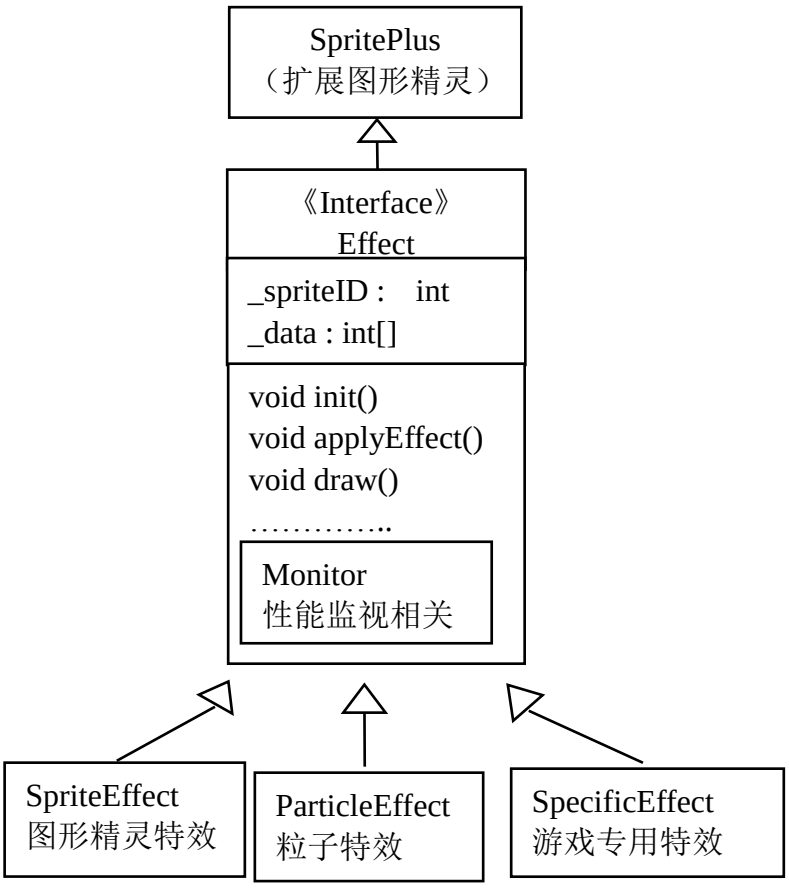


图 3.10 特效库继承关系图

另外，为了特效库，需要对图形精灵引擎进行一些改进。

下面将设计具体化。特效库的具体功能包括：

1. 针对特定游戏场景开发的专业游戏特效。例如赛车游戏里的雨雪天气，场景色调变化等。
2. 得到图形信息后进行分析处理特效的能力，这些能力比较具体，应该包括变形、变色、发光、半透明、雾化等等。

3. 粒子系统，其中包含了对美术资源作为基本单位的粒子效果以及纯粹的色彩颗粒的粒子这两种粒子的处理能力。
4. 系统的监视单元，底层的工具函数。

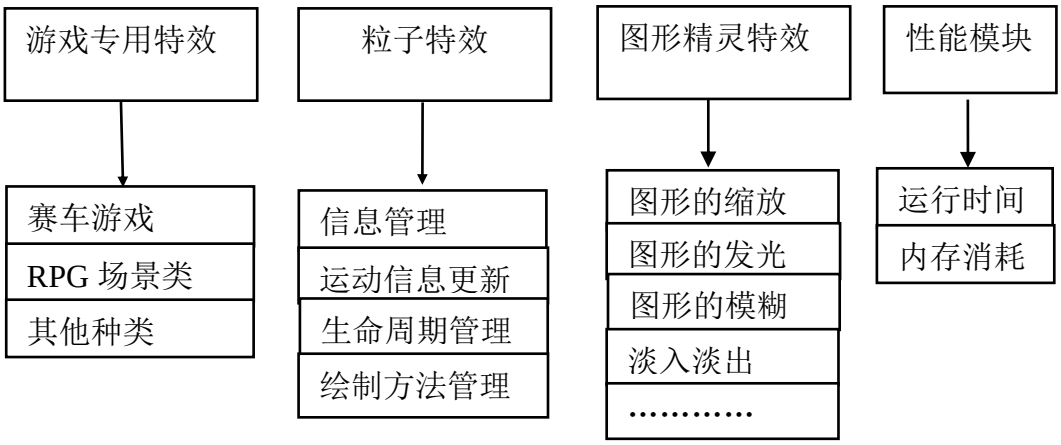


图 3.11 特效库详细需求设计

这几大块涵盖了特效库需要实现的内容。

### 3.5.2 游戏专用特效部分设计

传统的游戏在特效上的表现是比较缺乏的，通常只能够通过有效的美术手段来实现一些简单的特效，在这一部分我们结合业务逻辑与特效库的一些技术特点来实现游戏的特效。由于图像精灵引擎的改进，制作特效方便了很多。主要遵循以下的步骤：

1. 针对游戏特定应用场景，分析出该特效的运行逻辑。
2. 结合改进后的图像精灵以及特效库的技术对该特效进行分析。
3. 利用特效库的架构实现特效。

具体的特效实现逻辑往往与不同的游戏类型有密切关系，需要结合该游戏以及该应用场景详细的分析。下面针对不同类型的特效进行分析。

**第一，场景变换类特效。**

此类特效要求游戏在运行过程中场景进行变换，不可能制作很多的美术资源来进行绘制，此时我们需要灵活的运用调色板技术适时的修改调色板，来变换场景的颜色风格。

**第二，雨雪天气类特效。**

首先要明确此类天气特效在场景中与镜头的关系，是否为动态场景。然后确定要特效的运动与镜头运动的关系，确定各种天气效果的产生、发展、消失的规

律，确定此类特效的更新模型。鉴于 J2ME 的性能限制，在模型的选用上尽量采取简单为主。

**第三，场景要素类特效。**

当游戏场景中某个或者某类要素要求实现一项效果时，可以采用对该要素的图形资源进行特殊处理来完成。同时也要关注该要素在游戏中的逻辑，时间要素和与其他场景要素之间的互动等等。

### 3.5.3 基于改进图像精灵引擎的通用特效

MIDP 提供的图像精灵的功能还不能达到一个图形引擎的要求，只能作为一个制作动画和资源的工具使用。在通过对图像精灵的改造后，新的图像精灵引擎已经具备了切割图形碎片、调色板、半透明等强大的功能，在此基础上可以制作出很多强大的特效。

在图形引擎读入图像资源的数据信息后，此信息包含了一些基本的颜色值信息，通过对半透明程度的调整、调色板的实时变换、特定区域图像处理等可以很方便的实现许多特效。

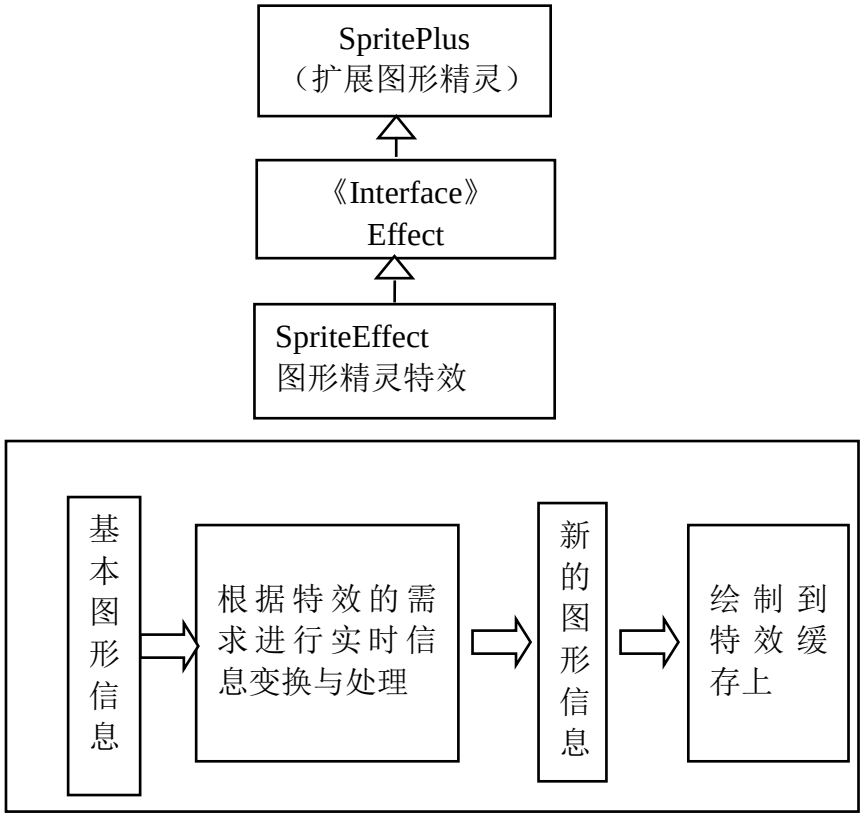


图 3.13 通用图形精灵特效开发

### 3.5.4 模块化的性能分析

特效类中的性能监控单元主要需要实现内存使用的监控和运行时间的监控，应该加上调试开关，即在需要调试的时候才运行。

内存使用部分：

设计一个计数器，在相关特效资源和变量初始化的时候记录下各种资源和变量消耗的内存数量；记录下缓冲使用的大小，同时记录下内存的峰值；在 IO 读取部分记下内存的消耗和时间的使用。

函数性能监控：

针对特效中不同函数建立计时器，在函数运行开始和结束分别设置时间节点，记录下函数的运行时间。

计时器信息输出：

首先定义好统一的接口，输出指定格式的性能数据，然后根据设置的不同，选择定义好的控制台输出、屏幕输出、文本文件输出等几种输出方式，方便开发者根据信息进行优化。

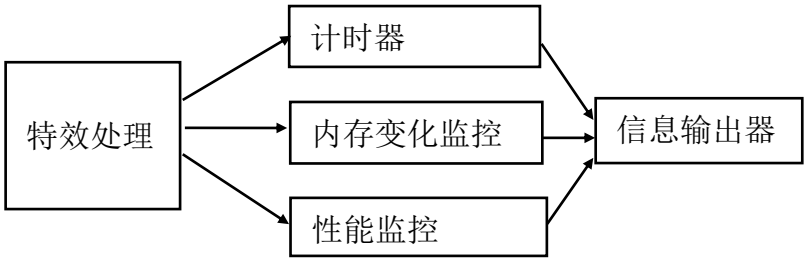


图 3.14 性能分析模块设计

信息输出器作为一个接口，输出一定规格的信息，可以用不同的显示模块来显示它的内容，可以是本文输出、控制台输出、手机实时输出等。

## 3.5 特效库简单粒子系统设计

粒子系统[14]是 Reeves 于 80 年代初提出的解决大规模小微粒运动模拟的方法。传统基于 J2ME 平台开发的游戏中，由于机能的限制，采用的粒子系统相对简单，适用于简单的场合。目前手机的性能在不断的加强，采用粒子系统的复杂度可以略微提高，但是仍然需要大量的实际测试和运用来验证性能。

在 J2ME 手机游戏中的粒子系统需要在传统的粒子系统中做一些改进以适用于手机游戏的开发：

1. 粒子信息采用一定的数据结构集中存储来代替一个粒子一个对象的做

法，比较节省内存。比如数组，位操作等。

2. 粒子信息更新的运算尽量简单，节约运算的时间。

3. 不必重绘所有粒子，适当的设置重绘标志。

对于粒子的特性部分，所有的特性都可以实现。

粒子系统的基本组成包括粒子的数据结构，粒子的生命周期的管理，粒子运动轨迹的更新，粒子特效的绘制四个方面。实现手机上使用的粒子特效系统需要关注整个过程中内存的使用，以及大规模粒子绘制的速度等。

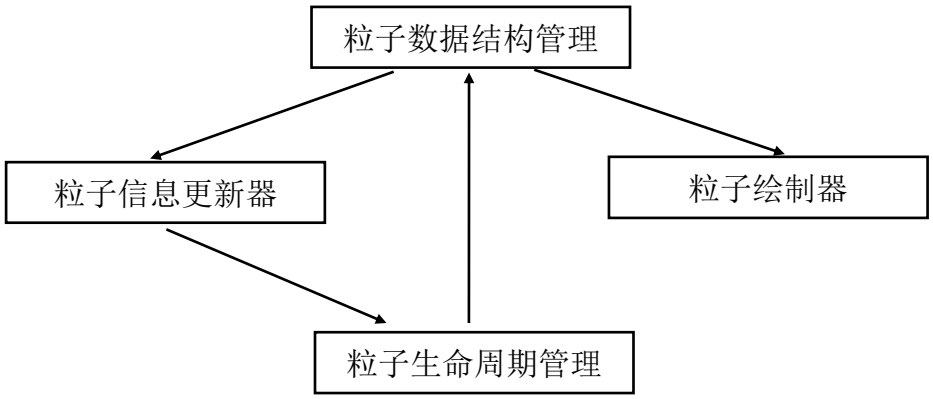


图 3.12 粒子系统结构图

其中，粒子的数据结构包含了存储了每个粒子信息的数据结构，最常用的就是数组，其中包含的信息不仅包括粒子的位置、速度、加速度、运动角度等信息，还包括了粒子之间的依存关系的信息，粒子的颜色信息等等。

粒子信息更新器应该具有的功能是提供一定的接口，对于不同实际环境下的粒子运动情况进行分析建模得出公式后，可以使用粒子更新器根据现有的条件来更新粒子的实时运动信息，并存储在粒子的数据结构中。粒子的运动信息可能被以下条件所影响：外在作用力、粒子本体算法中的生命周期等等。

粒子的生命周期包含着关于粒子当前生命情况的一些管理逻辑，不同的实际环境需要不同的逻辑。有一些粒子是一次性的，死亡后就不再重生；有些粒子可能要反复的重生；有些粒子是按照一定的条件死亡等等。根据每个实际运用场景的不同，需要建立不同的粒子模型。

粒子的绘制器部分实现了粒子在缓冲上的绘制，在得到粒子基本物理情况的前提下，它应该很多种方法来绘制粒子。

### 3.5.1 粒子的基本数据结构

在 J2ME 手机上实现粒子系统，无法像在 PC 机上一样每个粒子可以成为一个对象实例，那样的话势必给手机捉襟见肘的内存空间带来巨大的考验，无法正常

运行。因此在手机上实现粒子系统需要用管理器和数据结构来保存粒子的信息。

在不同的实际模拟情况下，有两种情况需要考虑：一种是粒子数目恒定的情况，另外一种是粒子数目变化较大的情况。对于第一种情况一般使用数组来存储粒子的信息，第二种情况有很多选择，对于内容足够，可以开辟比较大的内存来存储粒子的信息，对于性能比较弱的手机可以采用位存储、位操作等方式来尽量的节约内存的使用。下面来介绍一下一般的数据结构：

```
int PRATICLE_NUM_MAX = 200; //粒子的最大值;
int particlePostion [PRATICLE_NUM_MAX][2];//粒子的位置信息
int speed [PRATICLE_NUM_MAX];//粒子当前的速度
int accel [PRATICLE_NUM_MAX]; //粒子当前的加速度
int angle [PRATICLE_NUM_MAX]; //粒子当前的运动角度
int life [PRATICLE_NUM_MAX]; //粒子当前的生命值
int color [PRATICLE_NUM_MAX]; //粒子的颜色值
int weight [PRATICLE_NUM_MAX]; //粒子的质量
int mgrBit[PRATICLE_NUM_MAX]; //粒子的管理信息字段
```

其中粒子的管理信息字段采用 bit 操作的方式来存储，成熟的粒子系统应该包含比较丰富的信息，比如是否需要重新绘制该粒子等等。

粒子的生命值可以比较灵活的表达粒子的生命信息，对于按照时间拥有生命周期的粒子或者按照位置进行生命周期管理的粒子，可以自己定义含义。

这样的信息已经足够描述粒子的基本信息。在 J2ME 应用设计中，涉及到这样大规模的使用内存的情况，应该在程序初始化资源的时候一次性的将所需要的内存空间全部申请好。

### 3.5.2 粒子的信息更新模型

建立粒子系统的初衷就是为了模拟一些运动和特效，所以需要对一些实际的应用场合进行分析，整理出他们的运动规律和模型，进一步建立公式来描述粒子的运动。在 J2ME 手机游戏中，游戏的更新周期以帧为单位，每一帧对所有需要的信息进行一次更新和绘制。不同的手机的性能不同，完成每一帧运算的时间需要的也不同，为了模拟比较真实的效果，要使用 J2ME 的系统函数来取得每一帧的运行时间，得到这一帧时粒子系统应该运行的状态。

```
long timemilliseconds = System.currentTimeMillis();
```

得到的系统时间单位为毫秒，已经足够各种运动模拟的使用。

对于不同的运用场合，需要确定一些先决条件，比如初始速度，初始位置，初始加速度，粒子运动分布规律函数，外力作用函数等等。然后再每一帧的实时

更新时，根据当前的时间，当前的外力作用和已经存在的粒子的情况，使用已经总结好的粒子运动分布函数来得出位移和速度等的变化。

比如，假设粒子的初始状态为 StateInit，时间为 t，粒子运动函数为 F（x），外力以及其他作用元素为 G，那么得到的当前状态应该是：

```
StateNow = StateInit + F(t)+G;
```

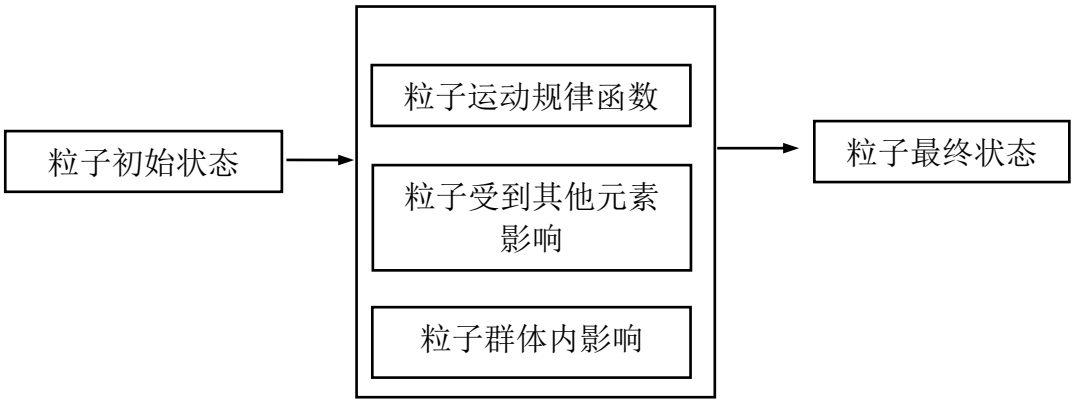


图 3.13 粒子更新模型

此步骤的关键即在于建立一个完善的粒子的系统模型，这无疑对设计者对所设计的内容的深入研究和理解，需要不断的实验和修正。但是一旦建立完善以后就可以很方便的反复使用。在 J2ME 手机上使用粒子系统所实现的都是比较简单的特效，可以比较容易的建立起模型。

在的特效库部分，设计一个粒子运动更新的接口函数，作为一个规范的接口，任何使用这个粒子系统的人都需要重写它来实现自己需要的内容。

```
public void updatePraticle(
int updateType, //更新方式，定义作为时间轴更新还是位置特征更新
int particlePostion[],  //作为参数的一些粒子系统的信息
int speed[],          //作为参数的一些粒子系统的信息
int life[],           //作为参数的一些粒子系统的信息
int systemTime,       //系统时间
……
)
```

有一些特效可能还需要添加特效群内互相影响的部分信息，需要另外设计一个功能。

### 3.5.3 粒子的生命周期管理

生命周期管理是很重要的一部分内容，在不同的实际运用的情况下，需要对

生命周期的管理作出不同的设计。

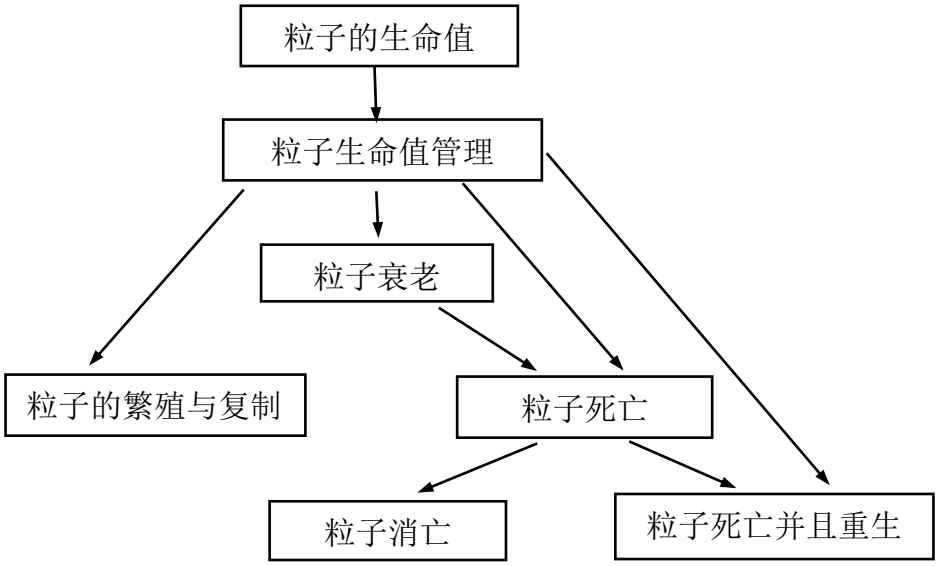


图 3.14 粒子生命周期更新模型

粒子在整个生命的周期有着不同的特征，在衰老期它存在着衰老期的一些特征，比如显示效果暗淡、速度减缓等。而粒子的死亡则有可能重生也有可能完全消亡。这都取决于在粒子系统设计的时候，需要设计怎么样的一个生命值管理模型来符合这个粒子系统的特性是一个关键性的任务。

在 J2ME 手机的特效中经常使用的有火焰、小火星、雪花、雨水等，在大多数情况下粒子的数量都是恒定的，在粒子死亡后都会重生。为了节约内存和优化系统性能，死亡后并且重生的粒子将使用原来死亡粒子占用的空间。而消亡的粒子所占空间也应该有标记，在有新粒子诞生时应该优先使用旧的空间。

```
public void updatePraticleLifeCycle(
        int life[],           //粒子的生命值
        int systemTime,       //系统时间
        ……
    )
```

对粒子不同的状态的存储，用不同的数值表示不同的状态即可。

### 3.5.4 基于 J2ME 和特效库的粒子绘制管理

在通过游戏的特有逻辑得到当前粒子群的信息后，需要对粒子进行绘制，在 MIDP2.0 版本中，系统以及提供了像素级别的半透明操作，为丰富粒子系统的效果提供了一个良好的条件。

前面本文介绍了使用调色板技术实现同一场景变色的特效，在这里同样使用相同的思想。根据在通过粒子更新信息后得到的粒子的位置，再结合拓展的特效

库中的相关技术，可以绘制不同颜色、透明度、大小的像素点。同时根据粒子的生命周期管理和相关的游戏逻辑，可以得到非常丰富的粒子系统的效果，比如燃烧的火焰由黄色慢慢变红等。

```
public static void drawParticle(
   int particlePostion[];//粒子的位置
   int life[],          //粒子的生命值
   int color,            //粒子颜色
   int drawType,      //绘制的特效
)throws exception
```

在绘制的方式 drawType 上可以输入不同的参数，不但可以绘制像素点，还可以输入一些资源进行绘制，以粒子系统为位置管理器，绘制由美工人员精心设计的图形，可以得到比较漂亮的效果。

## 3.6 小结

本章详细分析了传统 J2ME 游戏开发架构存在的不足，做出大量改进，提出了改进图像精灵的设计，为图像精灵添加了很多新的特效，提升了开发架构。随后针对游戏的需求从赛车游戏专用特效、通用特效、粒子特效等入手详细的设计了特效库的架构。还设计了特效库的性能监视分析模块，为特效的开发打下基础。

# 第四章 特效库的详细实现

本章将根据目前面的需求分析中所总结出的目前特效方面的不足以及面临的问题，对特效库的核心部分采用改进的技术进行实现。考虑到开发便捷与使用的便捷，针对不同的特效采取不同的实现方法，最后采用统一的上层接口，方便调用。最后对实现部分进行总结。

## 4.1 特效库赛车游戏特效的实现

上面一小节对特效库开发过程中作为基础的技术进行了实现。下面着重针对赛车等游戏的特效需求进行特效的开发。本文中介绍的特效的背景赛车游戏采用了比较先进的 2.5D 开发方法，保留了 3D 的速度场景感觉的同时也保留了 2D 的华丽美术风格，其中背景图片赛车等采用 2D 图片绘制，赛道旁边的建筑采用 3D 制作，整个赛道和场景都严格按照 3D 的透视效果来搭建，本部分主要介绍赛车游戏里的特殊特效的开发，并将其加入特效库中。包括场景的变化，天气效果，让赛车等游戏充满速度感的速度线等等。

### 4.1.1 利用动态调色板混合技术实现场景变色特效

在游戏中，通常有一个场景，让主角或者操作的赛车等在这个场景里进行游戏，一般来说场景只是一张普通的或者带有半透明效果的图片。在整个游戏中式一个单调的场景。在这里要进行一项具有创造性的实现，使用一张图片和调色版实现场景从白天到夜晚的无缝变色过程，从而使得游戏场景变得非常生动有趣，完全没有过去的单调感觉。

特效库中类名：EffectLib_DayToNight

类的声明：Public class EffectLib_DayToNight extends EffectLib{};

下面就来详细讨论该特效的实现以及需要注意的细节。

首先，进行资源准备，需要一张场景图片的 TileSet（即组成场景的图形精灵碎片，通过切碎的小图片可以重复利用组成比较大的一张图），即组成场景图片的图形碎片。还需要 1 个额外的调色板，原图对应白天的景色，额外的调色板则对应夜晚的景色，通过加深整个场景的颜色，将天空的颜色调成深紫色。

这样加上原来图片中的白天的调色板设置就有了两套颜色信息，假设为 palA[]和 palB[]。

然后，进行算法的设计。要想无缝达到 PalB[]中的黄昏色的效果，必须不断的缓慢改变缓存中场景图片的颜色信息，例如白天为蓝色偏白色，晚上为深紫色到黑色，需要将原有调色板中的 3 原色信息不断的进行变化，向目标调色板的颜色靠近。

最后，需要对 TileSet 的碎片进行缓存，每次修改后直接将颜色信息保存到缓存 Image 对象中。最后再绘制到屏幕上。

来看看详细实现。

首先，针对某一种色彩的变化，如果他的 ARGB 信息为 0X5F898989 变化到 0X5FF0F0F0。那么 3 原色的变化都是从 89 变化到 F0，需要对颜色的变化进行减法运算就可以得到颜色的差值，即（F0-89），由于单色有 256 个色彩，所以假设需要每次变化颜色差为 n 个单位，那么的步长为：

ColorTemp = （F0-89）*n /256;

优化一下：ColorTemp = ((ColorLater - ColorBefore) *n ) >>8;

其次，对于整个颜色信息，由于害怕加减的进位影响，将颜色的 A 和 G，R 和 B 分别分为 2 组进行运算。最后合并运算结果。

```
int tempA,tempB,ColorResult[];
While（--_nColors>=0）//遍历调色板里所有的颜色
{
   //计算 R 和 B 通道
   tempA = palA[i] & 0x00FF00FF;
   tempB = palB[i] & 0x00FF00FF;
   ColorResult[i] = (tempA +(((tempB-tempA) * n) >> 8 )) & 0x00FF00FF;
   //计算 A 和 G
   tempA = (palA[i] >> 8) & 0x00FF00FF;
   tempB = (palB[i] >> 8) & 0x00FF00FF;
   ColorResult [i] |= ((tempA<<8) +((tempB-tempA)*n)) & 0xFF00FF00;
}
```

这段代码里大量使用了与，或，移位操作，是一段比较优化的代码，相对于一张图片动辄上百种颜色，需要对代码非常的优化才能充分的提高运算的效果。

以上计算的是每次变色后的颜色值，作为一个游戏场景，需要对颜色变化的的差值 n 进行分配，才能均匀变化。假设游戏场景的总时间为 t 个单位，那么每个单位需要变化的色差值 n 为：

```
n = (palB-palA)/t;
```

在游戏主循环中，首先每次运算出要变化达到的颜色值，或者将所有时间点的颜色值一次性运算好存储起来使用。得到每一个画面的颜色值后，更新缓存里的图形碎片的颜色，使用这个图片碎片来实时绘制，即可以实现颜色无缝变化的特效了。那么伪代码为：

```
{  //初始化
   Palette PalA,PalB,PalTemp;
   EffectLib_DayToNight ._paletteData[0] = PalA;
   EffectLib_DayToNight ._paletteData[1] = PalB;
   ……
}
{  //实时更新部分
   //实时写入当前帧色彩信息
   EffectLib_DayToNight .ApplyPalBlend(n, PalTemp);
   EffectLib_DayToNight .renderTileSetImage(PalTemp);
    //用调色板作为参数，更新图形碎片信息
   ……
}
```

这个特效的实现已经结束了，还可以对此特效进行拓展。当替换调色板的颜色时候，可以将整个调色版中某个颜色变成特定的颜色，可以实现某些特定的效果，比如将草的绿色相关颜色独立提取出来建立一块调色板，将其改成对应黄色色调的一套颜色，这样就可以实现草地的变色。

下面为伪代码：

```
While（--_nColors>=0）//遍历调色板里所有的颜色
{
   if（palA == 草地的一种颜色）
      palA 替换为黄色色板里的对应颜色；
   else
      不变；
}
```

通过这些拓展，可以实现很多类似特效。

### 4.1.2 根据游戏特性制作速度线效果

在 PC 以及主机赛车游戏中，当汽车加速时，屏幕上会出现很多半透明的速

度线来加强速度感，下面在手机赛车游戏中将其实现。看一下下面这张图片的效果。


图 4.2 赛车游戏速度线效果图

在这张图片上汽车后面、屏幕的两边和上方有很多条速度线，需要将速度线的算法和绘制方法加入到特效库中去。

特效库中类名：EffectLib_SpeedLine

类的声明：Public class EffectLib_SpeedLine extends EffectLib{};

需要输入的参数：物体前进方向针对屏幕法线的角度。

速度线模拟的是风、空气中的细小水滴或者其他小物体在汽车快速运动时汽车内的人看到的轨迹，在手机游戏中通过随机绘制一些线条来模拟这个效果。

先简单的模拟计算出一些线段的数据值：

假设这些效果是从屏幕中间无限远的那个点射过来。首先确认屏幕地平线中中心点的左标，假定为（x,y），那么方法是先随机出以这个中心点向各个随机角度方向的一定数量的直线，再对直线的中间部分进行切割，只显示四周的一部分。为这些线建立一个 2 维数组，第一个元素表示它的角度，第 2 个元素表示它所要显示的长度。数组名称定为 s_speedLineData[][]，在绘制每一条时，假设屏幕宽高为 w,h，那么最远端的点坐标为：

```
X1 = x + w * sin(rand(0,360));
Y1 = y + h* cos(rand(0,360));
```

即屏幕中心点的 x 为原心， h 乘上（-1,1）之间一个值的为长度的那条半径。靠近中心点的坐标设定一个最靠近中心点的高宽后，用同样的方法计算出，假设中心半径 60 以外的地方开始画，则：

```
X2 = x + （60+rand(0,20)）*sin(rand(0,360));
Y2 = y + （60+rand(0,20)）*cos(rand(0,360));
```

其中 20 为随便设定的一个波动范围。这样再将得到的 X1、Y1、X2、Y2 作为参数，即可画出线段。在手机性能允许的前提下，可能不仅仅画线段，得到线段的信息后，可以在线段上绘制半透明线条，或者以半透明颗粒为单位的粒子直线等。

函数原型为：

```
int s_speedLineData[][] = new int [n][2];
int pos[] = new int [n][4];
EffectLib_SpeedLine.init(speedLineData);
While(--n>=0)
    DrawLine(pos[n][0],pos[n] [1],pos[n] [2],pos[n] [3]);
```

拓展讨论：

1. 其中 DrawLine 函数可以替换为其他任意绘制其他效果的函数。
2. 这样计算射线的方法只是简单的模拟，在真实 3D 场景中，可以通过一条直线在镜头平面上的投影来计算坐标。
3. 汽车尾部的速度线和汽车周身的速度线也可以通过类似方法实现。

### 4.1.3 利用半透明以及动态缩放技术制作汽车黑气

先分析汽车冒黑气的特征：刚冒出来的时候是一个小黑团，在不断的变大的过程中慢慢上浮，同时伴随着少量的左右位移。这部分的特效需要使用很小的资源实现逼真的效果。

具体方法是：先制作几块很小的半透明烟雾云团图片，使用少量颜色。利用特效库对资源的动态放大缩小功能（后续会有介绍），可以显示不同云团的大小；利用特效库的调色板替换技术，可以制作雾气的动态变色功能。针对黑气建立特效类，管理每个黑气的生命周期和运动轨迹等。

首先，制作一块或者多块半透明图片作为黑气的基本单位，当然也可以用粒子系统来实现黑气的散布，后续会单独介绍粒子系统。如下图，该图是半透明图片，背后的黑色部分是需要过滤掉的色值。

图 4.3 尾气资源

创建黑气类，主要管理尾气的资源和生命周期相关逻辑，类和函数原型设计为：

```
Public class EffectLib_Smoke extends EffectLib{};
void initSmoke ();
void updateSmoke (int updateType);
void drawSmoke();
```

先在内存中使用特效库的动态缩放功能，根据需求创建出不同大小的尾气图片，也可以在绘制的时候实时创建，依据性能而定。

使用 initSmoke()初始化一个黑气单位，使用随机数给予其一定范围内的生命周期值和一定的初始速度。

```
void initSmoke ()
{
   _life = 100;
   _speed = 20;
   _sizeLevel = 1;
   ……
}
```

在每一帧更新尾气的时候可以选择很多更新的方法，比如匀速运动，依据某些公式减速运动等等。通过 updateSmoke (int updateType)每一帧更新黑气的位置大小和速度等值，并且根据黑气的大小选择哪一张缩放图片来绘制。注意的是需要根据汽车运行的速度来更新黑气的位置，当汽车速度较慢时，尾气会一直朝上方运动，速度较快时由于视角的关系，尾气看来就一直拖在汽车的后方了。

```
updateSmoke (int updateType)
{
   //更新位置、速度、生命值
   updatePos(updateType);
   updateSpeed();
   updateLife();
   if(_dead) {……};//相关逻辑
```

```
}
```

在使用粒子系统来绘制尾气时，依旧使用这个设计来实现黑气的特效，不同的是在绘制部分，将绘制资源变成绘制粒子即可。

```
drawSmoke ()
{
   if(_useResource)//使用普通资源
      drawBySize(sizeLevel);
   else if (_usePraticle)//使用粒子
      drawByPraticle(sizeLevel);
}
```

此部分描述了单个黑气的逻辑，在游戏主循环里，可以设置尾气的数量，将他们作为一个组来管理，统一更新。

```
While(--i>=0){
    EffectLib_Smoke.update(i);
    EffectLib_Smoke.draw(i);
}
```

最后的效果见下图。

图 4.4 汽车尾部黑气效果

### 4.1.4 半透明技术制作玻璃打上雨水特效

前面一节实现了下雪的效果，这一节简单讨论一下雨水打落在玻璃上的特效的制作问题。

雨水打落在玻璃上的位置是没有规律的，有部分水珠比较大打到玻璃上后缓缓的滑落至底，也有部分雨水打在玻璃上呈长条形粘在一个位置不动。针对这两部分特点，制作两种不同的资源，一种是长条形的雨水粘在玻璃上，另外一种是雨水缓缓滑落的动画。两种动画资源都采用 80%半透明的效果。

同时建立数组来表示不同的雨水类型，位置信息。

```
int rainDotPos[][] = new int [MAX_RAIN_DOT_NUM*4];
int rainLinePos[][]= new int [MAX_RAIN_LINE_NUM*4];
```

其中其中每个二维数组的一个元素代表一个雨水的信息，第二位维的 4 个元

素分别为 X，Y 坐标，生存时间以及当前透明度。

粘在屏幕上的雨水采用随机分布的方式，将一定数量的雨水随机的散步在屏幕上方的一片区域内，给他们一个生命周期，在生命周期快结束时，缓缓的设置半透明值直到全部透明则重生这个雨水。伪代码如下：

```
遍历所有雨水
{
    if(雨水类型==长条形)
    {
        if(生命周期快结束)
            调整半透明值;
        if（全透明）
            重生雨水点至范围内的随机位置;
    }
    else if(雨水类型==点形)
    {
        if(运动完毕)
            重生雨点到随机位置;
        else
            根据生命周期播放动画;
    }
}
```

这样通过不断的循环这个过程，手机屏幕上就可以模拟出一边有大量的雨水停留，还有一些雨水缓慢的流下的特效，同时对雨水半透明处理使特效更加真实。

## 4.2 特效库基于增强图像精灵引擎的特效开发

上面一小节针对赛车等游戏中需要的一些独特的特效进行了实现，下面进行拓展，在前面工作的基础上讨论一下手机游戏开发中其他相关特效的开发，形成一个比较完善的特效库。

在 J2ME 游戏开发中，图片主要以像素图片为主，传统的开发技术比较局限。基于前面优化过的图像精灵 Sprite，可以对得出的图片数据进行丰富的处理从而实现对像素图片的常用特效，主要包括像素图片的缩放，发光，模糊等，并加入到特效库中。此部分特效作为一个通用特效部分，不仅可以用于赛车游戏还可以用于其它需要使用特效的场合。

### 4.2.1 基于平均值计算的区域或全屏模糊

在游戏中经常需要模糊的特效，比如在赛车撞击后、RPG 游戏被敌人攻击后，都需要一些全屏模糊的表现方式来使游戏更有趣。

传统的图片模糊采用高斯模糊[15]算法，即将目标点位置周围的一定范围内的像素建立矩阵，求矩阵的加权平均。算法公式为：

$$G(u,v)=\frac{1}{2\pi\sigma^2}e^{-(u^2+v^2)/(2\sigma^2)}$$

图 4.5 高斯模糊公式

图中 u, v 分别为水平、竖直距离。在 J2ME 中运算性能比较差的情况下，只要有模糊的效果即可，并不需要十分精确的效果。需要采取简单一点的方法来实现模糊，并不完全采用矩阵来进行运算。

根据 J2ME 的特点，采取在每一个像素的颜色值与前一个像素点的颜色值进行平均值计算的方法来实现模糊，这样大大减少了运算次数，缩减了每帧消耗的运算时间，同时经过测试也可以取得不错的模糊效果。在两个像素进行混合运算时取其差值乘系数 X（0<X<1）进行运算，可以通过这个 X 来调节图像的模糊程度。即：ColorA = （ColorA －ColorB）* X ；

那么，对于每一个像素点：

```
X = A >> 7;
r1 = (r2 + ((r1 - r2) * A >> 7)) & 0xFF0000;
g1 = (g2 + ((g1 - g2) * A >> 7)) & 0x00FF00;
b1 = (b2 + ((b1 - b2) * A >> 7)) & 0x0000FF;
```

分别对不同颜色值与前一个像素的颜色值进行比较，为了计算的精度，X用A右移7位即128分之A的形式来计算。

有了基础算法后，再进行拓展，建立读写函数：

```
EffectLib_ReadPixelData (
 int buffer[],//读取所用的缓存
 int offset, //读取信息在资源中的偏移
 int x, int y, int w, int h//读取的在屏幕位置上的 x，y 以及宽度高度
);
EffectLib_WritePixelData (
 Graphics g,//绘制的目的低 Graphics 对象
 int buffer[],//所用的缓存
```

```
int offset, //写入在资源中的偏移
int x, int y, int w, int h//写入的在屏幕位置上的 x，y 以及宽度高度
);
```

这样，在处理游戏屏幕内的模糊特效时，将屏幕的信息按照宽度分块处理，这样无需建立太大的缓存，读取对应的信息，进行处理，处理完毕后使用系统的 arrayCopy 函数拷贝回去，速度比较快。伪代码如下：

```
//初始化阶段
EffectLib_Initbuffer();
//更新阶段
While（所有信息行数）
{
  EffectLib_ReadPixelData（……);
  EffectLib_ApplyBlur（……); //模糊算法
  EffectLib_WritePixelData（……);
}
EffectLib_Draw(); //绘制到缓冲上
```

这样就完成了整个模糊处理的过程。

### 4.2.2 基于区域模糊以及半透明遮罩混合的区域图形发光

在游戏中普通的图形资源和文字看起来让人觉得很平淡，无法满足玩家的需求，需要加入对图片的自适应的发光特效，在游戏中偶尔的使用发光文字或者发光物品，是很不错的设计。传统的 J2ME 游戏特效制作中都是用美术资源来制作发光，在这里我们采用特效库的技术来实现。

主要的方法是在需要发光的区域绘制一层有透明度渐变效果的灰白色矩形，原图首先做模糊处理然后，将白色层与原图先进行混合，这样就得到了发光的外层。然后将发光增强后与原图混合，即可得到不错的发光效果。在游戏中，可以将所需要绘制的部分放到透明的背景上处理后再进行绘制，这样就不会影响其他内容。效果见下图：


图 4.6 发光特效效果图

首先，每帧得到透明渐变的白色矩形的像素值，这个值可以根据每帧的时间更新来运算得到。

然后，定义一副图的初始 RGB 信息为 R1G1B1，首先对这幅图进行水平位置上的模糊处理，利用前一小节的模糊方法，与白色透明矩形遮罩。进行模糊处理得到模糊后的 RGB 信息为 R2G2B2。

按照比例 x 分别对 R、G、B 通道进行混合：

```
r = (r2 + (r-r2) * x) & 0xFF0000;
g = (g2 + (r-g2) * x) & 0x00FF00;
b = (b2 + (r-b2) * x) & 0x0000FF;
R2G2B2 = 0xFF000000| r | g | b ;
```

其次，对其进行垂直方向用同样的方法的模糊处理，得到的 RGB 信息为 R3G3B3。

最后，将R3G3B3的结果按照系数intensity与原图进行混合，这样可以保证发光效果的基础上，不丢失原来图片的细节。

按照强度 intensity 分别对 R、G、B 通道进行混合：

```
r = (R1 + R3 * intensity) & 0xFF0000;
g = (G1 + G3 * intensity) & 0x00FF00;
b = (B1 + B3 * intensity) & 0x0000FF;
R4G4B4 = 0xFF000000| r | g | b ;
```

R4G4B4 即为需要的像素值。

如果需要的是静态发光的效果，而不是动态发光，那么只需要将原先用来混合的矩形部分固定为一个值即可。

拓展：通过同样的修改 RGB 信息的方法，可以任意的修改一幅图片资源的色调，饱和度等等。

### 4.2.3 利用额外信息调节场景部分区域色调及透明度

游戏中的场景通常都由图形碎片拼成，场景一般已经比较丰富，可以满足游戏开发的需求，可是在一些特殊场景时需要对场景的某一块小部分的色调和透明度进行处理，又不能制作单独的一张图片来绘制。这时候需要特殊的技巧来完成。

在制作游戏地图的时候，通常是很多个方块，加以标记来告诉代码绘制哪一块碎片。在这里在这一层的上方覆盖上另外一块灰度图，用以加入额外的信息。颜色最深的值假设为 100，最浅为 0，那么通过灰度图得到了每一个像素上额外的一个信息，通过这个信息作为标记，可以在将缓冲的内容绘制到屏幕上之前再修改屏幕的内容，用来对场景等图片做资源强化处理。

第一步，读出原有缓冲上的像素值和覆盖层的值。

第二步，根据覆盖层的值修改缓冲上对应的像素点信息。比如对一部分场景的色调进行强化，将原有的 RGB 信息与白色混合，而混合的强度正好又读入的覆盖层的值 X 来决定。

```
r = (r + （FF - r）* X /100) & 0XFF0000;
g = (g + （FF - g）* X /100) & 0X00FF00;
b = (b + （FF - b）* X /100) & 0X0000FF;
```

第三步，将运算好的 RGB 信息写入缓冲。

这样就通过覆盖单独的一块图来实现对原有图的效果增强，当然这不仅仅只有色调增强，还包括了透明度的调整以及其他特效等等。

### 4.2.4 利用双缓冲和透明度调节绘制淡入淡出效果

在游戏中有时候需要进行两屏画面之间的切换过渡，不需要生硬的黑屏，而是需要画面的淡入淡出。

这时候就需要这样特效处理方法：将第二屏的内容保存在第一个缓冲中，逐渐减少第二屏内容的透明度，将第二屏的信息绘制到第二个缓冲上。伪代码如下：

```
void EffectLib_processFadeInAndOut(Image img, int alpha)
{
    while(所有像素)
    {//处理 img 内容
      buffer[iPixel] = alpha | (buffer[iPixel] & 0x00FFFFFF);
    }
    绘制到 buffer 上（）;
```

```
}
```

其中 alpha 的值由 0 到 100 根据时间轴慢慢增长即可，如果原图包含 alpha 或者某些像素有 alpha，则根据参数按照比例乘以最大值即可。

每一帧都绘制新的画面将第二个缓冲的内容拷贝到屏幕上，这样就可以实现淡入淡出的效果了。

### 4.2.5 线性插值算法进行图形资源的缩放

在游戏中有时候需要图片的缩放效果，需要在特效库中实现图形资源的缩放功能。

图像的缩放算法有很多，需要尽量精简，选择在时间和内存上符合 J2ME 不同设备性能的方法，所以在设计这一部分的时候要让使用特效库的程序员有所选择。由于这类算法一般需要运算比较多，所以在速度比较慢的手机上不采取动态缩放，而是在装载资源到内存的时候，根据设定好的标签创建好缩放图片以供使用。通常采取线性插值[16]这种插值算法，在计算出需要插值的点的同时，利用前后的点的值进行插值计算。

先看看放大的部分，在性能比较弱的手机上，针对一个图片放大时：如果是整数倍则直接用 1 个像素的色值填充到其他像素点里；如果不是整数倍则使用插值算法，间隔 N 个像素添加 1 个像素，这个像素的色值根据周围的像素平均来获得。在性能不错的手机上，对应整数倍放大也采取线性插值算法来实现放大。首先创建好放大后图片大小的缓存，再将像素点的值填充到每一块的左上角，首先算出有像素值的每 4 个点正中间的点的色值再以此类推，就可以算出所有的像素的值。

图形缩小的时候根据比例删减像素，一样的原理。由于手机游戏本身的尺寸都比较小，一般整个屏幕的宽高都在 200-300 左右，所以缩放 2 次以后，图片资源就已经很小了，所以整数尺寸的缩放比较少，一般都是需要进行删减像素的处理。伪代码：

```
void EffectLib_processScale（in percent）
{
    根据 percent 判断放大还是缩小（）；
    创建缩放后后存储图片所需空间（）；
    if(放大)
    {
        计算需要插值的频率（）；
```

```
            填入原有图像信息（）;
            插值运算（）;
        }
        else if(缩小)
        {
            计算需要删减的像素频率（）;
            将原图有效像素填入（）;
        }
    }
```

插值的算法由于手机性能的限制，一般采取线性插值算法，计算像素周边像素的平均值填入。

## 4.3 利用特效库简单粒子思想的特效实现

上节介绍了几个手机赛车游戏中特效的实现，采取了少量的图形加适当的代码控制的方法来实现这些特效。下面结合设计中的简单粒子系统和 J2ME 的性能来实现一些特效。

### 4.3.1 简单独立粒子思想与场景实际结合实现雪花效果

传统赛车游戏中通常没有很华丽的雪花效果，因为这部分的实现比较复杂，本段利用简单的粒子思想，结合游戏的特点制作出雪花的效果。当赛车运动时，场景内有一定数量的雪花，雪花按照一定的速度和随机的角度下落，当驾驶的赛车或者控制的人物在改变视角方向时，雪花应该按照镜头转动的方向相应的运动。首先看雪花的逻辑处理过程：

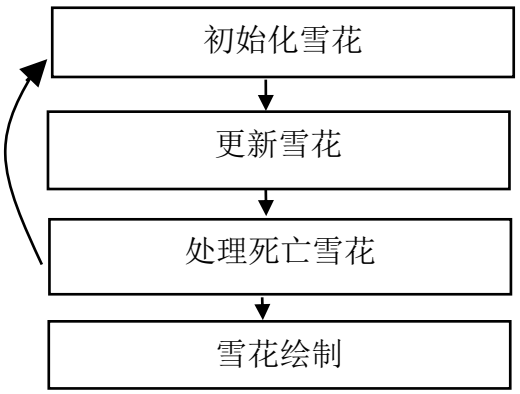


图 4.7 雪花特效实现结构

采用白色的像素点来代表雪花，一共三种大小，为宽度 1、2、3 的正方形，由于像素点比较小，这样已经足够表示。

首先设定屏幕内的雪花数量大小，假设为 120 个，先分配好雪花所需要的内存空间，避免游戏时反复申请，不仅产生内存碎片还浪费时间。需要声明的变量有雪花在屏幕上的 x、y 值，雪花缩放的大小，还有雪花经过换算过的在 x、y 方向上的速度。那么可得：

```
int  k_snowNum = 120;
int  snowX     =  new int[snowNum];
int  snowY     =  new int[snowNum];//位置
int  snowSpeedY =  new int[snowNum];
int  snowSpeedX =  new int[snowNum];//速度
int  snowSize  =  new int[snowNum];    //大小
```

**第一，雪花出生点的确定。**

这里我先讨论高速运动的情况：

在游戏静止的状态下，雪花应该是从屏幕的顶端均匀的下落，这毫无疑问。在汽车在场景中高速运动时，情况就不同了。由于汽车以较快速度行驶，而雪花仍然在缓慢下落，所以在镜头角度看来雪花不仅在下落，还在飞速的朝后方运动，这个时候由于速度比较快，看起来雪花出生的原点就是在前方而不是上方，X 轴方面雪花的出生点也集中于中间，而且是随着速度的加快向中间集中。同时这个时候雪花看起来就不是缓慢飘落了，而是物体不停的朝后方飞去。所以根据这一特点，还要设定一个焦点值，这一焦点值要根据车辆当前速度来改变。地平线在游戏中的高度为屏幕的一半加 10 个像素，那么为 320/2+10=170。在这里，假定汽车在时速 40 以下看到的雪花都是正常下降，而速度超过 210 时雪花看起来都是屏幕地平线中心飞出。这样取的原因是这样就是有 170 个单位和屏幕有 170 个像素点刚好吻合，方便于计算。那那么根据汽车的速度，这个焦点的坐标可以换算出。其中 carSpeed 为汽车速度。游戏的分辨率为 240*320。注意所有屏幕上坐标的速度单位都为像素/秒；

```
snowCenterY = (carSpeed* (170-0) / (210-40));//(40<=carSpeed<=210)
snowCenterY=0;// (0<=carSpeed<40)
snowCenterY=170;// (210<carSpeed)
```

而 snowCenterX 在速度大于 40 时一般可以认为趋近于屏幕中间，即

```
snowCenterX =120;
```

同时，在速度不变，镜头旋转的时候，雪花出生的 X 方向位置会稍微有所偏

移，根据赛车在转弯时与上一帧横向的位移来稍微调整 snowCenterX 值。由于赛道比较狭窄，这样的操作以及足以满足模拟的需要，在转弯时进行偏移，如果汽车回复直行，那么 snowCenterX 回复到中间。假定最多 snowCenterX 调整到屏幕 1/4 的位移，而赛道总宽度为 trackWidth，帧偏移值为 offX。则

```
snowCenterX = snowCenterX + (offX/trackWidth) *(240/4); // (offX!=0)
snowCenterX =120;// (offX!=0)
```

**第二，雪花运动速度。**

以上，有了对雪花产生焦点的基本模拟方法，下面要确认雪花的基本速度范围，后面在这个基本速度的基础上进行计算获得最终速度。对应于汽车的最高和最低速度，假定 40 时速以下时雪花的 x 方向为 0，最大时速 210 时雪花 X 方向的最大速度为 180，严格的来说这不是雪花的 X 方向速度，只是在屏幕这平面上看到的点的运动速度，因为在速度很高的时候所有的雪花看起来都是向后方飞，同样假定 Y 方向最小为 20，最大为 160。那么可以将赛车速度超过 40 时雪花飞行时偏移的基础速度根据速度的变化计算出来。

```
baseSpeedX = 0 + carSpeed* (180-0)/(210-40);// (40<=carSpeed<=210)
baseSpeedY = 0 + carSpeed* (160-20)/(210-40);// (40<=carSpeed<=210)
```

当然这个速度只是顺着前行方向的雪花的速度，前面还提到了焦点的问题，在焦点比较低的时候，焦点上方的雪花的运动是不一样的，在屏幕的视角看来，这些雪花是在上方漂浮着向后运动的，并不像侧面和下面的雪花是直接向后运动。这里要单独讨论一下，一般上方的雪花靠近中间的在镜头平面可以看做是朝上飞，而不靠近中间的是在 Y 方向不动而 X 方向有位移。

设定一个角度 20 度，在正上方左右 20 度以内的雪花让它们模拟向上运动，而其他焦点上方的雪花都向两边运动。如果在地平线下方，雪花并不是下落的很快，将其设定为 Y 速度的一半，同时由于是 Y 轴的负方向，所以设定为负数。

```
int tmp = abs (dx/dy);
if((tmp)>TAN_70)
    snowSpeedY = 0;
else
    {
        snowSpeedY = (TAN_70 - (tmp))*s_baseSpeedY/TAN_70;
        if(dy<0)
        {
            snowSpeedY >>=1;
            snowSpeedY = - snowSpeedY;
```

```
        }
    }
```

**第三，雪花静止状态。**

这样在运动情况下，雪花粒子在屏幕上运动的轨迹和速度都已经讨论完。下面讨论一下静止情况雪花的运动。由于在 Y 方向有 20 的速度，这点不用处理，而在 X 方向上为 0 这点并不科学，所以为这种情况下 X 方向设置一个初始速度 speedX，让雪花按照这个速度飘落 1 秒，在 1 秒后，由函数随机 RAND（0,1）来确定它是继续这样飘还是反方向飘，这样雪花就有了很多飘落的轨迹，得到了比较真实的感觉。那么代码这样：

```
if(timePassed>=1000)
{
   nextDirection = Rand(0,1);
   if(nextDirection!=curDirection)
     snowSpeedX = - snowSpeedX;
   timePassed -= 1000;
}
else
    timePassed += dt;
```

其中 nextDiretion 和 curDirection 代表下一个方向和目前的方向，而 timePassed 为时间计时器。

**第四，雪花的缩放以及生命周期**

雪花有三种大小，需要实现他们的自动缩放。所有的雪花基于当前的速度都会获得一个增量，当最小速度时设定为 15，最大速度时设定为 40，每一帧画面刷新都会加上这个增量，当增量累积到一定值时，比如 1000，就将雪花的大小增大一个等级，当雪花等级已经为最大时，则销毁这个雪花，使用这个雪花占用的内存空间在焦点附近新建一个随机位置和速度的雪花。

```
snowSizeZoomStep = 14 + carSpeed* (40-15)/(210-40);
snowSize[i] += snowSizeZoomStep;
if(snowSize[i]>3000)
  resetSnow(i);
```

其中，resetSnow()函数的功能就是销毁一个雪花，新建一个雪花。

到这里就已经讨论了所有情况下每个粒子基于屏幕中的位置和当前的赛车速度所用拥有的速度、缩放的规律以及生命周期的问题。还讨论了这些情况下焦点的存在位置。将其放到函数中方便接下来的逻辑中使用。

```
updateCenterPostion();//更新焦点的位置
updateSnowInfo(int carSpeed);//更新每一个雪花的位置和速度
```

下面要基于上面的成果对成群的雪花进行管理。前面已经定义过 k_snowNum 为雪花的数量，我首先需要初始化所有的雪花，遍历所有的雪花数量，在 X 方向给一个随机初始位置，x、y 方向的速度也给予范围内的随机值，雪花的大小也给予一个随机值。

```
newSnowDot(int snowID);
```

初始化以后使用前面提到的 2 个函数来更新雪花信息，每帧运行，同时将雪花绘制到缓冲上。最后进行判断，对大小大于 3 和出屏幕的雪花进行重生操作。

```
{
    ……
    updateCenterPostion();//更新焦点的位置
    updateSnowInfo(carSpeed);//更新每一个雪花的位置和速度
    drawAllSnowDotsOnBuffer();
    checkAndResetDots();
    ……
}
```

这样整个赛车专用的雪花特效类就设计完成了，该雪花特效类不仅可以用于赛车游戏，还可以用于赛道滑雪游戏、雪地摩托车游戏等相似题材的游戏中，下图是使用这个方法完成的游戏特效截图：

图 4.8 雪花特效效果图

对于雪花特效还有很多拓展工作可以进行，在这个简单的框架里可以改写绘

图部分，可以很容易的实现其他方式的雪花样式。当然也可以对雪花散布的方式进行拓展，给雪花一个三维的坐标，然后通过计算在不同面上的投影，即可以实现完全的三维场景里的雪花效果。

### 4.3.2 粒子簇思想设计赛车加速喷火特效

前面几小节详解了特效系统的实现，下面用特效系统来制作一下赛车游戏中使用加速功能后赛车的加速喷火装置的特效。这是一个典型的粒子簇的构造，需要根据火焰不同部位的特性来制作。

效果图如下：

图 4.9 喷火特效的粒子实现

首先，针对火焰的不同部分形成不同的粒子产生公式[17]：

详细分析外焰的现状，上面小部分称之为产生区，火焰由一点呈三角形形状或者梯形现状喷出。中间一部分称之为稳定区，是整体外形接近矩形或者椭圆形的稳定区域，这部分粒子的宽度变化不大。下面一部分为衰减区，粒子为一个倒梯形分布，但是衰减的幅度有限。最后一部分为消亡区，在衰减区的基础上粒子迅速减少，集中到一点。内焰的形状则基本上外焰的现状去除外部火苗粒子即可得到。最后得到粒子的分布公式 X=F（Y）。以下为示意图：

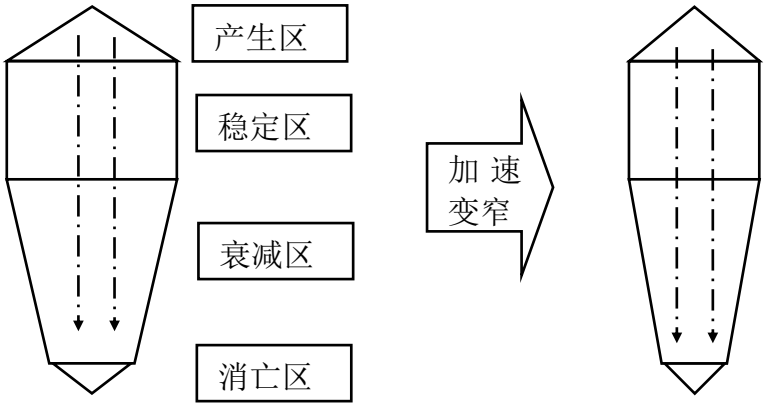


图 4.10 火焰粒子分析

然后，根据游戏逻辑中的速度，粒子簇整体的宽度将变窄，此时粒子簇的分布公式为 X=F’（Y）。同时注意到火焰的内焰变窄幅度比较小。

每一个粒子的运动轨迹都遵循着从产生区一直到消亡区的发展规律，他们的位移也根据不同区的形状而有所体现。

该特效设计的绘制逻辑主要是根据公式，围绕喷射的直线为中心小部分区域填充白色，其他区域填充淡红到红色。到此，该火焰特效设计完成了。

## 4.4 小结

本章实现了特效库从游戏需求方面的实际特效、一些通用方面和粒子系统的特效等一些代表性的特效，还有很多特效没有一一列出。

游戏专用特效部分从特定的游戏类型入手，描述了几种赛车等游戏特效的设计和实现过程，主要结合游戏的特性、利用优化好的图像精灵作为基础，重点详细分析业务逻辑，并结合 J2ME 性能特点制作特效。

通用特效方面主要基于图像精灵的改进，针对很多游戏通用的需求在图形资源的处理上开展工作。

粒子系统部分根据前面详细分析的粒子系统的特点，并在 J2ME 性能的承受能力基础上针对不同特效的需求给出了数据结构以及详细处理逻辑。

# 第五章 优化设计

在 J2ME 游戏开发的过程中，面临着各种各样的限制，来自于内存、性能、手机对 JAR 包大小的限制[18]等等，这一章结合游戏的总体设计，在特效库的实现方面进一步优化，提高游戏中特效库使用各种资源和系统的效率。

## 5.1 图像以及其他资源打包的优化

这一节里对图形部分的资源进行优化，主要从图形资源的容量，数据包的容量压缩方面入手。

### 5.1.1 图形文件格式改进

在手机游戏开发过程中，丰富的场景内容和不同特色的各种美观的画面是必不可少的元素，可是在实际的开发过程中图形资源在 JAR 包中占用的空间非常的大，如果不进行压缩直接使用 PNG 格式来开发游戏，那么一个游戏中的内容就非常有限了。手机游戏开发中通常都是使用 PNG 格式打包，PNG[19]的格式如下图：

表 5.1 PNG 文件格式表

| 文件头 | IHDR | PLTE | tRNS | IDAT | 文件尾 |
|---|---|---|---|---|---|

| Length | Chunk Type Code | Chunk Data | CRC |
|---|---|---|---|

除了文件头和文件尾，PNG 图片包含很多其他的信息块，每一个信息块又包含下面一张图中的信息：长度、类别、数据、CRC 码等。显然这样的格式决定了 PNG 文件的体积，有很多信息，最重要的是在数据部分很多数据在绘制的时候并不是所需要的，需要对其进行精简。

首先，在游戏中所使用的颜色数比较少，最多 256 个颜色。很多比较简单的图形资源所需的颜色数更少，一致按照一种颜色来存储显然会浪费很多空间。其次，在打包过程中应该舍弃那些不用的字段。为此，建立一种新的图片格式，命名它为.PG 文件，那么它的结构如下图：

表 5.2 新的文件格式表

| 文件总大小 | 调色板颜色数 | 调色板数据 | 图片大小和宽高信息 | 根据调色板索引的图片信息 |
|---|---|---|---|---|

新格式文件打包过程设计：在文件打包的过程中，首先提取计算出图片的颜色数量，一般手机游戏开发的过程中，对美术资源的颜色数量是严格控制的。一般有 8 色、16 色、32 色、64 色、128 色、256 色等，略大于某个级别的可以稍微减少一些颜色，尽量做到充分利用颜色数量。计算出颜色数量后，将这些颜色建立成调色板放入调色板数据块中。调色板部分如下图：

表 5.3 新的文件格式表

| 文件大小 | 调色板颜色数 | 调色板数据 |
|---|---|---|
| 固定使用 8 个字节 | 固定使用 2 个字节（16 位） | 颜色数 * 6 个字节（16 位*3） |

在 J2ME 游戏开发过程中，每个字节的容量都需要仔细使用。调色板数量考虑到最多为 256 种，所以使用 1 个字节足以表达出他的数量，而调色板的数据考虑到 RGB 信息每个需要使用 3 种颜色各 16 位，所以为 6 个字节。这样的格式足以存储所有的调色板信息和颜色信息。

计算完调色板的信息后，在后面紧接着填入图片的宽度和高度信息，各使用 4 个字节，一共 8 个字节即可。由于图片是矩形图片，所以高度乘以宽度即为图片尺寸的大小。

最后一部分即为图片的信息了，每一个像素采用一定的位数来索引调色板里对应的颜色，位数则由颜色数来决定。其中：

黑白图片每 8 个像素使用 1 个字节，即 1 位，4 色图片每个像素使用 2 位，8 色为 3 位，16 色位 4 位，32 色为 5 位，64 色 6 位，128 色 7 位，256 色 8 位。如果打包图片含有半透明信息，则在打包的图片信息后加入半透明的部分，如下表：

表 5.4 新的文件格式表

| 文件总大小 | 调色板颜色数 | 调色板数据 | 图片大小和宽高信息 | 根据调色板索引的图片信息 | 半透明级别数 | 半透明信息 |
|---|---|---|---|---|---|---|

首先半透明级别数部分可以用 1 个字节来表示，即从 0 到 255，具体读取时加上 1 即可，如果读取数值为 0，即只有 1 种那么就为不存在半透明效果。半透明的级别也根据需要的级别来计算字节数，最多的 256 个级别需要 8 位数据，即 1 个字节来表示。其他不同的颜色数所使用的数据位数量根据亲们的半透明级别数来定，与前面描述的颜色数类似。

这样，就完成了一个非常精简节约的数据结构。不但完整的表示了所有图片信息的内容，还可以通过图片的大小和文件的大小等对图片的完整性进行校验。所有数组计算完毕后，计算好总文件大小填入文件开始并打包成为一个 2 进制文件。

数据的打包部分完成了，那么数据的读取部分该如何设计呢?

显然需要特定的读取计算方式来读取，

（1）首先读取到文件的总大小信息，存储在一个 int 值中。假设 Read 为以位数为参数读取后自动偏移的底层函数。

```
int dataLen = Read(8);
```

（2）然后读取 1 个字节的调色板数：

```
unsigned byte palNum = Read(8);//0 到 255 使用时+1
```

（3）然后根据调色板数量来读取调色板数据。

```
int palData[] = new unsigned byte [palNum+1];
int i=0;
while(i++<256)
   palData[i] = Read(48) & 0XFFFFFF;
```

（4）读取文件的大小

```
int height = Read(32);
int width = Read(32);
```

(5) 根据调色板数量来算出每个像素的位数，再读取每个像素的索引值。假设这个函数为 GetBitByNum（int num）那么

```
int bitUsedPerPixel = GetBitByNum(palNum);
int colorData [] = new int [height* width];
int i=0;
while(i++<( height* width))
   colorData[i] = Read (bitUsedPerPixel);
```

(6) 采取和第 5 步类似的方法读取出半透明的值。

(7) 根据前面的图片大小信息和文件大小信息以及最后文件读取的偏移值来验证文件读取的完整性。

(8) 根据图片信息创建游戏中需要的 Image 对象或者是 RGB 数组信息。

这样就完成了读取部分函数的编写，将这个函数完善，即可作为标准读入使用。

小结：经过这样的图像格式的设计，可以说在 J2ME 开发的过程中图片信息资源的压缩和优化已经达到了极致。如果对图像信息和半透明信息部分再采用 RLE 压缩，那么实践中平均的文件大小仅为原始的 PNG 文件大小的 30%到 50%，如果颜色数比较少，有可能达到更低。同时数据打包成自定义的格式也让自己游戏中的资源不被其他人破解。

### 5.1.2 用自定义 RLE 算法对资源进行优化

RLE 算法[20]非常的通俗易懂，即将 AAABBBBBCCCCCC 这样的数据序列打包成为 A3B5C6 这样的数据结构，优点是在大量数据重复的情况下可以大量的缩短文件的长度，便于存储，是无损压缩。

在游戏的图像，音乐，文字等信息已经打包成 2 进制的基础上使用 RLE 算法进行压缩，可以取得不错的压缩率。同时可以在压缩前用几种不同的 RLE 压缩方法进行比较，取最优的方法。可以使资源的 2 进制包缩小到极致。

有以下两种方法：

（1）将数据打包成分成 1 位+3 位这样的结构循环读取。其中第一位为 0 或者 1，后面 3 为代表了前面 1 位内容的数量。由于不同的数据重复度情况不太一样，可以经过试验选取将 3 改成其他数字。如果数量段全为 0 代表有 1 个，其他依次类推加 1。见下图例子：

表 5.5 RLE 打包方式之一

| 0 | 0101 | 1 | 1010 |
|---|---|---|---|
| 后面重复的为 0 | 5+1 = 6 个重复的 0 | 后面重复的为 1 | 10+1 =11 个重复的 1 |

此图的数据表示 00000011111111111，原来的信息为 17 位，打包后为 10 位，减少了 7 位，此方法适用于重复次数较多的情况，实际使用中可以先通过 2 进制重复次数进行统计再觉决定后面数据段的长度。

(2) 将数据打包成 2 种，第一位 0 或者 1 表示重复数据和不重复数据，如果此为 0，下面一位 0 或者 1 表示内容，紧接着 N 为表示重复的数量。如果第一位为 1，则紧接着 M 位表示不重复数据段的位数，后面跟着不重复的数据段。见下图。同样个数+1 类推。

表 5.6 RLE 打包方式之二

| 0 | 1 | 101 | 1 | 1011 | 101100111010 | 0 | 1 | 011 |
|---|---|---|---|---|---|---|---|---|
| 重复内容 | 内容为 1 | 数量为 5+1 | 不重复内容 | 数量为 12 | 内容 | 重复内容 | 内容为 1 | 数量为 4 |

此数据的 N 为 3，M 为 4。解析为 6 个 1 加上 101100111010 加上 4 个 1，即：

111111 101100111010 1111

这此方法的 N 和 M 和第一种方法一样，需要对实际的数据进行分析统计，来决定重复的个数和实际的参数，才能实现最优化的性能。

### 5.1.3 使用 LZMA 等其他字典算法对资源压缩打包

在经过资源的处理和其他方法的优化后，还可以利用 LZMA[21]等成熟的字典压缩算法对每个单独的数据包进行压缩打包，可以减少很多容量。这类字典算法对数据进行分析，拿出大量的重复字段建立为字典，在原位置替换为索引，读取时将索引再替换为所需的值。

这样处理后的资源和其他的压缩算法一样在使用资源之前需要对资源进行解压。这样势必会损耗游戏载入消耗的时间。需要在对手机的性能和速度综合考量后再使用。

一般在游戏打包时采用 LZMA 算法在资源经过前两小节压缩的基础上可以再减少 5%左右的包容量。

### 5.1.4 混淆器优化代码

使用混淆器[22]可以使代码在打包时自动检测并删除没有用到的类、数据和方法，通过对代码的缩减来缩小游戏 JAR 包的大小。

目前最常使用的是 ProGuard[23]。

## 5.2 内存使用优化

本小节在内存的使用上对特效库的实现部分以及在 J2ME 游戏开发过程中的其他方面进行优化。

### 5.2.1 特效库资源的预装载优化

在使用特效库的游戏开发过程中，需要关注资源装载过程的配置以及相关对象、变量的使用，进行合理的规划配置可以减少很多内存使用不当带来的问题，大大加速后期的开发测试进程。

首先，特效库所使用的全屏缓冲在游戏启动时就已经创建好，如此大块且不断使用的内存应该开辟在前面，创建即不释放。

其次，将游戏上层开发工具比如场景编辑器等与特效库结合使用，合理分配好图片等资源的装载顺序，将大块的资源有限装载，同时游戏装载资源和游戏循环严格分开。对于不同的资源制定不同的使用策略：对于在复杂游戏场景中使用次数较多的资源，应该优先在内存中创建为 MIDP 支持良好的 Image 对象，可以提高使用效率；对于在静态场景中使用的大块的图片，由于对重绘性能要求不高可以创建成为 RGB 信息的数组储存在内存中，绘制时使用特效库的相关函数直接绘制即可。在这一部分，需要对所有的资源的使用频率、占用资源量和图片规模大小进行综合考量再分配使用的方式。

再次，对于使用频率较高的特效的相关资源预先装载。在很多特效中需要频繁的使用 RGB 缓存、不同的调色板信息等特殊的内容。一般这些资源都是在游戏丰富的场景时绘制，对于 RGB 缓存要创建足够大，对于使用次数频繁的调色板也可以直接创建为图片或者 RGB 数组信息，避免在图片绘制时频繁的临时进行调色板替换操作，同时也减少内存碎片产生的概率。

还有，要优化好游戏的状态机，在游戏菜单界面和游戏复杂场景之间切换时，合理的分配好装载和释放资源的操作，不用的资源尽早释放，在频繁装载释放时可以适当的进行主动的垃圾回收。同时还应注意在大量的进行 IO 操作的时候，尽可能的使用缓存，可以避免很多问题。

### 5.2.2 对象池变量池与优化

在游戏中除了游戏资源的装载，最容易产生内存问题的无疑是大量临时变量、全局变量和对象的申请与消亡，这类操作由于数量较多，极易产生大量的内存碎片。

解决的方法就是根据游戏的需要，对于不同的状态下预先申请好足够用的对象池，在每个对象创建时内部也可以预先保留一定数量的变量池。这样的好处是在游戏装载完毕后，游戏的主循环内基本上不需要再进行 new 和释放的操作，所有的操作都是针对申请好的内存的寻址操作，不仅改善了代码的规范性，也避免了这个过程中内存碎片的产生。

其次，减少类的数量，避免类过多产生的系统问题，一般在 J2ME 游戏开发中类的数量不超过 7 个。

其次，对于大规模的配置数组应该统一的打包到一个二进制文件中，游戏装载时通过载入资源的方式读入到内存中，这样可以减少静态区的使用，优化内存性能。

### 5.2.3 主动内存清理机制

这小节主要为了性能比较差的 J2ME 设备考虑，在制作一款游戏的过程中，性能比较强的手机在进行上面 2 小节的优化后一般不会有问题，然而由于硬件厂商的设计不同，很多手机存在着这样或者那样的内存缺陷，需要进行主动的处理。

首先分析内存不足的情况，要明确游戏对于内存的使用量以及手机的内存情况，大部分有问题的手机在经过减内存的优化后可以运行游戏，但是时常会在内存使用到达峰值的时候死机。在游戏逻辑内部经过评估和分析，找出和确定一些系统资源容易到达峰值的时间点，在逻辑之前主动进行内存清理。同时对此部分的场景进行优化，尽量使游戏的内存使用平缓。

然后分析由于手机设备问题产生内存碎片的情况。很多手机由于系统的原因，在游戏逻辑很完善的情况下仍然会产生大量的碎片，或者由于一些原因但是无法查明的情况下，需要主动的清理内存来确保游戏的流畅运行。这类游戏需要在游戏中内存不操作的时候以及场景简单不需要大量处理操作的时候主动的进行 GC()(JAVA 的系统清理垃圾函数)，来确保内存的使用。同时要避免在 IO 操作的时候进行 GC()。

最后，注意不使用的对象和数据一定要主动置 null。

## 5.3 绘制速度优化

本小节主要讨论特效库开发中的绘制优化技术。J2ME 循环中大部分的时间都用在了绘图上，所以优化是非常重要的。

### 5.3.1 灵活的重绘技巧

在游戏中，并不是所有场景所有内容每一帧都需要重绘，需要灵活的设置一些标签，来标记绘图函数中的一些逻辑在这一帧中是否要运行。这个标签与游戏逻辑息息相关，需要在一些模块设计时优先考虑好。

同时每绘制一个图像时也不需要刷新整个屏幕，对系统刷新函数 SetClip 的

使用也需要注意尽量只去刷新需要刷新的小区域，避免经常性的全屏幕刷新。

要尽量的平衡好各种特效以及半透明的覆盖程度，目前手机在绘制和处理半透明图像时速度比较缓慢，在游戏设计的过程中要平衡好游戏中使用不同的特效以及半透明处理的关系，尽量的平缓，不要在同一屏绘制太多的特效，这样会让游戏的速度难以控制。

在游戏的大循环中要加入限制帧数的代码，将游戏的速度限制到某一个能稳定达到的帧数，这样不至于在太简单的画面里游戏运行的速度太快导致让玩家感觉游戏时快时慢难以适应。

### 5.3.2 灵活的运用好缓冲

在特效库中，建立了一块离屏缓冲以及一些 RGB 缓存用来配合特效的使用，对于这些缓冲区的灵活使用，可以提高绘制的效率。

首先，对于大量场景的地图，一定要采用卷轴算法来避免临时的绘制地图。在这类游戏中，创建一块比屏幕宽 2 个基本地图块宽度的缓冲，在主角移动时，自动的更新最边缘的地图，那一部分并不显示在屏幕上，绘制时不要实时绘制地图，而是从缓冲中把地图拷贝到屏幕中。这样可以保证每次屏幕上显示的部分是已经创建好的，可以大大的提高速度。

其次，对于缓冲的内存区要灵活使用。很多缓冲是为了临时存储一些图像信息使用，可以提高这些缓冲的重复利用率，在缓冲空闲时可以提前读取资源绘制到缓冲上以备后面的游戏逻辑使用。这需要与不同的游戏逻辑相结合。

## 5.4 代码和调试工具协助优化

在 WTK 的工具中，提供了 Profiler 工具来协助开发者检测代码的效率问题，在运行完程序并且关闭后，通过设置 Edit-Preferences 里的 Monitor，勾上 Enable Profiling 即可打开性能监控窗口，在窗口中函数在模拟器运行的时间次数等一幕了然；要选中 Enable Memory Monitor 复选框即可实时的查看内存的使用情况。，可以通过此了解到自己游戏逻辑中代码的性能。当然这只是模拟器上的数据，还可以通过联机调试来测试性能。

在编写游戏代码的时候，对于需要浪费大量时间的 AI 运算部分和绘图部分，预先在 DEBUG 版本中写好时间计算代码，可以大量的节约调试时间，那一部分代码和绘图消耗时间最长可以很方便的看出。

在游戏模拟器方面，可以从网络上下载到很多比较好的模拟器，很多可以对内存，资源的使用情况可以方便的进行查看，协助开发者运行调试和优化。比较著名的有 JBLEND 等等。WTK 提供的模拟器也是一个不错的选择。

## 5.5 小结

本章针对 J2ME 的情况，从图形资源的打包、绘制、内存的优化等几个方面对特效库进行了优化设计，减少了图形资源的容量、内存消耗，提高了绘制速度。

# 第六章 结合特效库开发游戏

本文介绍的特效库已经大部分开发完善并且在实际项目中使用，本章简单介绍一下特效库在实际项目中的运用，并以一个实际的项目作为例子简单分析。

## 6.1 开发中加入特效库

在开发中加入特效库，代码部分只要将特效库编译后，打包成为.class 文件，在各种 IDE 中加入到项目目录里即可使用相关函数。在图形资源打包方面，使用已经打包好的 EXE 加入到自动化打包流程中，将图片自动打包成本文自定义的.PG 格式文件。并由特效库中提供的高层绘图函数来进行绘制。

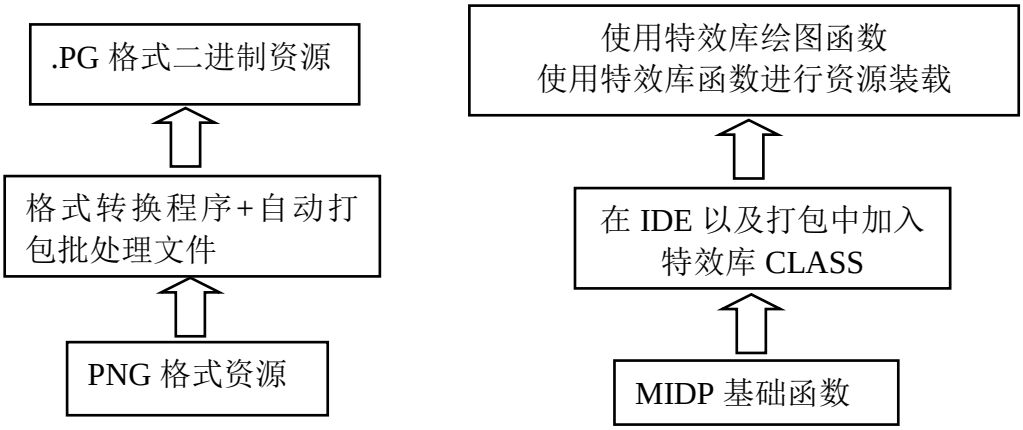


图 6.1 项目开发中绘图方式的变化

在代码打包时也需要将特效库 class 加入其中。自动打包一般可以采用批处理文件来完成。而图形资源则直接打包为二进制文件。

打包完毕处理后，在开发时，在游戏装载逻辑中写入相关装载的代码即可在不同的场景中使用特效。

## 6.2 使用特效库开发赛车游戏

下面简单介绍一下采用的特效库的不同种类游戏开发：

首先在游戏资源装载方面，不同特效需要的美术资源需要预先装载好，在游戏进入游戏操作界面之前也需要将所有需要的缓存创建完毕。

然后在游戏逻辑循环中根据需要改变各种特效的参数，实时的更新缓存进行绘制即可。

这款赛车游戏为目前正在开发的一款游戏，使用了一些场景的特效，天气变化、闪电，雨雪等效果，还有一些全屏幕的速度线效果，与此类游戏类似的还有滑雪游戏等。见图 6.2，左上角为菜单，采用了半透明技术制作的发光菜单，比较美观。右上角的图中有尾气的粒子效果，速度线效果，以及小光圈效果。左下角的图为轮胎与地面摩擦的特效以及下雨特效。右下角的图为下雪特效以及汽车冒烟特效。


图 6.2 赛车游戏中特效图

# 第七章 总结与展望

## 7.1 总结

手机游戏中的特效开发一直是手机游戏开发中最为重要的环节之一,为J2ME手机游戏开发的过程进行优化，加入特效库，可以极大的方便手机游戏的开发，增强手机游戏的图形限制效果，具有重要意义。本文针对这一方面，进行了如下的工作：

1. 研究并分析了现有市场上手机游戏特效方面的不足，从不同类型的游戏逐一分析，明确了目前游戏开发中对于特效的需求，分析了当前开发架构的不足和主要技术限制。
2. 针对目前常见游戏开发架构的不足，对游戏的开发架构进行改造。对图形图像处理的部分进行优化，将特效库置于开发架构中，使用特效库的技术简化开发流程，提高开发效率，提高游戏特效水平。
3. 改进了 J2ME 现有图形精灵引擎，并进行大量拓展，使之适应更高级的开发需求。
4. 结合现有需求，设计了特效库的架构，并针对操控类游戏的类型实现了部分特效。在此基础上还实现了简单的粒子系统和基于图形引擎的一些通用特效。
5. 改进优化了 J2ME 游戏开发中的图像压缩技术，提出了一种自定义的图像格式以及相应压缩方法，可以针对 J2ME 开发特点并结合特效库的特点大幅度的压缩图片资源。同时还提出了内存、绘制、调试等方面的一些优化技术。

本文的主要创新点为：

1. 基于现有 J2ME 的开发架构，针对 J2ME 游戏开发的特点进行研究，针对 J2ME 游戏开发框架的一些不足进行优化。
2. 提出和实现了适用于 J2ME 游戏使用的特效库。
3. 提出了一个适用手机开发的像素图片格式和一些新的优化技术。

## 7.2 展望

目前本文已经实现了一个特效库的框架，并且遵循这个开发框架开发了一些常用的特效，这些特效都已经在正在开发或者已经开发的一些游戏中得到了应用，在项目实践中得到了验证。但是游戏的类型非常的多，其中的需求也纷繁复杂，很多方面还需要进一步的完善：

1. 手机游戏类型非常的繁多，仅以一篇论文描述的内容是远远不够的，后期需要分析大量的不同类型的游戏，总结出更多通用的特效并加以实现，丰富特效库的内容。
2. 在特效库中的性能分析模块需要进行大量的拓展，在不同的特效使用时可以更详细的分析不同设备上的性能数据，以更好的优化特效库的开发性能和加速游戏的开发。

# 参考文献

[1]栾慧. 2009手机用户调研数据[EB/OL]. 艾瑞咨询.http://wenku.baidu.com/view/d908ce3383c4bb4cf7ecd1a3.html.2009:8,15

[2]杨光宏. 基于BREW平台应用开发研究[D].电子科技大学. 2005:10,13

[3]蒋媛. 智能手机在Symbian OS S60平台下的应用开发[D].成都理工大学. 2009:13,16

[4]储然. iPhone手机影响分析[J].现代电信科技. 2007(10):1,5

[5]党李成.基于Google Android智能手机平台的研究与应用[D].安徽大学.2010:16,27

[6]李振鹏,龚剑. J2ME手机游戏开发技术详解[M]. 清华大学出版社. 2006:5，27

[7]Antero Taivalsaari. JSR 30: Connected, Limited Device Configuration [S]. Sun Microsystems, Inc. 2000:

[8]覃宇. 基于J2ME平台的手机游戏开发中的若干关键问题的研究[D] . 电子科技大学. 2006:12,13

[9] 刘洪星，谢玉山.Eclipse开发平台及其应用[J].武汉理工大学学报(信息与管理工程版)(2). 2005:1,2

[10]Paul Su. JSR 118:Mobile Information Device Profile 2.0[S] . Aplix Corporation. 2006:

[11]王峰，王向阳，杨红颖.Windows环境下的逻辑调色板及其应用[J].计算机工程.1999(4):25,26

[12]李远泰,冯永晋.基于J2ME技术的手机游戏图形处理[J]. 中国科技信息.2005(16).2005:1,2

[13]谢海军，陶宏才.手机游戏中的2D卷轴算法及其实现[J].成都信息工程学院学报.2007(4):1,3

[14]王津.基于J2ME平台的手机游戏开发中的若干关键问题的研究[D] . 电子科技大学. 2007:43,44

[15]R.A. Haddad and A.N. Akansu. A Class of Fast Gaussian Binomial Filters for Speech and Image Processing[J], IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 39, March 1991：723，727

[16]Meijering, Erik. A chronology of interpolation: from ancient

astronomy to modern signal and image processing[J]. Proceedings of the IEEE 90 (3) .2002: 319,342

[17]齐路.基于粒子系统的流体模拟在J2ME平台上的研究与实现[D].中南大学.2009:22,23

[18]韩冬. 基于J2ME手机网络RPG游戏的性能优化[D].同济大学.2007:24,26

[19]朴红吉，孙宇哲.PNG图像压缩研究[J].现代计算机. (259):1,2

[20]苟列红，李学春.RLE压缩算法的分析及应用[J]. 现代计算机.1997(8):1,3

[21]Igor Pavlov.LZMA SDK (Software Development Kit)[EB/OL]. http://www.7-zip.org/sdk.html.2009:1

[22]史扬，曹立明，王小平.Java混淆器的设计与实现[J].计算机应用.2004(11):1,2

[23]刘飞.在嵌入设备中提升Java代码执行效率的研究[D] .武汉大学.2004:57,62

# 致 谢

本次论文写作从立题、搜集资料、起草到最后成文得到了许多老师、同学、同事和亲友的帮助。在此表示衷心地感谢。

首先要感谢徐迎晓老师。导师严谨的治学态度、强烈的敬业精神和开阔的专业视野给我留下了深刻的印象。在论文的选题、资料收集、撰写和最后的成文过程中，得到了导师悉心的指导。他在繁忙的教学任务中仍多次给我提出了许多宝贵的意见与修改意见。在此表示深深的谢意。

其次需要感谢张军平和李旻等老师。在读研究生期间，他们传授给我大量的学习方法和知识，这些都是我写作论文的基础。

还有要感谢我的同学谭哲峰、宋荣，在我平时的学习中给予了很多帮助。

最后需要感谢我的家人，尤其是我的妻子对我的支持，使我得以安心地进行论文的写作。

# 论文独创性声明

本论文是我个人在导师指导下进行的研究工作及取得的研究成果。论文中除了特别加以标注和致谢的地方外，不包含其他人或其它机构已经发表或撰写过的研究成果。其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意。

作者签名：_____ 日期：_____

# 论文使用授权声明

本人完全了解复旦大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。保密的论文在解密后遵守此规定。

作者签名：_____ 导师签名：_____ 日期：_____