Computer Science and Application 计算机科学与应用, 2026, 16(5), 508-513
Published Online May 2026 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2026.165201

Hans汉斯

基于分块处理的历程图快速绘制方法

印中杰，杜庆宣，董义俊，潘以晖，严　伟

杭州应用声学研究所，浙江 杭州

收稿日期：2026年4月23日；录用日期：2026年5月22日；发布日期：2026年5月29日

摘　要

在声呐设备显示控制终端中，历程图是方位能量谱、Demon/Lofar谱的常用展现形式，因其经常需要在大缓存背景下实现全图重绘，为保障显示效果，对显示数据的连续性和人机互动的响应速度要求极高。针对绘制大数据量历程图在历程图绘制与人机交互过程中存在的内存分配频繁、缓存命中率低等性能瓶颈，文章提出一种基于分块处理的快速绘制方法。该方法基于CPU内存访问特点与历程图实际绘制需求对后台数据进行合理分块，优化内存访问模式。实际验证结果表明，该分块处理方法显著提高了全图重绘的响应速度，提升了声呐设备显示控制终端绘制大数据历程图的性能与画面流畅度。

关键词

显示控制终端，大数据量历程图，分块处理，合理分块，内存访问模式

A Fast Drawing Method of Track Plot Based on Block Processing

Zhongjie Yin, Qingxuan Du, Yijun Dong, Yihui Pan, Wei Yan

Hangzhou Institute of Applied Acoustics, Hangzhou Zhejiang

Received: April 23, 2026; accepted: May 22, 2026; published: May 29, 2026

Abstract

In the display control terminal of sonar equipment, the track plot is a common form of presenting the azimuth energy spectrum and Demon/Lofar spectrum. Due to the frequent need for full-screen redrawing in a large cache background, the requirements for the continuity of display data and the response speed of human-computer interaction are extremely high. To address the performance bottlenecks, such as frequent memory allocation and low cache hit rate, in the process of drawing large data track plots, this paper proposes a fast drawing method based on block processing. This method rationally divides the background data based on the characteristics of CPU memory access

文章引用: 印中杰, 杜庆宣, 董义俊, 潘以晖, 严伟. 基于分块处理的历程图快速绘制方法[J]. 计算机科学与应用, 2026, 16(5): 508-513. DOI: 10.12677/csa.2026.165201

and the actual drawing requirements of the track plot, optimizing the memory access mode. The actual verification results show that this block processing method significantly improves the response speed of full-screen redrawing and enhances the performance and smoothness of drawing large data track plots in the display control terminal of sonar equipment.

Keywords

Display Control Terminal, Large Data Track Plot, Block Processing, Rational Division, Memory Access Mode

Copyright © 2026 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

1. 引言

在声呐设备显示控制终端[1]中，方位能量谱、Demon/Lofar 谱等图像的绘制常常采用历程图的形式，历程图能够通过各位置像素颜色或明暗深浅将探测结果很好地反馈于使用者。实际使用中，能量谱数据接收后往往需要按照整行去绘制历程图最上方一层像素点，通过逐行填充最终形成完整的能量谱图像。Qt 框架因为具备良好的封装机制、独特的信号槽机制以及友好的图形控件常常用于界面的开发[2]。QImage 类作为 Qt 框架中常用的图像处理类[3]，其像素颜色填充能力优秀，常常应用于历程图的绘制[4]。如图 1 所示，基于 QImage 能够便捷且高精度地将中间区域像素填充为红色，其他区域像素填充为蓝色。

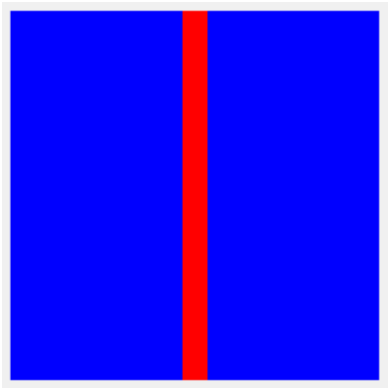

Figure 1. Example of QImage pixel filling
图 1. QImage 填充像素示例图

随着采集能力的提升、画面精度要求的提高，历程图所需展示的数据量也显著增大，庞大的数据量最终需要反馈于已经预先设定好尺寸的终端 UI 界面中。这种情况下，历程图若进行需要遍历全部原始数据的人机交互操作时(如全图幅度缩放、全图画面明暗变更、全图画面回溯重绘等操作)，将需要在短时间将大量后台数据传递给预先制定好尺寸的 UI 界面中；且此类操作可能为短时多次交互操作。此时，大数据传递过程反复进行，画面重绘等待时间长、画面流畅度下降。

传统方法采用组合优化的方法对历程图进行绘制：将上一次的绘制结果放置于临时画布上，只针对新一行的数据进行绘制[5]。该方法应用于小数据量历程图时能够显著提升效率，但是进行大数据量历程图全图绘制时，大块内存分配频繁、缓存命中率低将成为影响绘图效率的主要因素，仅采用组合优化的

绘制方法并不能完全解决这些问题。

针对上述问题，本文对 QImage 类的复制函数及缩放函数的实现原理进行了详细分析，提出一种对全图进行分块处理的历程图快速绘制方法，针对不同规模的图像计算出最佳分块结果，并基于不同分块大小进行多次测试，使用 QTime 类记录整个图像复制缩放开始前后时间，测试结果取平均值与计算结果进行比较，最终验证优化结果。

2. 主要过程耗时原因分析

2.1. 复制过程耗时分析

QImage::copy()函数的作用主要为通过深拷贝创建一个原始图像的复制对象，将原始图像中的部分或全部内容进行截取并复制给该对象，在历程图操作过程中，当预期观察区域为源图像的一部分时，需要通过该函数实现内容的截取。该函数源代码实现主要分为四个步骤：1) 边界检查；2) 创建复制图像并为其分配内存；3) 内存对齐检测与优化；4) 复制颜色表等元数据。

分析上述实现过程可知，当原始图像数据量很大时，为新对象分配内存与复制数据过程将成为主要耗时点，且因为数据量过大，可能会在过程中产生系统内存碎片；若源图像与新对象内存地址未能按缓存行进行对齐，执行过程则可能触发多次非对齐内存访问，使得整体运行效率进一步下降。另外，大数据图像在复制过程中，若高速缓存无法一次性容纳所有数据，则需要进行多次缓存的换入换出，在从主存读取数据过程浪费了较多时间，缓存命中率下降。

2.2. 缩放过程耗时分析

QImage::scaled()的作用为通过将源图像投射入指定大小区域进行显示，在历程图展示过程中，由于 UI 界面尺寸一般远小于源图像数据规模，所以当频繁进行全图操作时需要在短时间内将源图像进行多次缩放，从而在指定界面实现精细化数据展示。该函数源代码存在多种模式，每种模式对应不同算法，考虑到对于显示控制终端历程图，一般需要画面显示尽量精细化，因此本文只对显示质量更高的平滑模式进行分析，该模式使用的是双线性插值算法。

双线性插值算法需要对每个像素进行处理，不仅需要将每个输出的像素通过仿射变换与源图像坐标进行映射，还要获取当前像素周围 4 个像素的加权平均值，算法本身的复杂度较高、耗时需求较高[6]。当周围 4 个像素本身已经非连续存储，且源图像数据量过大时，则不仅算法本身耗时增长，而且由于数据无法全部存储在缓存中，缓存预取失败次数增加，缓存命中率降低，最终影响缩放过程的实时性。

3. 分块处理方法

通过上述分析可知，降低分配内存时间、减小数据规模，并且尽量使全部数据存储于高速缓存从而提高缓存命中率可以作为有效降低复制和缩放过程耗时的有效手段。针对“历程图在实际绘制能量谱时，往往在接收数据后需要绘制一整行或多个整行的数据”的情况，本文提出一种基于分块处理实现历程图快速绘制的方法：保持源图像长度不变，从高度入手进行拆分，核心理念是将大图像拆分成多个单独可完全存入缓存的小图像，实现减少每个过程的数据处理量，且使每个小图像的缓存可完全容纳于高速缓存；从而减少了内存分配时间，降低了产生的内存碎片数量，提高了缓存命中率。

以 1500 × 30000 的源图像需要映射到 1000 × 1000 的 UI 界面为例，RGB32 下的 QImage 源图像大小为 171.7 MB。保持行长度 1500 不变，对 30000 的高度进行拆分。拆分需要遵守两个原则：1) 对源图像进行整数分割，保证源图像数据全部映射；2) 每个分块大小合理，尽量保持大小可完全容纳于 CPU 内存层次中的高速缓存区。第一点容易理解，即拆分数量需要为 30000 的因数，着重对第二点原则进行分析。

现代 CPU 典型的内存层次结构分为如下层级：1) CPU 寄存器；2) 高速缓存区(按顺序分为 L1、L2、L3 三个层次)；3) 主存储器；4) 辅助存储器[7]。从访问速度角度，越靠近 CPU 的层级存储速度越快，即由 CPU 寄存器至辅助寄存器速度变慢；从缓存容量角度分析，越靠近 CPU 的层次缓存容量越小，即由 CPU 寄存器至辅助寄存器容量增大。其中高速缓存区由于其访问速度与容量均较适中，常常用于缓存 CPU 与主存之间的速度容量差异，也是图像处理最常使用的缓存区，其中 L1 层访问速度大约 1 至 4 个时钟周期，容量每个核心大约 32 至 64 KB；L2 层访问速度 10~12 个时钟周期，容量每个核心 256 KB 至 1 MB；L3 层访问速度较慢，一般 30 至 100 个时钟周期不等，容量 8 MB 至 64 MB。L1 层因为其规模极小，一般适用于存储最核心、最频繁使用的指令和数据片段，而在面对大数据量时，其访问速度优势无法填补容量小带来的弊端，造成访问次数巨额增加；L2 层能够处理较大的代码块与数据结构，因为其 MB 级别的数据存储与较快的访问速度，常常用于图像与图片的部分数据处理；L3 层拥有非常大的存储规模，但是访问速度较慢，该层的核心价值在于存储的数据可被多个核心快速共享，所以一般适用于存储规模较大且需要多核协同的工作集。因此，本文选择的合理分块大小应尽量保持全部处于 L2 层，即单块大小不超越 1 MB，以此保证在分配内存耗时合适的情况下，数据能够全部处于缓存中，大大提高缓存命中率，最终实现总耗时最小。

4. 实际应用与效果分析

为了验证本文分块处理方法对大数据历程图的绘制效率提升情况，本文以 1500 × 30000 的源图像映射到 1000 × 1000 的 UI 界面，以及 1500 × 50000 的源图像映射到 1000 × 1000 的 UI 界面这两个例子进行验证。为了方便实际绘制，保持行长度 1500 不变，同时为了实际绘制时像素精细度不降低，最大分块个数不能超过 UI 界面的像素行数 1000，宽度分块个数在整分的前提下由 1 至 1000 进行大致划分，并用 QTime 记录整个过程的开始与结束时间，整个实验过程进行多次，时间结果取平均值，测试环境见表 1。

Table 1. Test environment
表 1. 测试环境

测试环境	类型	参数
硬件环境	CPU	Intel (R) Core (TM) i7-12700
	核心数	12
	L1 缓存区大小	768 KB
	L2 缓存区大小	12 MB
	L3 缓存区大小	25 MB
软件环境	操作系统	Windows 10
	Qt 版本	5.12.8
	开发语言	C++

为保证源图像分块后能够将数据缓存于 L2 层级(大小不超过 1 MB)，且依据绘图方式保持行长度不变(长度 1500)，图像尺寸乘单个数据字节数可得其全部图像占用字节大小，由于数据源类型为浮点型占用 4 字节，所以经过简单计算可得结果：对于 1500 × 30000 的源图像来说，分块个数不得少于 173，对于 1500 × 50000 的源图像来说，分块个数不得少于 288。从图 2 和图 3 的耗时结果与分块大小曲线图可以观察到，分块个数从 1 开始时，因为分块的数量较少，每个分块数据量依旧庞大，增加步骤的同时不仅没有有效减少耗时，而且还略微增加了绘制时间，当分块个数逐渐增大后，数据量开始足够完全容纳

于高速缓存区 L3 层，但是内存分配、边界检查等操作的耗时依旧不少，相比于未分块时，效率有所提升。最终当图 2 代表的源图像分块个数从 100 跨越至 200 时，耗时处于最短水平，当图 3 代表的源图像分块个数从 250 跨越至 500 后，绘制耗时也来到最低水平。同时，经过对图像仔细观察，发现当分块大小达到最优分块大小后，若继续提升分块数量，效率不仅不会增加反而会降低。这是因为达到最优分块大小后，图像已能够完全存储于 L2 层，尺寸对于单个图像的处理效率影响并不会产生太多差异，若继续减小尺寸、增加分块数量，则会导致缓存换入换出更加频繁，分配次数显著增多，综合效率反而降低。尤其是分块数目达到一定数量时，图像尺寸甚至已经可以进入 L1 层，其内存访问次数相较于最优分块进行了巨额增加，虽然单个图像处理效率极高，但是综合效率相较于最优时显著降低。综上所述，实际操作时，需基于图像实际尺寸大小与绘制规则先计算出最优分块尺寸大小，再根据后台缓存数据大小与实际绘制 UI 大小进行整数分解，取满足 L2 缓存区容量尺寸且最接近最优分块大小的策略进行分块绘制。

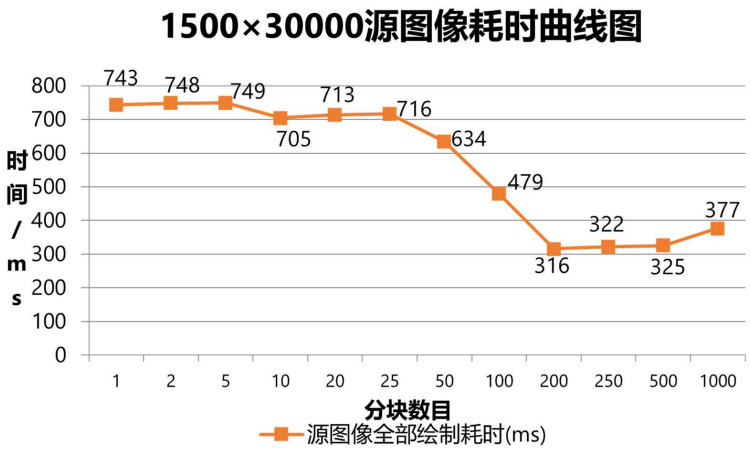

Figure 2. Time consumption waveform of 1500 × 30000 source image segmentation
图 2. 1500 × 30000 源图像分块耗时波形图

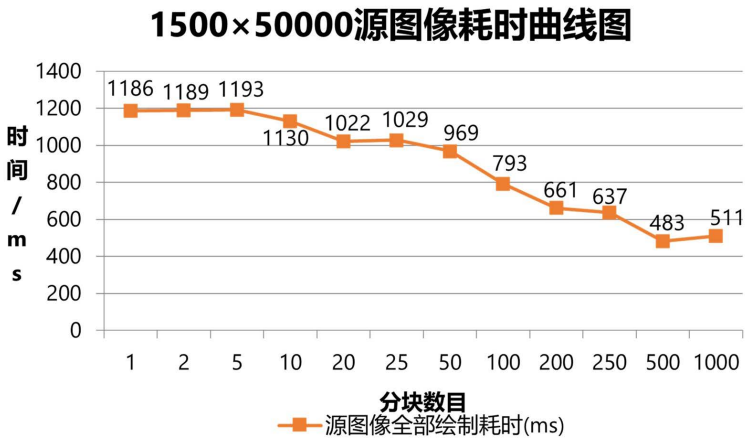

Figure 3. Time consumption waveform of 1500 × 50000 source image segmentation
图 3. 1500 × 50000 源图像分块耗时波形图

通过上述计算结果与测试结果的对比可知，达到最佳分块大小后可实现分块数据处于 L2 层级，并且耗时时间分别减少了 57%与 59%，这对于高频次的历程图全图绘制来说，是极大的性能提升。

5. 结语

本文依据历程图实际绘制需求与 CPU 内存访问特点，在对大数据源图像的复制过程与缩放过程进行详细分析后，针对内存分配时间、缓存命中率等主要耗时点提出了一种分块处理的历程图快速绘制方法。通过对当前 CPU 内存访问层级进行剖析，分析出最合理分块大小应容纳于高速缓存区的 L2 层。基于上述分析结果，以两种不同规模的源图像为例进行实际验证，分别计算出对应规模图像的合理分块数量，并在整分条件下依次分块、多次验证。验证结果表明，当分块个数增大至每个分块均满足上述分析结果时，耗时时间相较于未分块时降低 50%以上，使得大数据量历程图的全图重绘流畅度实现极大提升。

参考文献

[1] 李华庆. 基于 Qt 的声纳模拟显控软件设计与实现[J]. 电子技术与软件工程, 2019(7): 45.

[2] 王潇, 卢如意, 李博. 一种使用 Qt 的姿态传感器显控设计与实现[J]. 声学与电子工程, 2019(2): 21-24.

[3] 陆文周. Qt5 开发及实例[M]. 北京: 电子工业出版社, 2014.

[4] 邓子强, 王玉玫, 邓红艳, 等. 基于 Qt 绘图系统的图形应用优化研究与实现[J]. 软件, 2016, 37(12): 59-62.

[5] 王潇, 陈建冬. 一种可全图幅度缩放的历程图高速绘制方法[J]. 机电工程技术, 2025, 54(16): 91-95.

[6] 吕艳辉, 方亮. 基于双线性插值的单目标检测算法[J]. 火力与指挥控制, 2024, 49(1): 73-79+86.

[7] 王靖, 张路, 王鹏宇, 等. 面向图计算的内存系统优化技术综述[J]. 中国科学: 信息科学, 2019, 49(3): 295-313.