Modeling and Simulation 建模与仿真, 2024, 13(2), 1039-1049
Published Online March 2024 in Hans. https://www.hanspub.org/journal/mos
https://doi.org/10.12677/mos.2024.132099

Hans汉斯

基于OpenCL的车道线检测算法加速

葛品仕，刘甜甜

上海理工大学健康科学与工程学院，上海

收稿日期：2024年1月3日；录用日期：2024年3月1日；发布日期：2024年3月8日

摘　要

随着图像处理和算法复杂度的逐步提高，传统基于CPU的车道线检测算法在运行时会消耗大量时间，这极大地影响了算法的实时性。为了提升车道线检测算法的运行速度，本文采用了基于OpenCL的并行加速技术对传统的道路车道线检测算法进行优化。在处理过程中，首先对图像进行灰度化处理，然后利用OpenCL实现了高斯滤波、边缘检测和霍夫变换过程的内核调用，并在GPU上并行执行，这大大提升了这三个过程的运行速度，尤其是霍夫变换过程，其加速比达到了32.771。最后，通过筛选和聚类检测到的直线参数，实现了准确的车道线检测。在对1280 × 720大小的图片进行检测时，该算法相较于使用CPU处理的方法，其加速比能达到5.497，平均检测一张图片只需7.581 ms。这表明，本文提出的算法可以满足实时检测车道线的需求。

关键词

OpenCL，车道线检测，霍夫变换，异构并行

Acceleration of Lane Detection Algorithm Based on OpenCL

Pinshi Ge, Tiantian Liu

School of Health Science and Engineering, University of Shanghai for Science and Technology, Shanghai

Received: Jan. 3rd, 2024; accepted: Mar. 1st, 2024; published: Mar. 8th, 2024

Abstract

As the sophistication of image processing techniques and the complexity of algorithms continue to advance, traditional lane detection methodologies executed on Central Processing Units (CPUs) have been increasingly challenged by substantial computational demands, which greatly affects the real-time performance of the algorithm. In order to enhance the running speed of the lane detection algorithm, this paper adopts parallel acceleration technology based on OpenCL to optimize

文章引用: 葛品仕, 刘甜甜. 基于 OpenCL 的车道线检测算法加速[J]. 建模与仿真, 2024, 13(2): 1039-1049.
DOI: 10.12677/mos.2024.132099

the traditional road lane detection algorithm. In the processing phase, the image is first converted to greyscale, then Gaussian filtering, edge detection, and Hough transform processes are implemented using OpenCL kernel calls, and executed in parallel on the GPU. This greatly increases the running speed of these three processes, especially the Hough transform process, where the acceleration ratio reaches 32.771. Finally, by filtering and clustering the detected line parameters, accurate lane detection is achieved. When detecting images of size 1280 × 720, this algorithm, compared to the CPU processing method, has an acceleration ratio of 5.497, and it takes an average of only 7.581 milliseconds to detect an image. This demonstrates that the algorithm proposed in this paper can meet the demand for real-time lane detection.

Keywords

OpenCL, Lane Line Detection, Hough Transform, Heterogeneous Parallel

Copyright © 2024 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

随着中国汽车技术的迅猛进步以及汽车智能化程度的不断提高，国内的汽车保有量正呈现逐年增长的趋势。然而，由此引起的道路安全问题也日益严峻。为了确保车辆运行的安全性，基于计算机视觉的车辆辅助驾驶和智能车辆自动驾驶得到了广泛应用，而车道线的准确实时检测是其中一个重要的组成部分[1] [2]。车道是交通系统中用来划分道路、保证汽车安全有效行驶的交通标志，车道检测是一种自动检测道路标记的技术，其在自动驾驶方面发挥着重要作用[3]。实时准确的车道检测可以让自动驾驶车辆对其位置和状态做出正确的判断，从而确保安全驾驶。

在车道线检测方面，直线车道线的检测是研究最多的。传统车道线检测在经过图片预处理后会采用霍夫变换算法来检测直线，其在直线车道线检测领域表现出了高有效性和高精度。然而，霍夫变换算法的实施过程中存在计算量大和内存占用的问题，这在一定程度上限制了其在实时系统中的应用[4]。为了减少图像处理和直线检测的时间，提高检测的实时性，文献[5]通过限定斜率范围、设置平均斜率偏离阈值来快速除去噪声，将检测对象锁定在目标车道线上；文献[6]通过缩小感兴趣区域并动态跟踪感兴趣区域来减小霍夫变换的计算量；文献[7]则利用 OpenCV 库函数高效地实现了霍夫变换直线检测。

随着道路交通场景的不断复杂化，对应算法的复杂度也显著提升。由于车道线检测在实时性和准确性方面要求很高，以 CPU 为基础的传统算法已有其局限性。同时，伴随着并行计算的兴起，以 GPU 为核心的并行计算技术已经被越来越多的应用到各个领域[8]。基于 CPU + GPU 的异构计算架构正在逐渐成为一种高效、低能耗的主流计算方式，为多个应用场景提供了良好的计算平台[9]。OpenCL 通过统一的编程框架和接口能在 CPU + GPU 的异构计算环境中，构筑并行计算架构，允许应用程序利用多种硬件资源进行加速，这种并行计算可以同时处理多个任务或数据，显著提高了计算效率和程序性能[10]。因此，本文基于 OpenCL 异构计算架构，旨在通过并行化加速车道线检测算法，着重对车道线检测算法中常用的高斯滤波、边缘检测和霍夫变换等流程进行了 GPU 并行计算的实现和优化。

2. OpenCL 异构计算架构简介

OpenCL 包含一系列基础 API 和高效、快速、可移植的抽象层，使其能够在不同的硬件体系结构上进行异构计算程序开发，这能够更有效地利用现有的计算资源提高程序的性能和效率[11]。OpenCL 程序

包含主机端程序和内核程序。OpenCL 程序的运行流程如图 1 所示。计算任务由主机端开始并在主机端进行数据解包和预处理，在主机端选取设备端建立联系后，主机将数据发送到设备端并调用内核程序进行并行计算，计算完成后，主机端会读取计算结果进行后续处理。

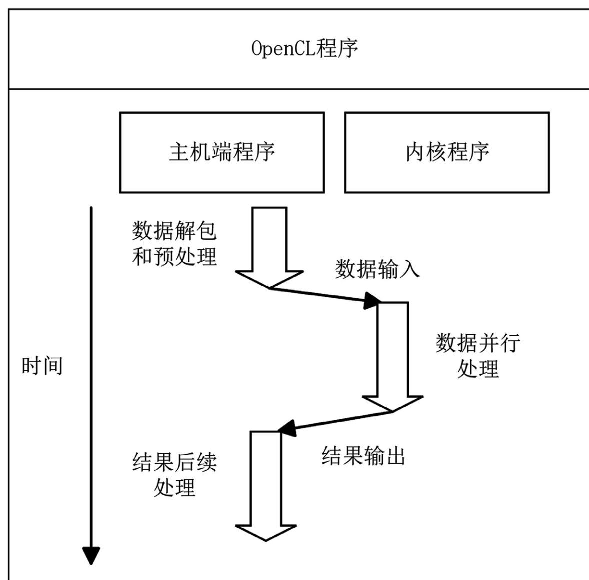

Figure 1. OpenCL program flow
图 1. OpenCL 程序运行流程

3. 基于 OpenCL 异构计算架构的车道线检测算法设计

本文研究的车道线检测任务主要分为图像预处理和车道线检测两个部分。为了减少噪声并突出车道线特征，将对含有车道线的图像进行预处理再进行后续的检测流程。这些预处理操作包括将图像转换为灰度图以降低数据的复杂性；应用高斯滤波器以平滑图像并抑制噪声；进行边缘检测以突出车道线的结构特征；以及提取图像中的感兴趣区域，即可能包含车道线的区域。随后进行车道线检测过程，该过程涉及霍夫变换来识别图像中的直线，并使用直线参数聚类技术以区分并选择出最合适的车道线。最终，挑选出的车道线将被准确地标绘在原始图像上，实现车道线的有效检测与视觉呈现。

在这个过程中将使用 OpenCL 异构计算架构来设计高斯滤波、边缘检测、提取感兴趣区域和霍夫变换等步骤的内核程序，并在 GPU 上进行实现，以加速整个车道线检测算法。整个算法的结构如图 2 所示。

3.1. 图像灰度化

彩色图像中的每个像素点包含红绿蓝(RGB)三个颜色通道，将彩色图像转化为灰度图像的步骤涉及对彩色图像中每个像素进行特定计算。可以通过给彩色图像的 RGB 三个分量施加不同的加权系数，从而进行加权处理将图像灰度化，其加权公式如下：

$$\mathrm{H}(i,j)=0.298\mathrm{R}(i,j)+0.587\mathrm{G}(i,j)+0.114\mathrm{B}(i,j) \tag{1}$$

彩色图像转换为灰度图像的过程是使用不同的权重提取亮度信息，从而得到灰度图像。灰度图像的每个像素的颜色值表示灰度，灰度的范围一般为 0 到 255，用于表示色彩的浓淡程度和图像的特征信息。图 3 为原图，图 4 为灰度化后的图像，与彩色图像相比，灰度图像的数据量更小，内存占用更低，从而使得处理速度加快，这对于需要实时反馈的车道线检测尤其关键。此外，灰度化处理能够在视觉上提高图像对比度，更加突出车道线。

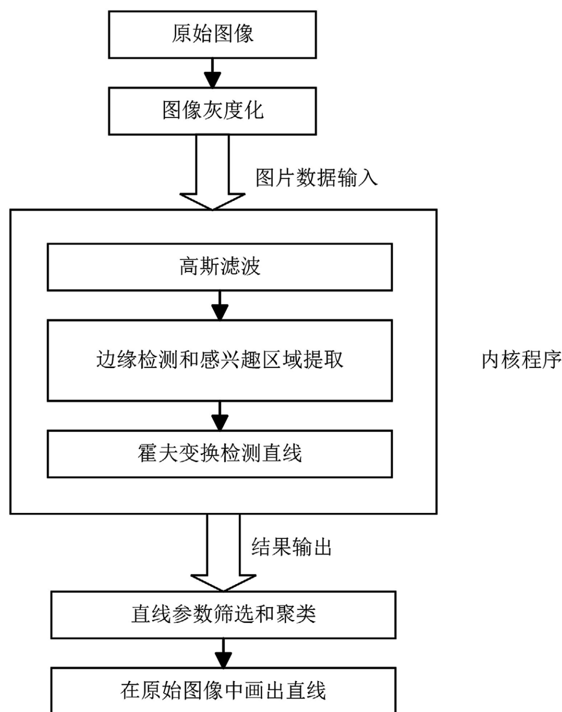

Figure 2. Structure of lane detection algorithm
图 2. 车道线检测算法结构

Figure 3. Original image
图 3. 原图

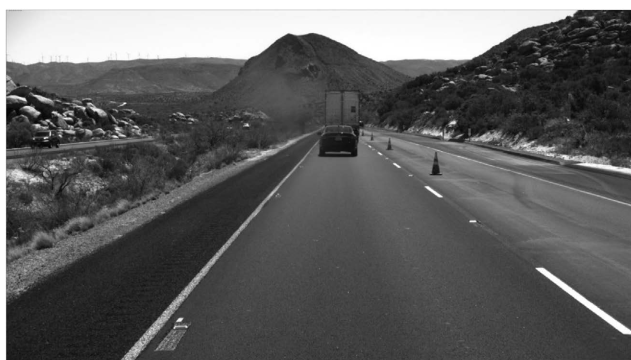

Figure 4. The gray-scaled image
图 4. 灰度化后的图像

3.2. 内核程序

在 OpenCL 框架中，内核程序被用来在 OpenCL 设备上执行并行计算任务。这类程序通常使用 OpenCL C 语言编写，该语言借鉴了 C99 语言的语法，并额外加入了并行编程所需的语法扩展。本研究中所构造的内核程序包括高斯滤波内核、边缘检测内核以及霍夫变换内核，其创建和调用过程均通过 OpenCL 提供的 API 实现，具体流程如图 5 所示。该过程主要包括以下步骤：

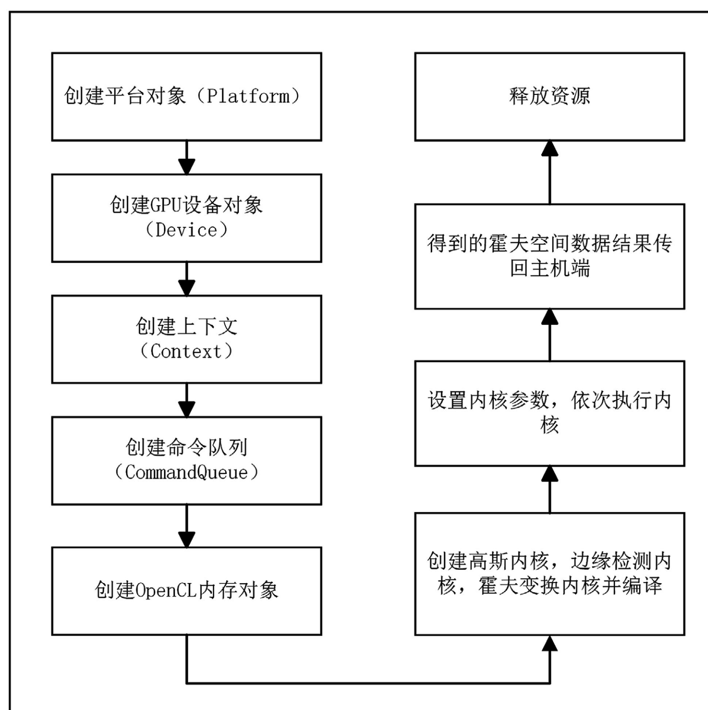

Figure 5. Creation and invocation of kernel programs
图 5. 内核程序的创建和调用

步骤一：在当前系统上利用 clGetPlatformInfo 函数来查找所有可用的 OpenCL 平台，并基于此选择最适合运行 OpenCL 程序的平台。在选定的平台上，通过 clGetDeviceIDs 函数来查询和选定可用的 OpenCL 设备。

步骤二：利用 clCreateContext 函数创建上下文环境，该环境主要管理 OpenCL 执行环境中的程序、内存、命令队列和设备等。

步骤三：利用 clCreateCommandQueue 函数在特定上下文中为特定设备创建命令队列，该队列主要用于控制内核的执行和内存操作。

步骤四：利用 clCreateBuffer 函数在设备上创建内存对象，用于存储数据。

步骤五：利用 clCreateProgramWithSource 函数创建一个程序对象，并加载 OpenCL C 代码。然后，通过 clBuildProgram 和 clCreateKernel 函数，可以进行内核的编译和创建，将 OpenCL C 代码转换为可执行代码，同时内核函数与主机程序关联，以便后续主机程序执行内核。

步骤六：利用 clSetKernelArg 函数设置内核参数，以便在主机程序与内核之间进行数据交换。通过设定内核的相关参数，可以保证数据在主机与设备间的正确传输，以确保算法的准确性与正确性。

步骤七：利用 clEnqueueNDRangeKernel 函数，根据设定的工作组和工作项并行执行内核，使内核在 GPU 上运行。

步骤八：利用 clEnqueueReadBuffer 函数将内核执行完毕后的结果传输至主机。

步骤九：在内核执行完毕后，释放所有已分配的资源和对象。

在 OpenCL 异构编程模型中，这些函数协同工作以准备和管理在异构系统上运行的程序。它提供一个统一的编程环境使开发者能够充分利用异构平台上的计算资源，降低了编写并行和异构代码的难度，减少了开发周期。

3.2.1. 高斯变换内核

对经过灰度化的图像进行滤波处理是车道线检测算法中的重要步骤。滤波的目的是为了减少图像中由于各种外部条件(如树木阴影、光线变化、路面标记等)引起的噪声，而噪声可能会对车道线特征的提取产生干扰。有效的滤波可以提高图像质量，使得车道线特征更加突出，便于后续的边缘检测和车道线识别步骤。在车道线检测中，高斯滤波是一种较为常用的选择，因为它在去除噪声的同时，能较好地保留车道线的边缘特征，有利于后续的边缘检测算法的准确执行，因此本文选择使用 3×3 的高斯核对车道线图像进行滤波处理。

高斯滤波过程如图 6 所示，经过灰度化处理后的图像可以通过并行处理使用高斯滤波核进行处理，通过将图像数据划分为多个工作项，每个工作项负责处理图像的一部分，并利用并行计算的能力，可以同时对多个像素进行高斯滤波处理。这样的并行处理方式可以显著提高图像处理的速度和效率。每个像素点 $H(i, j)$与 3×3 的高斯核进行卷积，得到新的像素值 $H^*(i, j)$。最后得到的高斯滤波后的图像如图 7 所示。

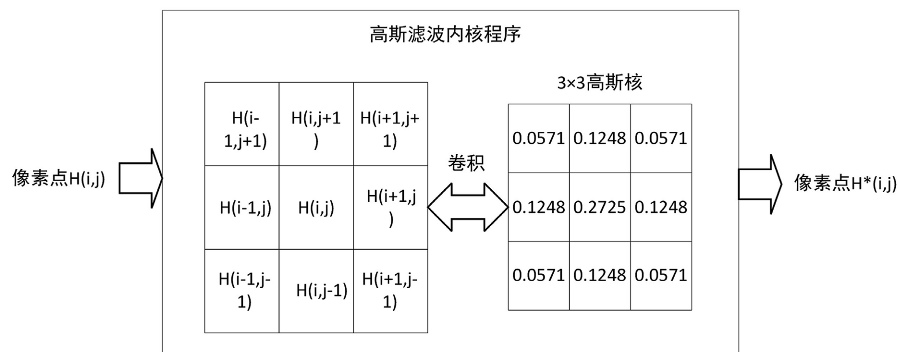

Figure 6. Gaussian filtering process
图 6. 高斯滤波过程

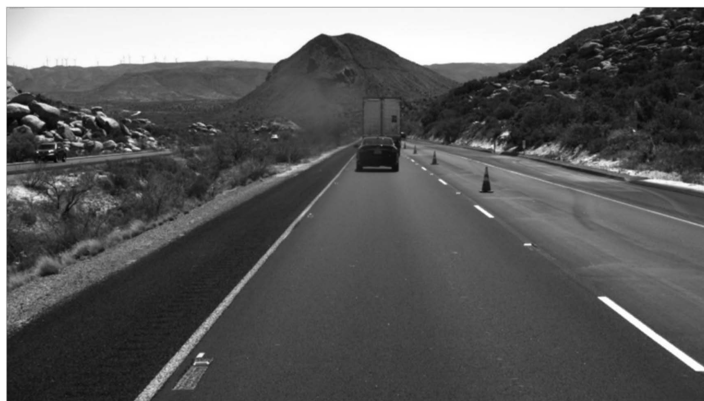

Figure 7. Gaussian filtered image
图 7. 高斯滤波后的图像

3.2.2. 边缘检测内核和感兴趣区域划分

边缘检测在图像处理中的作用是寻找灰度值波动较大的位置以生成二值化图像，如车道线与道路交界处的明显变化。常用的边缘检测算子包括 Prewitt、Sobel、Roberts 和 Canny 算子，其中 Canny 边缘检测算法是由 John F. Canny 在 1986 年开发的一个多级边缘检测工具，它旨在提供一个可靠的边缘检测算法，能够尽可能准确地检测图像中的边缘。在车道线检测的应用中，Canny 算子因其优秀的检测性能和准确性而被广泛使用。

本文中的 Canny 边缘检测内核的流程主要分为三步。首先，使用 Sobel 算子对图像进行梯度计算，以获得图像中每个像素点的梯度强度和方向信息。其次，通过非极大值抑制的方法，对梯度图像进行处理，去除非边缘的噪点，使得边缘更加细腻。在这一步中，只保留梯度图像上局部极大值点，即将具有最大梯度值的像素点保留下来，其他非极大值点则被抑制掉。最后，将采用两个预设的阈值：一个高阈值和一个低阈值，来辨识和连接图像中的边缘。像素强度超过高阈值的将被标记为明确的强边缘，而那些低于低阈值的像素将被认定为非边缘区域。至于那些介于两个阈值之间的像素，它们会根据是否紧邻已确认的边缘像素来决定身份，与强边缘相连的将被视作边缘，反之则视为非边缘。

为减少后续霍夫变换步骤的计算量，在完成边缘检测之后，对图像执行了感兴趣区域选取和区域分割。由于车道线通常集中在图像的某些特定部分，将图像切割以仅留下可能包含车道线的片段，并专注于这些片段进行霍夫变换可以更精确地提取车道线数据。这一策略的实施有效减少了处理所需的计算资源并提高了车道线检测的效率。边缘检测及感兴趣区域提取的结果如图 8 所示。

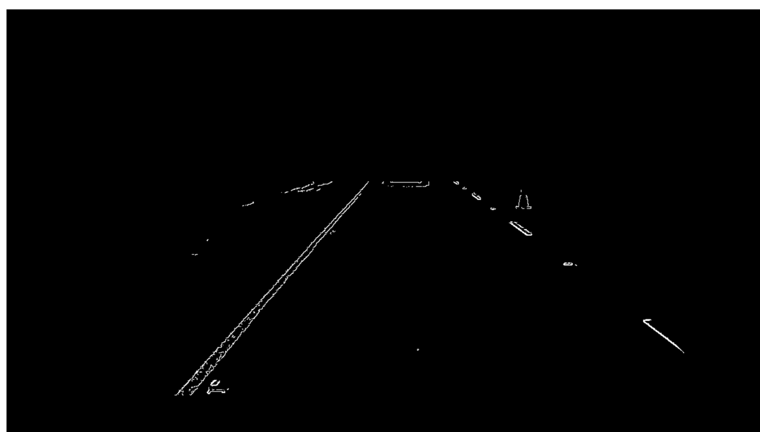

Figure 8. Results of edge detection and ROI extraction
图 8. 边缘检测和感兴趣区域提取的结果

3.2.3. 霍夫变换内核

边缘检测经过之前的图像预处理后，下一步是对图像中的直线信息进行检测，需要采用霍夫变换将图像中被检测为边缘的像素点从几何空间转换到霍夫空间。在霍夫变换中，几何空间中的直线将被映射到参数空间的一个点。具体来说，对于图像中的每个边缘点存在通过该点的无数条直线，这些直线映射到在霍夫空间中是许多正弦曲线。然后，在霍夫空间中对这些正弦曲线通过的点进行累积投票，以确定霍夫空间中的峰值点。这些峰值点代表了在几何空间中可能存在的直线的参数。通过在参数空间中寻找峰值点并根据峰值点的大小可以判断在几何空间中是否存在直线。因此，霍夫变换将判断直线是否存在的问题转化为在霍夫空间中寻找峰值点的任务。通过找到霍夫空间中的峰值点，可以确定几何空间中的直线参数，从而实现直线的检测和提取。这种转换过程使得直线检测问题变得更加简单和直观，并且能够有效应对图像中的噪声和变形。它的原理如图 9 所示：

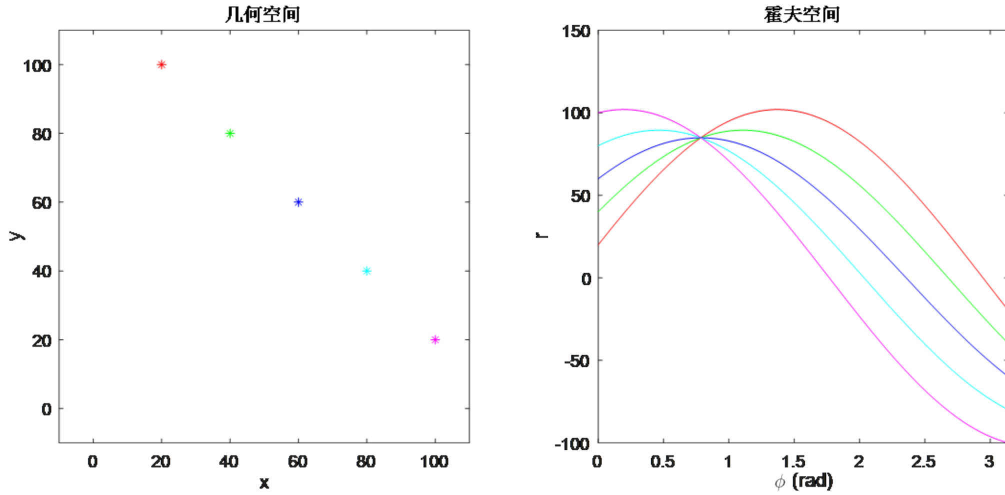

Figure 9. Principle of Hough transform
图 9. 霍夫变换的原理

假设在图中的几何空间中的点(x_i, y_i)位于同一直线上，在 x-y 坐标系下的直线方程可以通过式(2)表达，其中直线的斜率用 k 表示，截距用 b 表示。那么通过点(x_i, y_i)的所有直线可以由式(3)表示，直线到原点的距离用 r 表示，x 轴与直线的法线的夹角用 θ 表示。因此通过点(x_i, y_i)的所有直线经过式(3)的变换后都变成霍夫空间的正弦曲线，并且由于点(x_i, y_i)位于同一直线上，因此这些位于同一直线上的点(x_i, y_i)所对应的正弦曲线会相交于一点(r_0, θ_0)。通过寻找霍夫空间的峰值点就能得到点(r_0, θ_0)，从而计算出点(x_i, y_i)所在的直线的参数，r_0 表示原点到直线的距离，θ_0 表示直线的法线与 x 轴的夹角。

$$y_i = kx_i + b \tag{2}$$

$$r = x_i \cos\theta + y_i \sin\theta \tag{3}$$

因此在对边缘检测后得到的图像进行霍夫变换时，会对每个像素点进行判断和处理。对是边缘的像素点会根据其坐标信息进行霍夫变换，并在霍夫空间中进行投票，在所有边缘像素点完成投票后得到霍夫空间的数据。通过在霍夫空间中累积投票并设置大小合适的阈值，可以找到参数空间中的峰值点，这些峰值点对应着在几何空间中的直线。对于每个峰值点，可以通过反变换将其转换回几何空间，从而得到所需的直线的位置和方向。

3.3. 直线筛选聚类

通过 Canny 边缘检测与霍夫变换直线检测的配合使用，能够获得霍夫空间的投票数据。根据霍夫变换的结果进一步进行筛选和聚类，以获取更为精确的车道线参数。整个流程如下图 10 所示：

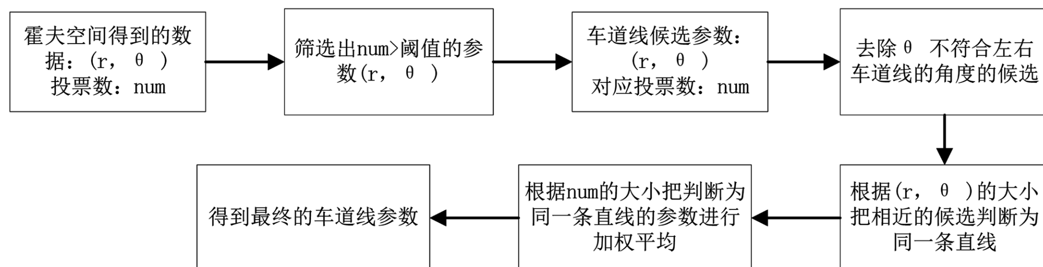

Figure 10. Linear screening clustering process
图 10. 直线筛选聚类流程

在得到霍夫变换后的数据后设置相应的阈值，将大于阈值的投票数对应的(r, θ)作为初步的车道线候选线的参数，由于候选线中包含大量的干扰线，为了得到准确的左右两条车道线需要进一步进行筛选。通过车道线候选线的 θ 参数，从而可以求出每条直线与水平方向的夹角。鉴于左右车道线与水平线的夹角通常位于特定区间，通过评估 θ 值的大小，可以有效地筛除一些噪声线条。之后会将具有相近(r, θ)参数的候选线判断为是同一条直线，并根据投票数 num 的大小对被判断为同一直线的(r, θ)参数做加权平均得到最终得车道线参数。得到最终得车道线参数后在原图中画出检测到得车道线如图 11 所示，可以看到该算法能很好的检测到左右两侧的车道线并在图中标注出来。

Figure 11. Detected lane lines
图 11. 检测到的车道线

4. 车道线检测结果与分析

本文用以下实验装置验证基于 OpenCL 的车道线检测方法：PC 机的 CPU 配置为 12th Gen Intel(R) Core(TM) i3-12100 3.30 GHz，GPU 配置为 Intel(R) UHD Graphics 730，运行内存为 16 GB，配置的 OpenCL 版本为 OpenCL3.0，以 C++作为编程语言。

为了对比 CPU 的处理时间，还配置了 OpenCV3.2.0 计算机视觉库，OpenCV 是一个优秀的计算机视觉库，其中包含了许多针对图像处理和计算机视觉任务的优化算法和实现，这些算法通常经过了专门的性能优化和高效实现，它们的实现进行了相当彻底的优化，包括但不限于底层并行化、SIMD 指令集优化、缓存友好的算法设计等。为了更多样的对比，在对图像灰度化，高斯滤波，边缘检测时决定采用 OpenCV 自带的函数，另外在对图像进行霍夫变换时采用 C++编写的霍夫变换函数。车道线检测的结果对比如图 12 所示，可以看出两种方式检测的结果相差不大，都可以很准确的识别到车道线的位置，表明基于 OpenCL 的算法能达到 CPU 执行的算法的效果。

分别选用不同大小的图片进行处理时间的对比。平均每张图片基于 OpenCL 异构并行处理与 CPU 处理的车道线检测时间对比如表 1 所示。其中显示了使用不同方式检测车道线所花的时间，使用不同大小的图片进行了测试对测试时间取平均得到每张图片的处理时间，从结果可以看出基于 OpenCL 的检测方法对比单独使用 CPU 进行检测有着明显的加速作用，在对 1280×720 大小的图片进行检测时能达到 5.497 的加速比，加速比为使用 CPU 下算法运行时间与使用 OpenCL 下算法运行时间之比。

表 2 表示使用 OpenCL 与使用 CPU 处理过程各个步骤所消耗的时间，包括高斯滤波、边缘检测和霍夫变换 3 个过程使用的时间对比，OpenCL 处理时只计算了内核所运行的时间，不包括 CPU 与 GPU 进行数据传输的时间。其中使用 CPU 进行高斯滤波和边缘检测时是直接采用 OpenCV 中的性能优化和高效实

现对应函数进行处理，霍夫变换过程没有采用 OpenCV 中的函数，而是另外编写的 C++函数，以此形成对比。使用 OpenCL 进行加速时采用 C++编写的内核程序，从各个内核的处理时间来看，其中霍夫变换内核有着高达 32.771 的加速比，其它内核的加速比在 2~4 之间。这表明基于 OpenCL 的算法实现对比单纯使用 CPU 有明显的加速效果。

Figure 12. OpenCL processing result (left), CPU processing result (right)
图 12. OpenCL 处理结果(左侧)，CPU 处理结果(右侧)

Table 1. Comparison of lane detection time
表 1. 车道线检测时间对比

图片大小	OpenCL 处理时间/ms	CPU 处理时间/ms	加速比
320 × 240	1.712	6.498	3.796
640 × 480	3.746	17.484	4.667
1280 × 720	7.581	41.674	5.497

Table 2. Comparison of elapsed time of each kernel
表 2. 各内核消耗时间对比

内核	OpenCL 处理时间/ms	CPU 处理时间/ms	加速比
高斯滤波	0.931	3.224	3.463
边缘检测	2.467	7.133	2.891
霍夫变换	0.863	28.281	32.771

5. 结论

本文采用 OpenCL 异构计算编程模型实现了对传统车道线检测算法的并行加速，整个算法包括图像预处理、直线检测和直线参数筛选和聚类，并且能准确检测到图像中的车道线并进行标注。为了验证算法的加速效果，采用大小相同的图片分别使用 OpenCL 编程模型和单纯使用 CPU 进行处理，并对比了两种方法的处理时间。在对 1280 × 720 大小的图片进行车道线检测时整个算法相比与单纯使用 CPU 进行检测能达到 5.497 的加速比，平均检测一张图片只需要 7.581 ms。算法中各内核函数的性能相比于使用 CPU 处理也有明显提升，其中霍夫变换内核的加速效果最好，达到了 32.771 的加速比，高斯滤波内核达到了 3.463 的加速比，边缘检测内核达到了 2.891 的加速比。经验证，算法大大缩短了检测时间，确保了车道

线检测的实时性。

参考文献

[1] 李亚娣, 黄海波, 李相鹏, 等. 基于 Canny 算子和 Hough 变换的夜间车道线检测[J]. 科学技术与工程, 2016, 16(31): 234-237+242.

[2] 杨喜宁, 段建民, 高德芝, 等. 基于改进 Hough 变换的车道线检测技术[J]. 计算机测量与控制, 2010, 18(2): 292-294+298.

[3] Zakaria, N.J., Shapiai, M.I., Ghani, R.A., Yassin, M.N.M., Ibrahim, M.Z. and Wahid, N. (2023) Lane Detection in Autonomous Vehicles: A Systematic Review. *IEEE Access*, **11**, 3729-3765. https://doi.org/10.1109/ACCESS.2023.3234442

[4] 陈政宏, 李爱娟, 王希波, 等. 基于改进 Hough 变换的结构化道路车道线识别[J]. 科学技术与工程, 2020, 20(26): 10829-10834.

[5] 周发华, 陈继清, 杨蓉. 基于改进 Hough 变换和斜率特征的车道线识别[J]. 现代电子技术, 2023, 46(7): 180-186.

[6] 顾友霖, 郑再象, 刘洋, 等. 基于霍夫变换的车道线检测跟踪及车道偏离预警[J]. 南方农机, 2023, 54(9): 129-132.

[7] 陈月正, 范文宇, 叶蕴森, 等. 基于 OpenCV 的车道线检测方法的研究[J]. 微纳电子与智能制造, 2022, 4(4): 82-87.

[8] 李旭. 基于 CUDA 加速的车道线检测算法研究[D]: [硕士学位论文]. 大连: 大连理工大学, 2020.

[9] Luo, G., Zhou, R.G., Liu, X., *et al.* (2018) Fuzzy Matching Based on Gray-scale Difference for Quantum Images. *International Journal of Theoretical Physics*, **57**, 2447-2460. https://doi.org/10.1007/s10773-018-3766-7

[10] 常新旭, 王珂, 张杨, 等. 异构并行编程框架综述[J]. 信息系统工程, 2022(8): 135-138.

[11] Munshi, A. (2009) The OpenCL Specification. 2009 *IEEE Hot Chips* 21 *Symposium* (*HCS*), Stanford, 23-25 August 2009, 1-314.https://doi.org/10.1109/HOTCHIPS.2009.7478342