Tabriz Journal of Electrical Engineering (TJEE), vol. XX, no. X, season X Serial no. XX
DOI: 10.22034/tjee.2025.66018.4970

Architectural Design and Hardware Implementation of Smoothing Filters in 3D Images

Jalal Babaie, Alireza Behrad*

Electrical Engineering Department, Faculty of Engineering, Shahed University, Tehran, Iran
E-mails: jalal_babaie@yahoo.com; behrad@shahed.ac.ir
* Corresponding author

Abstract
In recent years, 3D technologies have experienced significant growth and have impacted various fields such as robotics, entertainment, surveillance, and security. One of the critical processes in this domain is smoothing filters in 3D models. In this regard, this paper presents two fixed-point, parallel, and pipeline-based hardware architectures for two smoothing filters: the mean and Gaussian filters. In the proposed architecture, during the preprocessing stage, 3D image data is represented appropriately. Two different hardware architectures are proposed for the preprocessing stage. Additionally, two distinct hardware architectures are proposed for the mean and Gaussian filters, differing in the structure of 3D data representation during preprocessing. Furthermore, a lookup table method is used to compute the exponential function for the Gaussian filter. The proposed architectures are implemented using the Verilog hardware description language and simulated and synthesized using ISE and Vivado software. For a 3D image with 299 vertices and 562 faces, the operating frequency for the mean filter with coordinate and vector length structures is approximately 19 MHz, and the achieved processing rates are 37,367 and 28,081 images per second, respectively. For the Gaussian filter, the operating frequency is around 19 MHz, and the obtained processing rates are 37,299 and 28,336 images per second, respectively.

Keywords
Architectural design, hardware implementation, 3D image, smoothing filter.

Introduction

Image smoothing is widely used in numerous applications of image processing and machine vision, including object recognition and image matching. Although the hardware implementation of 2D image smoothing filters has been explored in various studies, its extension to 3D images has been less addressed. In 3D images, tools that generate 3D models may introduce noise, which can create challenges for measurements and processing. Therefore, there is a need for fast and efficient algorithms that preserve the geometric structure of the model [1, 2]. Consequently, this paper focuses on the hardware implementation of smoothing filters for 3D images and shapes using mesh representation and introduces novel architectures in this field.

Proposed Work and Methodology

This paper focuses on the hardware implementation of smoothing filters for images and 3D shapes using mesh representation and introduces new architectures in this field. In summary, the main innovations of this paper are as follows:

- Two new architectures for the mean filter with two different structures: 1) coordinate-based structure and 2) vector-length-based structure are introduced.
- Two new architectures for applying the filter based on two different structures, including coordinate-based and vector-length-based, are presented.
- The proposed architectures are implemented using the hardware description language Verilog, simulated and synthesized using ISE and Vivado software, and tested with numerous images.

The proposed architectures are implemented in Verilog and simulated/synthesized using ISE and Vivado software. For a 3D image with 299 vertices and 562 faces, the mean filter operates at approximately 19 MHz, achieving processing rates of 37,367 and 28,081 images per second for coordinate and vector-length structures, respectively. Similarly, the Gaussian filter operates at around 19 MHz, with processing rates of 37,299 and 28,336 images per second for the respective structures.

Conclusion

In comparing the proposed architectures for sparse representation of mesh data, the second proposed architecture consumes almost half the resources of the first architecture. The second architecture is about 38 times faster than the first. For the mean filter implementation, the vector-length-based architecture consumes about 26% fewer resources than the coordinate-based architecture. Inn terms of speed and frames processed per second, the coordinate-based architecture is 1.3 times faster than the vector-length-based architecture. For the Gaussian filter implementation, the vector-length-based architecture consumes about 33% fewer resources than the coordinate-based architecture. In terms of speed and frames processed per second, the coordinate-based architecture is 1.3 times faster than the vector-length-based architecture.

طراحی معماری و پیاده‌سازی سخت‌افزاری فیلترهای هموارساز در تصاویر سه‌بعدی

جلال بابایی
دانشجوی کارشناسی ارشد، دانشکده فنی مهندسی، دانشگاه شاهد، تهران، ایران

علیرضا بهراد
استاد، دانشکده فنی مهندسی، دانشگاه شاهد، تهران، ایران

چکیده
در سالهای اخیر فناوری‌های سه‌بعدی، رشد قابل ملاحظه‌ای داشته‌اند و در زمینه‌های مختلفی مانند رباتیک، سرگرمی، نظارت و امنیت تاثیر گذاشته است. یکی از پردازشهای مهم در این زمینه، فیلترهای هموارساز در مدل‌های سه‌بعدی است. در این راستا، در این مقاله دو معماری سخت‌افزاری به‌صورت ممیزثابت، موازی و مبتنی بر خط‌لوله برای دو فیلتر هموارساز میانگین و گاوسی ارائه شده است. در معماری پیشنهادی و در مرحله پیش‌پردازش، داده‌های تصویر سه‌بعدی به صورت مناسب بازنمایی می‌شوند. برای مرحله پیش پردازش دو معماری متفاوت سخت‌افزاری پیشنهاد شده است. برای فیلترهای میانگین‌گیری و گاوسی نیز دو معماری سخت‌افزاری متفاوتی پیشنهاد شده‌است که تفاوت آنها در ساختار بازنمایی داده‌های سه‌بعدی در مرحله پیش‌پردازش است. همچنین برای محاسبه تابع نمایی فیلتر گاوسی از روش جدول جستجو استفاده شده‌است. معماری‌های پیشنهاد شده با زبان سخت‌افزاری Verilog پیاده‌سازی و بوسیله نرم‌افزارهای ISE وVivado شبیه‌سازی و سنتز شده‌اند. برای یک تصویر سه‌بعدی با ۲۹۹رأس و ۵۶۲ وجه، فرکانس کاری برای فیلتر میانگین‌گیری با ساختارهای مختصات و طول بردار حدود ۱۹ مگاهرتز و نرخ پردازش حاصل شده به ترتیب برابر ۳۷۳۶۷ و ۲۸۰۸۱ تصویر بر ثانیه می‌باشد و برای فیلتر گاوسی نیز فرکانس کاری حدود ۱۹ مگاهرتز و نرخ پردازش بدست آمده به ترتیب برابر با ۳۷۲۹۹ و ۲۸۳۳۶ تصویر بر ثانیه است.

کلمات کلیدی
طراحی‌معماری، پیاده‌سازی سخت‌افزاری، تصویر سه‌بعدی، فیلتر هموارساز.

نام نویسنده مسئول: علیرضا بهراد
ایمیل نویسنده مسئول: behrad@shahed.ac.ir

تاریخ ارسال مقاله: ۱۴۰۳/۱۲/۰۳
تاریخ(های) اصلاح مقاله: ۱۴۰۴/۰۱/۲۹
تاریخ پذیرش مقاله: ۱۴۰۴/۰۲/۲۹

۱- مقدمه

در چند دهه اخیر فناوری‌های سه‌بعدی رشد قابل ملاحظه‌ای نمودند و بر زمینه‌های مختلفی مانند نظامی، رباتیک، سرگرمی، نظارت، امنیت، و پزشکی تاثیر گذاشته است. مدلسازی‌ها تا پیش از این به شکل تصاویر دوبعدی روی صفحه‌های نمایشگر یا روی کاغذ ارائه می‌شدند تا افراد با دیدن آنها درکی از آنچه طراحان در ذهنشان دارند بدست‌آورند. چاپگرهای سه‌بعدی توانایی تولید هر نوع قطعه‌ای با هر شکل و نما و طرحی را دارند. با استفاده از چاپگر سه‌بعدی می‌توان فرایند زمانبر شبیه‌سازی، ساخت ماکت و قطعات را تسریع بخشید و تنها با چاپ طرح سه‌بعدی در زمانی بسیار کم، به بررسی قطعه پرداخت. در حوزه پزشکی بخاطر اینکه اندام و اعضای بدن انسان از پیچیدگی و تنوع بالایی برخوردار است و همچنین محل‌های آسیب‌دیدگی غیرقابل پیش‌بینی هستند، دست یافتن به بخش‌بندی کامل و دقیق بافت از طریق تصاویر دوبعدی بسیار مشکل است. برای بافت‌هایی که توسط بافت‌های دیگر در برخی از زوایای پنهان محصور شده، تصاویر سه‌بعدی درک بالینی بهتری می‌دهند.

از نظر بازنمایی ونمایش داده‌ها، تصاویر سه‌بعدی به سه دسته ۱- روش حجم نگاری[1] ۲- روش تصویربرداری فاصله[2] و ۳- نمایش سطح سه‌بعدی[3] تقسیم‌بندی می‌شوند. در روش حجم‌نگاری ، فضای سه‌بعدی به مکعب‌های کوچکی به نام وکسل[3] تقسیم می‌شود. هر وکسل مانند یک پیکسل در تصاویر دوبعدی است، اما دارای بعد سوم (عمق) نیز است. در روش تصویربرداری فاصله، یک نمای دوبعدی از داده‌های سه‌بعدی ارائه می‌شود که در آن هر پیکسل فاصله از سنسور (مثل لیزر یا دوربین عمق) را نشان می‌دهد. این تصاویر معمولاً از اسکنرهای لیزری یا دوربین‌های عمق‌سنج تولید می‌شوند. این روش قادر به نمایش و بیان تصویر سه‌بعدی فقط از یک نما است. در نمایش سطح سه‌بعدی، سطح اشیای سه‌بعدی معمولا با استفاده از مش‌بندی[4] بیان می‌شود. مش‌بندی معمولاً از چندضلعی‌ها (مثل مثلث‌ها) تشکیل شده است. این روش بیان کاملی از نمایش سه‌بعدی است و در کاربردهای مختلفی از جمله چاپگر سه‌بعدی، کاربردهای صنعتی و گرافیگ مورد استفاده قرار می‌گیرد.

گرچه دستورات پردازش سیگنالی که به صورت درونی در پردازشگرها پیاده‌سازی شده‌اند، برای تصاویر دوبعدی و نیز تصاویر سه‌بعدی مبتنی بر

[1] Voxel based
[2] Range image
[3] 3D Surface

[3] Voxel
[4] Mesh

حجم‌نگاری و تصاویر سه‌بعدی فاصله قابل دسترس هستند. ولی به کارگیری این روشها برای تصاویر سه‌بعدی مبتنی بر نمایش سطح سه‌بعدی به صورت مستقیم قابل دسترس نیست و پیاده‌سازی سخت‌افزاری این تصاویر به دلیل چندوجهی بودن آنها دارای ملاحظات و پیچیدگی خاصی است.

هموارسازی تصویر یکی از مهمترین قسمت‌ها در کاربردهای پردازش‌تصویر و بینایی‌ماشین است. هموارسازی تصاویر در کاربردهای متعددی از پردازش تصویر و بینایی ماشین، از جمله در شناسایی شئ [۱، ۲]، تطبیق تصویر [۳]، قطعه‌بندی‌تصویر [۴] و بازیابی تصاویر [۵] بطور گسترده استفاده می‌شود. اگرچه پیاده‌سازی سخت‌افزاری فیلترهای هموارساز در مراجع متعددی بررسی شده‌است[۱۰-۶] ولی بسط آن به تصاویر سه‌بعدی کمتر صورت گرفته‌است. در تصاویر سه‌بعدی ابزارهایی که مدلهای سه‌بعدی را تهیه می‌کنند ممکن است دچار نویز شوند و برای اندازه‌گیری‌ها و پردازش‌ها مشکل ایجاد کنند، بنابراین به الگوریتم‌های سریع و کارآمد به‌صورتی که ساختار هندسی مدل حفظ شود نیاز است[۱۱].

براساس بررسی‌های انجام شده، گرچه برخی پیاده‌سازی‌های سخت‌افزاری برای استخراج ویژگی و هموارسازی تصاویر سه‌بعدی با نمایش حجمی و تصاویر سه‌بعدی با روش فاصله صورت گرفته، ولی به پیاده‌سازی سخت‌افزاری فیلتر هموارسازی تصاویر سه‌بعدی با نمایش مش‌بندی کمتر پرداخته شده‌است. هدف این مقاله پیاده‌سازی فیلترهای هموارساز برای تصاویر سه‌بعدی با نمایش سطح سه‌بعدی است. پیاده‌سازی سخت‌افزاری ساختارهای نمایش سطح سه‌بعدی به دلیل موازی بودن و داشتن نقاط زیاد به افزایش سرعت پردازش کمک شایانی می‌کند. علاوه بر آن، کاهش مصرف توان از مزایای اصلی پیاده‌سازی سخت‌افزاری در سامانه‌های نهفته است که استفاده از آنها را در وسایلی همانند روباتها که مصرف توان در آنها مهم است، اجتناب ناپذیر می‌سازد. روش پیشنهادی، علاوه بر FPGA می‌تواند در سامانه‌های مبتنی بر ASIC[۵] نیز پیاده‌سازی شود. لذا در این مقاله به پیاده‌سازی سخت‌افزاری فیلتر هموارسازی تصاویر و شکل‌های سه‌بعدی با بازنمایی مش پرداخته شده و معماری‌های جدیدی در این زمینه ارائه می‌شود. به صورت خلاصه نوآوریهای این مقاله به شرح زیر هستند:

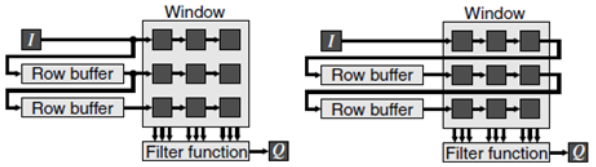
دو معماری جدید برای فیلتر میانگین با دو ساختار متفاوت، ۱-ساختار مختصات و ۲- ساختار طول‌بردار معرفی می‌شود.

دو معماری جدید برای اعمال فیلتر مبتنی بر دو ساختار متفاوت شامل، مختصات و طول بردار ارائه می‌شود.

معماری‌های پیشنهاد شده با زبان سخت‌افزاری Verilog پیاده‌سازی و بوسیله نرم‌افزارهای ISE وVivado شبیه‌سازی و سنتز شده و با تصاویر متعدد آزمایش می‌شوند.

۲- کارهای مرتبط

همانطور که ذکر شد، عموم روشهای پیاده‌سازی سخت‌افزاری بر روشهای هموارسازی فیلتر دوبعدی متمرکز هستند. در رویکرد نرم‌افزاری برای هموارسازی، تصویر ورودی و خروجی در بافر فریمها[۶] ذخیره می‌شوند. الگوریتم برای هر پیکسل خروجی تکرار می‌شود، پیکسل‌های داخل پنجره دوباره تعیین شده و تابع هموارسازی بر روی آنها اعمال می‌شود. این روش برای پیاده‌سازی سخت‌افزاری مناسب نبوده و در پیاده‌سازی سخت‌افزاری از روش شکل ۱ استفاده می‌شود. این روش بر این مبنا که لغزاندن پنجره روی تصویر معادل لغزاندن تصویر روی پنجره است، شکل گرفته است. بافرها می‌توانند بصورت موازی با پنجره قرار بگیرند یا از آنجایکه پنجره از یک سری ثبات شیفت‌دهنده[۷] تشکیل شده‌است، ، به صورت سری با پنجره قرار بگیرند. بافرهای موازی باید کمی بلندتر و عریض‌تر باشند (به اندازه عرض کامل تصویر). مزیتی که بافرهای موازی دارند این است که آنها مستقل از پنجره و فیلتر هستند. اگر منابع محدود باشد و بافر بیش از حد بزرگ باشد، تصویر بجای اینکه سطرهای تصویر را اسکن کند، می‌تواند ستونها را اسکن کند. در این صورت تصویر به صورت نوارهای عمودی بخش‌بندی شده و پردازش می‌شوند. گرچه معماری شکل ۱ می‌تواند در پردازش‌های تصاویر سه‌بعدی با نمایش حجمی و فاصله به کار گرفته شود ولی در نمایش سطح مش به دلیل ساختار چندوجهی امکان استفاده از این روش وجود ندارد.



(ب) (الف)

شکل ۱- نحوه قرار گیری بافرها با پنجره؛ (الف) سری با پنجره، (ب) موازی با پنجره [۷]

در مرجع [۸]، پیاده‌سازی سخت‌افزاری یک فیلتر گاوسی بررسی شده‌است. این فیلتر از فرآیندهای ضرب و جمع بین هسته و تصویر تشکیل شده‌است. که تصویر بوسیله ماتریسی که مقادیر آن بین ۰ تا ۲۵۵ است (۸ بیت) بیان می‌شود. هسته، بصورت یک ماتریس مربعی که مقادیر آن بین ۰ تا ۱ است در نظر گرفته شده است. در این روش یک تصویر ۵۱۲×۵۱۲ را بعنوان ورودی به FPGA داده شده است که تصویر به یک شکل برداری تبدیل می‌شود. برای کانوال[۸] کردن تصویر در یک هسته۳×۳ یک پنجره ۳×۳ از تصویر اصلی استخراج شده و سپس عناصر پنجره در هسته ضرب شوند و در آخر با هم جمع شوند و سپس یک مقدار بعنوان پیکسل خروجی معرفی می‌شود.

ارتباطات در تصاویر سه‌بعدی می‌تواند با یک ماتریس تنک[۹] بیان شود. برای یک تصویر سه‌بعدی با بیان سطح مش‌بندی با n گره، ماتریس تنک یک ماتریس n*n است که درایه سطر i و ستون j آن در صورت یک بودن بیانگر اتصال گره i به گره j است. در مرجع [۱۲]، پیاده‌سازی سخت‌افزاری ضرب ماتریس تنک مورد بررسی قرار گرفته و معماری سخت‌افزاری برای این منظور ارائه می‌شود. در این معماری، ماتریس تنک با استفاده از روش CSR[10] به صورت فشرده بازنمایی و ذخیره شده است. معماری اولیه شامل یک درخت دودویی[۱۱] با k گره برگ[۱۲] می‌باشد. هر گره برگ شامل یک ضرب کننده ممیز شناور و یک ثبات به منظور ذخیره داده ورودی می‌باشد. هر یک از k-1 برگ درخت، شامل یک جمع کننده ممیز شناور می‌باشد. جمع کننده (k-2)ام اصلی ترین برگ درخت است. خروجی درخت، یک مدار خلاصه‌ساز[۱۳] است که وظیفه جمع‌آوری اطلاعات میانی را برعهده دارد. یک واحد کنترل نیز برای کنترل پورت‌های ورودی خروجی و همچنین عملیات‌های ضرب و جمع در نظر گرفته شده‌است. در مرجع [۱۳] ابتدا روشهای محاسباتی ماتریس تنک بررسی و چالش‌های

[5] Application-specific integrated circuit
[6] Frame buffers
[7] Shift register
[8] Convolution
[9] Sparse
[10] Compressed sparse row
[11] Binary tree
[12] Leaf node
[13] Reduction circuit

مختلف پیاده‌سازی FPGA آن معرفی شده است. پس از آن، انواع شتاب‌دهنده‌های پیشرفته مبتنی بر FPGA را که برای محاسبات ماتریس تنک طراحی شده‌اند، معرفی و تحلیل شده است. علاوه بر این، یک تحلیل مقایسه‌ای از این شتاب‌دهنده‌ها انجام شده که شامل معیارهایی مانند نرخ فشرده‌سازی، توان عملیاتی و استفاده از منابع است.

Kim و همکارانش یک سیستم اندازه‌گیری شکل سه‌بعدی با استفاده از سیستم روی تراشه، که می‌تواند محاسبات پیچیده را به‌صورت موازی پردازش کند، پیشنهاد کردند که به‌طور مؤثری سرعت اندازه گیری را افزایش می‌دهد [۱۴].

Castro-Pareja و همکارانش یک معماری سخت‌افزاری برای ثبت بی‌درنگ تصاویر سه‌بعدی ارائه می‌دهند [۱۵]. آنها از روش مبتنی بر اطلاعات متقابل برای ثبت تصویر استفاده کرده و ترکیبی از معماری خط لوله و موازی را برای این منظور استفاده کردند. لیکن معماری آنها برروی تصاویر سه‌بعدی با نمایش حجمی متمرکز است.

در مرجع [۱۶] یک پردازشگر بیدرنگ و قابل پیکربندی مجدد برای پردازش تصاویر سه‌بعدی پیشنهاد می‌شود. این پردازشگر برای تصاویر سه‌بعدی با نمایش فاصله طراحی شده است که از سنسورهای اندازه‌گیری فاصله استفاده می‌کند و برای پردازش تصاویر سه‌بعدی با نمایش سطح مناسب نیست.

درمرجع [۱۷] یک معماری برای پیاده‌سازی FPGA بازسازی تصاویر سه‌بعدی پیشنهاد شده است. این معماری نیز برای بازسازی تصاویر سه‌بعدی با نمایش فاصله است.

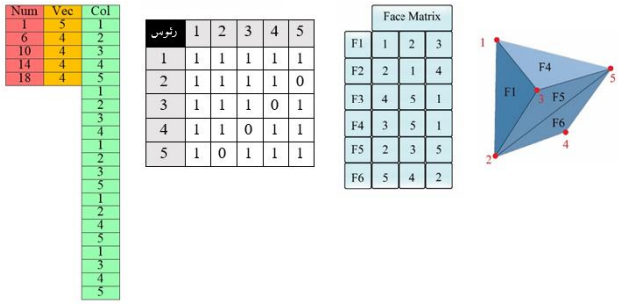
۳- معماریهای پیشنهادی

بیان تصاویر سه‌بعدی به صورت مش، دارای دو قسمت یکی فهرست رئوس وجه‌ها و دیگری مختصات رئوس است. معماریهای پیشنهادی در این مقاله دارای دو بخش است. در بخش اول یک مرحله پیش پردازش انجام می‌شود و اطلاعات مش به یک بازنمایی تنک تبدیل می‌شود. ما از دو ساختار برای این منظور استفاده کردیم که شامل ساختار طول‌بردار و ساختار مختصات است. در مورد بخش دوم عملیات هموارسازی روی مختصات رئوس انجام می‌شود. ما از روش ممیز ثابت برای هموارسازی استفاده می‌کنیم. لذا مختصات رئوس در عدد ۱۰۰۰ ضرب شده و مختصات به اعداد ممیز ثابت تبدیل می‌شوند.

ما برای هموارسازی تصاویر سه‌بعدی از فیلترهای هموارساز گاوسی و میانگین استفاده می‌کنیم و برای هر یک از آنها دو معماری پیشنهاد می‌کنیم و این دو معماری را از نظر سرعت و مصرف منابع‌های موجود در FPGA با هم مقایسه می‌کنیم. معماری‌های پیشنهاد شده بر اساس دو روش از روش‌های بازنمایی ماتریس تنک است.

۳ -۱- بازنمایی تنک اطلاعات مش

در این بخش فهرست رئوس وجه‌ها، که به آن ماتریس وجه نیز گفته می‌شود تبدیل به یک ماتریس تنک می‌شود. بعد از ایجاد ماتریس تنک باید آن را به صورت فشرده بازنمایی و به قالب‌های مناسب برای پردازش سخت‌افزاری تبدیل کنیم. ما دو قالب استاندارد برای بیان ماتریس تنک در نظر گرفته‌ایم. قالب اول، ساختار مختصات است که از دو بردار سطر (Row) و ستون (Col) تشکیل شده‌است. قالب استاندارد بعدی ما ساختار طول‌بردار است که از دو ماتریس Vec و Col تشکیل شده‌است. برای راحتی پیاده‌سازی سخت‌افزاری، یک بردار به ساختار طول‌بردار هم با عنوان num اضافه کردیم. شکل۲ ، یک شکل سه‌بعدی را به همراه ماتریس وجه و ماتریس تنک آن نشان می‌دهد. در ساختار مختصات، ماتریس تنک با دو بردار سطر (Row) و ستون (Col) بیان می‌شود که بیانگر شماره سطر و ستونهای عناصر غیر از صفر هستند. در ساختار طول بردار بهبودیافته که در شکل ۲-(ج) نشان داده شده است. بردار ستون(Col) شماره ستون عناصر غیرصفر ماتریس تنک را نشان می‌دهد و ماتریس Vec هم تعداد عناصر غیرصفر، در هر سطر از ماتریس تنک را نشان می‌دهد. به عنوان مثال در سطر ۱ از شکل ۲-(ج) همه ستونها دارای مقدار یک هستند لذا عدد ۵ به عنوان اولین درایه در بردار Vec ذخیره شده است. این عدد بیانگر این است که در ستون col بعد از ۵ عدد، اطلاعات سطر دوم از ماتریس رئوس شروع می‌شود. به همین صورت درایه دوم در بردار Vec برابر با ۴ است که بیانگر غیر صفر بودن ۴ درایه در سطر دوم ماتریس تنک است. ما در پیاده‌سازی موازی ماتریس Col را به چند قسمت تبدیل نموده و به‌صورت موازی آن را پردازش می‌کنیم، در نتیجه به یک بردار در کنار بردارهای‌های Col و Vec نیاز داریم (بردار num) که بیانگر شماره خانه شروع از بردار Col، برای پردازش است.



Num	Vec	Col
1	5	1
6	4	2
10	4	3
14	4	4
18	4	5

رئوس	1	2	3	4	5
1	1	1	1	1	1
2	1	1	1	1	0
3	1	1	1	0	1
4	1	1	0	1	1
5	1	0	1	1	1

Face Matrix			
F1	1	2	3
F2	2	1	4
F3	4	5	1
F4	3	5	1
F5	2	3	5
F6	5	4	2

(ج) (پ) (ب) (الف)

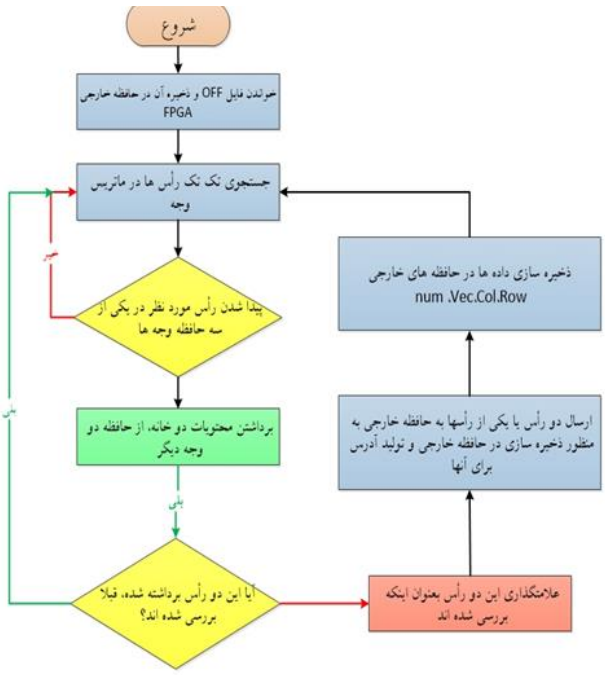
شکل ۲- الف) تصویر سه‌بعدی، ب) ماتریس وجه، پ) ماتریس تنک، ج) ساختار طول‌بردار بهبود یافته.

- **معماری پیشنهادی اول برای بازنمایی تنک اطلاعات مش**

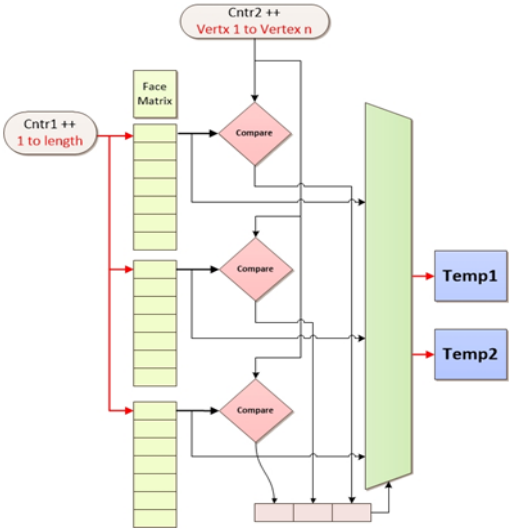
در این معماری فرض می‌شود که ماتریس وجه یک مدل مش مثلثی که از سه ستون تشکیل شده‌است، در سه حافظه ذخیره شده‌است. شکل ۳ روند کلی معماری پیشنهادی اول برای بازنمایی تنک اطلاعات مش را نشان می‌دهد.

در معماری پیشنهادی اول، باید ارتباطات بین رئوس کشف شوند که هر راس به چه راس های دیگری متصل است بنابراین از یک شمارنده که بیانگر راس مورد نظر جهت کشف ارتباطات آن می‌باشد استفاده میکنیم این شمارنده حافظه‌های سه‌گانه وجه (با طول length) برای راس مورد نظر پیمایش می‌شود که در شکل ۴ نشان داده شده اند. اگر در هر آدرسی از حافظه‌های سه‌گانه راس مورد نظر پیدا شد، مقادیر دو حافظه دیگر (با همان آدرس) را در دو رجیستر ذخیره می‌کنیم که بیانگر راسهای متصل شده به راس مورد نظر هستند. با توجه به اینکه اطلاعات اضلاع در وجههای مختلفی تکرار می‌شود در این مرحله مقادیر تکراری نادیده گرفته می‌شوند.

ما در حافظه خارجی، حافظه‌هایی برای ذخیره ساختار تنک تدارک دیده‌ایم، پس در هر کلاک بسته به حالت مدار، مقادیر دو رجیستر ، همچنین رأس مورد بازرسی شده را به حافظه‌های خارجی، انتقال می‌دهیم. حافظه‌های خارجی شامل اطلاعات بردارهای Row و Col در ساختار مختصات و همچنین بردارهای Col، Vec و num در ساختار طول بردار هستند.



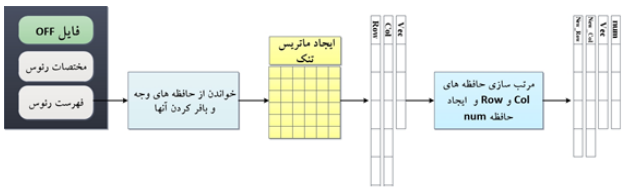
شکل۳- فلوچارت کلی معماری پیشنهادی اول برای بازنمایی تنک اطلاعات مش



شکل ۴- نحوه پیمایش شمارنده‌ها و استخراج رئوس از ماتریس وجه‌ها در معماری پیشنهادی اول برای بازنمایی تنک اطلاعات مش

- **معماری پیشنهادی دوم برای بازنمایی تنک اطلاعات مش**

روند اصلی معماری بدین صورت است که در مرحله اول داده‌ها را از حافظه مربوط به وجه‌ها می‌خوانیم و بافر می‌کنیم. در مرحله دوم ماتریس تنک آن را تشکیل می‌دهیم، در مرحله سوم برای هر یک از بردارهای Row، Col و Vec یک حافظه خارجی در نظر گرفته و داده‌ها را طبق آدرس‌های تولید شده در آن حافظه‌ها قرار می‌دهیم. سپس دوباره داده‌ها را از حافظه‌های خارجی Col و Row خوانده و مرتب نموده و عناصر صفر آن را حذف می‌کنیم و آدرس‌های مربوط به داده‌های ماتریس num را تولید می‌کنیم و در مرحله آخر داده‌ها را مطابق با آدرس‌های تولید شده در حافظه خارجی new_Row ، new_Col و num قرار می‌دهیم. روند اصلی این معماری در شکل ۵ نشان داده‌است که هر مرحله در یک کلاک انجام می‌شود و فرایند حالت خط لوله دارد.



شکل ۵: روند کلی برنامه مربوط به معماری پیشنهادی دوم برای بازنمایی تنک اطلاعات مش

ابتدا باید خانه های حافظه‌های سه‌گانه وجه‌ها (Face1، Face2 و Face3) خوانده شوند بدین منظور نیاز به یک شمارنده داریم که به‌عنوان آدرس خوانش حافظه‌های سه‌گانه وجه‌ها عمل می‌کند. با هر کلاک یک خانه از حافظه‌های سه‌گانه خوانده شده و بافر می‌شوند بنابراین ما در هر کلاک، سه مقدار بافر شده از رئوس را داریم(temp1~3). برای اینکه بفهمیم چه رئوسی را قبلا بررسی کرده‌ایم و دوباره آنها را بررسی نکنیم، باید یک ماتریس تنک نیز تشکیل دهیم. برای این منظور یک حافظه دو بعدی n×n به نام sparse_memory نیاز داریم. سپس مراحل زیر را برای بدست آوردن ماتریس تنک اجرا می‌کنیم:

- ✓ temp1 به temp2 متصل است یا خیر؟ اگر sparse_memory[temp1][temp2] برابر صفر بود یعنی این دو رأس قبلا بررسی نشده‌اند، پس ابتدا باید این موقعیت و موقعیت عکس آن یعنی sparse_memory[temp2][temp1]، را برابر یک قرار دهیم.
- ✓ temp1 به temp3 متصل است یا خیر؟ اگر sparse_memory[temp1][temp3] برابر صفر بود یعنی این دو رأس قبلا بررسی نشده‌اند ، پس ابتدا باید این موقعیت و موقعیت عکس آن یعنی sparse_memory[temp3][temp1] را برابر یک قرار دهیم.
- ✓ temp2 به temp3 متصل است یا خیر؟ اگر sparse_memory[temp2][temp3] برابر صفر بود یعنی این دو رأس قبلا بررسی نشده‌اند، پس ابتدا باید این موقعیت و موقعیت عکس آن یعنی sparse_memory[temp3][temp2]، را برابر یک قرار دهیم.

در این مرحله باید ساختارهای مختصات و طول بردار تشکیل شوند. بدین منظور یک حافظه به نام Cntr_mem به اندازه تعداد رئوس، یعنی n در نظر بگیریم، تا بفهمیم هر رأس به چه تعداد رأس دیگر متصل است و در نهایت سه حافظه خارجی col، numو Vec تشکیل می‌شود.

۳ -۲- معماری سخت‌افزاری برای پیاده‌سازی فیلتر هموارساز میانگین

در این بخش به نحوه پیاده‌سازی فیلتر میانگین برای دو ساختار طول‌بردار و مختصات خواهیم پرداخت.

- **پیاده‌سازی فیلتر میانگین با استفاده از ساختار مختصات**

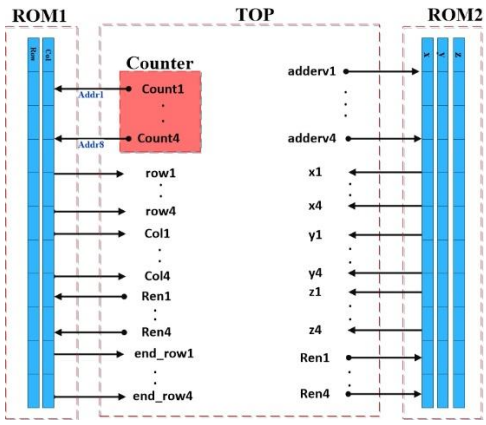
برای پیاده‌سازی فیلتر میانیگن با استفاده از ساختار مختصات، فرض می‌شود بردارهای مربوط به ساختار مختصات یعنی Col و Row را در حافظه خارجی ROM1 و همچنین مختصات رئوس را در حافظه خارجی ROM2 قرار دارد. ROM2 حاوی سه حافظه x، y و z می‌باشد. همچنین به منظور افزایش سرعت پردازش وانتقال اطلاعات، ROM2 را به ۴ قسمت تبدیل شده و ۴ قسمت بصورت موازی اجرا می‌شود.

بنابراین برای حافظه خارجی ROM1، ۴ درگاه بعنوان آدرس ، ۴ درگاه برای انتقال داده‌ها از حافظه Row و ۴ درگاه برای انتقال داده‌ها از حافظه Col به

ماژول اصلی در نظر می‌گیریم. همچنین ۴ درگاه برای اجازه دسترسی به حافظه ROM1تحت عنوان Ren1~4 در نظر می‌گیریم. یعنی اگر Ren مربوطه یک بود اجازه انتقال داده‌ها از حافظه ROM1به ماژول اصلی داده می‌شود.

برای حافظه خارجی ROM2 نیز ۴ درگاه آدرس، ۴ درگاه برای انتقال داده‌ها از حافظه x، ۴ درگاه برای انتقال داده‌ها از حافظه y و ۴ درگاه برای انتقال داده‌ها از حافظه z و همچنین ۴ در گاه برای صدور اجازه دسترسی به داده‌ها برای انتقال به ماژول اصلی تحت عنوان Ren در نظر گرفته شده‌است. شکل ۶ نحوه ارتباط حافظه ROM1 و ROM2 را با ماژول اصلی را نشان می‌دهد.

ما به کمک یک ماژول شمارنده، به ازای هر کلاک، ۴ داده از حافظه‌های Row و Col را می‌خوانیم. یک حافظه هم به نام index با اندازه تعداد رئوسی که داریم (Length_V) ایجاد می‌کنیم، این حافظه برای این است که بدانیم هر رأس به چه تعداد رأس دیگر متصل است. سپس به ازای هر داده‌ای که از حافظه Row می‌خوانیم، خانه متناظرش را در حافظه index را به اندازه یک واحد افزایش می‌دهیم.



شکل۶- نحوه ارتباط حافظه‌های خارجی ROM1 و ROM2 با ماژول اصلی.

برای خواندن داده‌ها از حافظه ROM2 باید آدرس آنرا، برابر داده‌هایی که از حافظه Col می‌خوانیم، قرار دهیم. بنابراین با آدرس‌دهی به حافظه ROM2 مختصات x، y و z بدست می‌آید. همچنین سه حافظه به نام outx، outy و outz به اندازه تعداد رئوس در نظر می‌گیریم. هنگامیکه در حال بررسی یک رأس هستیم و مختصات رئوس متصل به آن در اختیار ما قرار می‌گیرد آنها را مطابق رابطه (۱) با هم جمع می‌کنیم.

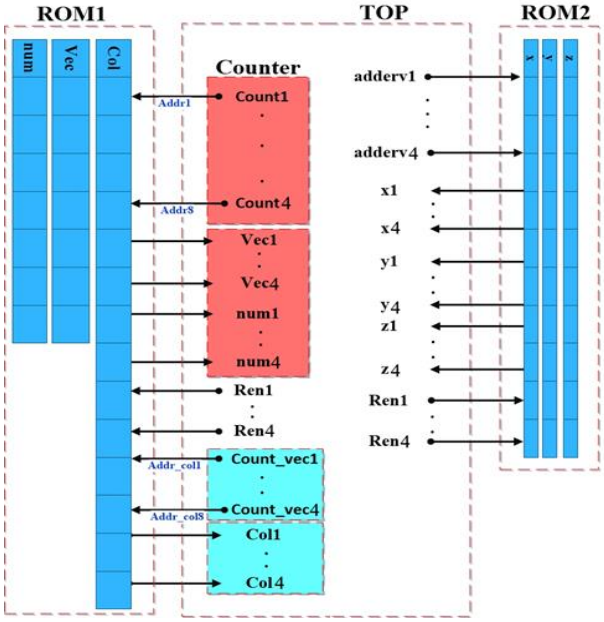
هنگامیکه بررسی رأس مربوطه پایان یافت و بررسی رأس بعدی شروع شد نتایج حاصله رابطه (۱) را بر تعداد رأس قبلی طبق رابطه (۲) تقسیم نموده تا مختصات هموار شده رأس مربوطه حاصل شود. سپس این مختصات توسط درگاه های خروجی برای ذخیره در حافظه خارجی قرار می‌گیرند.

$$\begin{cases} outx[i] = outx[i] + x_i \\ outy[i] = outy[i] + y_i \\ outz[i] = outz[i] + z_i \end{cases}, i = 1,2,3,4 \quad (۱)$$

$$\begin{cases} xf[i] = \dfrac{x[i]}{index[i]} \\ yf[i] = \dfrac{y[i]}{index[i]} \\ zf[i] = \dfrac{z[i]}{index[i]} \end{cases}, i = 1,2,3,4 \quad (۲)$$

- **پیاده‌سازی فیلتر میانگین برای ساختار طول بردار**

در این پیاده‌سازی نیز از دو حافظه خارجی به نام ROM1 و ROM2 بهره می‌بریم. ROM2 مانند بخش قبلی حاوی مختصات (x، y و z) رئوس بوده اما ROM1 حاوی ماتریس‌های Col، Vec و num است. در اینجا هم حافظه ROM1 را به ۴ قسمت تبدیل می‌کنیم و این ۴ قسمت باهم بصورت موازی پردازش می‌شوند. برای خواندن محتویات حافظه ROM1، به یک ماژول شمارنده نیاز داریم که مشابه معماری قبلی است. بنابراین ۴ خروجی شمارنده Addr1~4 به عنوان، آدرس حافظه ROM1 استفاده می‌شوند. با این آدرس فقط به خانه‌های حافظه‌های num و Vec دسترسی داریم و برای خواندن از حافظه Col از آدرس دیگری استفاده می‌شود که در ادامه توضیح داده می‌شود. شکل ۷ نحوه ارتباط با حافظه های ROM1 وROM2 را نشان می‌دهد.



شکل ۷- نحوه ارتباط با حافظه های ROM1 وROM2

شکل ۸ بلوک دیاگرام فیلتر میانگین با ساختار طول بردار و نحوه عملکرد آن را نشان می‌دهد. ما چهار عدد ماژول شمارنده به نام counter_vec1~4، نیز استفاده می‌کنیم. ما هنگامیکه خانه‌های حافظه ROM1 را می‌خوانیم، ۴ داده از هر کدام از حافظه‌های Col، Vec و num را در اختیار داریم، بنابراین num1~4 را بعنوان نقطه شروع شمارش به ماژول counter_vec1~4 می‌دهیم. هدف این است که با توجه به مقدار Vec1~4، که از حافظه ROM1 خوانده شده‌است، به این اندازه از خانه‌های حافظه Col را در دسترس داشته باشیم، نقطه شروع ادرس برای این حافظه در num1~4 قرار دارد. بنابراین نقطه شروع را به درگاه start ماژول شمارنده num1~4 انتقال داده و شمارنده شروع به شمارش می‌کند و نتایج را در درگاه‌های counter_vec1~4 قرار می‌دهد.

همچنین ما یک درگاه برای اطلاع از زمانیکه شمارش تمام شد به نام end_count_vec تعریف کردیم. این درگاه(end_count_vec) هنگامیکه یک شود، بدین مفهوم است که ما تمام مختصات رئوس متصل به آن رأس را استخراج کردیم و می‌توانیم رئوس متصل به رأس بعدی را استخراج کنیم. البته این ماژول یک درگاه برای فعالسازی به نام En_counter_vec دارد، که هر موقع یک بود ماژول فعال می‌شود. بنابراین ابتدا شمارنده اول(Counter) یک واحد می‌شمارد و طبق آن ما یک خانه از حافظه‌های num و Vec را برمی‌داریم و بر اساس آن، شمارنده دوم یعنی Counter_Vec را راه اندازی می‌کنیم.

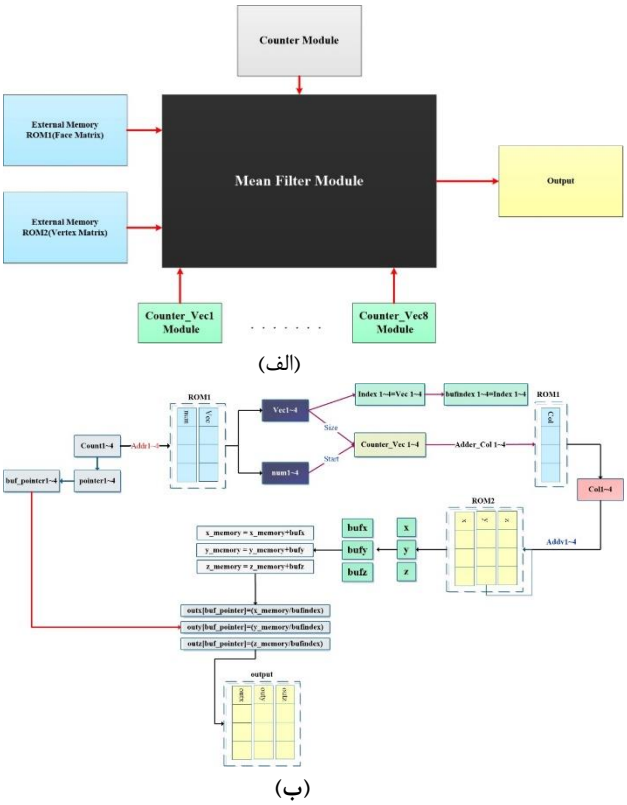
با قرار دادن خروجی ماژولcounter_vec، در آدرس Addr_col، به خانه‌های

حافظه Col دسترسی پیدا کرده و در نتیجه مقادیر x، y و z به دست می‌آیند. همچنین count1~4 را نیز بافر می‌کنیم یعنی ثباتهای pointer1~4 را برابر count1~4 قرار می‌دهیم تا در کلاک‌های بعدی از آن استفاده کنیم.

ثبات‌های vec1~4، بیانگر تعداد رئوس متصل به رأس در حال بررسی است و باید برای تنظیم زمانبندی آنها را نیز بافر نمائیم، برای این کار ثباتهای index1~4 را در نظر می‌گیریم سپس آنها را برابر با vec1~4 قرار می‌دهیم، سپس index1~4 را یک مرحله بافر می‌کنیم و مقادیر آن را در ثبات‌های bufindex1~4 ذخیره شوند.

همچنین سه حافظه به نام outx، outy و outz به اندازه تعداد رئوس در نظر می‌گیریم. هنگامیکه counter_vec1~4 شروع به شمارش می‌کند، محتویات خانه‌های Col را بر می‌داریم و به مختصات x، y و z آنها دسترسی پیدا می‌کنیم و برای تنظیم زمانبندی آنها را یک مرحله بافر می‌کنیم یعنی آنها را در ثبات‌های bufx، bufy و bufz ذخیره می‌کنیم، همچنین ثباتهای x_memory1~4، y_memory1~4 و z_memory1~4 برای ذخیره حاصل جمع مختصات رئوس در نظر می‌گیریم.

پس از پایان بررسی رأس مورد نظر و اتمام جمع مختصات رئوس، متصل به آن باید حاصل جمع مختصات رئوس را بر تعداد رئوس متصل به آن، تقسیم کنیم و نتایج را در حافظه‌های outx، outy و outz ذخیره کنیم.



شکل ۸- (الف) بلوک دیاگرام فیلتر میانگین با ساختار طول بردار. (ب) نحوه عملکرد فیلتر میانگین با ساختار طول بردار

۳-۳- معماری سخت‌افزاری برای پیاده‌سازی فیلتر هموارساز گاوسی

برای محاسبه تابع گاوسی نیاز به محاسبه تابع نمایی برای یک عدد منفی است. برای محاسبه تابع نمایی ما از روش جدول جستجو [۱۸] استفاده کردیم. هدف در طراحی جدول مراجعه، عدم درگیر شدن در محاسبات ممیز شناور بوده و جدول طوری طراحی شده است که عدد ورودی نقش آدرس حاصل تابع نمایی خودش را بازی کند و ما فقط با مراجعه به آدرس، حاصل را برداشته و روی آن پردازش نمائیم. برای رسیدن به این هدف به شیوه زیر عمل شد.

- ✓ برای رسیدن به دقت مطلوب و پردازش بصورت ممیز ثابت، عددی را که می‌خواهیم تابع نمایی آن را محاسبه کنیم در $۱۰^۶$ ضرب می‌کنیم عدد را تبدیل به قالب دودویی تبدیل می‌کنیم.
- ✓ بیت‌های ۱۲-۱۵ و ۱۶-۱۹و۲۰-۲۳ آن را جدا می‌کنیم.
- ✓ سپس آنها را کنار هم قرار می‌دهیم و آن را تبدیل به دسیمال می‌کنیم. عدد بدست آمده، بیانگر آدرسی از حافظه است که حاوی حاصل تابع نمایی می‌باشد.

جدول ۱ نتیجه اعمال روند بالا را به تعدادی عدد نمونه نشان می‌دهد. حال برای اینکه ما در برنامه عدد دسیمال را به آدرس حافظه بدهیم و حاصل نمایی آن را استخراج کنیم باید جدول ۱ را از راست به چپ پیاده‌سازی کنیم، یعنی ابتدا اعداد ۱ تا ۱۸۵۷ را تولید کنیم، سپس ۱۲ بیت صفر به آن اضافه می‌کنیم تا ۲۴ بیت شود، سپس آن را تبدیل به دسیمال می‌کنیم و نتایج را به برنامه متلب می‌دهیم سپس در برنامه متلب نتایج را بر $۱۰^۶$ تقسیم می‌کنیم سپس تابع نمایی منفی آنها را بدست می‌آوریم و برای اینکه نتایج را تبدیل به ممیز ثابت شود، آنها را به‌صورتی که مضربی از ۱۰۰۰ باشند در می‌آوریم و سپس آنها را گرد می‌کنیم. دلیل اینکه جدول از ۱ تا ۱۸۵۷ را تولید می‌کنیم این است که اگر روند بالا را روی ۱۸۵۷ به بعد، اجرا کنیم، حاصل تابع نمایی صفر می‌شود برای همین از این عدد به بعد را تولید نمی‌کنیم.

جدول ۱: نحوه محاسبه تابع نمایی برای یک عدد منفی

x	$Exp(-x)$	$x \times 10^6$	*Binary*	*Decimal*
0	1	0	0000 0000 0000 0000 0000 0000	0
0.01	0.99	10000	0000 0000 0010 0111 0001 0000	2
.	.	.	.	.
.	.	.	.	.
1	0.368	1000000	0000 0001 1000 0110 1010 0000	24
.	.	.	.	.
.	.	.	.	.
2	0.135	2000000	0001 1110 1000 0100 1000 0000	488
.	.	.	.	.
.	.	.	.	.
3	0.049	3000000	0010 1101 1100 0110 1100 0000	732
.	.	.	.	.
.	.	.	.	.
4	0.018	4000000	0011 1101 0000 1001 0000 0000	976
.	.	.	.	.
.	.	.	.	.

- **پیاده‌سازی فیلتر گاوسی با استفاده از ساختار مختصات**

مشابه فیلتر میانگین‌گیری، ماتریس‌های مربوط به ساختار مختصات یعنی Col و Row را در حافظه خارجی ROM1 قراردارند. برای دسترسی به حافظه خارجی ROM1، مانند فیلتر میانگین‌گیری از ماژول شمارنده (Counter) با همان تنظیمات استفاده می‌کنیم و حافظه خارجی ROM1 را به ۴ قسمت تقسیم نموده و این ۴ قسمت را باهم بصورت موازی پردازش می‌کنیم. حافظه خارجی ROM2 نیز شامل مختصات رئوس می‌باشد و مانند معماری فیلتر میانگین با

شمارنده‌ی که برایش تعریف شده‌است، قابل دسترسی است. تغییری که ROM2 معماری فیلتر گاوسی نسبت به معماری فیلتر میانگین دارد این است که ما علاوه بر درگاه‌های x1~4، y1~4 و y1~4 نیاز به درگاه‌های دیگری برای محاسبه تابع گاوسی داریم، بنابراین درگاه‌های آدرس Addrx_exp1~4، Addry_exp1~4 و Addrz_exp1~4 برای استخراج مختصات x، y و z بو درگاه های x_exp1~4، y_exp1~4 و z_exp1~4 برای انتقال آنها به ماژول اصلی استفاده می‌شود.

همچنین ما یک حافظه خارجی به نام ROM3 داریم که این حافظه حاوی همان جدول جستجوی ما است که توسط متلب تولید شده‌است که نحوه تولید آن را در بخش قبل توضیح دادیم. این حافظه دارای ۴ درگاه آدرس به نام‌های addr_expx1~4، addr_expy1~4 و addr_expz1~4 و ۴ درگاه که برای انتقال داده‌ها به ماژول اصلی به نام‌های dataoutx1~4، dataouty1~4 و dataoutz1~4 در نظر گرفته‌ایم.

نحوه عملکرد معماری فیلتر گاوسی با ساختار مختصات در شکل ۹ نشان داده شده است. هنگامیکه شمارنده یک واحد را شمارش می‌کند، ما یک مقدار از حافظه‌های Row و Col را می‌خوانیم. مانند روش فیلتر میانگین به ازای هر Row که می‌خوانیم خانه متناظر با آن Row را در حافظه index، یک واحد افزایش می‌دهیم. برای تنظیم زمانبندی، Row را نیز بافر می‌کنیم (buffrow1~4). سپس آدرس‌های addr_row1~4، که مربوط به حافظه ROM2 هستند را برابر buffrow1~4 قرار می‌دهیم، این عمل باعث می‌شود که مختصات x، y و z متناظر با buffrow1~4 را در اختیار ما قرار دهد.

آدرس‌های Addrv1~4 حافظه ROM2 را برابر Col1~4 قرار می‌دهیم در نتیجه مختصات x1~4، y1~4 و z1~4 در اختیار ما قرار می‌گیرد. سپس طبق رابطه (۳) محاسبات مربوطه را انجام می‌دهیم. در واقع ما در رابطه فیلتر گاوسی قسمت توان تابع نمایی را محاسبه می‌کنیم.

$$\begin{cases} gausx[i]=[(x[i]-x\_exp[i])^2]>>1 \\ gausy[i]=[(y[i]-y\_exp[i])^2]>>1, \ i=1,2,3,4 \\ gausz[i]=[(z[i]-z\_exp[i])^2]>>1 \end{cases} \quad (۳)$$

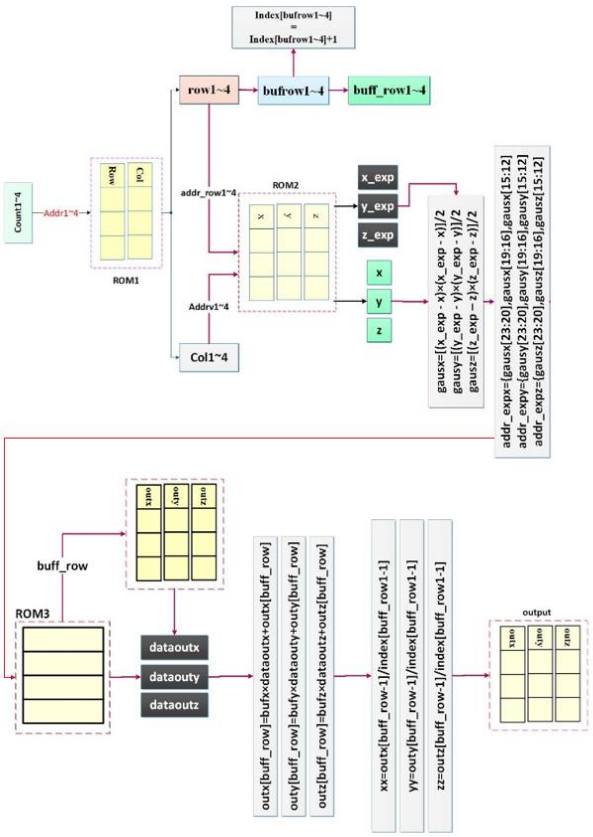
در رابطه (۳) یک شیفت به راست داریم که معادل تقسیم بر دو است. سپس آدرس‌های حافظه ROM3، را طبق برابر بیت‌های ۱۲تا ۲۴ رابطه (۳) قرار می‌دهیم. یعنی ما به آدرسی از حافظه ROM3 رجوع می‌کنیم که حاصل تابع نمایی آن عدد در آنجا قرار دارد.

با آدرس‌دهی حافظه ROM3، dataoutx1~4، dataouty1~4 و dataoutz1~4 در اختیار ما قرار می‌گیرد. البته این مقادیر همراه با یک کلاک تاخیر نسبت به x1~4، y1~4 و z1~4 خواهند بود، بنابراین لازم است x1~4، y1~4 و z1~4 را یک مرحله بافر کنیم

در این مرحله سه حافظه outx، outy و outz به اندازه تعداد رئوس تعریف و طبق رابطه (۴) مختصات رئوسی که به یک رأس هستند را در وزن گاوسی که بدست آوردیم، ضرب می‌کنیم و آنها را با هم جمع می‌کنیم.

$$\begin{cases} outx[buff\_rowi] = bufxi \times dataout1 + outx[buff\_rowi] \\ outy[buff\_rowi] = bufyi \times dataout1 + outy[buff\_rowi] \\ outz[buff\_rowi] = bufzi \times dataout1 + outz[buff\_rowi] \end{cases} \quad (۴)$$

buff_row1~4 در رابطه (۴) بافر شده buff_row1~4 می‌باشد. هنگامیکه کلیه محاسبات یک راس انجام شد، باید حاصل جمع را بر تعداد رئوس متصل به آن رأس تقسیم شود که روند مشابه فیلتر میانگین است.



شکل۹- نحوه عملکرد معماری فیلتر گاوسی با ساختار مختصات

- **پیاده‌سازی فیلتر گاوسی برای ساختار طول بردار**

در این ساختار سه حافظه خارجی به نام‌های ROM1، ROM2 و ROM3 وجود دارند. در حافظه خارجیROM1 ما سه بردار Col، Vec و num داریم. مانند بخش‌های قبل این حافظه نیز به ۴ قسمت تبدیل می‌شود و این قسمت‌ها با هم بصورت موازی پردازش می‌شوند. در این حافظه ۴ درگاه آدرس addrf1~4 که با آدرس‌دهی می‌توانیم ۴ داده از حافظه num و Vec استخراج کنیم، بنابراین نیاز به ۴ درگاه num1~4 برای انتقال داده‌های حافظه num به ماژول اصلی، داریم همچنین ۴ درگاه Vec1~4 نیز برای انتقال داده‌های حافظه Vec به ماژول اصلی نیاز داریم. برای انتقال داده‌های حافظه Col به ماژول اصلی نیز به ۴ درگاه آدرس به نام addrf_col1~4 و ۴ درگاه Col1~4 نیاز داریم. حافظه خارجی ROM2 شامل سه حافظه x، y و z است و دارای ۴ درگاه آدرس addrv1~4 برای انتقال داده‌های حافظه‌های x، y و z از طریق ۴ درگاه x1~4، ۴ درگاه y1~4 و ۴ درگاه z1~4 به ماژول اصلی می‌باشد. همچنین ۴ درگاه آدرس به نام addr_exp1~4 برای انتقال داده‌های حافظه‌های ، y و z از طریق ۴ درگاه x_exp1~4 ، ۴ درگاه y_exp1~4 و ۴ درگاه z_exp1~4 به ماژول اصلی در نظر گرفته‌ایم. برای حافظه خارجی ROM3 که همان جدول جستجوی تابع نمایی ما است، دقیقا همان حافظه ROM3 است که در بخش قبل بطور کامل در مورد آن توضیح دادیم. در این طرح مانند معماری بخش قبل درگاه‌های آدرس ROM1 یعنی addrf1~4 را برابر خروجی‌های ماژول شمارنده اول، یعنی Count1~4، قرار داده می‌شوند، در نتیجه داده‌های حافظه‌های Vec و num از طریق درگاه های Vec1~4 و num1~4 در اختیار ما قرار می‌گیرند. Vec1~4را طی دو مرحله بافر می‌کنیم. همچنین خروجی‌های ماژول Counter را (count1~4) را یک مرحله توسط ثبات‌های pointer1~4 بافر می‌کنیم. سپس از ۴ ماژول Counter_Vec برای پیمایش حافظه Col، استفاده می‌کنیم.

درگاه‌های start1~4، ماژول‌هایCounter_Vec1~4، را برابر داده‌های

درگاه‌های num1~4 قرار می‌دهیم همچنین درگاه‌های size1~4 این ماژول‌ها را نیز برابر داده‌های درگاه‌های Vec1~4، قرار می‌دهیم.

ماژول‌های Counter_Vec1~4 شروع به شمارش کرده و داده‌های حافظه Col را می‌خواند. بنابراین درگاه‌های آدرس حافظه ROM2 را برابر Col1~4 قرار می‌دهیم. در نتیجه مختصات x1~4، y1~4 و z1~4 متناظر با آن در اختیار ما قرار می‌گیرد. این مختصات‌ها را یک مرحله در ثبات‌های bufx1~4، bufy1~4 و bufz1~4 بافر می‌کنیم. همچنین خروجی‌های بافر شده Counter، یعنی pointer1~4 را به آدرس‌های addr_exp1~4 حافظه ROM2 می‌دهیم تا مختصات x، y و z متناظر با آنها را از طریق درگاه‌های x_exp1~4، y_exp1~4 و z_exp1~4 در اختیار ما قرار بگیرد. در واقع با این کار مختصات رأسی را در اختیار داریم که می‌خواهیم آن را هموار کنیم و مختصات رئوس متصل به آن در درگاه‌های x1~4، y1~4 و z1~4 قرار دارد. البته باید pointer1~4 را یک مرحله با ثبات‌های buf_x_exp1~4 بافر کنیم. حال مختصات رأسی که در حال بررسی هستیم را از رأسی که به آن متصل است، کم می‌کنیم و به توان ۲ می‌رسانیم و نتیجه را بر ۲ تقسیم می‌کنیم که exptx1~4، expty1~4 و exptz1~4 حاصل می‌شود. حال باید تابع نمایی اعداد بدست آمده را محاسبه کنیم برای این کار، آدرس‌های حافظه ROM3 را باید برابر exptx1~4، expty1~4 و exptxz1~4 قرار دهیم. در نتیجه حاصل نمایی آن اعداد بدست می‌آید و dataoutx1~4، dataouty1~4 و dataoutz1~4 در اختیار ما قرار می‌گیرد، اکنون باید این وزن‌های این رئوس را در مختصات خودشان ضرب کنیم و در آخر همه را با هم جمع کنیم. بقیه مراحل مشابه حالت ساختار مختصات بوده و هنگامیکه بررسی یک رأس تمام شد و خواستیم محاسبات رأس بعدی را شروع کنیم، حاصل جمع مرحله قبل را بر تعداد رئوس متصل به آن رأس تقسیم می‌کنیم.

۴- نتایج پیاده‌سازی

شبیه سازی‌های لازم در این مقاله با نرم افزار Vivado صورت گرفته و معماریهای پیشنهادی با مدلهای سه‌بعدی متعددی آزمایش شدند و نتایج را با خروجی‌های برنامه‌های متلب مقایسه شدند. مدلهای سه‌بعدی که در شبیه سازی‌های انجام شده مورد استفاده قرار گرفته است دارای ۲۹۹ رأس و ۵۶۲ وجه بودند. برای شبیه سازی از چیپ سری Virtex-7 (xc7vx485tffg1157) شرکت زایلینکس استفاده کردیم.

۴ -۱- نتایج سنتز معماری‌ها

- **نتایج سنتز معماری‌های بازنمایی تنک اطلاعات مش**

بر اساس نتایج سنتز، در معماری پیشنهادی اول به فرکانس ۳۷۳ مگاهرتز دست پیدا کردیم. در دومین معماری پیشنهادی معماری دارای قابلیت عملکرد تا فرکانس ۳۸۱ مگاهرتز دست دارد. جدول ۲ نتایج سنتز و منابع استفاده شده در معماری‌های بازنمایی تنک تصاویر سه‌بعدی را نشان میدهد.

- **نتایج سنتز معماری‌های فیلترهای هموارساز**

بر اساس نتایج سنتز، در معماری فیلتر میانگین با ساختار بازنمایی طول بردار و مختصات به فرکانس کاری ۱۹ مگاهرتز دست پیدا نمودیم. فرکانس کاری در در معماری فیلتر گاوسی نیز ۱۹/۲ و ۱۹/۳ مگاهرتز برای ساختار بازنمایی مختصات و طول بردار بود. جدول های ۳ و ۴ نتایج سنتز و منابع استفاده شده در معماری‌های فیلترهای هموارساز میانگین و گاوسی را نشان می‌دهد. نتایج جدولهای ۳ و ۴ نشان می‌دهد که فیلترهای هموارساز با ساختار بازنمایی طول بردار منابع کمتری را نسبت به ساختار مختصات نیاز دارند.

جدول ۲: نتایج سنتز و منابع استفاده شده در معماری‌های بازنمایی تنک اطلاعات مش

معماری	معماری پیشنهادی اول		معماری پیشنهادی دوم	
منابع	تعداد استفاده شده	درصد استفاده شده	تعداد استفاده شده	درصد استفاده شده
LUT[14]	۹۷	٪۳	۵۱	٪۲
FF[15]	۱۰۸	٪۲	۶۶	٪۱
IO[16]	۹۲	٪۱۵/۳۳	۱۹۹	٪۳۳/۱۷
BUFG[17]	۱	٪۳/۱۳	۱	٪۳/۱۳

جدول ۳: نتایج سنتز و منابع استفاده شده در معماری‌های فیلتر هموارساز میانگین

معماری	معماری فیلتر میانگین با ساختار طول بردار		معماری فیلتر میانگین با ساختار مختصات	
منابع	تعداد استفاده شده	درصد استفاده شده	تعداد استفاده شده	درصد استفاده شده
LUT	۱۰۳۱۶۱	٪۳۳/۹۸	۱۸۰۳۷۵	٪۵۹/۴۱
FF	۳۰۹۷۴	٪۵/۱	۳۹۳۸۰	٪۶/۴۹
IO	۱۰۸	٪۱۸	۴۹۴	٪۸۲/۳۳
BUFG	۱	٪۳/۱۳	۱	٪۳/۱۳

جدول ۴: نتایج سنتز و منابع استفاده شده در معماری‌های فیلتر هموارساز گاوسی

معماری	معماری فیلتر گاوسی با ساختار طول بردار		معماری فیلتر گاوسی با ساختار مختصات	
منابع	تعداد استفاده شده	درصد استفاده شده	تعداد استفاده شده	درصد استفاده شده
LUT	۸۷۷۱۵	٪۲۸/۸۹	۱۸۲۵۸۰	٪۶۰/۱۴
FF	۳۰۸۹۱	٪۵/۰۹	۳۹۴۹۷	٪۶/۵۰
DSP	۳۶	٪۱/۲۹	۲۴	٪۰/۸۶
IO	۱۰۸	٪۱۸	۴۹۴	٪۸۲/۳۳
BUFG	۱	٪۳/۱۳	۱	٪۳/۱۳

۴ -۲- نتایج شبیه‌سازیها

به منظور آزمایش عملکرد درست معماریهای پیشنهادی، عملیات شبیه‌سازی بر روی ۲۰ تصویر سه‌بعدی با سطح مش‌بندی انجام شده است و نتایج بدست آمده با نتایج شبیه‌سازی‌های همان تصاویر در متلب مقایسه شده‌است. تصاویر و مدلهای سه‌بعدی که در شبیه سازی‌های انجام شده مورد استفاده قرار گرفته است دارای ۲۹۹ رأس (گره) و ۵۶۲ وجه هستند و از اینترنت جمع آوری شده‌اند.

- **شبیه‌سازی معماری‌های مربوط به بازنمایی تنک اطلاعات مش**

برای بازنمایی فشرده، فایلی که دارای ۳×۲۹۹ داده مربوط به رئوس است

[14] Look-Up Table
[15] Flip Flop
[16] Input/Output
[17] Global Clock Buffer

و دارای ۳×۵۶۲ داده مربوط به وجه است، که نتایج شبیه‌سازی برای این معماری‌ها در جدول ۵ نشان داده شده‌است.. فرکانسی که در مرحله سنتز برای این معماری‌ها بدست آمد به ترتیب برابر ۳۷۳ و ۳۸۱ مگاهرتز برای معماریهای پیشنهادی اول و دوم بود.

جدول ۵: نتایج شبیه‌سازی معماری‌های پیشنهادی برای بازنمایی تنک اطلاعات مش

معماری پیشنهادی	فرکانس (MHz)	دوره تناوب هر کلاک (ns)
اول	۳۷۳	۲/۶۸۰
دوم	۳۸۱	۲/۶۲۰

168338

4441

- **شبیه‌سازی معماری‌های فیلترهای هموارساز**

نتایج شبیه سازی فیلترهای هموارساز از نظر سرعت پردازش تعداد کلاک و زمان فرآیند در جدول ۶ نشان داده شده است.

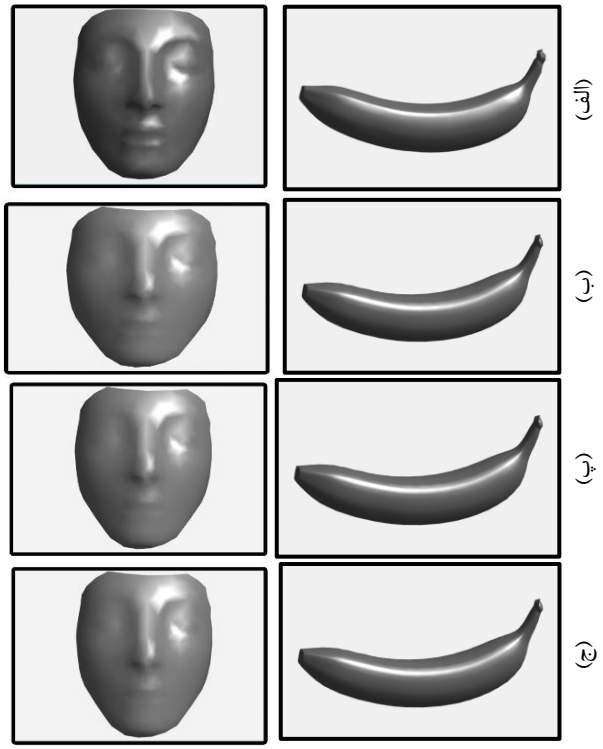
جدول ۶: نتایج شبیه‌سازی فیلترهای هموارساز از نظر سرعت پردازش

نوع فیلتر	فرکانس (MHz)	دوره تناوب هر کلاک (ns)	تعداد کلاک	زمان مورد نیاز (us)	نرخ پردازش (تصویر در ثانیه)
میانگین با ساختار مختصات	۱۹	۵۲/۳۸	۵۱۱	۲۶/۷۶	۳۷۳۶۹
میانگین با ساختار طول بردار	۱۹/۲	۵۱/۹۱	۶۸۶	۳۵/۶۱	۲۸۰۸۱
گاوسی با ساختار مختصات	۱۹	۵۲/۳۸	۵۱۲	۲۶/۸۱	۳۷۲۹۹
گاوسی با ساختار طول بردار	۱۹/۳	۵۱/۷۳	۶۸۷	۳۵/۲۹	۲۸۳۳۶

ما معماریهای پیشنهادی را با ۲۰ تصویر سه‌بعدی بوسیله FPGA شبیه سازی کرده و نتایج آن را با برنامه متلب مقایسه کردیم. در شکل‌های ۱۰ و ۱۱، نمونه‌هایی از تصاویر سه‌بعدی و نتایج اعمال فیلترهای هموارساز میانگین و گاوسی نشان داده شده است. برای مقایسه، نتایج اعمال فیلترهای هموارساز با استفاده از پیاده‌سازی متلب و پیاده‌سازی سخت‌افزاری نشان داده شده است. ما برای مقایسه، میانگین خطای مطلق[18] و خطای میانگین مربعات[19] (پیاده‌سازی FPGA نسبت به متلب) را مطابق با رابطه‌های (۵) و (۶) محاسبه کردیم. جدول ۷ میانگین خطای مطلق و خطای میانگین مربعات را برای معماریهای مختلف نشان می‌دهد.

$$MAE = \frac{\sum_{i=1}^{n} \left| y_i - x_i \right|}{n} \quad (۵)$$

$$MSE = \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n} \quad (۶)$$



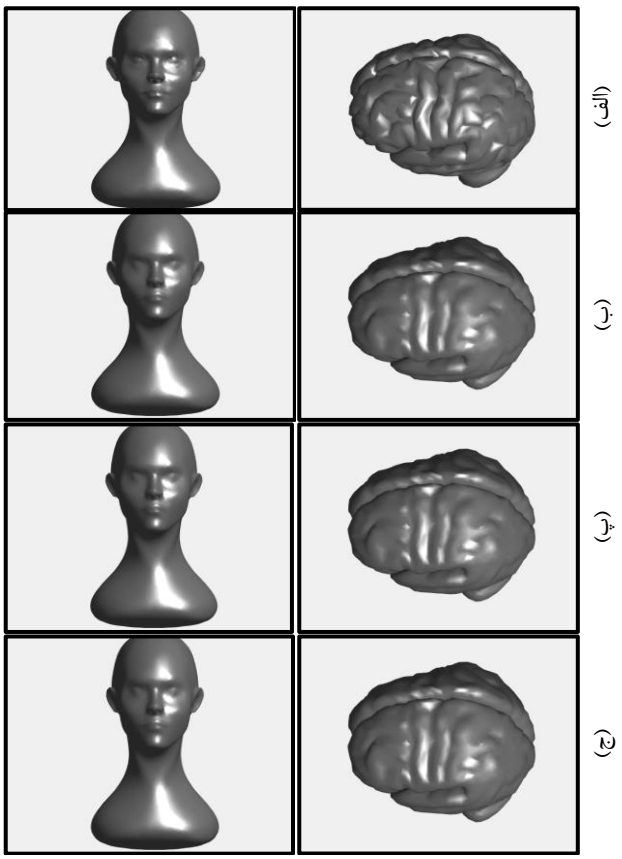
شکل۱۰- نتایج اعمال فیلتر هموارساز میانگین به دو نمونه تصویر سه‌بعدی با استفاده از پیاده‌سازیهای سخت‌افزاری و پیاده‌سازی متلب، الف) تصویر اصلی، (ب) خروجی فیلتر میانگین با پیاده‌سازی متلب (پ) خروجی پیاده‌سازی فیلتر میانگین با ساختار مختصات، (ج) خروجی پیاده‌سازی فیلتر میانگین با ساختار با ساختار طول بردار

- **مقایسه با سایر روشها**

همانطوریکه در کارهای مرتبط بررسی شد، عموم روشهای پیاده‌سازی سخت‌افزاری تصاویر سه‌بعدی، بر روی تصویربرداری فاصله و تصاویر سه‌بعدی با روش حجم‌نگاری نگاری تمرکز دارد. در این مقاله، روشی برای پیاده‌سازی سخت‌افزاری فیلترهای هموارساز برای تصاویر سه‌بعدی با نمایش سطح مش‌بندی ارائه شده است. بر اساس بررسی‌های صورت گرفته، پیاده‌سازی سخت‌افزاری مشابهی برای فیلترهای هموارساز میانگین در تصاویر سه‌بعدی با نمایش سطح مش‌بندی ارائه نشده است. لذا امکان مقایسه با سایر روشها وجود نداشت. تاحد اطلاع نویسندگان، روش ارائه شده دراین مقاله، اولین معماری ارائه شده برای پیاده‌سازی فیلترهای هموارساز تصاویر سه‌بعدی با نمایش سطح مش‌بندی است.

[18] Mean absolute error (MAE)

[19] Mean squared error (MSE)



شکل۱۱- نتایج اعمال فیلتر هموارساز گاوسی به دو نمونه تصویر سه‌بعدی با استفاده از پیاده‌سازیهای سخت‌افزاری و پیاده‌سازی متلب، (الف) تصویر اصلی، (ب) خروجی فیلتر گاوسی با پیاده‌سازی متلب (پ) خروجی پیاده‌سازی فیلتر گاوسی با ساختار مختصات، (ج) خروجی پیاده‌سازی فیلتر گاوسی با ساختار با ساختار طول بردار

جدول ۷: میانگین خطای مطلق و خطای میانگین مربعات بین پیاده‌سازی FPGA و پیاد سازی متلب برای معماریهای مختلف

معماری	میانگین خطای مطلق	میانگین خطای مربعات
فیلتر میانگین با ساختار مختصات	9×10^{-4}	6.7×10^{-6}
فیلتر میانگین با ساختار طول بردار	9.6×10^{-4}	54.4×10^{-6}
فیلتر گاوسی با ساختار مختصات	1.9×10^{-3}	1.2×10^{-4}
فیلتر گاوسی با ساختار طول بردار	1.6×10^{-3}	1.7×10^{-4}

۵- نتیجه گیری

در این مقاله معماری‌هایی برای هموارسازی تصاویر و مدلهای سه‌بعدی ارائه شد. معماریهای پیشنهادی شامل دو بخش شامل ۱- بازنمایی اطلاعات مش به صورت ماتریس تنک و ۲- اعمال فیلترهای هموارسازی بودند. ما دو معماری برای بازنمایی تنک تصویر سه‌بعدی و دو معماری متفاوت مبتنی بر ساختار مختصات و طول‌بردار برای فیلتر میانگین‌گیری و گاوسی ارائه کردیم. سپس به منظور آزمایش عملکرد درست معماریهای پیشنهادی، حدود ۲۰ مدل سه‌بعدی را بعنوان ورودی به معماری‌های ذکر شده، دادیم و نتایج حاصله را با نتایج حاصله شبیه‌سازی همین تصاویر در متلب مقایسه کردیم و میانگین خطای مطلق و خطای میانگین مربعات آنها را نسبت به نتایج برنامه متلب نیز بدست آوردیم.

در مقایسه معماریهای ارایه شده برای بازنمایی تنک اطلاعات مش، معماری پیشنهادی دوم تقریبا نصف معماری اول منابع را مصرف می‌کند، فرکانس کاری آنها تقریبا با هم برابر هستند اما معماری دوم نسبت به معماری اول، تقریبا ۳۸ برابر سریعتر است. لذا معماری پیشنهادی دوم گزینه بهتری برای پیاده‌سازی است. در پیاده‌سازی فیلتر میانگین، معماری مبتنی بر طول‌بردار حدود ۲۶ درصد کمتر از معماری مبتنی بر مختصات منابع را مصرف می‌کند. از نظر فرکانس تقریبا هر دو معماری در یک محدوده فرکانسی کار می‌کنند اما از نظر سرعت و پردازش هر فریم بر ثانیه، معماری مبتنی بر مختصات ۱/۳ برابر نسبت به معماری مبتنی بر طول‌بردار، سریعتر می‌باشد. در پیاده‌سازی فیلتر گاوسی، معماری مبتنی بر طول‌بردار حدود ۳۳ درصد کمتر از معماری مبتنی بر مختصات منابع را مصرف می‌کند. از نظر فرکانس تقریبا هر دو معماری در یک محدوده فرکانسی کار می‌کنند اما از نظر سرعت و پردازش هر فریم بر ثانیه، معماری مبتنی بر مختصات ۱/۳ برابر نسبت به معماری مبتنی بر طول‌بردار، سریعتر می‌باشد.

مراجع

[1] Seyed Saber Mohammadi, Yiming Wang, and Alessio Del Bue. "Pointview-gcn: 3D shape classification with multi-view point clouds." In 2021 IEEE International Conference on Image Processing (ICIP), pp. 3103-3107. IEEE, 2021.

[2] Viktoria Ehm, Paul Roetzer, Marvin Eisenberger, Maolin Gao, Florian Bernard, and Daniel Cremers. "Geometrically Consistent Partial Shape Matching." In 2024 International Conference on 3D Vision (3DV), pp. 914-922. IEEE Computer Society, 2024.

[3] Xingyu Jiang, Jiayi Ma, Guobao Xiao, Zhenfeng Shao, and Xiaojie Guo. "A review of multimodal image matching: Methods and applications." Information Fusion, vol. 73, pp. 22-71, 2021.

[4] Masoumeh Mohseni, Mehdi Ezoji, Reza Ghaderi "Image Segmentation based on Normalized Cut from the Perspective of the Discriminant Information" Tabriz Journal of Electrical Engineering vol. 46, no. 1, pp. 303-310, May 2016.

[5] Maryam Taghizadeh, and Abdollah Chalechale, "A model to image retrieval based on multiple-Query," Tabriz Journal of Electrical Engineering vol. 47, no. 3, pp.893-903, 2017.

[6] M. Hanumantharaju, M. Ravishankar, and D. Rameshbabu, "Design and FPGA implementation of an 2D Gaussian surround function with reduced on-chip memory utilization," in International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 604-609, 2013.

[7] D. G. Bailey, Design for embedded image processing on FPGAs. John Wiley & Sons, 2011.

[8] F. Cabello, J. León, Y. Iano, and R. Arthur, "Implementation of a fixed-point 2D Gaussian Filter for Image Processing based on FPGA," in Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), pp. 28-33 2015.

[9] Debasish Mukherjee and Susanta Mukhopadhyay. "Fast hardware architecture for fixed-point 2D Gaussian filter." AEU-International Journal of Electronics and Communications, vol.105, pp. 98-105, 2019.

[10] Wang, Biaobiao, and Qiang Xiang. "Fast median filter image processing algorithm and its FPGA implementation." Frontiers in Signal Processing, vol. 4, no. 4, pp. 88-94, 2020.

[11] H. Badri, M. El Hassouni, and D. Aboutajdine, "Kernel-based Laplacian smoothing method for 3D mesh denoising," in International Conference on Image and Signal Processing, pp. 77-84, 2012.

[12] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays, pp. 63-74. 2005.

[13] Liu, Yajing, Ruiqi Chen, Shuyang Li, Jing Yang, Shun Li, and Bruno da Silva. "FPGA-Based Sparse Matrix Multiplication Accelerators: From State-of-the-art to Future Opportunities." ACM Transactions on Reconfigurable Technology and Systems, vol. 17, no. 4, pp. 1-37, 2024.

[14] Tae-Hyeon Kim, Hyunki Lee, and Seung-Ho Ok. "Implementation of an FPGA-Based 3D Shape Measurement System Using High-Level Synthesis." Electronics 13, no. 16, p. 3282, 2024.

[15] C. R. Castro-Pareja, J. M. Jagadeesh and R. Shekhar, "FAIR: a hardware architecture for real-time 3-D image registration," IEEE Transactions on Information Technology in Biomedicine, vol. 7, no. 4, pp. 426-434, Dec. 2003.

[16] F. Chen, R. Ying, J. Xue, F. Wen and P. Liu, "A Configurable and Real-Time Multi-Frequency 3D Image Signal Processor for Indirect Time-of-Flight Sensors," in IEEE Sensors Journal, vol. 22, no. 8, pp. 7834-7845, 15 April15, 2022.

[17] Y. Wu, A. Aßmann, B. D. Stewart and A. M. Wallace, "Energy Efficient Approximate 3D Image Reconstruction," IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 4, pp. 1854-1866, 1 Oct.-Dec. 2022.

[18] D. Seidner, "Efficient implementation of 10Y lookup table in FPGA," 2009 IEEE International Symposium on Industrial Electronics, Seoul, Korea (South), pp. 686-689, 2009.